Supporting Cooperation in the SPADE-1 Environment

Sergio Bandinelli, *Member, IEEE Computer Society*, Elisabetta Di Nitto, *Student Member, IEEE, Computer Society*, and Alfonso Fuggetta, *Member, IEEE*

Abstract—Software development is a cooperative activity that heavily relies on the quality and effectiveness of the communication channels established within the development team and with the end-user. In the software engineering field, several Software Engineering Environments (SEE) have been developed to support and facilitate software development. The most recent generation of these environments, called Process-Centered SEE (PSEE), supports the definition and the execution of various phases of the software process. This is achieved by explicitly defining cooperation procedures, and by supporting synchronization and data sharing among its users.

Actually, cooperation support is a theme of general interest and applies to all domains where computers can be exploited to support human-intensive activities. This has generated a variety of research initiatives and support technology that is usually denoted by the acronym CSCW (Computer Supported Cooperative Work).

PSEE and CSCW technologies have been developed rather independently from each other, leading to a large amount of research results, tools and environments, and practical experiences. We argue that we have reached a stage in technology development where it is necessary to assess and evaluate the effectiveness of the research efforts carried out so far. Moreover, it is important to understand how to integrate and exploit the results of these different efforts.

The goal of the paper is to understand which kind of basic functionalities PSEE can and should offer, and how these environments can be integrated with other tools to effectively support cooperation in software development. In particular, the paper introduces a process model we have built to support a cooperative activity related to anomaly management in an industrial software factory. The core of the paper is then constituted by the presentation and discussion of the experiences and results that we have derived from this modeling activity, and how they related to the general problem of supporting cooperation in software development. The project was carried out using the SPADE PSEE and the ImagineDesk CSCW toolkit, both developed at Politecnico di Milano and CEFRIEL during the past four years.

Index Terms—Cooperative activities, CSCW, PSEE, software development environments, software processes.

1 Introduction

The development of nontrivial software applications is rarely carried out by a single programmer. Rather, it demands for the participation of (even large) teams of people. The interaction and exchange of information among team members is a key factor that determines the success or failure of any development initiative. Indeed, social and interpersonal aspects play a fundamental role in software development [19], [41]. Besides software development, there are many other examples of human activities where cooperation is a key issue. For this reason, a variety of research initiatives have been started with the goal of providing a general support to cooperative activities. This research field is traditionally called CSCW (Computer Supported Cooperative Work) [16] and has generated a large amount of methodological and technological results. In particular, several tools and environments have been developed, that make it possible to effectively support different kinds of cooperative activities. These activities can be classified in two broad categories, depending on temporal considerations [24]:

- *Asynchronous cooperation*: users work at different times on the same information; or, also, users exchange messages and data to coordinate their own work. For instance, the organization of meetings require users to exchange information on their preferred time schedules, and to share some documents to provide the background for the discussion.
- *Synchronous cooperation*: users "are and work" in the *same* (virtual) workspace at the *same* time. For instance, review meetings demand for on-line communication and discussion among people, who may be playing different roles, and may be located at dispersed sites. They need to share different types of information (e.g., documents and informal notes), and also to interact on-line, using a variety of media (e.g., video, audio, and shared boards).

An orthogonal tentative taxonomy of the main facets of cooperation is presented by Yang in [53]. He cites three main concepts:

- *Coordination*: (semi-)automatic sequencing of process steps.

- *S. Bandinelli is with the European Software Institute, Parque Tecnologico de Zamudio 204, E-48170 Bizkaia, Spain. E-mail: sergio.bandinelli@esi.es.*
- *E. Di Nitto is with Politecnico di Milano and CEFRIEL, Via Emanueli 15, 20126 Milano, Italy. E-mail: dinitto@elet.polimi.it.*
- *A. Fuggetta is with Politecnico di Milano and CEFRIEL, P.zza Leonardo da Vinci, 32 20133 Milano, Italy. E-mail: fuggetta@elet.polimi.it.*

Manuscript received Apr. 14, 1995.
Recommended for acceptance by D.E. Perry.
For information on obtaining reprints of this article, please send e-mail to: transse@computer.org, and reference IEEECS Log Number S95756.

- *Communication*: exchange of information among users.
- *Collaboration*: creation and management of shared information.

Coordination is an activity typically supported using asynchronous facilities (e.g., meeting schedulers), even if multiusers games (e.g., multiusers doom) manage coordination in a synchronous way. Collaboration and communication are often supported through both synchronous and asynchronous products (e.g., co-authoring systems, teleconference systems, e-mail, and bulletin boards). Table 1 combines both classifications and indicates some tools that support these activities. For example, asynchronous collaboration is supported by tools like Oval [42] and gIBIS [17]; coordination by Workflow Management Systems (WFMS) such as Regatta [51]; synchronous communication by real-time conferences and shared boards such as GroupKit [48] and ClearBoard [37]. In the reminder of the paper, we will mainly refer to the distinction between synchronous vs. asynchronous cooperation, since we believe this is the key factor to identify the relevant and critical design choices at the environment architectural level.

In the software engineering domain there has been an increasing interest in conceiving and developing SEE (Software Engineering Environments) that make it possible to explicitly specify the process that software developers are expected to follow. Support to cooperation is crucial for these environments (called PSEE or Process-Centered SEE) since software development is basically a team effort. For instance, the process to be followed to test a software system may require the testers and the developers to asynchronously share the same source code, and synchronously interact to solve an anomaly.

TABLE 1
FUNCTIONAL ASPECTS OF COOPERATION

	Asynchronous cooperation	Synchronous cooperation
Collaboration	traditional DBMS	on-line co-authoring
Communication	e-mail	teleconferencing
Coordination	agendas, meeting schedulers	multiuser games

A PSEE [27] is centered around an explicit process description, often called process model, that is defined using Process Modeling Languages (PML). These languages offer powerful capabilities to describe roles, manual and automated procedures, interaction among users, process artifacts, and constraints. The execution (enactment) of the process model within a PSEE provides support to process agents[1] in the execution of their work, for example, by offering guidance to them or by automating some parts of the process.

A PSEE can be seen as composed of three main parts [29]:

- an *enactment environment* for executing the process model;
- a *user interaction environment* (composed of tools such as compilers, editors, and control panel shells) for supporting the user's work and his interaction with the PSEE;
- a *repository* for storing process artifacts and process models as well.

The *enactment environment* executes the process model. It operates on the artifacts stored in the repository and coordinates the operations to be accomplished in the process. It can also invoke operations affecting the user interaction environment (see later on), such as launching tools, changing the state of active tools, and terminate their execution. The *user interaction environment* is the set of tools and interfaces that are visible to process agents. The process-relevant actions performed by process agents through the tools in the user interaction environment are captured by the enactment environment, that operates according to the defined process model. The *repository* stores and manages all the artifacts produced and manipulated during software development. It can be a dedicated database (e.g., an object-oriented database), the file system of the hosting environment, or even a combination of a database and the file system.

The problem of supporting cooperation in software development can be therefore tackled using different kinds of technologies. CSCW has produced a variety of tools that are supposed to support cooperation in different domains and contexts. PSEE are environments specifically conceived to support software development. From this initial discussion it is quite clear that these technologies share some commonalities, but present also significant differences. This paper tries to identify possible synergies between technology developed in the PSEE and CSCW fields, by addressing the following questions:

1) To what extent can a PSEE be used for supporting cooperative activities?
2) What kind of basic mechanisms should a PSEE offer to build different process-specific cooperation policies?
3) What are the differences and analogies between PSEE and CSCW environments?
4) Is it possible to identify reasonable strategies for integrating PSEE and CSCW environments?

To answer these questions, the paper retrospectively considers the work carried out by the authors in recent years. In particular, the paper reports on the experiences gained in building and using the SPADE-1 PSEE [7]. SPADE-1 provides a PML called SLANG, that is an extension of Petri nets, integrated with object-oriented data modeling facilities. To assess the effectiveness of SPADE-1 in supporting cooperative processes, we have modeled a significant fragment of an industrial software process, dealing with the evaluation and management of anomaly reports in a software factory. Based on this test-bed, we show how SPADE-1 can be used for supporting asynchronous cooperation. Finally, we address the issue of supporting synchronous cooperation in SPADE-1, and we outline the advantages and drawbacks of the approaches we have explored. In particular, we discuss and present three integration

1. In the paper we will use the following expressions:
- Process agents (or users): people involved in software development activities (i.e., "users" of the Software Engineering Environment).
- Process engineer: process expert who is charge of building the model of the process.
- End-users: users of the applications developed by process agents.

strategies between SPADE-1 and the ImagineDesk toolkit for CSCW applications [45]. We believe that most comments and observations that have emerged from this experience are general enough to be applicable in other contexts.

The paper is organized as follows:

- Section 2 briefly outlines the characteristics of the SPADE-1 environment, by introducing its architecture and the SLANG PML.
- Section 3 describes the process we have used as a testbed.
- Section 4 presents the SPADE-1 solution to support this process.
- Section 5 discusses the results of our experience.
- Section 6 points out the main characteristics and properties of CSCW environments and of the ImagineDesk toolkit, and presents the work we have accomplished to extend SPADE-1 in order to support synchronous cooperation.
- Finally, Sections 7 and 8 present the related work and draw some conclusions, respectively.

2 The SPADE-1 Environment

The goal of the SPADE project ([3], [6], and [7]) is to provide a software engineering environment for supporting Software Process Analysis, Design, and Enactment. The project has been carried out at CEFRIEL and Politecnico di Milano. The environment we developed, SPADE-1, is based on a process modeling language, called SLANG (SPADE Language), which is a high-level Petri net-based formalism. SLANG offers features for process modeling, enactment, and evolution. In addition, it provides suitable constructs for describing in a uniform style the interaction with tools and process agents.

The main features of the SLANG PML can be summarized as follows:

- Process models can be statically structured in a modular way using the *activity* construct. Activities (i.e., process model fragments) can be dynamically instantiated.
- Activities can be manipulated as data by other activities, i.e., SLANG supports computational reflection. This is an essential feature to support process evolution [29].
- Process artifacts, including process models, are modeled as tokens of a Petri net and are implemented as objects in an object-oriented database.
- Interaction between the PSEE and tools in the user interaction environment is modeled by using specific SLANG constructs supporting delegation of operations and detecting of external events.

In the remainder of this section we will detail the key concepts of SPADE-1 and SLANG. In particular, Section 2.1 presents the SPADE-1 architecture and describes the mechanisms offered for tools integration, while Section 2.2 provides a quick introduction to SLANG. A more detailed discussion of the SPADE-1 architecture and of SLANG can be found in [49].

2.1 SPADE-1 Architecture

SPADE-1 is the first implementation of the SPADE concepts. SPADE-1 architecture is based on the principle of separation of concerns between process model enactment and user interaction environment. It is structured in three different layers (Fig. 1): the Repository, the Enactment Environment (EE), and the User Interaction Environment (UIE) that includes the SPADE Communication Interface (SCI).

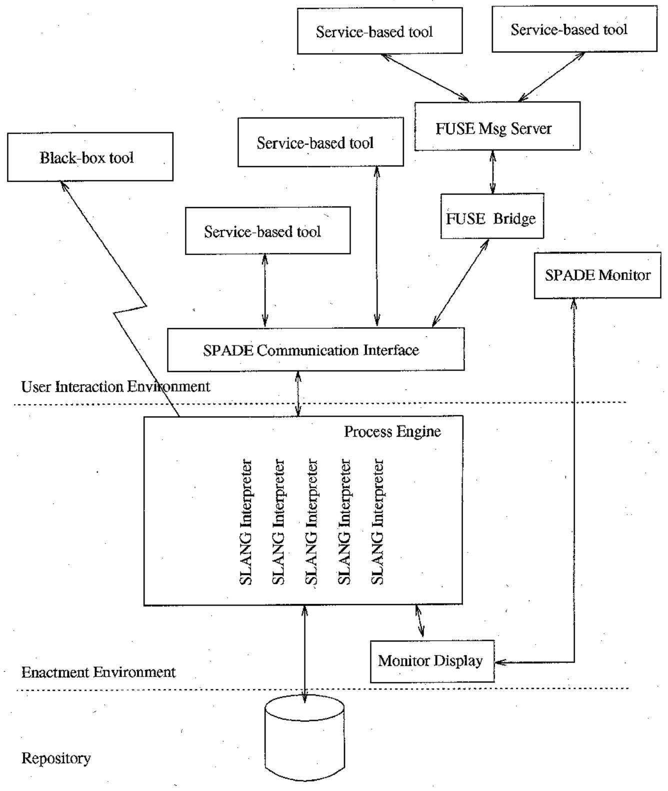


Fig. 1. SPADE-1 architecture.

The Repository stores both process model fragments and process artifacts. A process artifact can be any document or unit of information created during the software development process. The Repository is implemented on top of the object-oriented database management system O_2 [20]. The EE includes facilities for executing a SLANG specification, possibly creating and modifying process artifacts. Process model fragments and artifacts are represented in the process model as tokens. A SLANG process model is composed of different activities. During enactment, activities can be instantiated and executed. An activity instance is called *active copy*. It is possible to create many active copies, even from the same activity. Active copies are concurrently executed by different SLANG Interpreters, that are implemented as separate threads of a Process Engine. Finally, SPADE Monitor provides the users with a view of the execution state of the process model.

The UIE is the front-end of the PSEE to its users. It is composed of the tools used by the process agents involved in the development activities. SPADE-1 supplies well defined mechanisms to integrate tools from two different viewpoints:

- **Data integration**. SPADE-1 and a UIE tool are able to exchange process artifacts.
- **Control integration**. SPADE-1 is able to invoke and control functionalities (or services) provided by UIE tools.

The granularity of integration can vary depending on the characteristics of the tools and the process model being enacted. The different levels of granularity that are supported by SPADE-1 can be classified by considering the following classes of tools:

- *Black-box tools.* They are seen by the EE as single functions that receive some input and produce some output. From the control integration viewpoint, the environment manages only the invocation and the termination of these tools. From the data integration viewpoint, these tools use their own repository, typically the file system, for storing data. This means that process data stored in the O_2 database have to be explicitly converted and mirrored in the file system (or in another database) to make it possible for tools to operate on them. Black-box tools can be integrated in SPADE-1 without any modification of the source code. The integration is performed during the process model definition phase by exploiting the primitives of the SPADE PML to start a tool and detect its termination.
- *O_2-based black-box tools.* They achieve a higher level of data integration since they are able to manipulate O_2 objects directly, without any conversion. SPADE-1 can exchange data with these tools by exploiting an inter-O_2 client communication mechanism called ICCM [13].
- *Service-based tools.* They offer a programmatic interface through which it is possible to separately invoke the services they provide. Tools can invoke external services offered by other tools, or notify the environment of specific events or tool state changes. This kind of tools is highly integrated from the control viewpoint since it is possible to interleave and coordinate the execution of different tool services by properly managing the messages sent and received to/from tools. Service-based tools are basically those conceived in the FIELD environment [46] and in its commercial spin-offs DEC FUSE, Sun Tooltalk, and HP BMS.[2] Service-based tools can be integrated in SPADE-1 in two different ways: directly, using the SPADE Integration Protocol (offered by SCI, see later on), or by using COTS (Commercial-Off-The-Shelf) products, such as FUSE or Tooltalk. In the latter case, messages are managed and distributed by the message broadcasting server of the tool integration environment. All of these messages are also received by a specific SPADE component (called Bridge), which converts messages to the SPADE Integration Protocol and forwards them to SCI. SPADE-1 provides bridges for DEC FUSE, Tooltalk, DDE, and OLE2.
- *O_2 service-based tools.* They supply the highest level of control and data integration, since they support both the direct exchange of complex O_2 objects and a programmatic interface to support service-based integration.

2. We will refer to these products with the expression "tool integration environments."

Table 2 lists some tools offered by or integrated in SPADE-1, and classifies them according to the categories presented above:

1) SPADEShell is the interface through which process agents can send commands to the EE.
2) Agenda displays the list of tasks assigned to a user. It interacts with the EE to manage the list of tasks according to the policies specified in the process model.
3) ObjectEditor allows the user to edit any process artifact stored in the SPADE-1 Repository.
4) SLANG editor supports the editing and compilation of SLANG process models.
5) FUSE Editor is a service-based text editor.
6) FUSE Builder is an integrated C compiler and *make* utility.
7) *Emacs* and *cc* are an editor and a C compiler of the Unix environment.

TABLE 2
EXAMPLES OF SPADE TOOLS, TOGETHER WITH THEIR TYPES

	black-box	O_2 black-box	service-based	O_2 service-based
SPADEShell			•	
Agenda			•	
ObjectEditor		•		
SLANGEditor				•
FUSEEditor			•	
FUSEBuilder			•	
Emacs, cc	•			

SCI manages the communication between service-based tools and the EE, through the SPADE Integration Protocol. SCI is connected to all service-based tools based on the SPADE Integration Protocol, to the Bridges, and to the Process Engine executing the different SLANG Interpreters. SCI is able to redirect messages from the UIE to the enacting active copies. SCI can be dynamically configured by the EE as specified in the underlying process model (see next section).

2.2 SLANG

A SLANG process model is composed of two parts, the first one, called Process Types, defines a set of abstract data types (ADT) describing process data. The second one, called Process Activities, defines a set of activities:

$$SLANGModel = (Process\ Types,\ Process\ Activities)$$

Process Types are specified as O_2 classes and have a unique name, a type structure, and a set of operations (methods). SLANG provides the predefined type hierarchy shown in Fig. 2. The root of the hierarchy is type `Token`. Its instances are the tokens stored in SLANG nets. Process model-specific types can be created as subtypes in the subtree rooted at `ModelType`. The other subtrees contain process model independent types. In particular, the instances of type `Activity` represent activity definitions (e.g., each of them describe a specific activity); the instances of type `ActiveCopy` represent the state of the activity instances

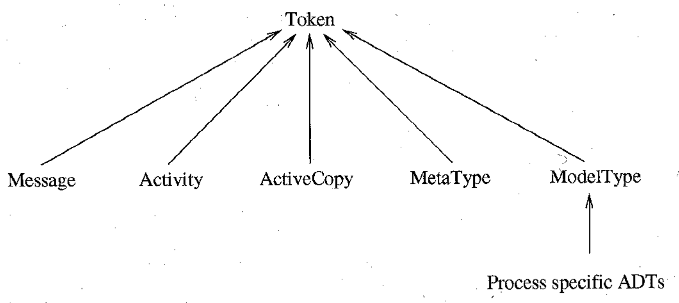


Fig. 2. Predefined SLANG type hierarchy.

being executed; the instances of `MetaType` contain type definitions; finally, instances of type `Message` are tokens modeling events occurring in the UIE. Tokens of type `Message` appear in special places, called user places, and are not produced by normal transition firings (see later on). Activities and types defining a process model can be seen and manipulated as tokens in the process model itself. It is, therefore, possible to support process evolution, by developing a process model fragment able to change the tokens representing the process model (types and activities).

Process Activities are specified as high-level Petri nets. Each activity specification is composed of an interface and an implementation. The implementation part may contain the invocation of other activities. The invocation relationship defines a hierarchical structure among activities. The root of this hierarchy is called *root activity*. Fig. 3a and 3b illustrate the interface and the implementation of a SLANG activity, respectively. The dashed box in Fig. 3b delimits the interface of the activity. In the example, places `P1`, `P2`, `P4`, `P5`, `P9`, and `P10` belong to the interface of the activity. `P1`, `P2`, `P9`, and `P10` are similar to the formal parameters defined for a procedure: when the activity is invoked, some of the places of the calling activity (the actual parameters) have to be bound to them. Transition `t1` is a starting event. Its firing causes the invocation of activity `A`. Transitions `t6`, and `t7` are *ending events*. Whenever one of them fires, the execution of the activity is terminated. Places `P4` and `P5` are called *shared places* for activity `A`. They behave as global variables, since their contents may be accessed by activity `A` and by the invoking activity. `P4` and `P5` are connected to the interface of the activity `A` through links. The implementation of activity `A` contains an invocation of activity `B`. When transition `t3` or `t4` fires, an instance of the activity `B` is invoked. In this case, `P3`, `P4`, `P5`, `P7`, `P8`, and `P12` are the actual parameters of the new instance of `B`, and `P6` is shared between `A` and `B`.

Transitions represent events. We distinguish between *white transitions*, that influence the internal state of the process model, and *black transitions* (in Fig. 3 transitions `t2` and `t9` are black transitions), that induce some changes in the UIE. Both types of transitions have a guard and an action. The guard is a boolean expression and must be verified by the tuple of tokens enabling the transition firing. The firing of the transition causes the corresponding action to be executed. The action is an O_2C code[3] fragment and it is implemented as a traditional ACID transaction. An action operates on the tokens of the enabling tuple, and produces tokens in the output places of the transition. A black transitions can be used to directly invoke the execution of a black-box tool, or to send a service request to a service-based tool. Black transition execution is asynchronous with respect to the execution of the active copy which the black transition belongs to. This means that other transitions in the activity can fire before a black transition terminates its execution, provided that they are enabled and their guard is satisfied. Tokens are placed in the output places of a black transition by a trigger that detects the termination of the tool spawned by the black transition.

In SLANG two kinds of places are defined: *normal places* (represented by single circles) and *user places* (represented by double circles). The content of a normal place may change only because of a transition firing. The content of a user place, instead, changes as a consequence of events occurring in the user interaction environment. An event in the UIE corresponds to the arrival of a message from a tool to SCI. SCI reifies external events by creating one token for each incoming message. The correspondence between user places and external events has to be explicitly specified as part of the process model. For example, black transition `t9` of Fig. 3 executes the following code fragment:

```
extAction = "ServiceRequest 0 ConfigSCI +
            * LoadFile P12";
```

`extAction` is a SLANG predefined variable. Its value is a string, which is considered by the SLANG interpreter as a command to be executed. In the example, the interpreter sends to SCI (identifier 0) a request for configuring its routing table in such a way that place `P12` of the current active copy is associated with the arrival of message "`LoadFile`" from any tool in the UIE. Character "`*`" in the string stands for "any tool." In case place `P12` has to be configured to receive messages from a specific tool, "`*`" should be replaced by the identifier of that tool. In the rest of the paper we discuss how SPADE-1/SLANG features may be exploited to support cooperative software development.

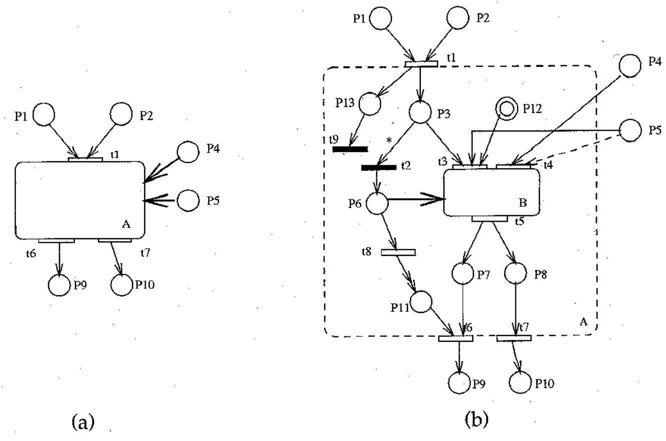


Fig. 3. SLANG activity definition.

3. O_2C is a C dialect embedding constructs to manipulate O_2 objects.

3 The Process Example

The process example discussed in this paper is a subset of a real process adopted by a telecommunication company. In [8], this process has been formally studied. The goal of that paper was to produce a formal specification of the quality manual of the company, in order to verify its soundness and consistency. In this paper, even if we use the same scenario, the goal is different. In particular, we aim at defining an *enactable* process model, able to support the company in the development process. As we outline in Section 5, the main modeling effort, in this case, has concerned the management of the interaction among the users and the tools used to guide or enforce the process, rather than the definition of the process itself.

The process we focus on deals with the management of anomalies reported during software testing and operation. When an anomaly is found, an Anomaly Report document is prepared and submitted to the manager responsible for the specific project. The manager is in charge of directing the procedures for anomaly analysis, error detection, and bug fixing. All these procedures are human-intensive. They require the project manager, the designers, and the programmers to intensively cooperate to effectively accomplish the above tasks.

In this kind of human-intensive process many of the tasks to be executed cannot be automated, but require process agents' creative work [30].

3.1 Roles in the Process

People working in the process can be associated with the following roles:

- *Project Manager* (*PM*): is responsible for the project.
- *System Administrator* (*SA*): is responsible for the management of the computing facilities used in the project.
- *Designer* (*DE*): is responsible for the design and implementation of software modules.
- *Programmer* (*PR*): is in charge of the coding activity.

3.2 Workspaces

Each user registered in the environment has a local workspace. This workspace is private and is not accessible by other process agents. Shared data is maintained in a global workspace, which is under configuration management. Process artifacts stored in the global workspace are partitioned in a hierarchy of folders. For each project, a different folder is defined, which, in turn, contains a folder for each product module (containing all the files related with the module, such as, design documents, source code files, include files, makefiles, object code files, modification reports, etc.), and other folders containing design documents, anomaly reports, modification reports, and any other document related with the project as a whole.

All of the above configuration items are identified by a name and a version number. The name determines the item position into the folders hierarchy. For example, `switchboard.callController.switch.c` is the name of the C source code of function `switch`, which is part of the `callController` module of product `switchboard`. The corresponding file is stored in folder `callController` located in folder `switchboard`. The version number has the form `release.level`. The version creator, the date of creation, and some textual comments are recorded together with each configuration item.

Users can perform operations on the global workspace according to their roles. PM may create the folder corresponding to a project, and may specify the names of DE enabled to add new module folders. In turn, DE may specify the names of the PR who can access and modify the items of a module.

Configuration items can be checked out from the global workspace. When this operation is performed, the item is locked, and a copy of it is stored in the local workspace of the user. When an item is checked in, a new version is created in the global workspace.

3.3 The Process Steps

Anomalies are usually detected during system test and operation. Each anomaly is described by an anomaly report (AR) stored under configuration management. When an AR is created, its initial state is "originated." PM considers AR and decides how to handle them. Three alternatives are possible:

- The reported anomaly is considered not to be a real anomaly (e.g., it was erroneously signaled by the end-user). The AR state is set to "rejected."
- The anomaly is recognized as relevant. The appropriate corrective actions have to be taken immediately. The AR state becomes "approved."
- The decision is delayed and the AR state becomes "postponed."

Each approved AR is passed to DE to be analyzed. Each DE considers the module(s) she/he is responsible for and determines the necessary modifications that should be implemented (if any). For each proposed module modification she/he generates a modification report (MR) with initial state "originated." When all DE have analyzed the AR, PM schedules a Configuration Control Board (CCB) meeting in order to decide which MRs have to be considered for the next release of the product. Each MR may be either "approved," i.e., the modifications have to be implemented for the next release; "postponed," i.e., the MR has to be reconsidered in the future; or "rejected," i.e., the proposed MR is not accepted. Once all MRs of an AR have been considered, the AR state is set to "defined." The modification proposed in a MR may be directly performed by the DE or it may be delegated to a PR. Upon termination, the MR state is set to "done." When all MRs associated with an AR are done, the AR state is set to "solved."

Note that the distribution of the AR to DE represents a decomposition of the original task ("Analyze AR") in multiple subtasks delegated to DE. In this case, PM and DE cooperate to accomplish the original "Analyze AR" task, by sharing a common resource, i.e., the AR. The task decomposition and delegation activity does not necessarily require the face-to-face interaction between PM and DE, and it is accomplished asynchronously through e-mail or internal memos. In this process, task delegation is an asynchronous coordination activity. Conversely, some of the steps in the AR management process require face-to-face (synchronous) cooperation. For instance, during CCB meeting PM and DE interact to decide the release strategy. Interaction is among all the meeting participants, and it is less formal. The shared resources are the AR and the MR.

4 The SPADE Solution

The SPADE solution for the process described in the previous section is articulated in two steps:

1) Definition of the UIE, i.e., selection, development, and integration of the users' tools.
2) Design and implementation of the SLANG process model that will be enacted by the EE. In turn, this activity can be decomposed in the definition of the process types and of the process activities.

Clearly, these two steps are strongly related. For instance, the selection of a specific tool has an impact on the structure of the information to be stored in the Repository, and on the process model fragment in charge of controlling it.

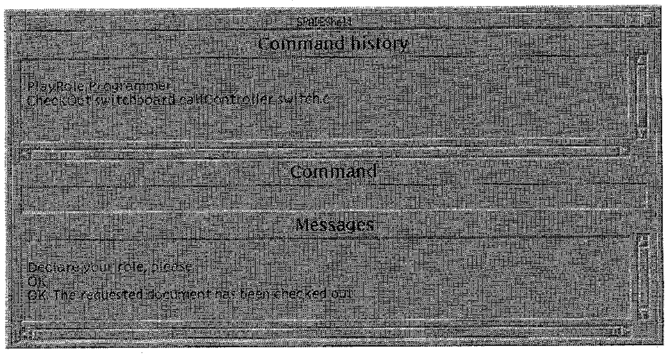
In the remainder of this section we first present the tools, the general architecture of the solution, the process types, and, finally, the process activities.

The proposed solution does not take into account the issues related to synchronous cooperation. In fact, the version of SPADE-1 that we used for this experiment did not include components to support this kind of cooperation. Support to synchronous cooperation in SPADE-1 is discussed in Section 6.

4.1 Tools

The tools we use in the example are a subset of those listed in Table 2. They are: SPADEShell, ObjectEditor, and Agenda. As discussed in Section 2.1, SPADEShell is a service-based tool. It simply accepts sequences of characters typed by the users. SPADEShell is totally unaware of the semantics of the character sequences it receives. They are delivered as messages to SCI that reifies them as tokens in some enacting active copy. The SLANG interpreter in charge of the active copy execution manipulates these tokens according to the process model being enacted. In this way, a request issued by the user through SPADEShell is eventually addressed by some process model fragment. In the process example, process agents use SPADEShell to perform all the operations related to access control and configuration management. Fig. 4 shows SPADEShell user interface. Table 3 shows the list of commands that can be issued through the SPADEShell and that are recognized by the process model.

ObjectEditor is an O_2 black-box tool. It allows users to access and modify O_2 objects (i.e., all the entities manipulated by the process and stored in the O_2 database). In the example, ObjectEditor is used to manipulate several process artifacts such as AR and MR. Agenda is a service-based tool that is used to manage the list of tasks belonging to a specific user. A task is a unit of work described by a set of structured data, called task attributes. This information can be updated either by the user or by the EE, depending on the process model, its state, and users' decisions and preferences. Each task is also associated with a set of states. States represent properties of a task. State transitions may be determined by the enacted process model or by an explicit action of the user.



```
PlayRole Programmer
CheckOut switchboard callController switch.c
```

```
Declare your role, please
OK
OK. The requested document has been checked out
```

Fig. 4. SPADEShell user interface.

TABLE 3
COMMANDS ISSUED THROUGH SPADESHELL

Command	Parameter
PlayRole	role
Logout	
Register	user role
UnRegistered	
ListLogged	
AddProject	Proj.
AddDEsigner	proj. user role
OpenProject	proj.
CloseProject	proj.
NewRelease	proj.
AddModule	proj. module
AddProgrammer	proj. module user role
OpenModule	proj. module
CloseModule	proj. module
NewRelease	proj. module
Submit	proj. module doc.
CheckOut	proj. module doc. [ver.] [ReadOnly]
CheckIn	proj. module doc. [ver.]
UnLock	proj. module doc.
ListProjects	
ListModules	proj. [ver.]
ListElements	proj. module [ver.]
CreateFile	proj. module doc. [ver.]
CreateAR	proj. module doc. [ver.]
CreateMR	proj. module doc. [ver.]
Edit	proj. module doc. [ver.]
Delete	proj. module doc. [ver.]
ListLocal	
Shutdown	

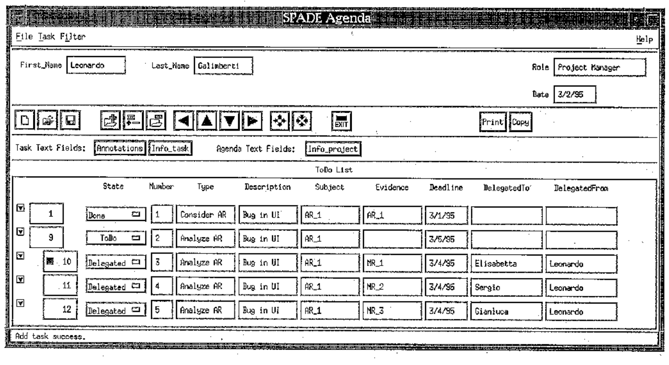
In general, states and all the other attributes related to a task depend on the process being supported, on its social context, and on the roles of participating process agents. Therefore, this information cannot be statically hard-coded in Agenda. Rather, the process engineer must be able to flexibly define this information according to process characteristics and requirements. For these reasons, Agenda can be dynamically tailored through a configuration file, that includes the definitions of task attributes, states, state transitions, and other minor features. The configuration file is loaded by Agenda at start-up, or even during Agenda execution, if explicitly required.

The predefined operations offered by Agenda are basically the insertion and deletion of tasks, and the modification of task states and attributes. These operations can be invoked by the user through the iconic buttons in the Agenda interface, or by the process model, through messages sent to Agenda via black transitions.

As discussed in Section 3.3, in the process example some tasks may be delegated to other users. For instance, a DE may request a PR to modify a module according to the corresponding MR. To support delegation, the Agenda configuration file defines three task states: `ToDo`, `Delegated`, and `Done`. A task assigned to a user is initially in state `ToDo`. The task can be delegated to another team member, either explicitly by its owner, or by the EE as a consequence of an operation specified in the process model. In both cases, the procedure used to manage delegation is coded in the process model. According to this procedure, the state of the delegated task in the delegator's agenda is set to `Delegated`. The same task is also sent to the delegatee's agenda. In this agenda its state is set to `ToDo`.

As soon as the delegatee sets the state of the task to `Done`, his/her Agenda sends a message to the process engine to signal the state transition. The token representing this message enables the execution of some transition in the process model, that eventually sends a message to the delegator's Agenda. This message requests the transition of the task state from `Delegated` to `Done`. Clearly, being this procedure completely specified in SLANG, it can be enriched or modified according to specific process needs and requirements. For instance, it is possible to write a SLANG net that, as soon as a delegated task is completed, sends a message to the project manager and creates a record on a log file.

Fig. 5 shows the user interface of the Agenda used in the process example. A fragment of the configuration file defining the Agenda behavior is shown in Fig. 6. In particular, the `[States]` section specifies task states. Task state transitions are defined by their initial and final states (specified in the first and the second column of the `[UserTransitions]` section, respectively). Finally, the task attributes are specified in the `[Structure]` section. Each attribute is described by its length (expressed in number of characters), its type, and its control mode. The control mode indicates whether a change in the value of an attribute or of a task state has to be authorized by the process model (`WAITREPLY`), or can be accomplished autonomously by the users (`SIMPLENOT`).



First_Name Leonardo   Last_Name Galimberti   Role Project Manager
Date 3/2/95

Task Text Fields: Annotations Info_task   Agenda Text Fields: Info_project

ToDo List

	State	Number	Type	Description	Subject	Evidence	Deadline	DelegatedTo	DelegatedFrom
1	Done	1	Consider AR	Bug in UI	AR_1	AR_1	3/1/95		
9	ToDo	2	Analyze AR	Bug in UI	AR_1		3/5/95		
10	Delegated	3	Analyze AR	Bug in UI	AR_1	MR_1	3/4/95	Elisabetta	Leonardo
11	Delegated	4	Analyze AR	Bug in UI	AR_1	MR_2	3/4/95	Sergio	Leonardo
12	Delegated	5	Analyze AR	Bug in UI	AR_1	MR_3	3/4/95	Gianluca	Leonardo

Add task success.

Fig. 5. Agenda user interface.

```
[States]
ToDo Delegated Done

[UserTransitions]
ToDo          Delegated   WAITREPLY
Delegated     Done        WAITREPLY
ToDo          Done        WAITREPLY

[Structure]
Number         3          STRING    WAITREPLY
Type           12         STRING    WAITREPLY
Description    12         STRING    WAITREPLY
Subject        12         STRING    WAITREPLY
Evidence       12         STRING    WAITREPLY
Deadline       8          DATE      WAITREPLY
DelegatedTo    15         STRING    WAITREPLY
DelegatedFrom  15         STRING    WAITREPLY
Annotations    10         TEXT      SIMPLENOT
Info_ask       10         TEXT      WAITREPLY
```

Fig. 6. A fragment of Agenda configuration file used in the process example.

4.2 Architecture of the Solution

The general architecture of the solution is summarized in Fig. 7. Each user is logged on its workstation and runs a copy of SPADEShell and SPADE Agenda. Through Agenda a user is enabled to check the list of tasks she/he is assigned to, to start them, or delegate them to other users. These actions are notified to the EE that reacts to them according to the process model being enacted. Users may also invoke commands through SPADEShell. Again, these commands are reified as tokens in the Petri net, and are manipulated in the process model.

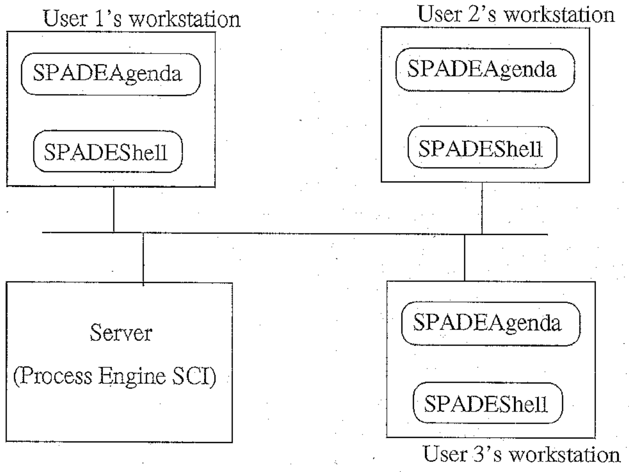


Fig. 7. Architecture of the solution.

Thus, cooperation is achieved by providing users with tools that enable them to send and receive information, and to issue requests to the PSEE. All these requests are received by the EE that acts as system coordinator and dispatcher of information among the different users.

4.3 Process Model Types

Fig. 8 shows the main types defined for our example:

- `User` defines the information about the users. They are the name of the user, his/her role (PM, DE, or PR), the workstation she/he is working on.
- `Item` and its subtypes define the structure of the objects under configuration management. They are projects, modules, and documents (e.g., AR and MR).
- `Task` defines the task data exchanged between the process model and the users. It reflects the task structure specified in Agenda configuration file. The type definition is the following:

```
class Task inherit ModelType
public type
   tuple (
     owner          : User,
     state          : string,
     taskId         : integer,
     type           : string,
     description    : string,
     subject        : string,
     evidence       : string,
     deadline       : string,
     delegatedTo    : User,
     delegatedFrom  : User) end;
```

- `owner` is the user who is in charge of executing the task; `state` is the task state; `type` indicates the type of the current task; admissible values are "Consider AR," "Analyze AR," and "Implement MR." These values represent the steps discussed in Section 3.3. `description` contains an informal description of the task. `subject` is the name of the document provided as input to the task. `evidence` contains the name of the artifact produced as output of the task. `evidence` is filled in by the user when she/he terminates the execution of the task. `delegatedTo` and `delegated-From` contain the reference to delegator and delegatee, respectively.
- `Workspace` defines the data related to the private workspace of each user. Basically a workspace contains the items the user is working on and the description of the tasks she/he is in charge of.

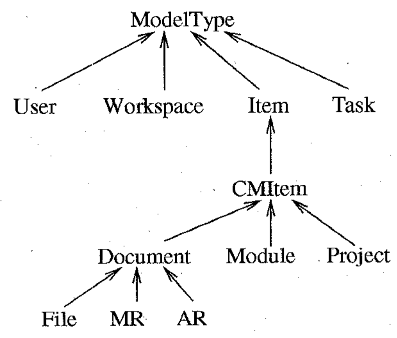


Fig. 8. The main types used in the process example.

4.4 Process Model Activities

The process model root activity is shown in Fig. 9. The root activity contains some initialization actions and the invocation of three other activities. The initialization involves the registration of the user place `LoginMsg` to receive a token each time a SPADEShell is launched. This registration is performed in the black transition `ConfigUserPlace`, which sends a registration request to SCI. The invoked activities are the following ones:

- `AccessControl`. It controls users' access, and allows new users to be registered by the PM or the SA.
- `ConfigurationManagement`. It implements the global workspace in which process data is stored, and the configuration management policy applied to items in this workspace.
- `SessionManager`. It supports the interaction of users with the environment.

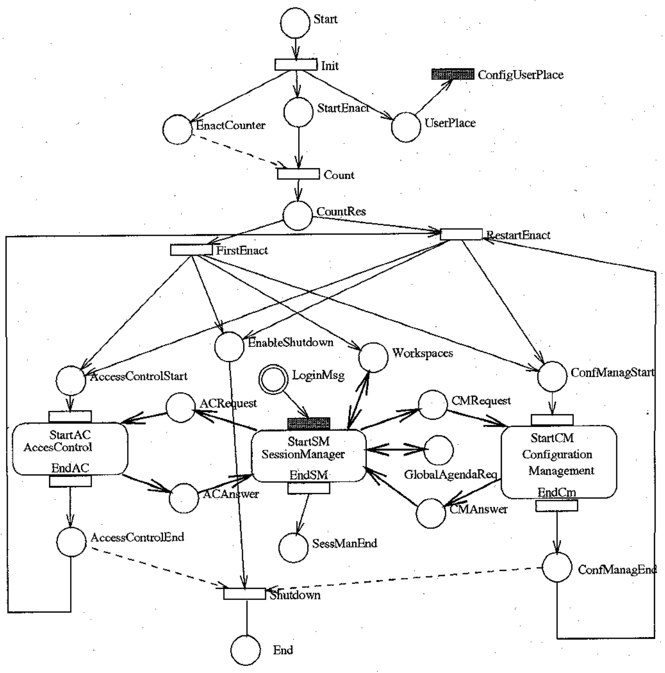


Fig. 9. The root activity used in the example.

After the registration of the user place, one active copy for each of the activities `AccessControl` and `Configuration-Management` is created. These two active copies remain active for all the duration of the process. When SPADEShell is started, it sends a `LoginMsg` to the EE. The process model we have defined reacts by creating an active copy of activity `SessionManager`. Thus, the number of active copies of `SessionManager` activity at a given time depends on the copies of SPADEShells that are presently running (typically one SPADEShell per user session). Each of these active copies of `SessionManager` is in charge of controlling the interaction with one of the process agents. Activity `SessionManager` receives and interprets user commands issued through SPADEShell. These commands may require the execution of local actions on the private workspace of the user, or global actions on either the global workspace or the local workspaces of other users. Local actions are directly managed by `SessionManager`. Global actions require the execution of transitions defined in other activities.

In particular, shared places `CMRequest` and `ACRequest` (Fig. 9) control the enabling of transitions in `Configura-tionManagement` and `AccessControl` activities, respectively. For example, if a user wants to check in a document in the global workspace, a token is inserted in `CMRequest` to request the corresponding `CheckIn` operation to the `ConfigurationManagement` activity. Shared places `CMAnswer` and `ACAnswer` collect the results of the required operations. Shared place `GlobalAgendaReq` supports communication among different instances of `SessionManager`. For example, a token is inserted in this place if a user requests a task delegation.

When a user logs in (i.e., a token is created in place `LoginMsg`), an active copy of activity `SessionManager` is instantiated. At this point, the user may declare his/her role using the SPADEShell command "`PlayRole`." If his/her role has been correctly registered, `SessionManager` invokes activity `AgendaManager`, which is in charge of managing the interaction of the user with the Agenda (see Fig. 10). Once initiated, `AgendaManager` starts the Agenda tool on the user's workstation. Information about the address of the user's workstation is retrieved by transition `StartAM` from place `Owner` of type `User`. Agenda is initialized with the user's task list. The list is read from place `LocalWS` (i.e., the user's local workspace), and each single task in the list is stored in place `TasksToBeAdded` as a distinct token. Tokens in `TasksToBeAdded` are then consumed by `InitAddTaskReq`, that generates tokens of type `AgendaMessages` (the definition of this type has not been provided for the sake of simplicity). They, in turn, are used to compose the "`AddTask`" requests sent to Agenda through black transition `SendAgRequest`. Tasks are also stored in place `Tasks`, which maintains data about tasks until the current working session terminates.

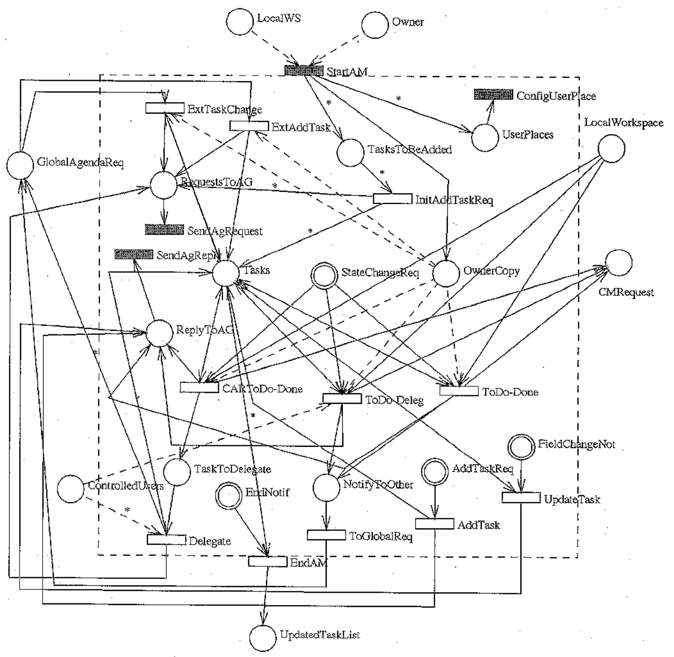


Fig. 10. AgendaManager activity definition.

The execution of transition `ConfigUserPlace` registers user places `AddTaskReq`, `FieldChangeNot`, `StateChangeReq`, and `EndNotif` to receive the messages coming from Agenda as tokens. In particular, `AddTaskReq` receives a token when the user adds a new task in his Agenda. `FieldChangeNot` receives a token when the user changes the contents of a task attribute. `StateChangeReq` receives a token each time a user tries to change the state of a task appearing in his/her Agenda. Finally, `EndNotif` receives a token when Agenda terminates its execution. Upon completion of the start-up operations discussed above, activity `AgendaManager` behaves as a reactive system, being ready to receive requests or notifications from the Agenda, or from other active copies through place `GlobalAgendaReq`.

Let us analyze the sequence of actions accomplished by the EE to react to a task state change request issued by a user. A user can request a task state change, depending on his role. Table 4 summarizes the state transitions allowed for each type of task. It also indicates the entity that can invoke them. Some state transitions, in fact, cannot be requested by users, but can be enforced only by the process model, as a consequence of some events.

TABLE 4
STATE TRANSITIONS AND TASK TYPES

	ToDo → Done	ToDo → Delegated	Delegated → Done
Consider AR	PM	nobody	nobody
Analyze AR	DE	the process model	the process model
Implement AR	DE, PR	DE	the process model

When a user requests a task state transition from `ToDo` to `Done`, a token appears in place `StateChangeReq`. This token triggers the evaluation of the guards of transitions having `StateChangeReq` as input. In particular, transition `ToDo-Done` in `AgendaManager` activity is enabled only if all of the following conditions hold:

- The user has a role that enables him/her to terminate the current task (according to what is specified in Table 4).
- The artifact to be produced as output of the task (specified by field evidence of the token extracted from place `Tasks`) is in the local workspace of the user.
- The task name is not "Consider AR." The case "Consider AR" is managed by transition `CARToDo-Done`.

As a result of transition execution, the produced artifact is checked in the global workspace (a token representing the corresponding check in request is stored in place `CMRequest`), the state of the current task is changed to `Done`, and a confirmation message is sent to Agenda (through transition `SendAgReply`). Finally, if the task has been delegated to the user by some other team member, the delegator is notified of the termination of the task and of the name of the produced artifact (a token is stored in place `GlobalAgendaReq` through transition `ToGlobalReq`). When a user requests a state transition from `ToDo` to `Delegated`, the transition `ToDo-Deleg` fires only if the delegator is enabled to delegate a task to another user, and if the delegatee belongs to the team member the delegator is responsible for (i.e., a token describing the delegatee is contained in `ControlledUsers` place).

Task delegation is managed in the following way:

1) The data related to the task is cloned and sent to the delegatee's local workspace by transition `ToGlobalReq` that stores a token in shared place `GlobalAgendaReq`.
2) The state of the task in place `Tasks` is changed to `Delegated`.
3) The input artifacts needed to execute the task are moved from the local to the global workspace (a token containing them is put into place `CMRequest`). In this way, these artifacts can be checked out by the delegatee later on.
4) A confirmation message is sent to the users's Agenda to authorize the requested state transition.

Notice that any change in the task state in the delegatee's Agenda is mirrored in the delegator's Agenda as well. For example, as explained above, when the delegatee's task is set to state `Done`, the same applies to the corresponding delegator's task. The delegator, therefore, is kept informed of the task progress. When a user requests state transition from `ToDo` to `Done` for task "Consider AR," transition `CARToDo-Done` fires if the following guard holds:

```
guard
StateChangeReq->name == "Consider AR" &&
StateChangeReq->parameter == "Done" &&
Tasks->state == "ToDo" &&
OwnerCopy->role == "ProjManager" &&
LocalWorkspace->element == Tasks->subject &&
(LocalWorkspace->element->state == "Rejected"||
 LocalWorkspace->element->state == "Postponed"||
 LocalWorkspace->element->state == "Approved")
end_guard
```

The guard checks if the role of the requesting user is PM, and allows the "Consider AR" task to be terminated only if the user has properly modified the AR. When transition `CARToDo-Done` fires, the following actions are performed:

1) The state of the task is set to `Done`.
2) The anomaly report (`LocalWorkspace->element`) is put under configuration management.
3) A message is sent to Agenda, to authorize the requested task state transition.
4) If the state of the AR has been set to "Accepted," a task of type "Analyze AR" is generated. It is inserted in place `TaskToDelegate`. It is then delegated to all DEs by transition `Delegate`.

Step 2 guarantees that the AR is available in the global workspace and can be used as input document for the "Analyze AR" task. Step 4 allows PM to send to other team members the data related to the delegated tasks.

Transition `Delegate` generates a token representing a task delegation request for each token in `ControlledUsers`. It also forwards (enabling transition `SendAgRequest`) the "`AddTask`" requests to the local Agenda. In this way, a delegated task appears in the delegatee's Agenda with state `ToDo`, and in the local user's Agenda with state `Delegated`. Therefore, transitions `CARToDo-Done` and `Delegate` implement task decomposition and delegation.

Transition `ExtTaskChange` manages requests "`SetTaskState`" and "`SetTaskField`" coming as tokens in place `GlobalAgendaReq` from the other instances of `AgendaManager`. Transition `ExtTaskChange` updates the task list stored in place `Tasks`. Moreover, it has the request forwarded to Agenda tool through a token stored in place `RequestsToAG`. Transition `ExtAddTask` manages request "`AddTask`" in a similar way.

`ConfigurationManagement` activity (see Fig. 11) implements a policy for asynchronous data sharing among the users. The activity interface is composed of two shared places, `CMRequest` and `CMAnswer`, one input place, `ConfManagStart`, and one output place, `ConfManagEnd`. The implementation of the activity contains only one place, `CMItemDB`, storing all the items under configuration management. The policy implemented by the activity is quite simple. It guarantees that when an item is checked out in read-write mode, exclusive access is granted by locking the item; there are no restrictions for an item checked out in read-only mode. In addition, the policy determines that the only authorized persons that may check out items in read-write mode are the PM, the DE responsible for the module which the item belongs to, and the PR working on the module.

Within `ConfigurationManagement` activity, transition `CheckOut` checks out an element from the global workspace. The following fragment of transition body looks in the global workspace for a document to be checked out in read-write mode. It verifies that the user who requested the operation is enabled to accomplish it, and that the document has not been locked.

```
for (p in in_CMItemDB where p->id->name ==
     CMRequest->projId->name && p->opened)
 for (m in p->modules where m->id->name ==
      CMRequest->modId->name && m->opened)
  if (!CMRequest->readOnly)
  for (u in m->programmers where u->name ==
       CMRequest->from->name && u->role ==
       CMRequest->from->role)
   for (e in m->elements where (e->id ==
        CMRequest->elementId && e->lastVersion)
   if (!e->locked)
   {
   CMAnswer->answer = "CheckOut";
   CMAnswer->projName = CMRequest->projId->name;
   CMAnswer->modName = CMRequest->modId->name;
   CMAnswer->element = (o2 Element)e->deep_copy;
   e->locked = true;
   e->lockedBy =
           (o2 User)CMRequest->from->deep_copy;
   }
...
```

5 Evaluation

In this section, we provide our evaluation of the experience we have conducted with SPADE-1. We strongly believe that the issues we discuss here, even if based on our own experience, have a significant relevance at a general level.

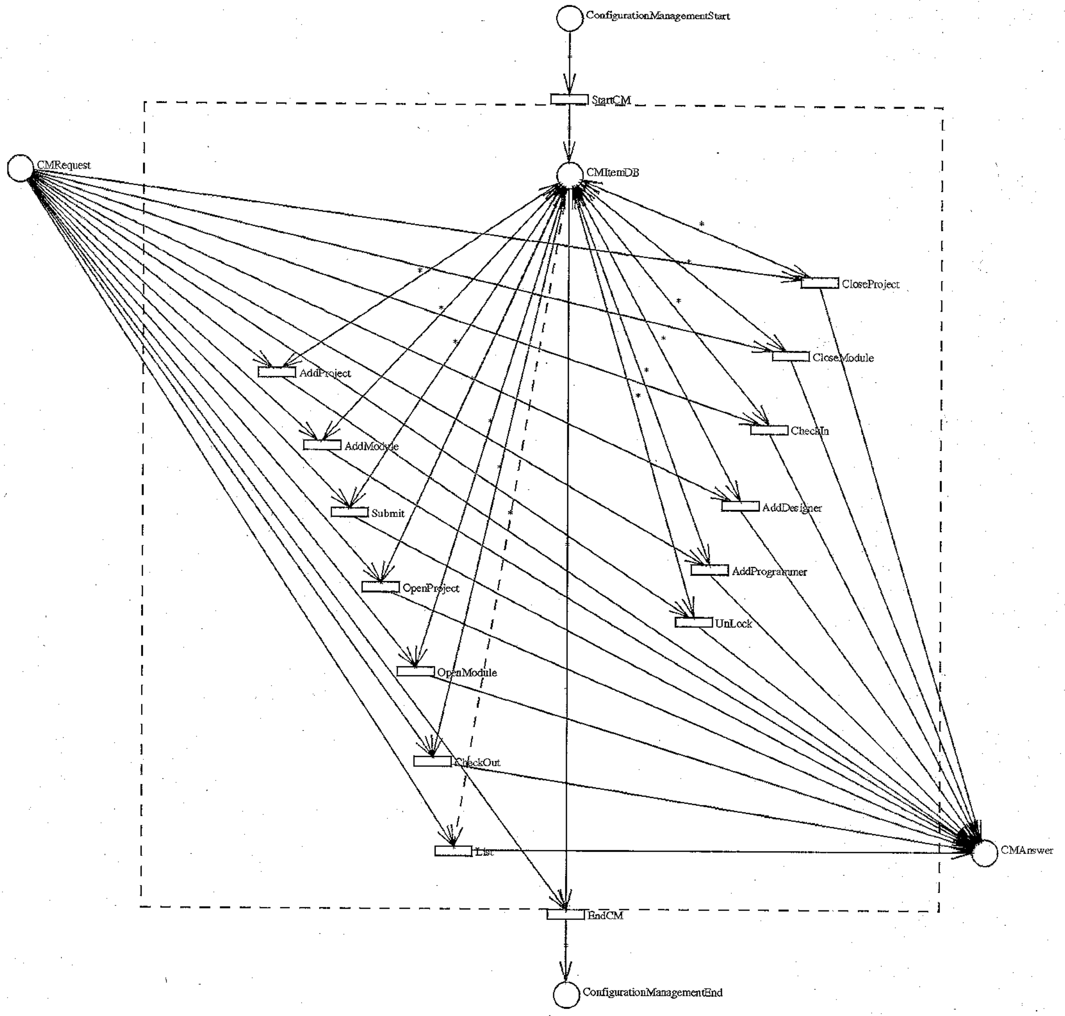


Fig. 11. ConfigurationManagement activity.

We argue that supporting cooperation in software processes is a critical task that has many facets and implications. A feasible and effective solution can be built only if a variety of aspects are properly evaluated and taken into account. In particular, we want to stress the relevance of the interplay among three factors:

1) Architecture of the PSEE.
2) Characteristics/features of the PML and of the tools being used.
3) Methodology and "style" [31] used in building tools and process models.

5.1 PML/PSEE Features and Tool Integration

Cooperation is perceived by the users of a PSEE through the tools they use. Namely, the quality and effectiveness of the PSEE heavily depend on the "quality" of the tools that compose its UIE. In particular, an essential quality factor that make it possible to support the cooperation of tools (and hence of the users behind them) is their degree of "control integration." In this context, control integration has two complementary meanings:

1) integration among tools and
2) integration between tools and the PSEE.

The adoption of FUSE, or similar environments such as Tooltalk or BMS derived from the FIELD project [46], makes it possible for a tool to invoke services offered by other tools. This feature is extremely useful to support different kinds of cooperative activities. In particular, it is possible to follow up an operation accomplished by a user using a tool, by sending a message to other tools. For instance, the activation of task "teleconference on product X" in the product manager's Agenda might trigger the invocation of a teleconference application involving different software engineers located at dispersed sites.

In order to support process integration (i.e., the ability to control tools according to a defined process [28]), it is also mandatory that tools properly interact with the PSEE. This means that the messages exchanged by tools must be explicitly controlled and managed by the PSEE. In FUSE, for example, if during the compilation of a file an error is discovered, the normal behavior is that an "openfile" and a "highlightText" messages are forwarded to the FUSE editor to highlight the corresponding source code fragment. If we want to enforce some process-specific access control policy, we might want to specify in the process model how to check whether the file can be edited by the user who invoked the compiler. This means that the "openfile" message generated by the compiler has to be captured by the PSEE and then forwarded to the FUSE editor only if the requested operation is consistent with the process model.

We argue that, to effectively integrate a tool into tool integration environments, it is necessary to conceive the tool as a collection of services since the very beginning (a priori tool integration). A posteriori tool integration (e.g., by means of wrappers) could be less effective since a tool is still seen as a monolithic "operator."

Besides usage of integration features, our experience has also indicated that there are specific requirements for the PML/PSEE. In particular, PML must provide specific constructs to model and control the messages exchanged among tools, and the architecture of the PSEE must be designed accordingly. In SPADE these issues have been addressed as follows:

- **PML**. SLANG provides two constructs for tool control: the black transition and the user place (see Section 2). The black transition is the basic construct to request the execution of an operation to the hosting environment. In particular, it can be used to send a message to some tool, thus solving one aspect of the problem, i.e., the communication between the EE and the "external world." The user place makes it possible to reify any information generated in the "external world" as a process model datum (i.e., as a token). These two SLANG constructs support the basic actions of "delegating" an operation to tools outside the EE and "detecting" an event occurring in the UIE, respectively. These constructs have been designed in a way that is consistent and orthogonal to the rest of the language.
- **PSEE**. It is necessary to provide means to "map" the firing of black transitions with the execution of services offered by tools in the UIE, and, conversely, messages coming from the UIE with the appearance of tokens in some user place. In SPADE-1, this is achieved by SCI, which acts as the interface between the EE and the UIE. If we consider the tool integration facility (e.g., Tooltalk) as a "software bus" to support intertool communication, then SCI is the "interface board" to connect the EE to this bus.

We want to stress that these basic mechanisms and features are essential ingredients to enable the creation of highly integrated environments that effectively support the cooperation among multiple users.

5.2 Interaction Paradigms and Characteristics of the PML/PSEE

The UIE can be built by exploiting different "interaction paradigms." An interaction paradigm specifies the main metaphors and concepts that are visible to the users of the environment. In particular, these metaphors and concepts allow users to specify commands and invoke environment services. They are also used to provide feedback and indications to the user after some significant event has occurred. A particular type of interaction paradigm is provided by the agendas offered by many PSEE (e.g., SPADE-1, HP Synervision [36], and Leu [22]). An agenda manages the list of tasks that are relevant for a specific user.

Other PSEE offer a different interaction paradigm. For instance, in Merlin the interaction between the users and the environment is based on the concept of "workcontext" [38]. A workcontext displays on a graphical interface the documents to be manipulated by each user (each document is represented by a box). Each document is associated with a menu that provides the list of actions that can be invoked on the selected document. Moreover, dependencies among documents are displayed in a graphical form. The execution of an action on a document can have effects on documents belonging to the same or to other workcontexts. These effects may vary from the creation of a new document in a workcontext to the modification of the state of other documents. In this latter case, a flag in the workcontext affected by the change indicates that an update has been accomplished. When explicitly requested by the user, the update is actually propagated to the workcontext.

In Marvel [15], the UIE is mainly centered on the visualization of the set of rules a user can invoke. These rules represent the goals to be accomplished. The activation of a rule can trigger backward or forward chaining, depending on the state of the Marvel objectbase. This kind of paradigm is also adopted or advocated by several other PSEE (e.g., PEACE [1]). In Marvel, it is also possible to see the structure of the artifacts that are currently manipulated by the PSEE.

An interaction paradigm suggests or even imposes a specific view of the process. For instance, agendas offer a "task-oriented view" of the process; workcontexts provide a "document-oriented view"; rule sets define "a goal-oriented view." In addition, it is possible to identify intermediate approaches. For instance, Process WEAVER Agenda [26] displays workcontexts, composed of both task descriptions and documents to be used or produced. The interaction paradigm implicitly defines also the metaphors used by users to interact and cooperate among each other. For instance, in task-oriented approaches, the main means to support cooperation is the creation and delegation/distribution of tasks.

Therefore, the interaction paradigm is essential in defining the quality and style of the support to cooperation provided by the PSEE. We argue that none of the aforementioned paradigms should be considered as the general and universal solution for building UIEs. Rather, different interaction paradigms should be used to support different categories of users in different classes of processes or even in different stages of the same process. For instance, a software developer might be interested in seeing the goals of his work, more than the specific steps to be accomplished. Conversely, the project manager might be interested in understanding the sequence of tasks still to be completed. Finally, these needs can change over time even for the same class of users. Thus, UIEs should be changeable and customizable according to the specific requirements of the process being supported, and should make it possible for different paradigms to coexist.

Actually, the real challenge of PML/PSEEs is to be able to consistently create and manage different views of the state of the enacted process. For this reason, we have tried to understand, from an architectural and conceptual viewpoint, the relationship between the user interaction paradigm(s) and the underlying PML/PSEE. We have identified two extreme approaches [4]:

- *Coupled UIE*. The UIE is coupled with the PML/PSEE. Namely, the user interaction paradigm is derived from the PML/PSEE semantics: it is not explicitly described as part of the process model. Marvel is an example of a PSEE based on the coupled approach. In Marvel a process model is described as a set of rules operating on a set of artifacts. As we said, its user interaction environment is based on a rule menu that shows all the visible rules and the network of artifacts that are created or manipulated in the process.[4] When the user selects a rule, it is executed (backward and forward chaining is applied). The execution yields a new state in the objectbase. Notice that it is not necessary to specify this behavior in the Marvel process model. The interaction environment is automatically managed so that it remains consistent with the enacted process model. The user interaction paradigm is not defined as part of the process model, but it is directly based on the PML/PSEE.
- *Decoupled UIE*. The UIE is completely decoupled from the PML/PSEE. In this case, the interaction paradigm is implemented as part of the process model. An example of PSEE that adopts the decoupled approach is Process WEAVER. As we said, in Process WEAVER user interaction is supported through the Process WEAVER Agenda. Each user has an instance of this tool that contains a set of workcontexts assigned to that user. From the process engineer viewpoint, a workcontext is a datum that can be manipulated as any other process artifact. For example, one possible policy implemented in Process WEAVER could be: when a task is assigned to a process agent, a workcontext is created and inserted in his/her Process WEAVER Agenda. Conversely, when the task is finished, the workcontext is removed from this tool. These operations must be specified in the process model. The PSEE does not have any understanding of the Process WEAVER Agenda semantics. This means that the process model must explicitly include all the operations to ensure the consistency between the state of the enacted process model (e.g., the Petri net marking, in Process WEAVER) and the state of user interaction environment (e.g., the contents of the Process WEAVER Agenda).

In summary, in the coupled approach the user interaction environment is bound, to a large extent, to the PML paradigm and to the supporting environment. This forces the process agents to follow a predetermined user interaction paradigm that is implicit in the language semantics, but frees the process engineer from explicitly managing the consistency between the state of the enacted process model and the state of the user interaction environment. The decoupled approach allows the process engineer and the PSEE designer to design the appropriate interaction paradigm in a way that is fairly independent of the paradigm supported by the PML. Indeed, by using a decoupled approach, it is possible to define different interaction styles (goal-oriented, task-oriented, document-oriented, or hybrid) for different classes of users. However, the drawback of this approach is that the process model must include operations to explicitly guarantee the consistency between the state of the enacted process model and the information offered to users.

Two main decisions characterize SPADE-1 and the process example discussed in Section 4 with respect to interaction paradigms. First, SPADE-1 adopts a decoupled approach, which has an impact on the PSEE at the architectural level (e.g., in SPADE-1, EE, and UIE are distinct architectural elements and the interface managing communication between the two environments is precisely defined). Second, the example provides a task-oriented view of the process, which has an impact in the structure of the process model. For instance, in a Petri net-based notation, process steps might be represented by sequences of transitions. Had we chosen a coupled approach, a task in the SPADE-1 Agenda could have corresponded to the firing of some transitions in the model. In a decoupled approach, tasks in SPADE-1 Agenda are explicitly represented by tokens of type Task stored in some places of the net and manipulated as prescribed by the model in order to achieve the required behavior in the UIE. This solution is not specific to Petri nets. Basically, in a decoupled approach the metaphors and concepts used in the UIE are reified as data in the process model, and explicitly managed in order to keep the state of the UIE consistent with the state of the EE.

We would like to stress that by adopting a decoupled approach, the paradigm of the SPADE UIE is mainly determined by the tools of the UIE itself and the process model fragments controlling them. For instance, SPADEShell and SPADE-1 Agenda implement a specific interaction paradigm. By building different tools and process models, based on different metaphor and concepts, we would have obtained a different interaction paradigm, without changing the semantics and architecture of SPADE-1/SLANG.

5.3 Process Modeling and Enforcement Modeling

A relevant problem that is very frequently discussed among researchers is the degree of enforcement and guidance that should be provided to the user. For instance, [23] mentions four different levels of process support:

- *Passive guidance*. Basically, it consists of passive support offered upon user's request only.
- *Active guidance*. The system is able to solicit the user intervention when required.
- *Process enforcement*. The user is forced to do what is specified in the process model.
- *Process automation*. The system performs actions with no user intervention.

A similar classification is proposed in [35]:

- *Process monitoring*. The system does not influence what it is going on, but keeps track of the actions being carried out.
- *Process guidance*. The system inspects and controls the outcomes of process enactment.

4. While the Marvel administrator's menu shows all the rules, process agents' menus do not show rules marked as "hidden."

- *Loose process enforcement*. the system controls some of the activities being enacted.
- *Strict process enforcement*. The system controls all the activities being enacted.

Even if the above classifications may be somewhat vague, the issue is clearly important. Specific activities may demand for weak control, while other more critical ones may require strict enforcement. This issue is fairly orthogonal to the interaction paradigm. For example, in a task-oriented paradigm users may be allowed to choose the more convenient sequence of task execution, or may be constrained to follow a predefined sequence. Or also, in a goal-oriented paradigm, the goals to be achieved can be indicated by the system or autonomously chosen by the user. In [44] this concept is investigated as far as the "locus of control" is concerned. In particular, it is underlined how in the Intermediate/Interact approach three entities (the process model, the humans, and the PSEE) control and direct the process enactment.

As far as cooperation support is concerned, this issue is clearly related to the degree of control and enforcement that is adopted to coordinate the operations of several users. For instance, when a user completes some operation (say the release of a document), this event may immediately trigger a mandatory follow-up request to another user or, conversely, may just imply a notification of the event.

According to our experience, there is no single approach that can be used in any situation. Actually, we select the level of enforcement case by case, depending on the specific requirements of the process fragment being modeled.

Consistently, the level of enforcement determines the way tools are built and, more important, the way they are managed by the controlling process model. For instance, in the process example we discuss in Section 4, Agenda leaves the user free to decide what to do (e.g., which task has to be activated or delegated), without forcing any specific behavior. The process model simply prohibits the execution of those operations that violate some process constraint. In this case, the level of enforcement is very low, and we therefore obtain a nonintrusive behavior. Notice that this behavior is solely determined by the way we have specified the process model controlling Agenda, and it is not imposed by the semantics/architecture of the PML/PSEE and of Agenda itself.

We argue, therefore, that, like the interaction paradigm, the level of enforcement should be designed and specified as part the process model. It does change from process to process, or even within the same process depending on the role of the user and other context-specific constraints.

It is, therefore, essential to realize that enforcement modeling is an important part of process modeling. Clearly, a complete enforcement of human behavior can never be achieved. This is ethically unacceptable and technically impossible. The issue here is to provide enough flexibility to tune the behavior of the PSEE depending on the specific process fragment being modeled.

5.4 Reactive vs. Proactive Process Execution

It has been extensively argued that PSEEs must offer both proactive and reactive *process execution styles* (see for example [2]).

- Proactive control means that the PSEE initiates and controls the operations to be carried out by a user. Basically, the PSEE notifies the user that something has to be done, and makes it sure that the operation is eventually completed.
- Reactive control means that it is the user who starts the operation, by issuing a request to the PSEE. In this case, the PSEE "reacts" to some user-generated event.

The process execution style is related to cooperation support, since it can influence the view that users have of their shared workplace. In general, a cooperative process is a combination of reactive and proactive operations. For instance, a user might initiate an operation that requires coordination with other people. The PSEE might react by providing specific information about the persons who should be involved in the cooperative activity. In addition, it could pro-actively contact these other users to define the terms and modality for accomplishing the cooperative activity.

We, therefore, argue that it is not suitable to fix the process execution style in the PSEE. This must be definable at the process model level or, in an even more dynamic fashion, as an input parameter or pragma when the specific operation is started.

The process execution style can be considered fairly orthogonal to the level of enforcement. For instance, an operation started by the PSEE (e.g., proactively) can either provide just some guidance or even force a specific behavior.

In SPADE-1, the basic mechanisms for tool integration and control (discussed in Section 5.1) make it possible to "specify" at the process model level the process execution style. For instance, a user can be enabled to send messages to the PSEE via SPADEShell, and the process model can be built in such a way to just react to these messages. This would implement a "pure" reactive behavior. Conversely, the process model can include the invocation of tool services. Or also, it can embed operations that are started when significant events occur, independently of users' actions or requests. For instance, when too many activities related to the testing of a module are late, it is desirable that the PSEE starts some notification and control activity to inform the project manager of what it is going on. This can be easily implemented by an activity that sends a message to the project manager's SPADEShell when the number of tokens representing modules being tested is greater than the acceptable threshold. Thus, it is also possible to achieve a proactive behavior. Even more, by properly composing SPADE-1 basic mechanisms, it is possible to identify the best combination of reactive and proactive control for each specific process.

5.5 Support for Cooperation Policies

A cooperative process (i.e., a set of cooperative activities) is carried out by process agents who, in general, play different roles and behave according to procedures and rules that depend on the specific process they are involved in. For instance, in a software inspection process there is a coordinator, several reviewers, and a recorder. They interact to discover and report defects in a software product. Different individuals have responsibilities and expectations that depend on their specific role, and on the characteristics of the software inspection process. The policy used to support and control a cooperative process depends on the specific domain for which the process has been defined, the roles being used, the procedures that manage cooperation, and many other domain-dependent factors. It is our opinion that, at this stage of technology development, it is quite difficult to identify general high-level cooperation policies and concepts that can be used in any cooperative process. For instance, the policy for delegating tasks we discussed in Section 4 is reasonable for the process we have considered, but might be inadequate or even not applicable in other contexts.

To effectively support cooperation, therefore, an environment must be built in such a way that any cooperative tool can be built and/or integrated, and any potential cooperation policy can be modeled and used. Even more, cooperation policies do change to take into account new process requirements or changes in the market or in the organization itself. Therefore, cooperation processes must be evolvable to take these changes into account.

To achieve these goals, it is essential to identify the general and process independent mechanisms and features that have to be provided by the PML/PSEE. In building SPADE-1, we realized that the basic mechanisms and linguistic features discussed in Section 2 are general enough to achieve the aforementioned goal. Actually, cooperation is supported in SPADE-1 by 1) building/integrating specific tools and 2) developing process models that are able to properly control these tools according to the desired cooperation policy. As for this issue, an important aspect to be dealt with is concurrency control. In advanced domains such as software processes, it is often claimed that it is necessary to offer advanced transactions that extend traditional ACID semantics. These approaches exploit specific mechanisms to share and make it visible intermediate results of the computation [9]. In SLANG, advanced cooperative transactions can be modeled by properly composing different Petri net transitions (i.e., ACID transactions). Although this approach is less expressive than other advanced cooperative transaction systems, it makes it possible to flexibly define at the process model level the desired concurrency control policy.[5]

In the example we have shown in Section 4, the cooperation support (i.e., the roles and the procedures used to support the cooperative process) is implemented by two tools (Agenda and SPADEShell) and by some SLANG activities, that properly implement a cooperation policy, and control the different instances of Agenda and SPADEShell. It basically defines a concurrency control policy among the different users.

Changes in a cooperation policy can be accomplished exploiting SLANG reflective capabilities. In turn, this might even request the modification of some tool. Notice that in our example, we decided to build a customizable Agenda to flexibly modify its structure and organization, in order to easily reflect at the tool level the changes in the cooperation policy (e.g., how task delegation is accomplished).

5. A discussion of how different advanced cooperative transactions can be implemented as a Petri net is presented in [5].

5.6 Remarks

The implementation of an enactable process model must be accomplished by taking into account a variety of aspects and problems. For instance, an appropriate interaction paradigm must be selected, and the level of enforcement has to be identified. Thus, such a process model specifies how the process should be supported by the PSEE. This is what the SLANG model we presented in Section 4 does.

These aspects are not relevant when the goal of the modeling activity is just to produce a description of a process, with the purpose of facilitating its understanding and communication. In this latter case, one should focus on the procedures to be executed, the precedence among them, the flow of data among the activities, and the goals to be eventually accomplished [8].

We argue, therefore, that we can distinguish between (at least) two types of modeling activities: *modeling to enact* and *modeling to describe*. In [50], the former has been called *process programming* to emphasize this difference. In our experience, even when the same language is used (in our case SLANG), the models that can be derived by accomplishing these modeling activities are different, because their purpose radically changes. This is quite evident when comparing the process model described in [8] and the one presented in this paper. As we mentioned in Section 4, they model the same process, but the former is an high-level specification, while the latter is an enactable model.

6 Integrating SPADE-1 and CSCW Technology

The evaluation summarized in the previous section shows how SPADE-1 can be exploited to support different kinds of asynchronous cooperative processes. Even more, the mechanisms and linguistics constructs that SPADE-1 offers to integrate tools and to build process models make it possible to flexibly define many different types of cooperative processes, possibly exploiting different interaction, enforcement, and process execution styles. These features provide a level of flexibility that is very often missing in existing CSCW environments, where the cooperation policies and the interaction paradigms can be modified or customized in a very limited way (see Section 7). However, some features offered by CSCW environments are still missing in the SPADE-1 version described so far. It, in fact, does not offer support for synchronous cooperation.

As Ellis and Wainer state in [25], the main vehicle for problem solving are meetings. Clearly, this observation applies also to software processes, where meetings are actuated to analyze and discuss alternative solutions and produce results that impact on the whole development process. For instance, in the process example discussed in this paper, CCB meeting is performed to prepare the plan for the next release of the software product.

These considerations suggested that we should explore the possibility of introducing synchronous cooperation support in SPADE-1. Following the underlining idea that tool integration is the main way to add new features to a simple kernel, we have explored the possibility of adding new synchronous cooperation features to SPADE-1 by integrating it with a CSCW toolkit. As we will discuss in the reminder of this section, this approach presents several advantages for both SPADE-1 and the CSCW toolkit.

6.1 Supporting Synchronous Cooperation

Synchronous cooperation has been widely addressed in the CSCW field, and has led to the development of many prototypes supporting video conferences, cooperative writing, and other classes of cooperative applications [24]. These applications are, often but not solely, based on the shared workspace metaphor: each participant may perform actions in this workspace that are made visible to all the other users.

In general, the application defines a set of roles, often partially customizable, that embody the actions that can be performed on the shared workspace. Moreover, a policy of cooperation is given in terms of join and leave operations associated with each role. For example, given the listener and the speaker roles in a video conferencing system, the listener role may be joined and left by anybody at any time. Conversely, the speaker role may be left at any time, but joined only if the role is currently vacant.

This simple way of expressing cooperation policies is not suitable to express flow of activities and synchronization among them, as it is typical in software development processes, but it is powerful enough to model simple constraints for specific synchronous activities.

Synchronous cooperation requires an organization phase in which all the actions (scheduling, invitations, material preparation) needed to start up the cooperation are performed. This organization phase is itself a process that can be more or less complex depending on the specific activity to be prepared. Several synchronous cooperative applications ignore this phase. In some cases, it is assumed that users are all ready to start cooperating at application start-up; i.e., they previously reached an agreement about the time and topic of the activity. In other cases, a user may activate the cooperative application and wait for the other participants to join in.

In order to support the development of CSCW applications, some toolkits have been implemented to support the architectural design and/or the implementation phases of the application life-cycle [18], [48]. Usually they provide a common infrastructure that is in charge of managing the communication and the control aspects of the cooperation. This infrastructure is then tailored and plugged in the design of the final cooperative application.

We argue that the integration of a CSCW application or of a toolkit in a PSEE can solve some limitations of both CSCW and PSEE environments. In particular, on one side, the process modeling features offered by PSEE can be profitably used to model the organization of a synchronous cooperation and the cooperation policy that is adopted during the execution of the activity. On the other side, the capabilities of CSCW applications/toolkits to manage synchronous communication and information exchange can be exploited by PSEE to enhance its capability of supporting software processes. To experiment this integration, we used SPADE-1 and a toolkit called ImagineDesk [45]. The advantage of integrating a toolkit in SPADE-1 is that this enables the interaction between SPADE-1 and an entire class of applications (all those that can be built using the toolkit).

6.2 A Quick Overview of ImagineDesk

ImagineDesk supports the development and operation of synchronous, distributed, and multimedia cooperative applications. The architecture of an ImagineDesk cooperative application is illustrated in Fig. 12. In the figure, boxes represent processes, while arrows are used to specify communication channels; "c" and "d" labels indicate whether the channel is used for control or data transmission, respectively. Each process composing the architecture is devoted to a specific task:

- *Tools* implement the user interface of the cooperative application. New instances of Tools are created for each user who is involved in the cooperative activity. Tools provide users with a view of the shared workspace. They also allow users to operate on the shared workspace and to join or leave a role in the cooperative activity.
- *Communication Coordinators* are devoted to control data flows among Tools. If the application manages different types of media (e.g., images and text), different Communication Coordinators, one for each type of media, are used. They are connected to all the tools through bidirectional data communication channels, and are controlled by a Conversation Coordinator.
- *Conversation Coordinators* are devoted to control the cooperation. They manage the roles that can be covered by the users of the application, and implement one or more policies of cooperation that are followed during the execution of the application. A Conversation Coordinator receives the requests of role change from Tools through a bidirectional control channel, and manages them according to the policy of cooperation that is currently in place. It also controls Communication Coordinators by sending them, through unidirectional control channels, information about user access rights to the shared workspace. In particular, whenever a user changes his/her role, his/her new access rights are communicated to the Communication Coordinators through a control message. The hierarchy of Coordinators that has to be executed to control the cooperative application is defined at start-up. This means that, for example, it is possible to develop a catalogue of Conversation Coordinators supporting different cooperation policies, and to choose the more appropriate one at start-up time.
- *Supervisor* provides services to start and manage a cooperative application. Supervisor maintains a list of all the cooperations to be started, and offers services to add a new cooperation to the list and to specify its participants, its scheduling, and the cooperative application to be used. Supervisor starts cooperative applications according to their schedule, managing their initialization phase.

ImagineDesk provides, as parts of the toolkit, the Supervisor and the skeleton of Coordinators. They are supposed to be specialized according to the application to be built, the roles involved in the cooperation, and the cooperation policy to be adopted.

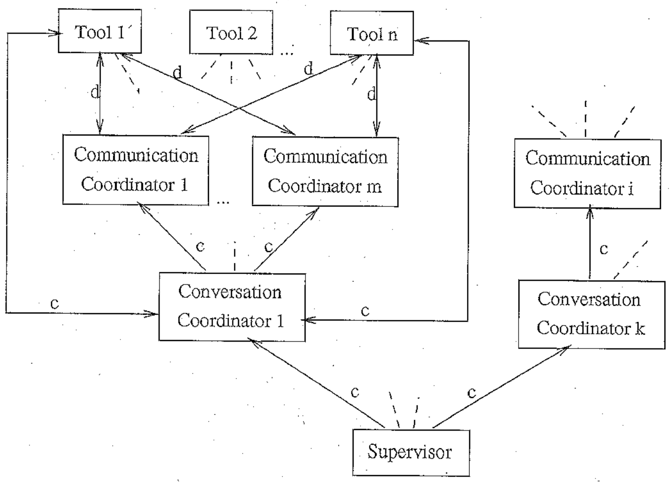

Tool 1
Tool 2
Tool n
d
Communication Coordinator 1
Communication Coordinator m
Communication Coordinator i
c
Conversation Coordinator 1
Conversation Coordinator k
Supervisor

Fig. 12. The ImagineDesk application architecture.

6.3 Integration Strategies

SPADE-1 and ImagineDesk offer complementary functionalities that can be jointly exploited to provide more effective support to software developers. In particular, SPADE-1 offers explicit process modeling and enactment, tool integration, and product modeling and management facilities. ImagineDesk offers support for building and executing multimedia, synchronous applications, but does not support flexible and dynamic definition of cooperation policies.

As in the case of traditional tools, different integration strategies can be envisaged that provide the PSEE with less or more control over the CSCW application. We have identified and implemented three levels of integration, that are currently being evaluated [21]. We believe that they are rather general, since both SPADE-1 and ImagineDesk present most typical characteristics of state-of-the-art PSEE and CSCW technologies.

6.3.1 Simple Integration

The management of the cooperation is entirely delegated to ImagineDesk: SPADE-1 manages only the activation of ImagineDesk (through the services offered by the SCI). Thus, ImagineDesk manages both the organization and the execution of the cooperative activity in a way that is fully independent of SPADE. At this level of integration the CSCW tool is seen as black-box (see Section 2.1). Thus, the implementation of simple integration does not require any change in the source code of ImagineDesk.

6.3.2 Intermediate Integration

The organization of the cooperative activity is managed by SPADE-1. The control of the synchronous cooperation is demanded to ImagineDesk and it is performed according to the cooperation policies ImagineDesk implements. The goal of the cooperative activity, the number of participants, the schedule, the presentation material, and the technological support are chosen according to the information held in the state of the enacted process model.

In the example of Section 4, the organization of the CCB meeting would be started by the Project Manager as soon as the "Analyze AR" tasks she/he has delegated to Designers are done (when this happen, the task in his Agenda are set to done). She/he would then directly contact the user who notified the occurrence of the anomaly in order to identify a set of possible dates for the meeting. Finally, she/he would request the design team to select a date, based on the workload of each component of the team. Alternatively, this activity might be automatically performed by SPADE-1 on the basis of the information of the team members' Agendas.

The architecture resulting from intermediate integration is shown in Fig. 13. From the technical viewpoint this solution has required the reengineering of the Supervisor; it is the ImagineDesk tool in charge of managing the start up of the cooperative applications. In particular, the Supervisor has been interfaced with the SCI, in order to make the EE able to request the start up of a cooperative application. A tool, called spadeCEI (SPADE Conference Environment Interface) has been developed in order to support users in the organization phase. This tool is integrated in SPADE-1 according to the service-based paradigm, and it is controlled by a process model fragment that implements the policy adopted in the organization phase. Fig. 14 shows the user interface of the spadeCEI. This tool has been implemented as a Java applet [34].

The advantage of the intermediate integration is simplicity. Integration is achieved by simply modeling in SLANG the conference start-up process (possibly, exploiting the facilities offered by the spadeCEI) and by invoking the service `StartCooperation` provided by the Supervisor. This request carries the information on the number and the identity of the participants to the cooperation and on the cooperation policy to be used.

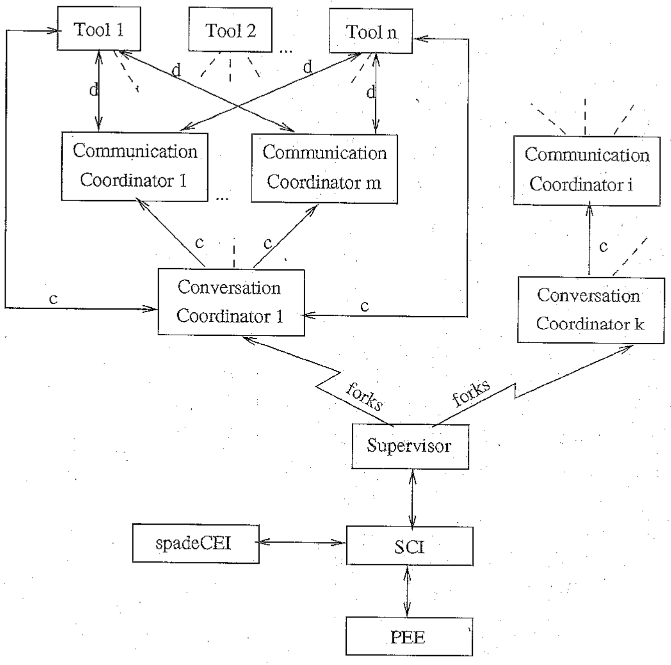
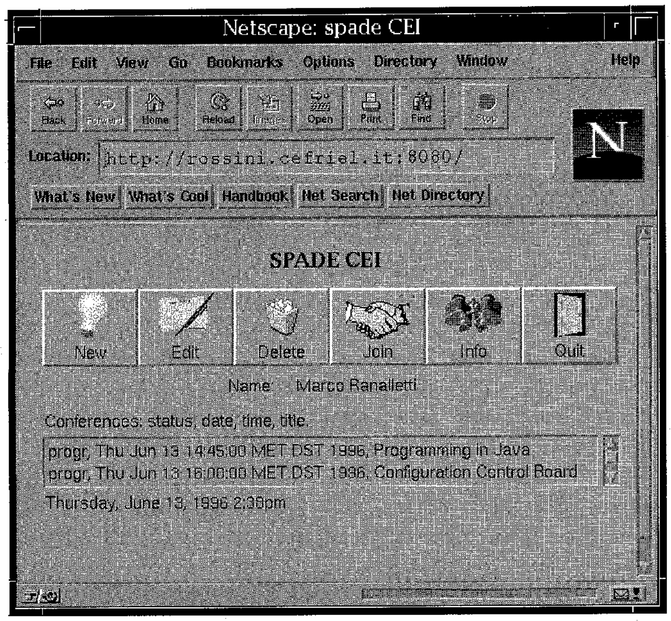

Tool 1
Tool 2
Tool n
d
Communication Coordinator 1
Communication Coordinator m
Communication Coordinator i
c
Conversation Coordinator 1
Conversation Coordinator k
forks
forks
Supervisor
spadeCEI
SCI
PEE

Fig. 13. Intermediate integration between SPADE-1 and ImagineDesk.


Netscape: spade CEI
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New What's Cool Handbook Net Search Net Directory
New Edit Delete Join Info Quit

Location: http://rossini.cefriel.it:8080/

SPADE CEI

Name: Marco Ranalletti

Conferences: status, date, time, title.

progr, Thu Jun 13 14:45:00 MET DST 1996, Programming in Java
progr, Thu Jun 13 16:00:00 MET DST 1996, Configuration Control Board

Thursday, June 13, 1996 2:38pm

Fig. 14. spadeCEI user interface.

On the other hand, in the intermediate integration it is impossible to integrate the cooperation and the process policies since they are independently described in the ImagineDesk Conversation Coordinators and the SLANG process model. Actually, a Conversation Coordinator implements a simple process, whose semantics is hard-coded and defined by its implementation. In many cases, it would be important to "blend" and integrate the two processes in order to reduce redundancy, avoid inconsistencies, and increase effectiveness. For instance, a Conversation Coordinator includes information on users and their roles. Similar information is usually stored in the software process model. Decisions related to the invocation of specific conversation operations might depend on the state of the software process being executed (and vice versa). Moreover, if the policies of cooperation are kept into ImagineDesk Conversation Coordinators, it is not possible to take advantages of SLANG reflective features to dynamically change them.

6.3.3 Process Integration

Process integration supports the organization and the execution of the cooperative activity under the full control of SPADE-1. In this case, the ImagineDesk Coordinator at the highest-level of the hierarchy is controlled by SPADE-1 as any other service-based tool.

From the technical viewpoint, getting the Coordinator hierarchy integrated with SPADE-1 is not a complex task. In fact, the Coordinators have been originally conceived according to a service-request approach. That is, they receive and execute service requests coming from the upper levels of the hierarchy (if any). Thus, integration can be accomplished developing a *bridge* that manages the problems related with protocol conversion between the two environments. The resulting architecture is shown in Fig. 15. The component named spade2ID provides both the functionalities of a bridge and of the Supervisor, thus managing both format conversions and the start-up of ImagineDesk components.

This solution basically extracts all the cooperation policies from ImagineDesk and makes them explicit as SLANG process model fragments. These model fragments describe the following entities:

1) the policy used to organize and to start up the cooperation;
2) the policy of cooperation and the management of role change requests coming from the Tools;
3) the way the roles defined at the process model level are mapped on the roles defined in the controlled Coordinator.

This means that the CSCW components are devoted to the management of specific issues related to media management and real-time distribution of information (i.e., managing the flow of data in a video-conference). All the policies are moved to the PSEE, where they can be integrated with software development policies.

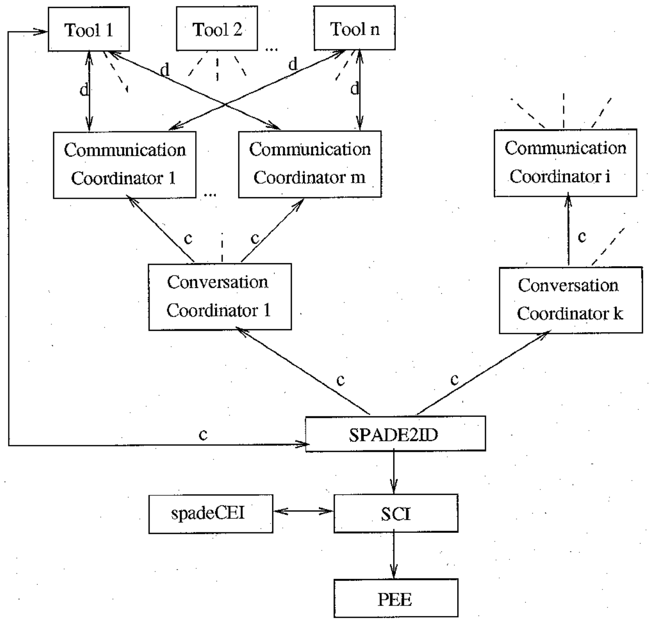

Tool 1
Tool 2
...
Tool n
d
d
d
d
Communication Coordinator 1
...
Communication Coordinator m
Communication Coordinator i
c
c
c
Conversation Coordinator 1
Conversation Coordinator k
c
c
c
SPADE2ID
spadeCEI
SCI
PEE

Fig. 15. Process integration between SPADE-1 and ImagineDesk.

6.3.4 Remark

All the integration strategies described above have been implemented and tested. Currently, the CCB meeting mentioned in Section 3 is being modeled in SPADE-1 using the three integration strategies. Early results show advantages of both the intermediate and process integration in terms of flexibility in the resulting environment over the simple integration.

7 Related Work

In this paper we discussed how to support cooperative activities in software development. There are three classes of products/technologies that address this issue:

1) Process-Centered Software Engineering Environments (PSEE),
2) Workflow Management Systems (WFMS) [32], and
3) Computer Supported Cooperative Work (CSCW) technology.

Actually, the distinction between these categories of products is not sharp. For instance, very often a WFMS is considered a subset of CSCW technology (we followed this approach in Section 1). At the same time, the features offered by WFMS are very similar to those provided by PSEE. In this section we present significant examples of systems from these three categories, in order to better understand differences and similarities and to put our work into a proper broad context.

7.1 PSEE

Most PSEE support (at least partially) asynchronous cooperation. Few systems, however, deal with synchronous cooperation. A significant ongoing project where this issue has been tackled is Oz.

Oz [10], developed at Columbia University as a successor of Marvel [39], supports asynchronous and synchronous cooperation. To address these issues, pre-defined policies are provided as basic modeling constructs at the PML level. In particular, Oz provides predefined linguistic constructs to describe and manage tasks (which correspond to Marvel rules). Also delegation and synchronous and asynchronous interaction have their corresponding constructs, with a pre-defined semantics. This approach is different from the one adopted in SPADE-1. It results in a strict coupling between the PML and the UIE interaction paradigm.

Fig. 16 shows an Oz process model fragment describing a rule whose execution is delegated to a user. Rule `analyze_bug` starts the process of software module testing. The rule is executed if the rule precondition is satisfied (i.e., the file belongs to a module of the system being developed and it is contained in a workspace). When enabled, the execution of the task is delegated to the owner of the file. Dynamic binding is used to associate tasks to the users. If many users may be selected by the rule, only one of them is chosen. This choice may be performed by the PML interpreter randomly, or with human assistance. In conclusion, the delegate command is part of the PML: Its semantics is predefined.

```
analyzebug[?tr:TEST_RUN, ?c:CFILE];
(and
    (forall MODULE ?m suchthat
       (member [?m.cfiles ?c]))
    (exists WORKSPACE ?w suchthat
       (likto [?w.module ?m])))

delegate[?w.owner]:
....
```

Fig. 16. A fragment of Oz rule.

As far as synchronous cooperation is concerned, Oz supplies a language construct to specify its participants. The execution environment supports the initialization phase of the cooperation, making each callee aware that the cooperation is about to start. Again, the policy used to manage this phase is embedded in the semantics of the corresponding Oz constructs.

To interface the synchronous cooperative application, Oz adopts the simple integration approach we discuss in Section 6.3. No control over the cooperation policy is therefore possible in the referenced version of Oz [10].

In general, Oz offers different levels of tool integration strategies. With respect to our approach, Oz does not exploit the service-based interface provided by the tools belonging to integration environments. It, instead, provides some "wrappers" to instrument and provide inputs to the tools [52].

Another system which is relevant in the context of this paper is the Desert Software Development Environment [47]. It represents a significant extension of the FIELD system since it provides advanced tool integration facilities. In Desert it is possible to specify simple rules that control the routing of messages. However, these rules cannot be considered a general process modeling technique since their scope and applicability is just limited to control tool behavior. Thus, Desert does not provide the same modeling policies offered by SPADE-1.

7.2 WFMS

WFMS grew up in the CSCW area and have been often considered as a subset of CSCW technology. At least from a market viewpoint, however, they are assuming an autonomous visibility and presence [32]. In general, workflow management consists of three areas: process modeling, process reengineering, and workflow implementation, and automation [32]. These expressions identify the same problems addressed in software processes, namely process modeling, improvement, and automation. WFMS is, therefore, very close to PSEE. We will further explore this analogy by considering a couple of significant examples.

Action Workflow is a notable example of a WFMS [43]. In this system, a workflow is defined in term of Action Workflow loops (see Fig. 17). A loop corresponds to the execution of an activity up to obtain customer satisfaction. The agents involved in the Action Workflow loop are a customer and a performer.

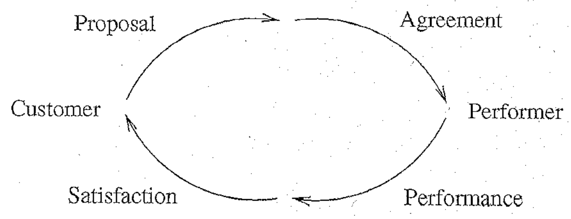


Fig. 17. Action workflow loop.

A workflow is composed of many loops related each other in three possible ways:

- *Subordinate workflow loops.* They need to be started and completed before completion of the main loop.
- *Independent triggered workflow loops.* They are triggered by some actions in another workflow, and proceed independently.
- *Resolving workflow loops.* Their result determines the acceptance or the rejection of another loop.

The conceptual architecture of Action Workflow has many different implementations. The architectural core is the workflow management server, that stores and interprets a workflow. Different applications may be integrated in the environment, and used and controlled as UIE tools. Some STF (Standard Transaction Format) processors act as bridges between the applications and the workflow management server.

Regatta is another interesting project in this area [51]. It provides end-users with support for defining their personal processes to accomplish tasks. In Regatta a workflow is called colloquy, and it is composed of a shared data space and a collection of plans. In turn, a plan is composed of a net of tasks. A task is a request from a person to another person (the assignee). It may be declined or accepted (notice that this approach is similar to Action Workflow loops). In this last case, the assignee can choose among three different ways of accomplishing the task: to perform it manually, to instantiate a plan template, or to create a new plan from scratch. After task completion, the terminating event is sent to the plan that generated the task execution request. Synchronization is thus achieved between the requester and the assignee.

7.3 CSCW Technology

Ellis classifies CSCW environments in four categories [25]:

- *keepers*: manage shared data;
- *synchronizers*: enforce action sequencing;
- *communicators*: enable people to communicate with each other;
- *agents*: perform specialized actions within a group setting.

In this section, we will consider significant examples from some of these classes.

ConversationBuilder (CB) [40], [11] is classified as a CSCW system, but presents many features typical of PSEE and of WFMS. CB activities are called conversations, and may involve one or more users. Each action in an activity is seen as an "utterance" in the corresponding conversation.

CB architecture is shown in Fig. 18. UIE tools are integrated from the control view point through a message bus. The Conversation Engine controls the cooperative activities progress and supports data integration among tools. The Conversation Engine is basically composed of a kernel, that supplies some primitive objects and services. The objects may be connected by hypertextual links. Examples of objects are process artifacts. They may be located in shared or individual conversation spaces. *Protocols* define an arrangement of actions (their semantics, their temporal relationship ...). Their instances are the conversations. *Obligations* graphically define a process structure, in term tasks assigned to users and of dependencies among these tasks. The internal structure of an obligation defines the network of actions to be accomplished in order to complete the obligation.

According to Ellis classification, CB is a keeper (user data may be stored in shared and individual workspaces), a synchronizer (protocols may define task sequences), and a communicator (Display and Conversation Managers are in charge of propagating users' actions to other users). These concepts are being further exploited in the wOrlds project [12].

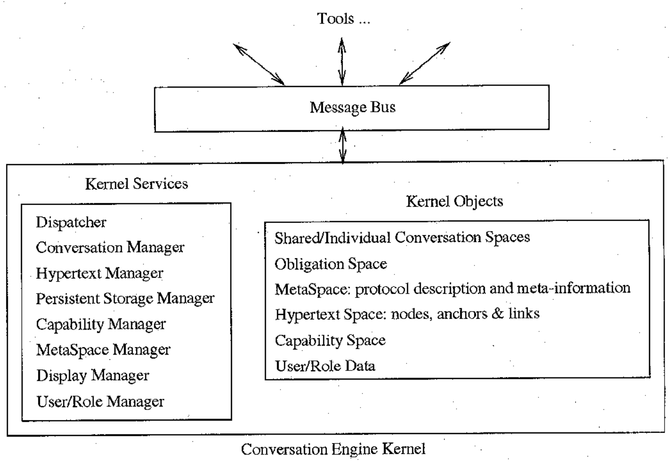


Fig. 18. ConversationBuilder architecture.

Oval [42] is a "radically tailorable" tool that allows end users to build asynchronous cooperative applications in a straightforward way. The basic constructs offered by Oval to build applications are the following ones:

- *Objects*. They represent people, tasks, messages ...
- *Views*. They are means to visualize and manipulate collections of objects.
- *Agents*. They are similar to active rules in database world. They are activated by events, and can modify, add, delete, and mail objects.
- *Links*. They represent relationship among objects.

Object sharing is performed through shared files or through e-mail. Each user involved in a cooperative activity has his/her own copy of the shared objects. The copies are updated in an asynchronous way. Basically, Oval belongs to the category of keepers. The possibility of defining links between different objects can effectively help in maintaining connections between related artifacts (e.g., a specification and its implementations). Some simple cooperation and synchronization policies may be described in term of agents.

CSDL [18] is a language for specifying and designing communicators. Some basic terminology in CSDL is similar to the one used in ImagineDesk. Both projects, in fact, started from the same ideas and background. The basic unit of CSDL is the Coordinator. More Coordinators may be composed in a hierarchy. Each Coordinator is composed of a specification, a body, and a context. The specification defines the roles and the policies of role change. The body deals with data communication. The context defines, whenever a coordinator hierarchy is specified, the relationships between the Coordinator and its descendants. The body is specified only for the Coordinators that are leaves of the hierarchy (the Communication Coordinators). In CSDL the cooperation policies express the conditions to be satisfied in order to accept changing role requests coming from the users. The language does not provide support to specifying more complex procedures and process steps.

ClearBoard [37] is a cooperative drawing tool designed to support the interaction between two remote users. Each user sees the other as she/he were on the other side of a glass that is used as a drawing surface. This way, the interaction is not just limited to the shared drawings, instead, it is also supported by the gestures and the eyes movements of the two participants. The last version of ClearBoard provides users with "TeamPaint," a multiuser computer-based paint editor. Users draw with a digital pen. Their gestures are registered by a camera and are displayed as background of the paint editor. A mirroring system is used to solve reversing problems. According to Ellis classification, ClearBoard is basically a communicator. It focuses on the interaction metaphor in order to provide the most familiar and natural way to allow people to shift between the *interpersonal space* and the *shared workspace*. ClearBoard does not provide support for definition and enactment of cooperation policies.

7.4 A Final Remark

CSCW, PSEE, and WFMS are three technologies that address the same basic issue, i.e., how to support cooperative activities in human-centered processes. We argue that most differences in goals and objectives are mainly related to different terminology and background of the domains where these technology have been originated. For instance, PSEEs emphasize the importance of "supporting process change" and "tolerating deviations," while in the CSCW and WFMS domains there is a strong interest in "supporting process exceptions." Basically, these goals are related to the same basic issue of accommodating and supporting human creativity and unexpected situations.

We argue that WFMS and PSEE are very closely related. Even more, we believe that they have the same basic characteristics. The main difference is the application domain for these tools, i.e., software development for PSEE, and Information Systems/Business Processes for WFMS. This difference, however, tends to disappear. Many PSEE have been used to support business processes as well, taking advantage of their flexible modeling and tool integration facilities. Even more, they are often classified as WFMS. For instance, [32] mentions at least three PSEE as WFMS (Process Weaver, Process Wise, and Synervision). In general, both PSEE and WFMS are oriented towards supporting asynchronous cooperation. A similar view is advocated in [14].

In general, CSCW technology tends to have a wide application domain and comprises many different types of tools. We argue that, in general, the support they offer to policy definition tends to be weaker than in WFMS and PSEE. Conversely, CSCW environments emphasize synchronous cooperation support and advanced interaction metaphors. This observation cannot be considered general and universal. However, it indicates a relevant and realistic trend.

The approach presented in this paper aims at blending and jointly exploiting the main advantages and features that originated in the CSCW and PSEE domains. We believe that SPADE-1, being focused on openness and high tailorability, may support general processes and act as a general shell in which specialized CSCW environments, devoted to support specific activities (e.g., shared editors or real-time conferencing systems), may be integrated and controlled according to well defined and modifiable policies written in SLANG.

8 Conclusions

Software development is a critical activity in which many process agents cooperate according to different interaction styles. In different phases of the development process, cooperation may be either asynchronous or synchronous. In this paper we discussed how SPADE-1 supports cooperation in software development, and we derived some general results from our experience. We argue that there are technical and methodological factors that enhance the ability of SPADE-1 to support different cooperative activities. We identified the following technical factors:

- A *decoupled UIE*, i.e., a UIE independent from the PML paradigm. This approach ensures a higher level of UIE customization, depending on specific users' needs. Decoupling may be obtained by strictly separating the UIE from the EE. In SPADE-1 this issue has been addressed by introducing the SPADE Communication Interface that acts as an interface between EE and UIE.
- An *open architecture*, i.e., the possibility of adding new tools to the environment, to support specific software development activities. PSEE, in fact, do not have to provide software developers with a wide predefined set of facilities. Rather, they have to offer well-defined mechanisms to build new tools and integrate existing ones. The SPADE-1 solution to this problem is a well-defined interface between UIE and EE, and bridges to guarantee process integration with standard tool integration environments.
- A *PML* which provides linguistic constructs to describe interaction with the UIE tools and cooperation policies. In SLANG the constructs addressing these issues are user places and transitions. User places detect events in the user interaction environment. Black transitions invoke operations in the user interaction environment. By using user places and black transitions, it is possible, for example, to coordinate single-user tools in such a way that the actions performed by one user on a tool (for example a task state change operation on an Agenda instance) may affect the state of other users' tools. In the same way, white transitions may be composed by the process engineer to implement different concurrency control policies.

Notice that the basic design choice in designing SLANG has been to provide elementary and orthogonal constructs to build different process-specific policies. This choice has been motivated by the observation that there is still little consensus on a set of process-specific policies that can be reasonably frozen as PML constructs. Process models and tools supporting specific processes have to be developed exploiting the above infrastructure. Besides basic process modeling aspects (e.g., activities, roles, products), their design (in particular process model design) must be based on the evaluation of four additional important dimensions (see Fig. 19):

- the interaction paradigm, i.e., the metaphors and concepts used by the PSEE to interact with users;
- the level of enforcement, i.e., the approach used by the PSEE to guide and/or constrain users during process execution;

- the process execution style, i.e., the proper combination of reactive and proactive behavior;
- the cooperation policy, i.e., the procedures and steps through which people interact with the PSEE and among each other.

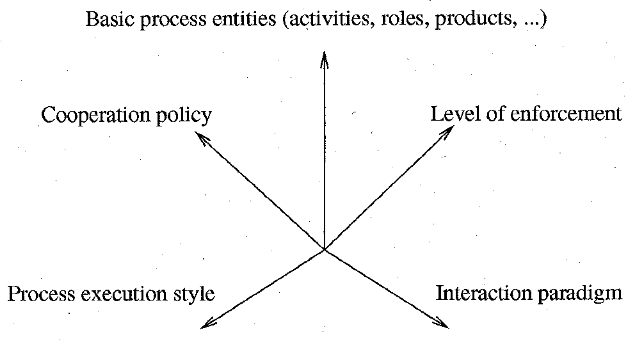


Fig. 19. Dimensions in software process modeling.

We assessed these concepts through the application of SPADE-1 to a real software process. In particular, we have shown how SPADE-1 can be used to support a significant and general cooperative process dealing with anomaly management. The process model we proposed is based on a task-oriented interaction paradigm. Users see the process to be carried out in terms of tasks. The provided level of enforcement is low. According to the classification proposed in [35] and mentioned in Section 5.3, the process model provides process guidance, i.e., it offers a fairly reactive behavior. The policy of cooperation indicates how to perform delegation, who can delegate, and the result of the termination of a delegated task. No support to synchronous cooperation was originally provided.

SPADE-1 original features did not offer any metaphor and related mechanism for synchronous cooperation. For this reason, we have evaluated CSCW environments offering this kind of functionality. In particular, we selected the ImagineDesk toolkit to exploit its ability of managing synchronous multimedia data flows. On the other side, SPADE-1 can enhance ImagineDesk by offering flexible and evolvable cooperation policies.

We identified three different integration strategies. In the simple integration strategy, the cooperative application is launched by SPADE-1, but its operation and cooperation policies are kept distinct from the SLANG process model. In the intermediate integration strategy, the start-up of the cooperative activity is accomplished by SPADE, but the actual management of the cooperation is delegated to the CSCW environment. Finally, in the process integration approach, there is a single process model implementing the software process policies and the synchronous cooperation policies.

We argue that the results and experiences of our work can be generalized and reused in a wider context. In particular, our answers to the questions raised in Section 1 can be summarized as follows:

1) *To what extent can a PSEE be used to support cooperative activities?* In general, PSEEs effectively support asynchronous cooperation. Process models, in fact, may define cooperation policies and describe interaction among UIE tools. The example discussed in Sections 3 and 4 illustrates SPADE-1 ability to deal with this problem.
2) *What kind of basic mechanisms should a PSEE offer to build different process-specific cooperation policies?* PSEE should supply flexible mechanisms for tool integration and control. A decoupled approach makes it possible to explicitly define and control the interaction with tools at the process model level. In turn, this enables a flexible and evolvable specification of the interaction paradigm, of the level of enforcement, and of the process execution style.
3) *What are the differences and analogies between PSEE and CSCW environments?* PSEE may be considered as particular CSCW environments. PSEE and CSCW environments, however, grew in separate fields, and focus on different aspects. PSEE technology mainly focuses on explicit process modeling. CSCW environments provide advanced metaphors and concepts to support users interaction and synchronous cooperation.
4) *Is it possible to identify reasonable strategies to integrate PSEE and CSCW environments?* Both PSEE and CSCW environments may gain advantage from this integration. In fact, PSEE can exploit the sophisticated interaction metaphors provided by CSCW environments. In turn, CSCW environments can benefit from the definition of flexible cooperation policies.

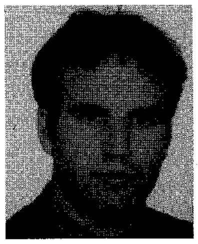
Acknowledgments

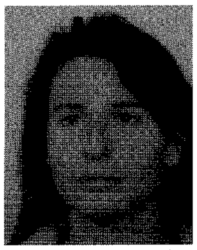
SPADE-1 has been developed at CEFRIEL and Politecnico di Milano with the contribution of many friends and colleagues. We wish to thank Gianluca Bassini, Antonio Carzaniga, Cesare Colombo, Leonardo Galimberti, Luigi Lavazza, Marco Ranalletti, and Giovanni Vigna. Their enthusiasm, technical competence, and commitment have been an invaluable ingredient for the accomplishment of the work discussed in this paper. Moreover, we thank Carlo Ghezzi who gave us many useful comments on a preliminary version of the paper.

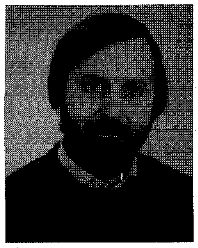
References

[1] S. Rabbi and F. Oquendo, "PEACE: Goal-Oriented Logic-Based Formalism for Process Nodeling," *Software Process Modelling and Technology*, Research Studies Press Ltd, 1994.
[2] S. Bandinelli, "Report on the Workshop on Software Process Architectures," *Software Process: Improvement and Practice*, vol. 2, no. 1, pp. 54-72. John Wiley & Sons Ltd, Mar. 1996.
[3] S. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza, "The Architecture of the SPADE-1 Process-Centered SEE," *Proc. Third. European Workshop Software Process Technology*, Lecture Notes in Computer Science 772, Springer-Verlag, 1994.
[4] S. Bandinelli, E. Di Nitto, A. Fuggetta, and L. Lavazza, "Coupled vs. Decoupled User Interaction Environments in PSEEs," *Proc. Ninth Int'l Workshop Software Process (ISPW9)*, Oct. 1994.
[5] S. Bandinelli, S. Ceri, and M. Felder, "TANGO: A Notation for Describing Advanced Transaction Models," *Proc. Int'l Conf. Information Systems Analysis and Synthesis*, July 1996.
[6] S. Bandinelli, A. Fuggetta, and C. Ghezzi, "Process Model Evolution in the SPADE Environment," *IEEE Trans. Software Eng.*, vol. 19, no. 12, pp. 1,128-1,144, Dec. 1993.
[7] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza, "SPADE: An Environment for Software Process Analysis, Design and

Enactment," *Software Process Modelling and Technology*, Research Studies Press Ltd, 1994.
[8] S. Bandinelli, A. Fuggetta, L. Lavazza, M. Loi, and G.P. Picco, "Modeling and Improving an Industrial Software Process," *IEEE Trans. Software Eng*, vol. 21, no. 5, pp. 440-454, May 1995.
[9] N. Barghouti and G. Kaiser, "Concurrency Control in Advanced Database Applications," *ACM Computing Surveys*, vol. 23, no. 3, pp. 269-317. Sept. 1991.
[10] I. Ben-Shaul and G.E. Kaiser, "Process Support for Synchronous Groupware Activities," Technical Report CUCS-002-95, Columbia Univ., New York, 1995.
[11] D.P. Bogia, W.J. Tolone, S.M. Kaplan, and E. de la Tribouille, "Support Dynamic Interdependencies Among Collaborative Activities," *Proc. Conf. Organizational Computing Systems*, Nov. 1993.
[12] D.P. Bogia and S.M. Kaplan, "Flexibility and Control for Dynamic Workflow in the wOrlds Environment," *Proc. Conf. Organizational Computing Systems*, Nov. 1995.
[13] A. Carzaniga, G.P. Picco, and G. Vigna, "Designing and Implementing Inter-client Communication in the O_2 Object Oriented Data Base Management System," *Proc. Object-Oriented Methodologies and Systems* (*ISOOMS'94*), Lecture Notes in Computer Science 858, Springer-Verlag, 1994.
[14] G. Chroust, "Interpretable Process Models for Software Development and Workflow," *Proc. Fourth European Workshop Software Process Technology*, Lecture Notes in Computer Science 913, Springer-Verlag, Apr. 1995.
[15] Programming Systems Laboratory, *Marvel 3.1.1 Manuals*. Columbia Univ., New York, 1995.
[16] "Special Number on Collaborative Computing," *Comm. ACM*, vol. 34, no. 12. Dec. 1991.
[17] J. Conklin and M. Begeman, "gIBIS; A Hypertext Tool for Exploratory Policy Discussion," *Proc. ACM Conf. Computer Supported Cooperative Work '88*, Sept. 1988.
[18] F. De Paoli and F. Tisato, "CSDL: A Language for Cooperative Systems Design," *IEEE Trans. Software Eng.*, vol. 20, no. 8, Aug. 1994.
[19] P. Dewan and B. Krishnamurthy, "Relations Between CSCW and Software Process Research: A Position Statement," *Proc. Ninth Int'l Workshop Software Process* (*ISPW9*), Oct. 1994.
[20] O. Deux, "The O_2 System," *Comm. ACM*, vol. 34, no. 10, Oct. 1991.
[21] E. Di Nitto and A. Fuggetta, "Integrating Process Technology and CSCW," *Proc. Fourth European Workshop Software Process Technology*, Lecture Notes in Computer Science 913, Springer-Verlag, Apr. 1995.
[22] G. Dinkhoff, V. Gruhn, A. Saalmann, and M. Zielonka, "Business Process Modeling in the Workflow-Management Environment Leu," *Proc. Entity Relationship Conf.*, Dec. 1994.
[23] M. Dowson and C. Fernström, "Towards Requirements for Enactment Mechanisms," *Proc. Third European Workshop Software Process Technology*, Lecture Notes in Computer Science 772, Springer-Verlag, 1994.
[24] C.A. Ellis, S.J. Gibbs, and G.L. Rein, "Groupware: Some Issues and Experiences," *Comm. ACM*, vol. 34, no. 1, Jan. 1991.
[25] C.A. Ellis and J. Wainer, "Goal-Based Models of Collaboration," *Collaborative Computing*, vol. 1, pp. 61-86, Mar. 1994.
[26] C. Fernström, "Process WEAVER: Adding Process Support to Unix," *Proc. Second Int'l Conf. Software Process*, 1993.
[27] A. Finkelstein, J. Kramer, and B. Nuseibeh, *Software Process Modelling and Technology*. Research Studies Press Ltd, 1994.
[28] A. Fuggetta, "A Classification of CASE Technology," *Computer*, vol. 26, no. 12, pp. 25-38, Dec. 1993.
[29] A. Fuggetta and C. Ghezzi, "Process Modeling Language Must Be Fully-Reflective," *Proc. Eighth Int'l Software Process Workshop (ISPW8)*, Mar. 1993.
[30] A. Fuggetta and C. Ghezzi, "State of the Art and Open Issues in Process-Centered Software Engineering Environments," *J. Systems and Software*, vol. 26, no. 1, pp. 53-60, July 1994.
[31] D. Garlan, R. Allen, and J. Ockerbloom, "Exploiting Style in Architectural Design Environments," *Proc. Second ACM SIGSOFT Symp. Foundations of Software Eng.*, Dec. 1994.
[32] D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases*, no. 3, pp. 119-153, 1995.
[33] *Proc. Ninth Int'l Workshop Software Process (ISPW9)*, C. Ghezzi, ed., Oct. 1994.
[34] J. Gosling and H. McGilton, "The Java Language Environment: A White Paper," Technical Report, Sun Microsystems. Oct. 1995.
[35] V. Gruhn, "Interpersonal Process Support Systems," Season Report 1/94, Lion GmbH, Bochum, Germany, 1994.
[36] "Developing Synervision Processes," Hewlett-Packard, Palo Alto, Calif., Part no.: B3261-90003, May 1993.
[37] H. Ishii, M. Kobayashi, and J. Grudin, "Integration of Interpersonal Space and Shared Workspace: ClearBoard Design and Experiments," *Proc. ACM Conf. Computer Supported Cooperative Work 92*, Nov. 1992.
[38] G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf, "MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment," *Software Process Modelling and Technology*, Research Studies Press Ltd, 1994.
[39] G.E. Kaiser, P.H. Feiler, and S. Popovich, "Intelligent Assistance for Software Development and Maintenance," *IEEE Software*, vol. 5, no. 2, May 1988.
[40] S.M. Kaplan, W.J. Tolone, A.M. Carrol, D.P. Bogia, and C. Bignoli, "Supporting Collaborative Software Development with Conversation Builder," *Proc. Fifth ACM SIGSOFT Symp. Software Development Environments*, Dec. 1992.
[41] K. Kishida and D.E. Perry, "Team Efforts—Session Summary," *Proc. Sixth Int'l Workshop Software Process* (*ISPW6*), Oct. 1990.
[42] T.W. Malone, K. Lai, and C. Fry, "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work," *Proc. ACM Conf. Computer Supported Cooperative Work 92*, Nov. 1992.
[43] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores, "The Action Workflow Approach to Workflow Management Technology," *Proc. ACM Conf. Computer Supported Cooperative Work 92*, Nov. 1992.
[44] D.E. Perry, "Enactment Control in Interact/Intermediate," *Proc. Fourth European Workshop Software Process Technology*, Lecture Notes in Computer Science 913, Springer-Verlag, Apr. 1995.
[45] S. Pozzi and E. Di Nitto, "ImagineDesk: A Software Platform Supporting Cooperative Applications," *Proc. ACM 1994 Computer Science Conf.* (*CSC94*), Mar. 1994.
[46] S. Reiss, "Connecting Tools Using Message Passing in the FIELD Program Development Environment," *IEEE Software*, vol. 7, no. 4, pp. 57-67, July 1990.
[47] S. Reiss, "Simplifying Data Integration: The Design of the Desert Software Development Environment," *Proc. 18th. Int'l Conf. Software Eng.* (*ICSE 18*), 1996.
[48] M. Roseman and S. Greenberg, "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications," *Proc. ACM Conf. Computer Supported Cooperative Work 92*, Nov. 1992.
[49] SPADE Team, *SPADE 3.1 Manuals*. Centro per la ricerca e la formazione in tecnologia dell'informazione (CEFRIEL), Politecnico di Milano, Italy, Mar. 1995.
[50] S.M. Sutton, D. Heimbigner, and L.J. Osterweil, "APPL/A: A Language for Software-Process Programming," *ACM Trans. Software Eng. Methodology*, vol. 4, no. 3, July 1995.
[51] K.D. Swenson, R.J. Maxwell, T. Matsumoto, B. Saghari, and K. Irwin, "A Business Process Environment Supporting Collaborative Planning," *Collaborative Computing*, vol. 1, pp. 15-34, Mar. 1994.
[52] G. Valetto and G. Kaiser, "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," *Proc. IEEE Seventh Int'l Workshop Computer-Aided Software Eng.*, July 1995.
[53] Y. Yang, "Coordination for Process Support is Not Enough!" *Proc. Fourth European Workshop Software Process Technology*, Lecture Notes in Computer Science 913, Springer-Verlag, Apr. 1995.


Sergio Bandinelli received his Dr degree in computer science from ESLAI, University of Lujan, Argentina, in 1989, and his PhD degree in informatics engineering from Politecnico di Milano, Italy, in 1995. From 1991 to 1995 he was a researcher at CEFRIEL and teaching instructor at Politecnico di Milano. Currently, he is a member of the technical staff at ESI (European Software Institute) in Bilbao, Spain, where he is leading projects in the area of software reuse. Dr. Bandinelli is the author of several scientific publications. His research interests are in process modeling, process improvement, process technologies, and software reuse. He is a member of the IEEE Computer Society.


Elisabetta Di Nitto received her PhD in informatics engineering from Politecnico di Milano, Italy, in 1996. Currently, she is a researcher at CEFRIEL, a research institution created by Politecnico di Milano and several IT industries. Her research interests are in process support technology, inconsistency and deviation management, and development of advanced telecommunication services. She is a member of the IEEE Computer Society.


Alfonso Fuggetta (M'90) is an associate professor of software engineering at Politecnico di Milano, Italy. He is also a senior researcher at CEFRIEL, a research institution created by Politecnico di Milano and several IT industries. His research interests are in process technology, process improvement, architecture of distributed processes and systems, and inconsistency management. He is a member of the IEEE and the IEEE Computer Society.