HEWLETT PACKARD

Distributed Workflow Management: The TEAM Model

Giacomo Piccinelli*
Extended Enterprise Laboratory
HP Laboratories Bristol
HPL-98-56
March, 1998

E-mail: giapicc@hplb.hpl.hp.com

workflow,
process management,
distributed systems

In recent years, workflow systems have been the basis for business-process re-engineering (BPR) but the characteristics of processes are changing and new types of problems need to be addressed. Different organizations dynamically group together to support complex projects requiring different competencies: coordination is fundamental.

Focusing on the interaction among distinct distributed PCEs (process-centered environments), we present a network-based system for the definition and enactment of federated (distributed) processes. The distribution and multi-organization problems are transparent to the process designer, who is provided with a CSP-like process-definition language, while the enactment infrastructure exploits the interconnection capability offered by network technology in order to support the cooperation among possibly-distributed execution engines (workflow servers). The interface between a local PCE (workflow client) and the related execution engine depends on both the characteristics of the PCE itself and the autonomy requirements of each organization: fault tolerance and privacy are some of the main issues we tackle.

*Contributions to this work came from F. Marcello and G. Zugliani (University of Pisa) and many colleagues at Hewlett-Packard Laboratories Bristol.

1 Introduction

The evolution of distributed objects technology [14,16,18] is having a big impact on both the definition and enactment of processes in modern organizations. Software tools become more complex and specialized but there is also a lot of emphasis on the integration of different components into global environments [18,20], and then into global processes [6,11,17,21]. Network technology, Internet in particular, offers an unprecedented interconnection capability [12] and network-oriented object architectures allows the creation of location-independent environments [4,19].
The resources needed to enact a single project (process) may come from different parts of an organization and/or from different organizations: in any case cooperation and coordination are crucial [1,2,3]. We refer to this multi-organization scenario as *federation* [3,5] and in this context we locate our work. Focusing on the coordination and information exchange aspects, we present a system for the definition and enactment of federated processes. After a brief overview of the more popular distributed objects architectures we present and discuss both the abstract cooperation model (TEAM) at the base of our system and its implementation. The process-definition language and the different components of the overall enactment infrastructure are described and problems related to integration with local environments and architecture deployment are discussed.

2 Distributed Object Architectures

Object model is having a huge impact on the engineering of software systems. The component paradigm enforced by object abstraction is fundamental for the modularization of applications but its impact on software architectures goes beyond the boundaries of a single application. Object models like COM (Component Object Model) by Microsoft [18], the OMA (Object Management Architecture) by OMG (Object Management Group) [14] and Java RMI (Remote Methods Invocation) [16] enforce two major aspects of an application: strong modularization (components) and location transparency.
Although location transparency is quite important for application components, the big impact of distributed object technologies on process-centered environments depends on the "automation" [18] features they introduce. The mechanisms may be slightly different but the result is the same: applications may "talk" with other applications. Meta-information is available to one application in order to understand dynamically how to interact with other applications in terms of the services that can be requested and the type of data to exchange. Extra layers are built on top of basic architectures (like OLE - object linking and embedding – for COM or Common Facilities in the OMA [15]) and the image an application offer of itself is more solution-oriented than technology-oriented.
In terms of the actual infrastructure we build to support the federation process, we focus on Java: RMI basic services are then a natural choice. CORBA (OMA basic layer) and OLE interaction is investigated and the actual situations in which integration is needed are discussed.

3 Cooperation Model

The purpose of a cooperation process is to organize resources and know-how of different organizations[1] in order to reach a common achievement. We use the term *federation* to indicate both the set of entities involved in the process and the process itself [3,5]. The term federation may suggest geographical distribution and/or low degree of homogeneity among the *members* but these elements are not essential. The peculiar aspect of a federation is that a pool of independent and autonomous organizations agrees on a common process and the members share part of their resources and expertise in order to enact such a process.

We propose a solution based on the paradigm of a common workspace. Every organization is associated with a part of this space called *workspace component* (Fig.1) representing its interface to the federation: the union of the workspace components represents the *federation workspace* (Fig.2).

The main elements of the workspace component (W) are the task space, the object space and the message space. The names give an indication on the kind of information we expect to find in each part of a W but we need to think in terms of both federation workspace and federated process in order to understand the dynamics of the system.

In its own *object space*, an organization puts data it needs to share with its partners and it

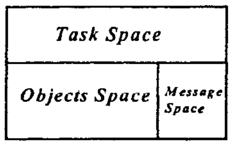

Fig.1: Workspace component

can retrieve data produced by its partners and which are relevant for the tasks it has to perform. Data are moved, replicated or deleted from the object space depending on the federated process definition and following specific rules automatically enforced by the federation infrastructure. An organization has immediate access only to the data inside its own object space and these are the only data exposed to the federated process: organization autonomy is preserved. Each organization shares all and only the data it agreed to release and under the circumstances defined in the federated process. At the same time each organization receives all and only the data it is entitled (requested) to work on.

Both the content and the dynamics of the message space follow the schema of the object space but the intended meaning of a message is different from the meaning of a piece of data. While data (objects) are the result of an activity or row material to work on ("artifact" or "work item" in the common workflow terminology [10]), messages represent information on the state of either the system or the process. They are intended to be mainly a reference for the decisions concerning the flow of control during the enactment of a process. As for the data in the object space, each organization has a view of the federation state tightly dependent on the role it plays in the execution of the

[1]We refer to a generic interpretation of the term *organization* indicating an autonomous and independent entity [5,10].

federated process. It gives all and only the information on its own internal state that were specified during the definition of the process and the same happens for the information about other members of the federation.
The tasks an organization is requested to perform are related to atomic operations like the execution of an activity or the manipulation (insert, withdraw, process) of data and messages. All the tasks in W can be executed either in parallel or in any order the organization prefers. When a task has been accomplished, the organization can mark it as finished. The flow (control) logic within the overall process is transparent to the federation members during the enactment phase, unless explicitly provided.

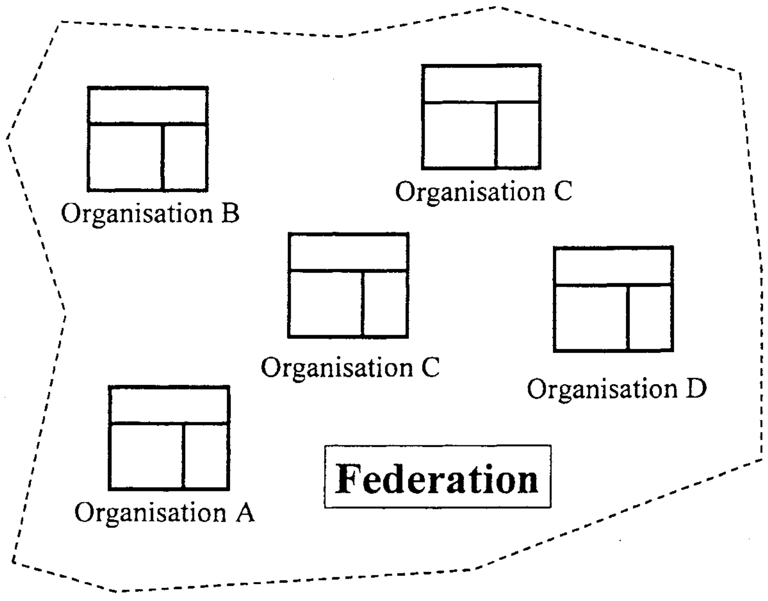

Fig. 2: Federation workspace

The purpose of a federation infrastructure is to manage the federation workspace in a way that, at any time, each organization knows exactly what to do and has available the resources it needs. The peculiar aspects of the cooperation model we propose are the presence of a management entity and a structured common working space.
The management entity (ME) is independent from any single organization but it cooperates with each member of the federation in order both to support the work of the single member and to make sure each member does what it agreed to do in the process definition phase. The management entity is trusted by all the components of the federation and manages the entire federation workspace (FW) but it cannot interfere with the internal mechanisms of any organization: autonomy is preserved. These choices are reflected in the actual implementation of the federation infrastructure but an organization

is allowed to partially relax these constraints using proxy wrappers for its workspace component with direct access to its resources.

3 Process Definition Language

The basic operations in a cooperative process are related to the exchange of artifacts, the exchange of synchronization (control) information and the execution of activities related to internal tasks or supporting the work of other members of the federation. The value added by a process-based organization depends on the fact that atomic components may be organized into complex activities (under the control of automatic systems) and the organic execution of many basic steps produces global high-value results. The aspects of a process related to the complexity of the single step need to be considered on a case-by-case base therefore we keep our system open to different options.

Push (OrgA, OrgB, Obj)
Pull (OrgA, OrgB, Obj)
Message (OrgA, OrgB, Msg)
Service (OrgA, OrgB, Srv, Obj)
Task (OrgX, Act)

Tab. 1: Basic Operations

In Tab.1 we present the *basic operations* that our PSL (process specification language) provides for the definition of the federated process while in Tab.2-3 we list the *composition operators*. The influence on the formalism coming from languages like Hoare's CSP [9] and Milner's CCS [13] is quite strong but we explicitly target the peculiarities of a federated process instead of working with generic distributed processes. The actual semantic of the language has been formalized following an approach (operational style) similar to the C-FAM (concurrent functional abstract machine) used for FACILE [7]: the result is TEAM (teamwork enabler abstract machine). We first give an intuitive description of the various elements in the formalism and then we present their formalization in the TEAM framework.

3.1 Language Description

The point of view taken during the design of a process is the one of an impartial coordinator that looks at the members of the federation as resources to organize in order to achieve a specific result. An organization may supply (push) data to other organizations and send them control information (messages) as well as asking (pull) for data. An organization may be asked to perform a specific task related to an aspect of the process it is immediately responsible for but, in order to support the central role of cooperation in the federation, it may also be asked to help one of its partners (service). The general semantic for a "service" implies that (1) an organization A receives some data from the organization B, (2) A processes the data and then (3) it sends back the result to B.

$P_1 ; P_2$	Sequential Composition
$< P_1 \& \dots \& P_n >$	Parallel Composition
(expr) $[P_1 + \dots + P_n]$	Choice Operator

Tab. 2: Composition Operators

Concerning the definition of complex processes out of the basic operations, the balance is between expressiveness and complexity. In order to preserve expressiveness without being redundant we focus on the three operators listed in Tab. 2 plus the possibility to encapsulate sub-processes into procedures (Tab. 3). P_i are generic processes and **nil** represents a null process.

Label(Var1:T1,..,VarN:Tn) { P }	Procedure definition
Label(Val1,..,ValN)	Procedure call

Tab. 3: Procedures

All the definitions are recursive but we notice that they do not introduce loops. The *sequential* operator ";" indicates that all the tasks in the process P_1 need to be completed before starting any task indicated in P_2: the overall process ends when P_2 ends. The *parallel composition* operator allows multiple execution threads within the process and the resulting process ends when all P_i processes are completed. The purpose of the *choice* operator is to choose one and only one process among the P_i depending on the state of the federation. An expression is evaluated and we expect an integer result k in the range [1,n]: only P_k is executed and when it ends, the overall process ends. If the condition is not specified the choice is random. The scope of the conditional expression is the entire

federation workspace but only simple operations (like test of presence) are supported in the present version of the system.
Procedures (Tab.3) are introduced mainly for modularization purposes but they also offer the possibility to specify recursive process definitions (thus loops). All the procedure definitions became part of a single execution environment and they may be accessed at every point in the process: the environment is flat and at the moment we don't support nested definitions. There is the possibility to define collections of procedures (libraries) and the process designer may reuse these definitions during the specification of any process. We enforce a "late" evaluation policy concerning procedure-call evaluation and it is therefore possible to have simple as well as mutual recursion in the definitions. Procedure mechanisms allow the definition of module skeletons focused on specific aspects of the cooperation process.

3.2 Formal Specification

The formal semantic of the process-specification language is given (operational style) in terms of the state evolution of a TEAM (teamwork enabler abstract machine). A TEAM represents the abstraction of a federation infrastructure in which the cooperation paradigm is the *federation workspace* (Fig. 2) and the actual federated process is managed putting specific data and control information in the workspace component associated to each organization. The order in which information flows in the workspace components depends on the synchronization constraints specified in the process.
A (T, O, M) triple models a workspace component and the state of the abstract machine is composed by a set of these triples, one for each organization. The expected actions associated to the tasks posted in the T field are quire intuitive while some explanations are necessary to clarify the modeling of user (PCE) interaction.

Task accomplished:

$$\frac{\{\ \ldots (T \cup \{\tau\}, O, M) \ldots \}{::}\ \xi}{\{\ \ldots (T \cup \{\tau^*\}, O, M) \ldots \}{::}\ \xi} \qquad \frac{\{\ \ldots (T \cup \{\tau^*\}, O, M) \ldots \}{::}\ \xi}{\{\ \ldots (T \cup \{\tau^\wedge\}, O, M) \ldots \}{::}\ \xi}$$

The fact that the user notify the system about the accomplishment of a task is modelled through the (non-deterministic) possibility to mark it (*) in the task space. In the same way, the system may want to perform some actions (ex. controls) before actually considering the task completed and when the revision process ends the mark on the task is changed into (^).

Push:

$$\frac{\{\ (T_A, O_A, M_A)\ \ldots\ (T_B, O_B, M_B)\}{::}\ \text{Push}\ (A, B, d)}{\{(T_A \cup \{\text{putObj}\ (d)\}, O_A, M_A)\ \ldots\ (T_B, O_B, M_B)\}{::}\ \text{Push}\ (A, B, d)}$$

$$\frac{\{(T_A \cup \{\text{putObj}(d)^{\wedge}\}, O_A \cup \{d\}, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Push}(A, B, d)}{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{getObj}(d)\}, O_B \cup \{d\}, M_B)\} :: \text{Push}(A, B, d)}$$

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{getObj}(d)^{\wedge}\}, O_B \cup \{d\}, M_B)\} :: \text{Push}(A, B, d)}{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{nil}}$$

Pull:

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Pull}(A, B, d)}{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{putObj}(d)\}, O_B, M_B)\} :: \text{Pull}(A, B, d)}$$

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{putObj}(d)^{\wedge}\}, O_B \cup \{d\}, M_B)\} :: \text{Pull}(A, B, d)}{\{(T_A \cup \{\text{getObj}(d)\}, O_A \cup \{d\}, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Pull}(A, B, d)}$$

$$\frac{\{(T_A \cup \{\text{getObj}(d)^{\wedge}\}, O_A \cup \{d\}, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Pull}(A, B, d)}{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{nil}}$$

Push and pull semantics is quite immediate. We ask the owner of the object to put (putObj) it into the *object space* of its workspace component (W), we move the object into the W of the receiver organisation and we then ask this organisation to collect it. For the message exchange the procedure is similar but we use the *message space* in W.

Message:

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Message}(A, B, m)}{\{(T_A \cup \{\text{putMsg}(m)\}, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Message}(A, B, m)}$$

$$\frac{\{(T_A \cup \{\text{putMsg}(m)^{\wedge}\}, O_A, M_A \cup \{m\}) \ldots (T_B, O_B, M_B)\} :: \text{Message}(A, B, m)}{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{getMsg}(m)\}, O_B, M_B \cup \{m\})\} :: \text{Message}(A, B, m)}$$

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{getMsg}(m)^{\wedge}\}, O_B, M_B \cup \{m\})\} :: \text{Message}(A, B, m)}{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{nil}}$$

Task:

$$\frac{\{\ldots(T_A, O_A, M_A) \ldots\} :: \text{Task}(A,\ t)}{\{\ldots (T_A \cup \{\text{exec}(t)\}, O_A, M_A) \ldots\} :: \text{Task}(A, t)}$$

$$\frac{\{\ldots(T_A \cup \{\text{exec}(t)\hat{}\}, O_A, M_A) \ldots\} :: \text{Task}(A,\ t)}{\{\ldots (T_A, O_A, M_A) \ldots\} :: \text{nil}}$$

Tasks are also quite immediate while services need more attention. The crucial point is that we use the same object to carry both the data to be processed (if any) and the result of the processing activity (if any). When an organisation is requested to offer a service, it can find the data to work on in its W while the organisation that asked for the service can find the result in the same container in which it putted the original data.

Service:

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Service}(A, B, s, d)}{\{(T_A \cup \{\text{servPar}(d)\}, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Service}(A, B, s, d)}$$

$$\frac{\{(T_A \cup \{\text{servPar}(d)\hat{}\}, O_A \cup \{d\}, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Service}(A, B, s, d)}{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{service}(s, d)\}, O_B \cup \{d\}, M_B)\} :: \text{Service}(A, B, s, d)}$$

$$\frac{\{(T_A, O_A, M_A) \ldots (T_B \cup \{\text{service}(s, d)\hat{}\}, O_B \cup \{d\}, M_B)\} :: \text{Service}(A, B, s, d)}{\{(T_A \cup \{\text{servRes}(d)\}, O_A \cup \{d\}, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Service}(A, B, s, d)}$$

$$\frac{\{(T_A \cup \{\text{servRes}(d)\hat{}\}, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{Service}(A, B, s, d)}{\{(T_A, O_A, M_A) \ldots (T_B, O_B, M_B)\} :: \text{nil}}$$

Concerning the procedure call, the set of typed variable $\{V_i\}$ may be empty and P is a generic process definition in which the variables may occur (free or bounded): the procedure acts as a scope binder. We assume standard rules for variable instantiation but we require that, when a procedure call is evaluated, values of the correct type are provided for all the variables. The types we allow for variables are: *org* (organization), *msg* (message), *obj* (object), *act* (activity/task) and *srv* (service).

Procedure Call:

$$\frac{\text{Label (Val1,..,ValN)}}{\text{P \{Var1/Val1\}...\{VarN/ValN\}}}$$

Where - Label (Var1,..., VarN) { P} - is in the definition environment.

In the rule for the sequential operator we model the fact that the left operand (P) has to be executed first and the right operand (Q) is considered only when P is completed.

Sequential:

$$\frac{\vartheta :: P \Rightarrow \vartheta' :: P'}{\vartheta :: P\ ;Q \Rightarrow \vartheta' :: P'; Q}$$

$$\vartheta :: \text{nil}\ ; Q \Rightarrow \vartheta :: Q$$

The semantics of choice operator introduces the *Eval* function we need to evaluate the condition (cond). We expect an integer result *k* in the range [1,n] and only the process P_k will be executed.

Choice:

$$\frac{\mathit{Eval}\,(\text{cond}) = i \quad i \in \{1 \,..\, n\}}{\vartheta :: (\text{cond})[P_1, \ldots ,P_n] \Rightarrow \vartheta :: P_i}$$

The parallel operator introduces the consistency problem for multiple actions on the same environment. The solution we adopted is to associate a copy of the original environment to each process and to let them evolve independently. All the threads are completely autonomous and they can evolve either in a truly concurrent or in an interleaved way. The parallel operator is completed when all the threads are completed and the final state of the system derives from the merge of the final state of all the threads.

Parallel:

$$\frac{\vartheta :: \ < P_1 \& \ldots \& P_n >}{<< \vartheta::P_1 \& \ldots \& \vartheta::P_n >>}$$

$$\frac{\forall k \in \{1 \,..\, n\} : \vartheta_k :: P_k \Rightarrow \vartheta_k' :: P_k'\ i}{<< \vartheta_1::P_1 \& \ldots \& \vartheta_n::P_n >> \ \Rightarrow \ << \vartheta_1'::P_1' \& \ldots \& \vartheta_n'::P_n' >>}$$

$$<< \vartheta_1::nil \;\&\; \ldots \;\&\; \vartheta_n:: nil >> \;\Rightarrow\; \vartheta_1 \cup \ldots \cup \vartheta_n :: nil$$

3.3 Example

As an example of a very simple but very reusable procedure, we present a possible definition for the basic interaction model used in the PSEE Oz: the *summit* [5]. The first step in a summit is to arrange for all the participants to be ready to start, then a cooperative activity takes place and the final step is to provide indications to the participants on what to do after the core activity is finished.

```
OpenSummitWith(X:org, Y:org, Z:org){

 Task(X, "summit initialisation");
 < message(X, Y, "begin summit");
   Task(Y,"start");
   message(Y, X, "1-ready")
&
   message(X, Z, "begin summit");
   Task(Z,"start");
   message(Z, X, "2-ready")
>
}
```

```
CloseSummitWIth(X:org, Y:org, Z:org, res:obj){

 < pull(X, Y, res);
   message(X, Y, " 1end summit")
&
   pull(X, Z, res);
   message(X, Z, "2 end summit")
>
}
```

```
CentralizedProcessing(X:org, Y:org, A:act, O:obj){

pull(X, Y, obj);
message(X, Y, "thanks");
Task(X ,act)
}
```

Fig.3: Library procedures

Let us suppose to have in our library the three procedures represented in (Fig. 3) and that the core activity of the summit is to make organization A processing tables coming from the organizations B – C, and then to share with them the result. We can define the overall summit process as:

```
MySummitWith (Member1:org, Member2: org, Task:act){

 <OpenSummit (A, Member1, Member2) & Task(A, start_operation)>;
 <
  CentralizedProcesing (A, Member1, Task, table1)
 &
  CentralizedProcesing (A, Member2, Task, table1)
 >;
 Task(A, join_table_processing_results);

 CloseSummit (A, Member1, Member2, result);
}
```

The actual execution of the process may be triggered by a call like:

MySummitWith(B, C, FindMax)

As we can see, procedures encourage process reuse but they are also a fundamental modularization tool.

4 Federation Infrastructure

Main components of the support infrastructure are the compiler, the enactment engine(s) and the interface wrappers.

4.1 Compiler

The purpose of the compiler is, starting from a single process definition, to extract information about the role of each organization in the process. In terms of cooperation, there are two main aspects we need to identify and they are related to: (1) the activities an organization has to perform and (2) the way in which activities performed by different organizations (or multiple activities within an organization) are synchronized.
The definition language enforces the point of view of an independent manager who wants to coordinate the work of different resources (the organizations) in order to achieve a specific result. This approach allows compact and easily understandable process definitions but for the actual enactment of the processes we take a completely opposite approach. The impact of this choice on the flexibility of the overall infrastructure will be clear looking at the enactment architecture.
For each organization we build a version V_{org} of the federated process that contains the specification of all and only the tasks the organization is requested to do and the synchronization points it has to maintain with respect to its partners. Without going into all the details of the specific compilation techniques, we focus on some of their crucial aspects. Basic operations are easy to map into V_{org} while the synchronization problems come with the composition operators. In this version of the system we do not allow

higher-order procedures, that means the parameters of procedure cannot be other procedures, so their mapping is quite linear. The problem we have, for example with sequential composition, is pictured in the following example:

$$< A(xx) \ \& \ B(xx) > ; < A(yy) \ \& \ B(yy) >$$

If *xx* is completed in A but B is still working on it, A has to wait until also B completes *xx* before starting *yy*: if B is faster than A the situation is the same. The compiler manages these situations with specific solutions that assure the intended semantic of the operators is preserved. Because each organization needs to be sure that all its partners will conform their behavior to the same set of rules, symmetry is fundamental.
This organization-centric approach allows a modular structure for the enactment infrastructure with major benefits also in terms of autonomy and security as well as fault tolerance. An organization may follow its own process independently from other members of the federation (autonomy) unless explicit synchronization points are specified. Benefits from a fault tolerance perspective derive from the autonomy of the organizations: if an organization experiences (temporary) problems its partners may not be affected.

4.2 Enactment Engines

The enactment paradigm has strong dependencies with both the cooperation model and the compiler techniques presented in the previous sections. In the actual enactment infrastructure we distinguish three main components (Fig. 4): workspace components (W), engines (E) and the interconnection support.
Focusing on a single organization, the engine has complete access to the workspace component and it can also communicate with other engines but, in a normal situation, it cannot interact directly with any PCE. Each engine E_x enacts the projection P_x of the federated process produced by the compiler for the organization X: its main job is related to messages and data management, task posting and synchronization. Also for the engine implementation, the complexity is concentrated in the support for multiple execution threads, sequential integrity and choice-step consistency.
Choice-step consistency problems, for example, depends on the fact that if a path (Pi) is chosen (choice operator) for one of the projections of the global process, we need to follow the same path in the enactment of all other projections. Major issue is that we allow different execution speeds in different organizations. In order not to introduce implicit synchronization points (with solutions like waiting for all the organization involved in the choice to reach the evaluation point), specific solutions need to be enforced both in the engine and in the compiler.

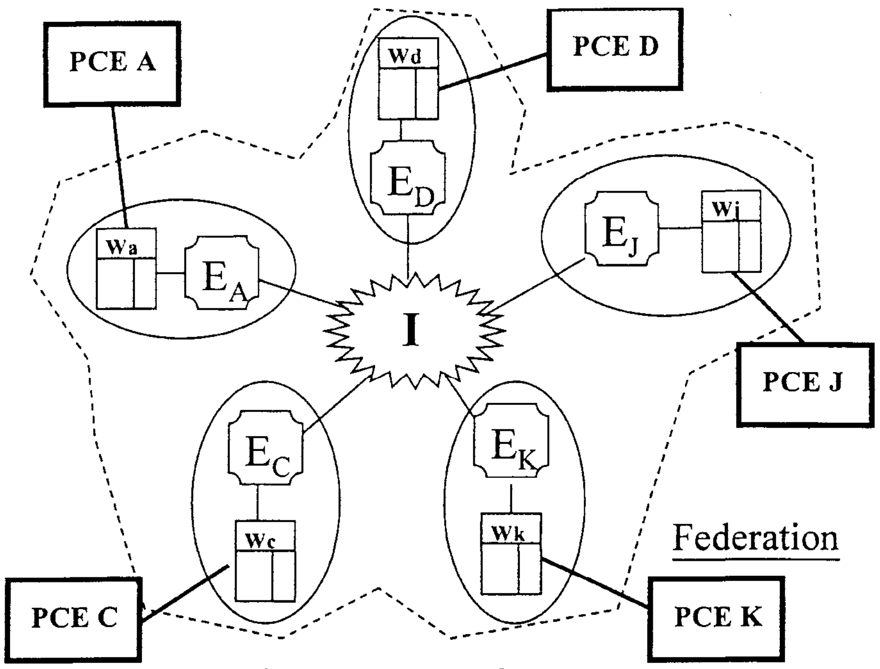

Fig. 4: Enactment Infrastructure

4.3 PCE Interface

The logic interface an organization has to the federation is provided by its workspace component. In practice the PCE of an organization needs a bridge to W in order: (1) to put and get messages and data and (2) to access the indications on the tasks it has to perform. W is mainly a container for data and information, and the bridge to the PCE depends on the level of automation it enforces. In our investigation we focused on two extremes (full automation and pure presentation) but solutions in between are also possible. Concerning the technology, we focused on Java and Corba though OLE is also under investigation.

We can consider, for example, the case of full automation based on Java. The wrapper uses an event-based mechanism in order to receive a notification every time the engine posts a new task in W. A one-to-one association between tasks and objects is established so that as soon as a task is posted the correspondent method is activated. The association of a task to a structured set of methods invocations is not directly supported but this limitation can be bypassed using a proxy method whose body contains the desired invocation sequences. In order to put data or messages into the workspace component the PCE can invoke specific methods of the wrapper API.

5 Deployment

Deployment considerations had a big impact on the actual implementation of the enactment infrastructure and flexibility was our main reference.

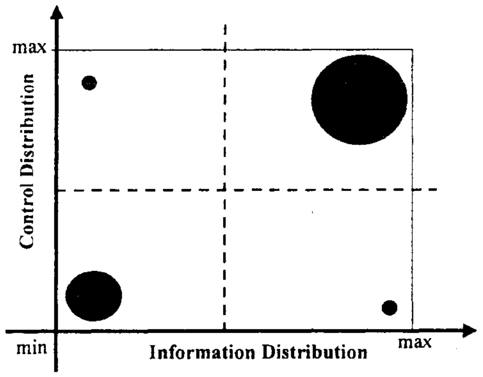

Fig. 5: Deployment of federation infrastructures

Looking at the problem of *where* to deploy the various components of the federation infrastructure associated to both *information* and *process* management, we notice (Fig. 5) that there is a tendency [2,3,5] to associate data with process logic. Alternative approaches have been investigated [8] but we decided that, given the dynamic characteristics of a federation context, the ability of our infrastructure to adapt to different situations without being re-engineered was a major goal.
The main components of the physical architecture (engines, workspace components and wrappers) are built on top of Java RMI infrastructure and they can be deployed following the distribution pattern that better suits the configuration of the federation.

6 Conclusions

Distributed object architectures (DCOM, CORBA, Java RMI) coupled with Internet and Intranet technology have a great impact on process-centered environments and distributed workflow management, both in terms of connectivity and applications automation. As projects become more complex they may span over different organizations and/or different parts of an organization and the coordination need of different PCE (federation) is the target of our work.
We present a complete infrastructure supporting federated workflow starting from the definition of the process to its actual enactment. Few basic operators and the possibility to build high-level modules are the design environment we provide while the enactment environment is based on a distribution-oriented compiler and a RMI (Java) based infrastructure. The TEAM (teamwork enabler abstract machine) abstraction is used to model the system and to specify the process semantics.

Bibliography

[1] S. Bandinelli, E. Di Nitto and A. Fuggetta. Supporting cooperation in the SPADE-1 environment. In *IEEE Transactions on Software Engineering,* Vol. 22, no. 12, December 1996.

[2] N.S. Barghouti. Supporting cooperation in the Marvel process-centered SDE. In *Fifth ACM SIGSOFT Symposium on Software Development Environments.* Herbert Weber (ed.), 1992.

[3] C. Basile, S. Calanna, E. Di Nitto, A. Fuggetta and M.Gemo. Mechanisms and policies for federated PSEEs: basic concepts and open issues. In *Proc. 5th Europeen Workshop on Software Process Technology.* Nancy, France, 1996.

[4] I.Z. Ben-Shaul, A. Cohen, O. Holder and B. Lavva. HADAS: A Network-centric framework for interoperability programming. In *Proc 2nd Inter. Conference on Cooperative Information Systems.* June 1997.

[5] I.Z. Ben-Shaul and G.E. Kaiser. Federating process-centered environments: the Oz experience. In *Automated Software Engineering*, Vol. 5. Kluwer Academic Publisher, 1998.

[6] I.Z. Ben-Shaul and G.E. Kaiser. Integrating groupware activities into workflow management. In *Proc. 7 th Israeli Conference on Computer Based Systems and Software Engineering.* June 1996.

[7] A. Giacalone, P. Mishra and S. Prasad. FACILE: A symmetric integration of concurrent and functional programming. In *Proc. of TAPSOFT'89,* Vol.2. Lecture Notes in Computer Science (LNCS 352). Springer-Verlag, 1989.

[8] D. Heimbigner. The Process Wall: a process state server approach to process programing. In *Proc. 5th SIGSOFT Symposium on Software Development Environments.* ACM Press, December 1992.

[9] C.A.R. Hoare. Communicating Sequential Processes. *Series in Computer Science.* Prentice-Hall, 1985.

[10] D. Hollingsworth. The workflow reference model. Workflow Management Coalition (WfMC), TC00-1003, November 1994.

[11] N. Krishnakumar and A. Sheth. Managing heterogeneous multi-system task to support enterprise-wide operations. In *Distributed and Parallel Databases.* Kuwler Academic Publishers, 1995.

[12] J. Miller, A. Sheth, K. Kochout and D. Palaniswami. The future of Web-based workflow. In *Proc. of the International Workshop on Research Directions in Process Technology*. Nancy, France, July 1997.

[13] R. Milner. A calculus of communicating systems. *Lecture Notes in computer Science* Vol. 32. Springer-Verlag, 1980.

[14] Object Management Group (OMG). A discussion of the object management architecture. January 1997.

[15] Object Management Group (OMG). CORBA facilities: common facilities architecture V4.0. November 1995.

[16] R. Orfali and D. Harkey. Client/Server programing with Java and CORBA. Wiley Computer Publishing, 1997.

[17] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In *Modern Database Systems: the Object Model Interoperability and Beyond*. ACM Press and Addison-Wesley, 1995.

[18] Creating Programmable Applications with OLE Automation. Vol. 1 and 2. Microsoft Press, 1994.

[19] A. Sheth, D. Georgakopulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden and A. Wolf. Report from the NSF workshop on workflow and process automation in information systems. In *Proc. NSF workshop on workflow and process automation in information systems: state-of-the-art and future directions*. A. Sheth (ed.), May 1996.

[20] K.D. Swenson and K. Irwin. Workflow technology: tradeoffs for Business Process Reengineering. In *Proc. Conference on Organizational Computing Sysyetms*. ACM Press, August 1995.

[21] G. Valetto and G.E. Kaiser. Enveloping sophisticated tools into process-centered environments. In *Proc. 7th IEEE International Workshop on CASE*. IEEE Computer Society Press, 1995.