

Architectures for Process Support System Interoperability

*J. Estublier, P.Y. Cunin, N. Belkhatir
L.S.R. Actimart, Bat 8, Av de Vignate
38610 Gieres FRANCE
{jacky/cunin/belkhatir/amiour/dami}@imag.fr*

Abstract

The problem we would like to solve is to find a way to build a real and wide scope process support based on a number of existing Process Support Systems (PSS). In other words how to make these PSSs interoperate in order to build the ideal process support system. Interoperability is a goal that has been pursued for a long time, in all domains. This paper evaluates the different paradigms that have been used in the domain of process support.

The most common approach is based on Corba like technology. In this approach, each component declares the services it provides. The process itself is executed by a special PSS which asks the “right” PSS to execute the “right” service when required. It is *control based* interoperability. We show that this kind of technology is suited as long as the process model can be partitioned in such a way that each partition (called a component) can be handled entirely by a single PSS.

The other basic approach is based on the availability of the state of a common virtual process. Each PSS can see the state of the common process, can change it and react to changes performed on that state by other components. Each component collaborates freely to the federation behaviour. It is *state based* interoperability. We show that this approach is flexible but it does not provide any control of the process.

It is shown how these dual approaches can be reconciliated, keeping the best of both paradigms. We claim that the choice of a paradigm or a combination must be dictated by the kind of interoperability required, not by platform technological limitations. This paper also claims that we need a clear distinction between the functional and operational aspects of a federation.

We have designed and implemented the APEL interoperability platform which supports both paradigms and their combination. This paper presents the architecture of that platform, and discusses its major characteristics. We discuss the experience gained and the issues still to be addressed.

1 Introduction

One major objective within the process support domain is to implement, in a company, a complete process support system, which includes the tools, systems and techniques that process participants (developpers, managers, ..) are used to.

This objective is exemplified by the following (typical) scenario. Let us consider an organism (e.g. a company department) that has been developing software systems since several years. People in that organism are using many different tools, most of them being commercial tools, to support parts of their technical or managerial activities, e.g. specification, planning or mailing tools. What they would like to have is a flexible working environment able to provide efficient support for their daily work process while incorporating in a transparent way their usual methodologies, techniques and tools.

Powerful (commercial) tools have specific characteristics, e.g.:

- they usually embed process or methodology features, a *Local Process*
- they manage their own data with some persistency aspects, a *Local State*.

Such tools are Process Sensitive Systems (called thereafter PSS) able to manage internally the state of their view of the process they are involved in. The concept of interoperability within the process support domain is a very important one if we consider the fragmented structure of complex processes and the heterogeneity of their representations.

A federation of such tools can be seen as a virtual PE which execute a process, called the *global process*. This global process is the result of all executions, thus it does exist, but it is not formalised in any way. Building a federation always consist in identifying a part of the global process, called the *common process*, which is responsible, in one way or another, for federation cohesiveness. This paper identifies architecture styles and show how they use the common process/common state to design and execute a federation of PSSs.

We have identified two basic families of PSS federation architecture styles:

the **control based** architecture style. A federation is control based when each PSS is a core component relying on the principle of encapsulation and when interaction between PSSs is based on process routines calls. By encapsulating the process, each PSS is autonomous and responsible of its (sub) processes, but it requires that the common process be decomposed into such independent process components. An environment based on this style requires explicit control flow and explicit invocation of PSS for executing activities. It is a centralized control.

The **state based** architecture style. A federation is state based when each PSS shares a common representation describing the current progress of the global on-going process. It is the means by which PSSs can interoperate and deliver their services. Invocation of activities in this style is implicit and is more a result of events rather than a source of control. A PSS cannot control the order of invocations but can see the effect on the common state of others PSSs. So decomposing the process into components is not necessary in a state based architecture. The shared state increases PSSs cohesion but decreases PSSs coupling.

The paper is structured as follows:

Section 2 presents the control based architecture style and section 3 the state based approach. Section 4 proposes to combine the two styles of architectures and develops a flexible architecture allowing full interoperability of PSSs together with a high level of consistency and control.

Section 5 presents the APEL architecture we have developped. It is a concrete and working instance of such a flexible architecture

Section 6 concludes by stressing the importance of having a general purpose architecture able to support various specialized ones (e.g. tuned to a specific method or paradigm, etc) and allowing dynamic evolution of cooperation and control strategies.

2 The control based approach.

The basic idea behind this architecture is to abstract (i.e. encapsulate) the services provided by each tool, in order to hide the heterogeneity of each tool (formalism, platform,...). The tools services are defined in a common formalism, a Process Interface Description Language. In this way, a tool can call another one, whatever their respective formalisms and the communication technology used. It is a Corba-like philosophy [28][30].

A basic tool provides the services (sub-processes) defined in its interface (in PIDL language). A tool (T0 in Fig 1) plays the role of supervisor, and calls the others (tools).

The supervisor model expresses formally the relative ordering of sub-processes to be executed by the other tools; it can contain tool allocation strategies (for load balancing for exemple), consistency constraints or network control. This supervisor model formalizes, at a high abstraction level, the process performed by the federation; this is why it is called the “common” model.

This common model expresses formally the *goal* of the federation. The basic tools do not know to which goal (process) they contribute to, and have no possibility to (directly) influence the common process.

The supervisor has no knowledge about how a service will be provided. From its point of view, a service is atomic. It means the supervisor process is a composition of process fragments, each fragment being executed by a single tool. It also means that the supervisor model does not provide any information about the sub-processes. To program a federation, a modeller will still have to write models directly for each tool of the federation in its specific formalism.

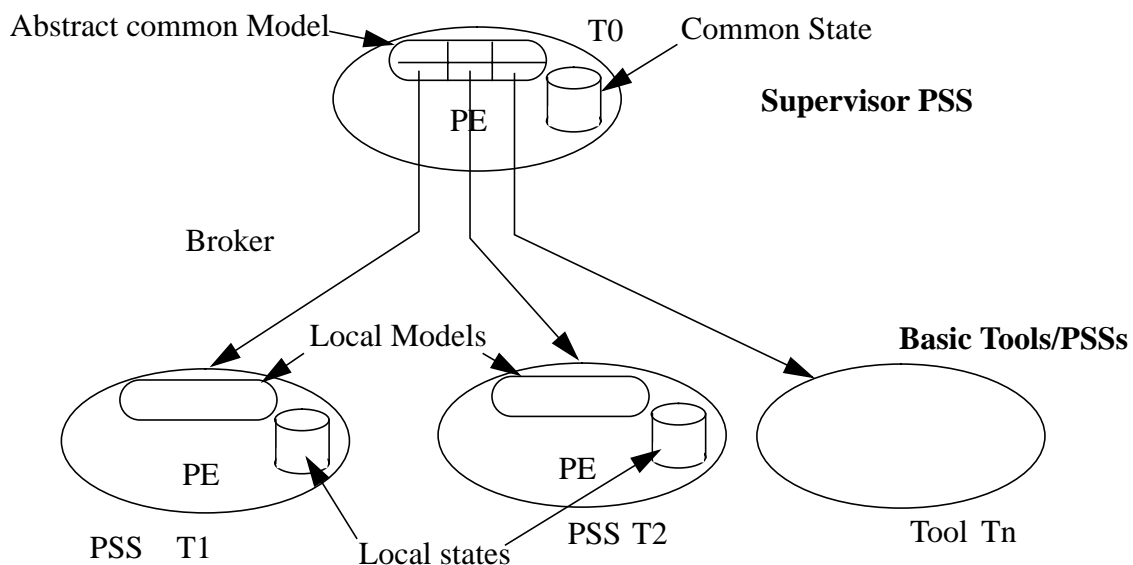


Figure 1: Conceptual architecture of a Control based Federation

In this approach, each tool is considered as a process component, known only through its interfaces and maintains its autonomy. The meta model is not used to translate the local process model into the common model, but only to define the tool interfaces.

The main characteristics of the approach are that:

- the supervisor has the knowledge of the goal of the federation,
- the supervisor is in full control of the way to reach the goal (its process execution),

-
- tools are independent and autonomous.

It has the following drawbacks:

- the supervisor model must be decomposed in such a way that each fragment is executed entirely on a tool,
- any change in the federation implies changing the supervisor model,
- the supervisor model is a rough abstraction of the global model.

This paradigm is the major one today. Many work have as goal to extend that paradigm, either to provide new services on top of Corba or for specific process support [3][8].

Let us use the society metaphor to illustrate process federation architectures. Each society agent (PSSs) is a human or a company and has its own model and goal, and is capable of providing some services in an autonomous way.

The service paradigm belongs to a centralized society. The goal and the rules of the society are clearly defined and enforced by a centralized government (the supervisor). The place and role of each human/company is controlled by the government. In turn each company can use the same model, or another one, to control its own workers and sub-companies.

3 The state based approach

The approach above works fine as long as each model partition is executed by a single tool.

If tools forming the federation are PSS COST tools, like MSProject, Lotus Notes, Adele and a monitoring tool, the hypothesis may be the following, it is likely that the same process fragment may need to be handled by more than one PSS simultaneously; in which case the partitioning hypothesis does not hold.

Many atomic aspects in the process require the fine grained collaboration of different PSSs. For example, creating an activity in the common model, (which is an atomic operation) may involve MSProject (for planing), Lotus Notes (for notification), Adele (for work Space creation) and the monitoring tool for the team leader controlboard. Whatever the structure of the process, and the grain of components, no partitioning can be found.

From the point of view of an external user, the common model is the behavior of the PSS federation thus the common state exists. That common state is an abstraction of the local states found in the collaborating PSSs.

The common state provides a very good basis on which the collaborating PSSs can synchronize their work, because the PSSs in the federation can observe the common state, and update their local state accordingly, or change the common state according to changes performed in their local state when executing their local model. In the example, the creation of a common activity changes the common state, which is noticed by the PSSs, each one executing their specific aspect: planning (MSproject), notification, workspace and controlboard.

The implementation, of such an approach is exemplified by the ProcessWall[23]. There is no specialized common PSS, the common model does not exist, the architecture is only based on the effective presence of the common state, on which all PSSs synchronize their activity during execution. This is why this approach is called *state based*. The Database community have addressed the interoperability problem from this point of view (active databases[13] and process support databases [9] [33]).

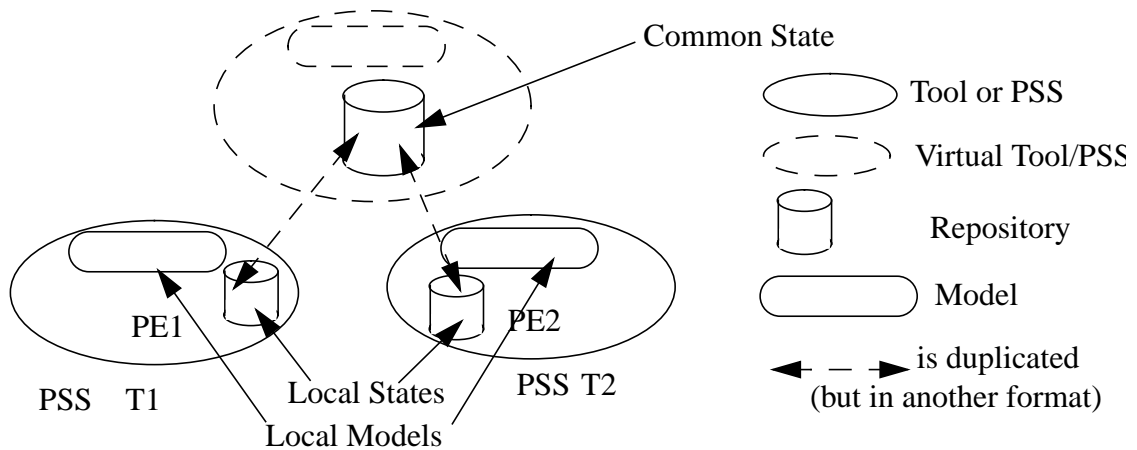


Figure 2: Conceptual architecture of a state based federation

If the PSSs are heterogeneous, the format of the local states and common state are different; the common state format plays the role of a common format, making it possible to synchronize at execution. Thus this approach requires the definition of a common Meta Model where the relevant concepts are defined (i.e. the process concepts), and a language defining the format in which these concepts will be represented. However, in this approach, there is no need to define the semantics of these concepts, because it is the responsibility of each PSS to execute the process.

This approach was followed by the workflow management coalition when defining the WPD [22] [35] [36] [37] (Workflow Process Definition Language) or the PIF (Process Interchange Format) [27]. WPD is a language in which process concepts are represented, but since there is no semantics, this language is not executable.

The state based architecture have the following positive characteristics:

- It allows PSS with overlapping functionalities to collaborate,
- Collaborating PSSs do not need to know each other (but to see their effects)
- At any time a PSS can join the federation and collaborate,
- External users can monitor the common process, observing the common state,

It also has the following drawbacks:

- The common model is not explicit,
- The whole architecture relies on the “correct” behavior of each PSS,
- There is no consistency control

With respect to the previous approach, the common PSS and common model are now virtuals, only the common state is real, and each PSS can access and change the common state. The limitation of partitioning is removed, but there is no formal common model, no explicit goal and rules.

In our society metaphor, the state based paradigm corresponds to an ultra liberal society, where each human/company observe the state of the society (common state) and decides to “collaborate” freely. Groups of humans can handle a work in common observing the actions of the others (cooperation between overlapping PSSs). No common process (the goal i.e. the desired future) is defined, no rules (correct behaviors and laws) are enforced. It is an anarchic society; which can work only if each component (human) behaves “correctly”, which is unlikely in a human society, but not in a PSS federation.

4 Toward a general approach

In the control based approach, PSSs do not know in what they are participating in, but there is formal knowledge of exactly what will be executed (the common process). In the state approach, each PSS knows to what it contributes, but there is nowhere the knowledge of what will happen.

The first approach provides the goal and control desirable in a federation, but introduces severe limitations of partitioning and lack of flexibility. The state based approach provides the desirable flexibility and generality, but at the cost of losing global control (goal and rule enforcement). There is probably an intermediate position between dictatorship and anarchy.

The process based approach (process state + process model)

To avoid anarchy, the common model should be formally defined, for modelers and managers to understand, tune, customize, optimize the company process. If PSSs provide overlapping functionalities, this common model subsumes and abstracts part of the heterogeneous local models; it provides a single global view of the process. This common model should be really executed, and each one of the PSSs comprising the federation should be able to execute the part it is responsible for. In other words, the goal of the federation becomes explicit and enforceable. It is semantic interoperability[23] [25] [26].

This approach requires at least:

- The *common model* to be formal and executable, thus
- A *common process formalism* to be formal and executable, thus
- A *common meta model* with clearly defined semantics.

In this context, the common model includes part of the local models, but also specific parts, if needed. If the PSS are heterogeneous, the formalisms in which the local models and the common model are defined are not the same. The formalism of the common process plays the role of the common formalism of the federation. It can be used to exchange process fragments between heterogeneous PSS, like the WPDL [37] but in contrast with WPDL, the meta model as it has semantics, the common model can be executed. The common PSS can be real. In this case, the architecture is the following:

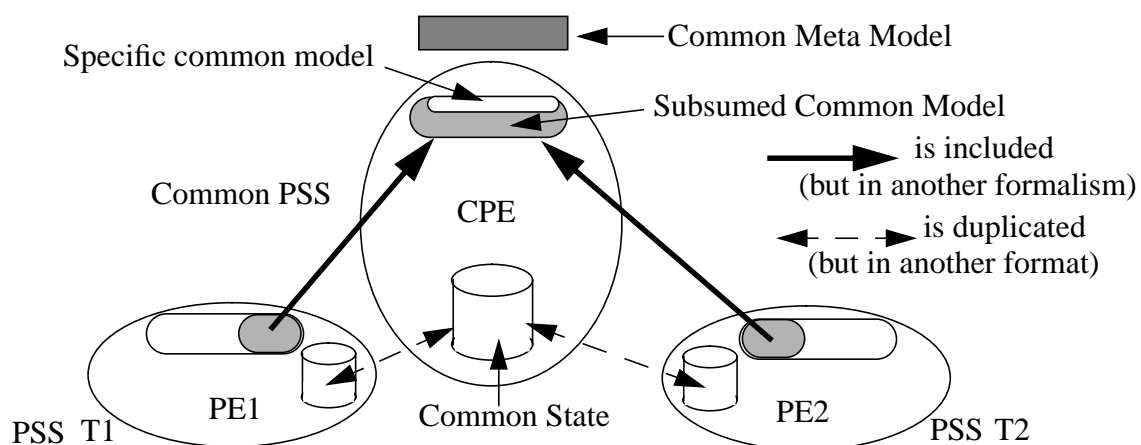


Figure 3: Conceptual architecture of a Process based federation

The common PSS executes the common process model with respect to the common meta model semantics, and makes the common state evolve; it ignores the other PSSs. Each one of the other PSSs, observing the evolution of the common state, can perform the actions they are specialized for. For example, suppose the common model dictates the creation of an activity A; this is executed by the common Process Engine, which is reflected by a change in the common state. This common state change alerts the other PSS: MSProject updates the planning accordingly, Lotus notes notifies the users, Adele builds the work space etc. In turn these actions may make evolve the common state.

The technology involved here is similar to the state based approach; it is event based (implicit and asynchronous invocation).

This solution can be seen as a compromise; we retained the common PSS (from control based) and the common state (from state base). We keep the existence of the common model, and thus the visibility of a global goal with the flexibility and ability to handle overlapping PSSs.

The characteristics of such an approach are the following:

- It allows PSSs with overlapping functionalities to collaborate,
- The common process is explicit and enforceable,
- Collaborating PSSs do not need to know each other (but to see their effects)
- At any time a PSS can join the federation and collaborate,

It also has the following drawbacks:

- The whole architecture relies partially on the “correct” behavior of each PSS,
- There is no consistency control

In its most achieved implementation, the PSSs can not only consult and change the common state but can also consult and change the common process model. This opens interesting possibilities for evolution control and it allows a more “collegial” way of managing the federation.

In our metaphor, the process based approach corresponds to a society where there exist a government (the common PSS) and a global society development plan (common process). The plan is enforced by initiating large works (like building highways, i.e. executing the common process); humans and companies react to these events, and contribute to the correct realization of that work. Most often only the order to build is sent by government, the actual work is performed cooperatively by individuals and companies, based on the knowledge of the common goal, the current state of that goal, and the actions performed by the other actors (observing the common state).

In this society, the realization of a common goal may require the fine grained collaboration of many agents, but this not managed by the government. A human may add a common task (that may be handled by others), as well as “contribute” freely to the common state. In this society, there are no laws and no control, it makes the assumption everybody will contribute honestly and efficiently to the common process.

The General architecture (Process + control)

The process based approach still lacks global control and consistency enforcement. There is no way to enforce how the different PSSs should contribute. For example, when creating an activity, there is no way to execute workspace creation before user notification (no event ordering); nor to undo everything if one of the aspect fails (no transaction control); nor to avoid that more than one PSS perform the same task (no missing or duplicate reaction control).

The common model contains the “functional” aspects of the common process, not the operational one. We propose to add an *interoperability model*, the purpose of which is to focus on technical “details” of how aspects mentioned in the common process should or must be handled. This model is interpreted by an interoperability PSS, which has the privilege to know which the PSSs are composing the federation PSSs, to know (a sub-set of) the services each one provide, and has the capability to explicitly invoke these services.

The interoperability model contains information related with the consistency control of the common model like the ordering of PSS reaction to the same event or transaction control; and information related with the federation distribution, like PSS localization, network topology, name space of services, and so on.

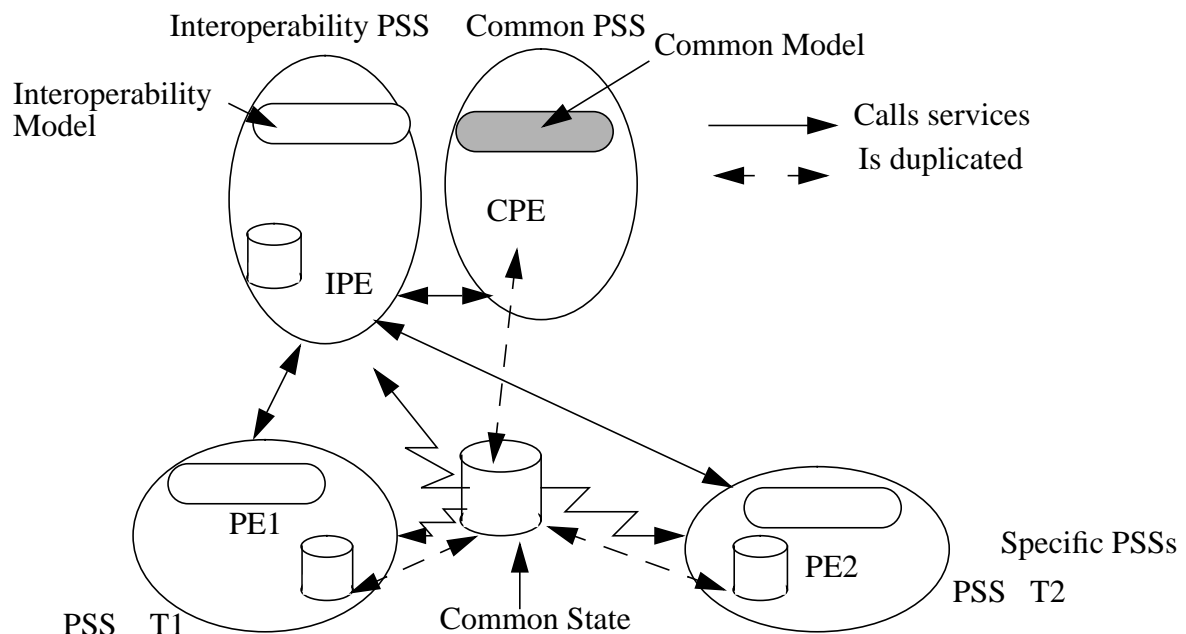


Figure 4: Architecture of a general federation

This architecture covers the complete spectrum from control based to state based. The “pure” control based approach Fig 1, interoperability and common PSS are merged; the common state is hidden. In the “pure” state based approach, the interoperability PSS and common model are missing. In process based approach, the interoperability PSS is missing.

However, what make the originality and interest of the approach is that:

- All cooperation paradigms can be implemented.
- There is a clear distinction between the functional aspects (common PSS) and the operational ones (interoperability).
- Mixture between all paradigms can be used, for any aspect.

Take again the example of the creation of an activity A. If A is present in the common model, the creation is performed by the CPE; otherwise by an arbitrary PSS. This creation changes the common state. The desirable behavior is that MSproject, Lotus, Adele and the monitor contribute to the A creation. If consistency control is needed (ordering or transaction control for example), it may be wise to let the interoperability PSS capture the event “A is created” and then call all the PSSs performing the required control (it is a control based paradigm); otherwise it may be simpler to let each PSS react freely to the event “A is created”. It may also be the case that a sub-set of PSS need control, while another sub-set not; in which case the interoperability

PSS will be in charge of controlling the first sub-set, the other PSSs reacting “freely” to the same event.

It shows that the use of a control or a state based approach should be strategy decision, and not, as in today systems, imposed by the technology. It shows also that the “good” compromise may evolve during time, depends on the semantics of the common model and the characteristics of the others PSSs.

It is very important to note that changing between state based to control based can be performed without any side effect on the common model, and even without any side effect on the collaborating PSSs, as soon as the interoperability PSS can dynamically inhibits PSS event reception. This is a significant improvement toward reusability of models and COTS PSSs.

Following our metaphor, this is a modern and democratic society where the choice between order and rules v.s. liberty and free enterprise is up to the society itself. Some aspects can be very closely controlled, both for goals (common PSS) and means (interoperability PSS) like defense, education, health; while others can be left to the appreciation of each agent like commerce. It is striking to notice that in societies too the compromise between control and liberty is eagerly discussed and constantly evolve dynamically.

5 The APEL Architecture

The APEL process support environment has evolved over the last 4 years, and has experimented a number of the previous paradigms [15]. The current APEL architecture is an instance of the general architecture mentioned above. This chapter shows the feasibility of the general architecture, provides some details on the APEL architecture and gives some feedback on the experience with that tool.

The APEL architecture is designed essentially for supporting interoperability between PSSs [16] and relies on the following concepts:

Common Meta-Model (CMM). To interoperate in the APEL environment all PSSs, must agree on the semantics of concepts that enable them to exchange knowledge with a common understanding. The shared concepts must be general enough to allow a reasonable range of common processes to be defined and managed among different kinds of interoperating PSSs[14]. At the same time, they must have sufficient semantics to enable semantic process interoperability.

Common Process Model (CM). The Common Process Model, written in the Common Proecess Modeling Language.

Process Server (PS). At execution time, a Process Server contains the process and model under execution that is, it contains both a reification of the Common Model and a reification of all entities created by the process under execution. The PS interface allows components to create any entity and to arbitrarily change the current process, as well as the process model. This capability is the basis for evolution support.

Event Server (ES). The event server catches all APEL events (as defined in the CMM) and dispatches the event, along with associated annotation to all components that suscribe to that event.

Common Process Engine. The common Process Engine, based on the events received from the event server, enforces the semantics of the process present in the Process server, and is in charge of executing the Common Model.

Interoperability Model (IM). This model, in the current implementation this model is limited to handling inconsistencies, non determinism and ownership of some entities.

Interoperability Process Engine (IPE). The Interoperability process Engine receives requests from the Process engines, and suscribe to events. It is in charge of providing an answer to requests and to transform some events into requests to other PSSs. It is in charge of ensuring some degree of consistency in the federation, by defining ownership of components over some PS artifacts, vetoable changes or default handling component.

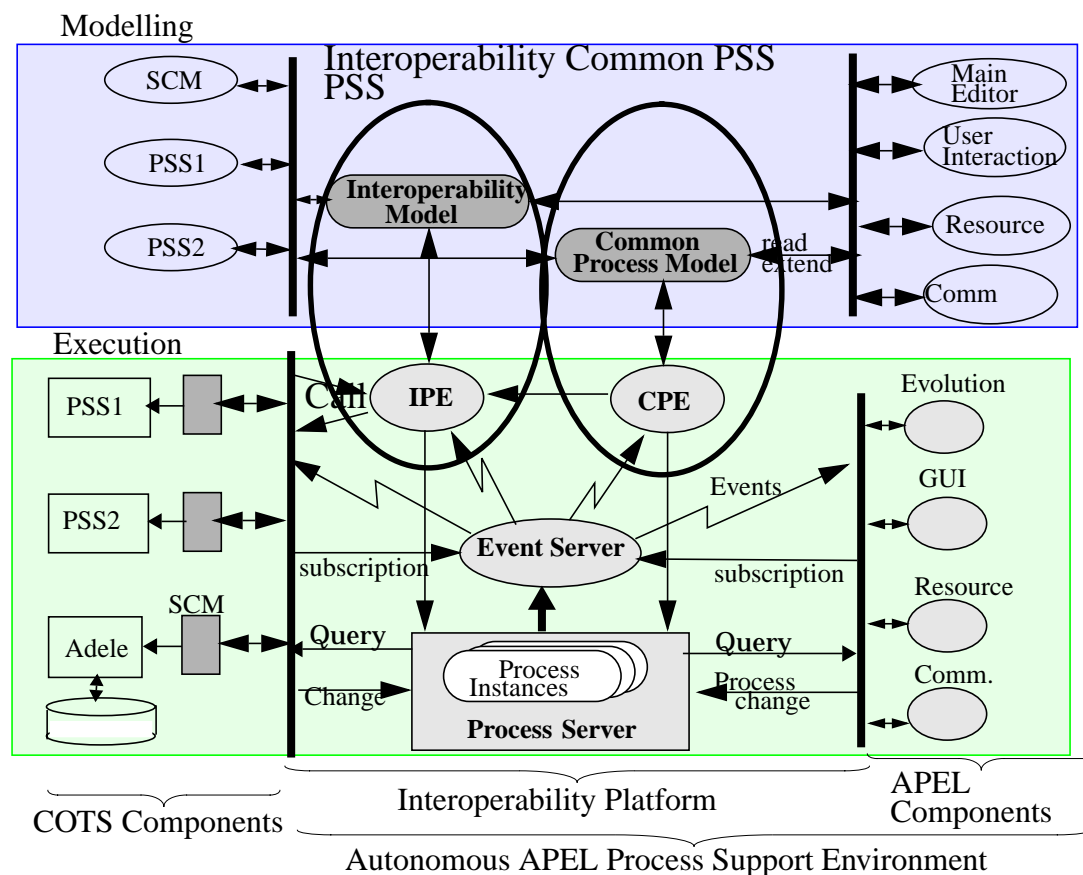


Figure 5: The APEL Federation

We have two kinds of components in the APEL federation: the ones we have built on purpose, in order to provide a complete process support environment; and the COTS PSSs. The on purpose PSSs are

- main editor. A graphical tool for modeling processes based on the CMM. Its Process Engine is the Common Process Engine.
- Resource manager. In charge of modeling resources and organizational aspects. At execution, its PE is in charge for allocating these resources.
- Communication manager. In charge of modeling message flow. At execution its PE is in charge of the effective handling of messages [1].
- User interface. Is in charge of providing process performers with a comfortable interface, based on the desktop paradigm.

-
- Evolution manager. Is in charge to provide process managers the help required in order to design and perform a process change (model and/or instance level).

The COTS PSSs in the APEL federation are the software configuration management (SCM), based on the Adele PSS [17], [18] and the MSProject PSS for planning purposes. Experimentation with COTS PSSs has just started. We expect more experimentation shortly.

6 Conclusion

Until now, in all domains, systems targeting interoperability have focussed on a single paradigm either control based (the Corba way) or state based (event based systems). We have shown that each paradigm has characteristics that make it convenient in some cases, but not in others. The control based paradigm emphasizes order and precision; the state based paradigm emphasizes flexibility and liberty.

It is our claim that not only we need both paradigms but also that, depending on the desirable compromise between control and flexibility, we need various degrees of combination. We also claim that this compromise is very evolutive, and that making evolve that balance must be possible at very low cost.

Many systems focussing interoperability are homogeneous for example Spade [4][5], LEU [21], Oz [10]). Their point is to distribute the model (in a single formalism) and/or its execution, not to make interoperate COTS tools; which is our point. In our terminology they are a single (but replicated) PSS federation (!). In Endeavor[11], low levels ad-hoc handlers are interoperating, not COTS PSSs.

Heterogeneous interoperability is exhibited in Provence [6][7], but at a low level of abstraction. The point was not to make interoperate COTS but to extend Marvel with file system control. In Meteor [31],

Many systems have focussed on the concept of point of view [19][25][29] [32][34]. These work address the problem of how to model a complex system by a combination of point of view, which bears some similarity with our approach. However the focus of the point of view approach is consistency checking at model level, not the interoperability of heterogeneous PSS, nor architectural issues.

We think our work contributes in at least two directions.

- We show that a general architecture supporting “all” the current interoperability paradigms is feasible, and indeed it has been experimented in the APEL platform.
- We claim that there is a need for separating clearly the functional aspects (the common process) from the control and consistency aspects (interoperability model).

The compromise between control based v.s. state based paradigms is a matter of strategy which is permanently subject to revision. It seems to us fundamental to provide designers with a platform in which this compromise is easy to define and make evolve. We do not know other systems exhibiting this kind of functionality.

Currently some systems exhibit paradigms combinations. For example COM/DCOM propose a paradigm where a Com document (the common state) contains artifacts managed by different applications (Word and Excel for example). Changing an element of the common state calls the relevant application only (control based paradigm). Java Beans adds collaboration to the COM strategy. When a component is changed (on the screen) not only the component (bean) respon-

sible of the element is called, but also the other components (beans) linked to that element (by event). Java Beans features common state (screen), strict control (owner of a bean) and implicit and asynchronous reactions (event based).

A platform like APEL is policy free: it does not emphasize any paradigm but allows “all” paradigms to be implemented. On top of such a platform, a variety of specialized architectures based on various component nature and interoperability strategies can be defined and supported.

Software systems also are emphasizing component based architectures. As exemplified by Corba, COM and Beans, the platform we propose fits also pretty well software architecture needs. The question is to know to which extent the framework we propose fits any component based architecture in any application domain, not only the process support domain.

The idea of separating the common process from the control have been implicitly found by others. In Oz [12], treaties is an interoperability process, based on which a piece of common model (the treaty) is transferred to PSSs. In LEU, the binding and cooperation between models is explicitly described: it is an interoperability model. In [2], LCP is only the interoperability language. However, in these systems, there is no clear separation of concern.

It is interesting to note that the clear separation between juridic (interoperability) and operational (common) powers has been adopted for long by liberal societies, but is seldom found in computerized interoperable systems. We think that the amount of control information is getting bigger and bigger with the distribution, autonomy and heterogeneity of PSSs, and that it is important to make now this distinction too.

References

- [1] M. Amieur and J. Estublier. “A Support for Communication in Software Processes”. Submitted to SEKE’98.
- [2] J.M. Andreoli, J.L. Meunier, D. Pagani. Process Enactment and coordination. *EWSPT’96, European Workshop on Software Process Technology*, Nancy, France. October 1996. LNCS 1149, P195-216.
- [3] D. Avriilionis, N. Belkhatir, P.Y. Cunin. A Unified Framework for Software Process Enactment and improvement. 4th International Conference on Software Process ICSP4. 2-6 december 1996 Brighton. UK.
- [4] S. Bandinelli, M. Barga, A. Fuggetta, C. Ghezzi, and L. Lavazza. “SPADE An Environment for Software Process Analysis, Design and Enactment”. In [20] , 1994.
- [5] S. Bandinelli, M. Barga, A. Fuggetta, C. Ghezzi, and L. Lavazza. “SPADE An Environment for Software Process Analysis, Design and Enactment”. *Software Process Modeling and Technology (eds. A. Finkelstein, J. Krammer, B. Nuseibeh)*, Research Studies Press, Taunton, 1994.
- [6] N. S. Barghouti and B. Krishnamurthy. “ An Open Environment for Process Modeling and Enactment”. Proc. of the 8th. Int. Software Process Workshop. Wadern, Germany, March, 1993.
- [7] N.S. Barghouti and B. Krishnamurthy. Using Event Contexts and Matching Constraints to Monitor Software Processes. In *Proc. ICSE17, 17th International Conference on Software Engineering*, Seattle WA, USA, 1995. IEEE Computer Society Press. Pages 83-94.
- [8] N. Belkhatir. An Object Oriented Framework for Distributed, Interoperable Process Engineering Environments. IEEE Tools’23 conference. Technology of Object oriented Languages and systems. 20th Int. Conf. July 28- August 1, 1997 SantaBarbara, CA. , USA.
- [9] N. Belkhatir, J. Estublier, and W. L. Melo. “Adele 2: a support to large software development process.” In First Int’l Conference on the Software Process. Los Angeles. pages 159–170.
- [10] I. Z. Ben-Shaul and G. E. Kaiser. “Federating Process-Centered Environments: the Oz Experience”. ASE journal (Automated Software Engineering), Vol. 5, Issue 1, January 1998, Kluwer Academic Publishers.

-
- [11] G. A. Bolcer and R. N. Taylor. "Endeavors: A Process System Integration Infrastructure". 4th. Int'l Conference on Software Process ICSP4, December 2-6, 1996.
 - [12] A. Bouguettaya, B. Benatallah and A. Elmagarmid. "Introduction to Multidatabase Systems". Technical report.
 - [13] C. Collet and T. Copaye and T. Stevens. NAOS Efficient and modular reactive capabilities in an Object Oriented Database system. 20th VLDB. Santiago, Chile May 1994.
 - [14] R. Conradi, C. Fernstrom, A. Fuggetta, and B. Snowdown. Towards a Reference Framework for Process Concepts. In *Proc. EWSPT'92, 2nd European Workshop on Software Process Technology*, Trondheim, Norway, Sept. 1992
 - [15] S. Dami, J. Estublier and M. Amiour. "APEL: a Graphical Yet Executable Formalism for Process Modeling", Kluwer Academic Publishers, pages 60 - 96. Boston, January 1998.
 - [16] Jacky Estublier and Naser S. Barghouti . *Interoperability and Distribution of Process-Sensitive Systems*. Software Engineering for Parallel and Distributed Systems (PDSE'98). Kyoto April 19-25, 1998.
 - [17] J. Estublier, S. Dami and M. Amiour. "High level process modelling for SCM". 7th. *International Workshop on Software Configuration Management (SCM7)*, Boston, 19-20 May 1997.
 - [18] J. Estublier and R. Casallas. *The Adele Software Configuration Manager*, chapter 4, pages 99–139. Trends in Software. J. Wiley and Sons, Baffins Lane, Chichester West Sussex, PO19 1UD, England, 1994.
 - [19] J. Estublier and N. Belkhatir. "A generalized multi view approach." In W. Shaeffer, editor, *European Workshop on Software Process Technology (EWSPT4)*, LNCS 913, pages 179, 185, Leiden, The Netherlands, April 1995. Springer Verlag.
 - [20] B. N. A. Finkelstein, J. Kramer, editor. *Software Process Modeling and Technology*, volume 1. John Willey and Son Inc. Research Study Press, Tauton Somerset, England, 1994.
 - [21] G. Graw, V. Gruhn, "Process Management in-the-Many". 4th. European Workshop, EWSPT'95, April 1995.
 - [22] D. Hollingsworth. Workflow Management Coalition. The Workflow Reference Model. 1994.
 - [23] S. Heiler. Semantic Interoperability. *ACM Computing Surveys*, 27(2):271-273, June, 1995.
 - [24] D. Heimbigner. "The ProcessWall: a Process State Server Approach to Process Programming". ACM-SDE, December 1992.
 - [25] V. Kasshyap, A. Shet. Semantic and Schematic Similarities between Database Objects: A context Based approach. VLDB Journal. The International Journal on Very Large Data Bases. 1996.
 - [26] W. Kim, J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *Computer Journal*, 1991, P 12-18.
 - [27] J. Lee, G. Yost, and the PIF WorkGroup. The PIF Process Interchange Format and Framework. Technical report, PIF WorkGroup, 1996. <http://ccs.mit.edu/Pifmain.html>.
 - [28] J.A. Miller, A. Sheth, K.J. Kochut and X. Wang. CORBA-Based Run-Time Architectures for Workflow Management Systems. *Journal of Database Management, Special Issue on Multidatabases*, 7(1):16-27.
 - [29] B. Nuseibeh and A. Finkelstein. "Viewpoints: A vehicle for method and tool integration." In *Proc. of 5th Int'l Workshop on CASE; IEEE CS Press*, pages 50–60, Montreal, Canada, July 1992
 - [30] The Object Management Group. The Common Object Request Broker: Architecture and Specification. Revision 2.0, July 1995.
 - [31] A. Sheth, K. Kochut, J. Miller et al. Supporting State-wide Immunization Tracking Using MultiParadigm Workflow Tech. Proc. 22nd VLDB Conf., Bombay, India, 9/1996.
 - [32] I. Sommerville, G. Kotonya, S. Viller, P. Sawyer, "Process Viewpoints", 4th. European Workshop on Process Technology (EWST'95), Noordwijkerhout (NL), 1995, PP. 2-8.
 - [33] D. Tombros, A. Geppert and K. R. Dittrich. "The Broker/Services Model for the Design of Cooperative Process-Oriented Environments". Technical Report 97.06, Department of Computer Science, University of Zurich, Switzerland, 1997.

-
- [34] M. Verlage. "Multi-view modeling of software processes." In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [35] Workflow Management Coalition. Interface 1: Process Definition Interface. WfMC TC-1016. August 1996 WfMC@eyam.be
- [36] Workflow Management Coalition, Coalition Overview, Sept. 1995, <http://www.aiai.ed.ac.uk/WfMC/overview.html>.
- [37] WFMC. "WPD". Document WFMC TC-0020 , Workflow Management Coalition.