Interaction Modelling in Federated Process-Centered Environments

Giacomo Piccinelli [♣]

Extended Enterprise Laboratory
Internet Business Management Department
Hewlett-Packard Laboratories
Bristol BS12 6QZ, U.K.

Tel.: +44 (0)117 9229610 – Fax: +44 (0)117 9229250
Email: giapicc@hplb.hpl.hp.com

Abstract

The evolution of distributed object architectures (mainly COM, CORBA and Java) opens an unprecedented range of possibilities in terms of automatic process management and PCEs (process-centered environments) play a key role in the software process re-engineering induced by these technologies. The human factor is still the major component of the software process but, as complexity and precision requirements increase, its role is changing and so does the kind of support it needs.

Process management is important within a single organization but it becomes fundamental for projects spanning over distinct organizations. Focusing on the interaction among distinct distributed PCEs (heterogeneous federation), we present a web-based process-oriented system for the definition and enactment of federated processes. The distribution and multi-organization problems are transparent to the process designer that is provided with a CSP-like definition language. The compiler and execution infrastructure exploit the interconnection capability offered by the web in order to support the cooperation among execution engines. The interface between a local PCE and the related execution engine depends on the characteristics of the PCE itself and the autonomy requirements (constraints) of each organization: some basic solutions are investigated.

[♣] Useful contributions to this work came from F. Marcello (University of Pisa), G. Zugliani (University of Pisa) and many members of EEL (Hewlett-Packard Laboratories, Bristol).

1 Introduction

It is difficult improving what we can not measure but it is extremely difficult measuring what we can not handle and process-based infrastructures offer a natural support for managing problem complexity [20]. The definition of a process embeds the know-how of an organization and the role of PCEs[1] (process-centered environments) in modern organizations is to manage such knowledge in order to turn it into procedural effectiveness and productivity.

Although the human factor is still the most important element in the software process [22], the evolution of new technologies [16,18] dramatically impacts on both the way in which processes are designed and the role human beings have in these processes. Software development tools become more specialized but, at the same time, there is a lot of emphasis on the integration of different components into global environments [18] and then into global processes [6,11,21]. Internet technology offers an unprecedented interconnection capability [12] that distributed object architectures [14,16,18] exploit in order to boost the creation of domain-focused and location-transparent environments [4,19]. LAN-based Intranets and even isolated workstations offer a great potential in terms of process-centered environments: the cooperation and coordination aspects become crucial [1,2,3].

While in the past PCEs supported the work of big groups of people with a wide variety of competencies, the personal-productivity tools now available allow small groups (even a single person) to develop substantial parts of a project if adequately supported. The split of the work tends to be based on vertical competencies more then on physic aggregation and the notion of "environment" reflects the technologic shift: the problem is the extents to what PCEs reflect this change. Specialization becomes instrumental for effectiveness and the notion of a general purpose PCE evolves towards a more realistic vision in which a common bedrock is preserved but domain-specific support is also offered to the users [10]. In this scenario, a new layer of management (PCE) is needed in order to support projects involving multiple competencies and the added value is in the process coordination. We refer to this scenario as *federation* [3,5] and in this context we locate our work. The benefits of the split between specific competencies and high-level process management are mainly in terms of modularity and flexibility. The use of PCEs as macro-resources allows a more compact view of the overall process and enables an easier integration of components coming from different organizations (ex. for joint projects) where privacy and autonomy are balanced by the need to achieve a common goal.

Focusing on the coordination and information exchange aspects, we present a system for the definition and enactment of federated processes involving different organizations and/or different parts of the same organization. After a brief overview of the more popular distributed object architectures and their impact on PCEs, we present and discuss the cooperation paradigm we enforce. The process-definition formalism is presented together with some example of its application on well-known cases. The architecture of the enactment infrastructure is described and we focus on its integration with local environments.

---

[1] We use the term PCE to indicate both the environment and the system that manages the definition and enactment of a process in that environment. In case of ambiguity we will specify the distinction between the two concepts.

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 architecture 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 ask other applications to perform task and/or to supply information. Moreover, meta-information is also 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. The execution (enactment) of a process is supported by a specific application (process engine) and the possibility to actually interact with the environment (other applications) allows not involving human beings in mechanic operations leaving them free to concentrate in the creative aspects of the process.

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 presented and discussed.

3 Cooperation Model

The purpose of a cooperation process is to organize resources and know-how of different organizations[2] 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 the fact 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. Despite the commitment to the common goal and the need to exchange data and services, autonomy and secrecy are fundamental issues for the members of a federation and any infrastructure that targets federative process support has to deal with these requirements.

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

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 and 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 federation workspace and federated process in order to understand the dynamics of the system.

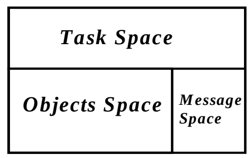

Fig.1: Workspace component

In its *object space*, an organization puts the data it needs to share with its partners and it can retrieve data produced by its partners and relevant for the tasks it has to perform in the in the context of a specific federated process. Data are moved, replicated or deleted from the object space depending on the definition the federated process and following specific rules automatically enforced by the federation infrastructure. Each operation allowed by the process definition language has a specific semantics in terms of the effects on the federation workspace and this semantics cannot be modified by the organizations. An organization has immediate access only to the data in the object space of its W and these are the only data exposed to the federated process: autonomy of the organizations 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.
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 federated process. It gives all and only the information on its internal state it agreed on during the definition of the process and the same happens for the information is receives on other members of the federation.
While objects and messages represents static assets for an organization, the dynamic aspects of the federated process are modeled posting explicit task requests into the W task space of the federation members. 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 in the associated W. Multiple tasks may be submitted to an organization at the same time and the intended execution model enforces both truly concurrent and non-deterministic sequential paradigms. All the tasks indicated in the W can be executed either in parallel or in any order the organization prefers and

when a task has been accomplished, the organization can mark it as finished. There may be no visible effect on the workspace caused by the execution of a task and in this case the trust relationship that characterizes a federation is fundamental. The flow (control) logic within the overall process is transparent to the federation members during the enactment phase, unless explicitly provided.

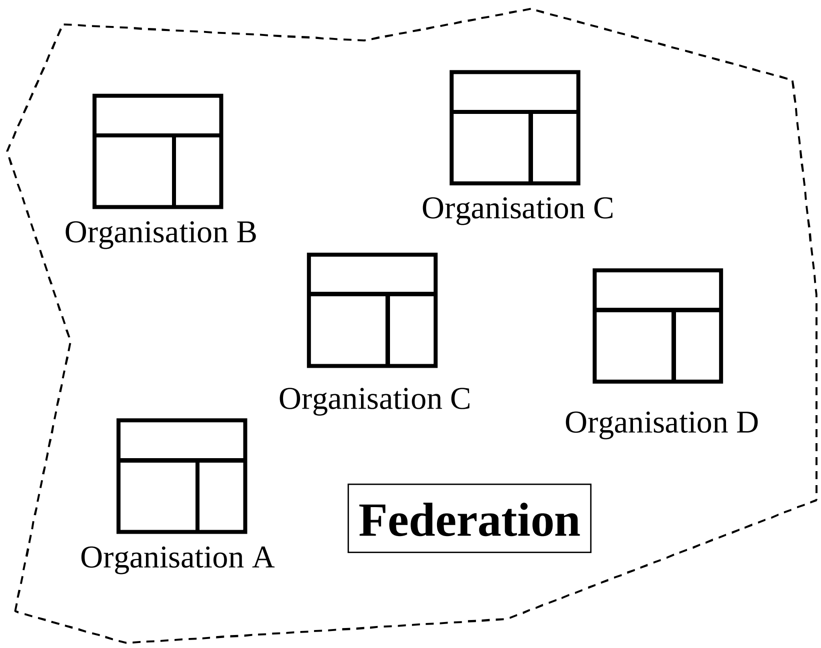

Fig .2: Federation workspace

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 single management entity (encapsulated in the enactment infrastructure) 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 PCE 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 (W).

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 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 entire language has been formalized following an approach (operational style) similar to the C-FAM (concurrent functional abstract machine) used for FACILE [7] but we prefer to give a more direct description of the meaning of the various elements in the formalism. 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 **&** … **&** P_n **>**	Parallel Composition
(expr**)** **[**P_1 + … + 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 and we focused on the three operators listed in Tab. 2 plus the possibility to have procedures. P_i are generic processes and **nil** represents a null process.
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). The set of typed variables $\{V_i\}$ 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 and the execution semantics may be given in terms of the classic rewriting rule:

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

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). 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. The designer may concentrate on some aspects of the federation once and then reuse the solutions in different places within the same process or for the definition of new processes. Procedure libraries become precious assets as they actually collect know-how on process management. Moreover, compositional features offered by the language allow a continuous knowledge evolution in terms modeling complex behaviors (for example ACID transaction schemas [17]) building on top of existing components.

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

Tab. 3: Procedures

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, Y, 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)

Although we may not plan to reuse the peculiar structure of this process, we can wrap it into a procedure definition in order to split into manageable modules a more complex definition.

4 Federation Infrastructure

Main components of the support infrastructure for the implementation of a federated process are the compiler, the enactment engine(s) and the interface wrappers. We briefly present the compiler and wrapper technology while we focus on the enactment engine that are a fundamental element in our architecture.

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 enactment of the process. In terms of cooperation, there are two main types of information 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) has to be synchronized in order to preserve the semantic of the language

constructors. 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 developed, 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 procedures themselves, 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 we take the point of view of A, we may expect to execute the task *xx* and when *xx* is finished we would like to start the task *yy*. The problem is that semantics for the sequential composition " ; " indicates that both A and B have to complete the task *xx* before any if them can start the execution of *yy*. If *xx* is completed in A but B is still working on it, A has to wait until also B completes *xx*: 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 global process is preserved. Symmetry is fundamental in the compilation process as each organization needs to be sure that all its partners will conform their behavior to the same set of rules.
This organization-centric approach allows a modular organization of 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, unless explicit synchronization points are specified, independently from other members of the federation (autonomy). Security is enforced by the fact that the compiler is consistent with the PSL semantics and all the actions an organization is requested to perform derive from the agreement specified in the process definition. Benefits from a fault tolerance perspective derive from the autonomy of the organizations: if an organization experiences (temporary) problems its partner may not be affected.
The equivalence between the global federated process and the set of single-organization processes is based on the evaluation of the changes of the federation workspace and on the recursive structure of the process. We anticipate that the result of the compilation is location transparent meaning that we model independent components but information on the physic location of the organizations (components) is ignored.

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 (all the links between components). The content of a workspace component has already been discussed but now we discover its first access point (the interface with the related engine): we will see later its interface with the PCE of an organization (wrapper).
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 Ex enacts the projection Px of the federated process produced by the compiler for the organization X and 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.

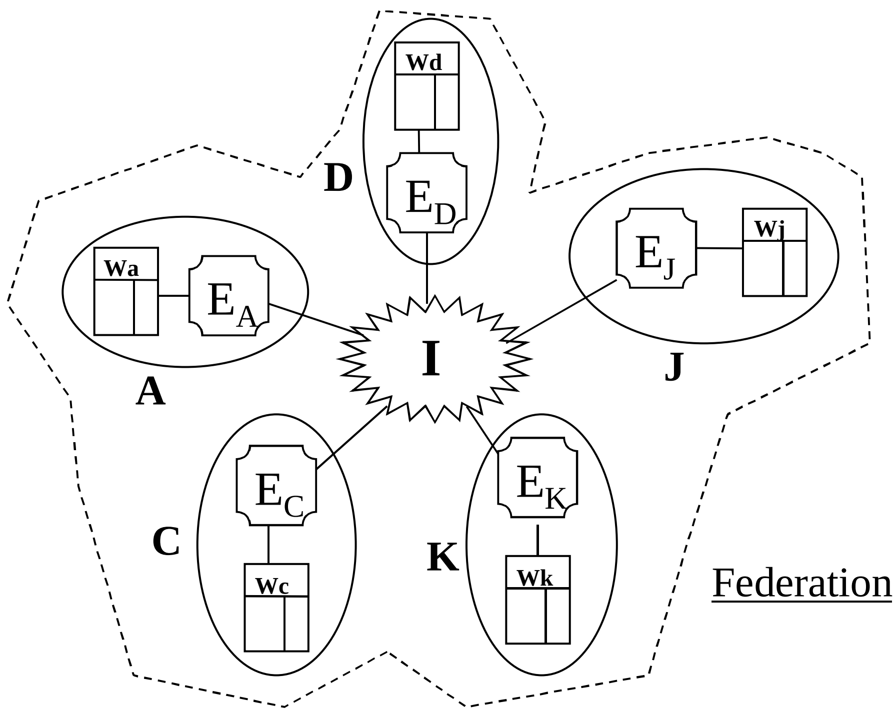

Fig.4: Enactment Infrastructure

Choice-step consistency problems, for example, depends on the fact that if a path (Pi) is chosen, within a choice operator, for one of the projections of the global process then also in the enactment of all other projections we need to follow the same path. Major issue is that we allow different execution speed in different organizations and, 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.

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 the W in order: (1)

to put and get messages and data as indicated by the tasks posted by the process and (2) to access the indications on the tasks it has to perform. The W is mainly a container of data and information, and the bridge to the PCE depends on the level of interactivity and 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 the 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.
In general the characteristics of the wrappers are related to the level of detail in the process specification and to security (autonomy) considerations. We may want, for example, to integrate a complex tool [21] into the federation allowing automatic access to the functions it provides. In the process specification we need detailed procedures encapsulating information on how to access those functions and the wrapper for the W of the organization that hold the tool should have direct access to the tool in order not to slow down the process. The drawback for the owner of the tool is that, if its PCE does not enforce access policies to the resources, the organization may loose part of the control over its own environment.

5 Deployment

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

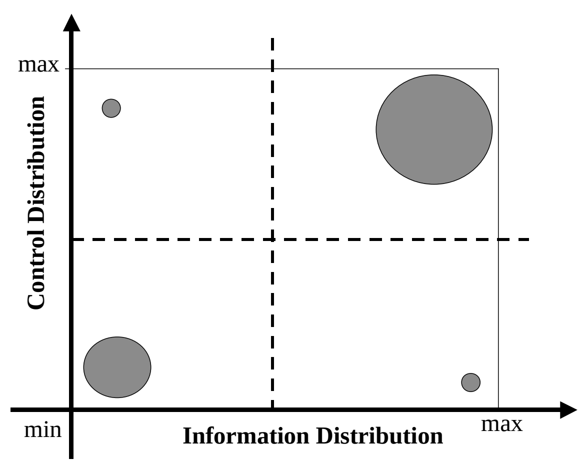

Fig.5: Deployment of federation infrastructures

Considering the problem of *where* to deploy the various components of the federation infrastructure, focusing on *information* and *process* management components, we noticed (Fig. 5) that there is a tendency [2,3,5] to associate information 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 our architecture (enactment aspect) are the engines, the workspace components and wrappers and thanks to the support of Java RMI (remote methods invocation) we enforced the possibility to allocate them in different ways without major changes.
Solution in which all the Ws are in the same physic location may be enforced if, for example, an organization offers efficient storage facilities to the federation and privacy problems are not dramatic. The volume of data moved in/out the W by an organization and the exchanges among the various Ws must be analyzed. If performance optimizations are important, we may decide to install on the same machine (system) also the Ws keeping in the local PCE only the wrapper or a proxy.

6 Future developments

Higher-order procedures are among our objectives for the evolution of the language but a more flexible approach to variable and data structure is also under investigation. Concerning implementation aspects, a deeper integration of wrappers with OLE (without Java or Corba proxy) is a near term objective in the perspective of tools automation.

7 Conclusions

Distributed object architectures (DCOM, CORBA, Java RMI) coupled with Internet and Intranet technology have a great impact in process-centered environments both in terms of connectivity and application automation. As software projects become more complex they 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 the federated process starting from its definition to its actual enactment. Few simple basic operators and the possibility to build high-level modules and process libraries are the design environment we provide while the enactment environment is based on the result of a distribution-oriented compiler and specific cooperation environment. Concerning the deployment of the federation infrastructure, the components are built taking into consideration location transparency problems therefore we can tune the deployment process on the peculiarities of the federation.

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.

[22] A.L. Wolf and D.S. Rosenblum. Process-centered environments (only) support environment-centered processes. In *Proc. 8th Inter. Software Process Workshop (ISPW8)*, Wadern, Germany, March 1993.