Federating Process-Centered Environments: the Oz Experience

Israel Z. Ben-Shaul
Technion-Israel Institute of Technology
Department of Electrical Engineering
Technion City, Haifa 32000
ISRAEL
issy@ee.technion.ac.il
+972-4-8294680
fax +972-4-8323041

Gail E. Kaiser
Columbia University
Department of Computer Science
New York, NY 10027
UNITED STATES
kaiser@cs.columbia.edu
212-939-7081
fax 212-939-7084

CUCS-006-97
March 10, 1997

Abstract

We describe two models for federating process-centered environments (PCEs): homogeneous *federation among distinct instances of the same environment framework enacting the same or different process models, and* heterogeneous *federation among diverse process enactment systems. We identify the requirements and consider possible architectures for each model, although we concentrate primarily on the homogeneous case. The bulk of the paper presents our choice of architecture, and corresponding infrastructure, for homogeneous federation among* MARVEL *environment instances as realized in the* OZ *system. We briefly consider how a single* MARVEL *environment, or an* OZ *federation of* MARVEL *environments, might be integrated into a heterogeneous federation based on ProcessWall's facilities for interoperating PCEs.*

Keywords: Collaborative work, Distributed system, Enterprise-wide environment, Geographical distribution, Internet, Process interoperability, Software process, Workflow management

1 Introduction

Large-scale software engineering projects are not always confined to a single organization (e.g., group, department or lab), or even to a single institution (e.g., in a subcontracting or consortium relationship). A project may span *multiple* teams located at geographically dispersed sites connected by a wide area network (WAN) such as an organizational intranet or the Internet. Distinct teams may each have their own software development practices, favored tools, use different programming languages, etc. Yet the teams may still need to collaborate frequently in real-time, i.e., operate concurrently rather than sequentially, share part or all of their code and document base, perform tasks on behalf of each other and/or jointly, and so on.

Note we generally use the term "site" to mean an administratively cohesive domain, in which most (but not necessarily all) machines share a single network file system name space, e.g., cs.columbia.edu, as opposed to either a single host such as westend.psl.cs.columbia.edu, a lab subnet within an administrative domain such as psl.cs.columbia.edu, or a campus backbone such as columbia.edu. However, as we shall see, we sometimes use the term "site" in an alternative sense where a single local area network (LAN) or even a single machine may be home to multiple sites — when multiple teams happen to do their work on that same LAN or machine, respectively. That is, a site is whereever a team does its work.

Consider, for example, several teams each responsible for a separate set of "features", all intended to be included in an upcoming Microsoft product release [17]. Imagine some of these teams have been subcontracted from various independent software houses located outside Microsoft's main development lab, perhaps even outside the United States. Although Microsoft documents recommend vendor processes, it seems unlikely that these teams would follow identical software engineering practices, use exactly the same tools, etc. They may not be willing to publicize (even among themselves) their proprietary software development "trade secrets".

There are various approaches to software development environment (SDE) support for multi-site projects. For the purposes of this paper, we organize these approaches along two orthogonal axes: tightness of coupling and degree of heterogeneity. At one end of the coupling spectrum, each team chooses its own SDE (which may happen to be copies of the same environment in some of the sites) and there may be more or less concern with whether the different team's SDEs are compatible with each other.

A little further along the coupling spectrum, the teams may choose the same (homogeneous) SDE, to minimize data conversion and supply a common vocabulary, or they may use heterogeneous SDEs but agree on a shared data interchange format. In either of these cases, sharing and collaboration between teams is done *outside* the environment — unless some special "glue" is added on top to bind them together into a federation (i.e., a common data format alone is not sufficient for them to work together at run-time), as explained below.

Another important intermediate range is covered when the teams share the same instance of what we call a *multi-site* SDE, which distinguishes among teams (who may reside at the same or different sites) in some way, but provides facilities for sharing and collaboration between teams *inside* the environment. That is, the glue (or perhaps "cement" in this case) is part of the environment framework itself. The degree of independence afforded each team determines the point within the subrange. The heterogeneous version of this intermediate range consists of *multi-SDEs*, that is, interacting but distinct SDEs with the glue consisting of a shared standard event notification scheme [5] or other control facilities in addition to a common data interchange format.

Finally, the far extreme is a geographically distributed SDE that does not distinguish among teams — all the users are treated as members of one very large team sharing everything. We choose the terms "multi-site" and "geographically distributed" here because many SDEs are said to be "distributed", meaning they have multiple internal components that may execute on different hosts on a LAN or WAN.[1]

The geographically distributed SDE end of the spectrum is analogous to distributed database systems, where there is transparent access to distributed data, while the independent choice of SDE end is comparable to a collection of independent databases. The database community has also delineated an intermediate range, often termed "federated databases" [64, 58]. Federated databases generally permit a high degree of autonomy with respect to one or both of two criteria, schema and system: local components of a single database system with intrinsic federation glue may devise and administer their own schema independently (known as a *homogeneous federation*), and/or the local components may correspond to different database systems from among those supported by extrinsic federation glue (*heterogeneous federation*) — in which case even conceptually equivalent schemas may appear in different forms due to system-specific data definition languages.

We are concerned in this paper with the subclass of SDEs known as *process-centered environments* (PCEs) [60, 13]. In general, a PCE is a generic environment framework, or kernel, intended to be parameterized by a *process model* that defines the software development process for a specific instance of the environment. The PCE's *process engine* interprets, executes or "enacts" the defined process, to assist the users in carrying out the process by guiding them from one step to another, enforcing the constraints and implications of process steps as well as any sequencing or synchronization requirements, and/or automating portions of the process. A federated PCE might coordinate users from multiple teams working on collaborative tasks, inform one team when it should perform some task on behalf of another, notify one team on completion of some task it has been waiting for another to perform, and transfer process state and product artifacts (design documents, source code, executables, test cases, etc.) among local components of the federation as needed for this work.

It is important to note that in both multi-site PCEs and multi-PCEs, we treat process as the *integrating principle* of federation. That is, the federation is intended to fulfill the semantics expressed explicitly in the (global) process, and this has a crucial impact on the design of the federated architecture. We do not address non-process-centered SDE federations further in this paper.

A multi-site PCE is analogous to a homogeneous database federation. In particular, the PCE process model fills the role of the database schema with respect to homogeneous federation: the local components of the multi-site PCE are identical, except that they are tailored by and thus enact different process models (or possibly reflect different instantiations of the same process model). A multi-PCE is analogous to a heterogeneous database federation, and similarly requires that each separate PCE is independently (from the others) capable of interfacing to federation glue that makes it possible for them to work together. Generally the process modeling formalism as well as the process model are different at each site, although the process model could be conceptually the same while expressed differently. In either case, again the process model fills the role of the database schema, although we note that generally each PCE also supports some schema for a data repository containing its software development artifacts and process state.

Figure 1 illustrates the space of approaches, highlighting the two "federated" grey areas, which

[1] Some authors have used the terms "multi-site" and "geographically distributed" interchangeably, but here they refer to different concepts.

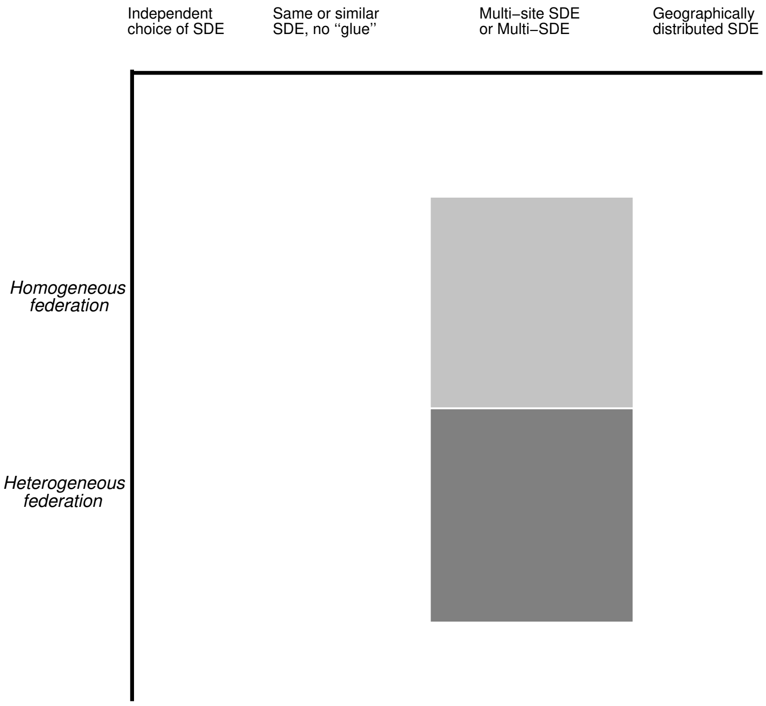

Figure 1: Multiple Team SDE Spectrum

serve as the context of this paper. That is, we are concerned with federated PCEs that exhibit at least some degree of coupling but also at least some degree of independence, i.e., not transparent distribution; and we do not consider completely homogeneous approaches, where even the processes must be identical, or completely heterogeneous approaches, where it is impossible to introduce any sort of run-time integration, and the only possible integration is at definition-time, through a process definition exchange format, e.g., as promoted by WfMC [47].

A federated PCE for cross-organization projects should permit each team to specify its own local process model, along with the desired collaboration with other teams through shared subprocesses, tool sets, data subschemas, data instances, etc. Thus, a noticeable difference from database federation is that the focus here is on interoperability among heterogeneous processes, i.e., the application semantics, as opposed to (only) heterogeneous data schemas, i.e., the data on which applications operate.

One approach to homogeneous PCE federation, where every team runs a component of the same multi-site PCE but enacts a different process, is taken by our Oz PCE [9]. Oz was devised to scale up our earlier Marvel PCE [36] to multi-process, multi-team, geographically dispersed software engineering projects. Oz introduces an *International Alliance* metaphor whereby each team autonomously devises its own local process (supported by a local Oz component that is essentially an extended instance of Marvel), analogous to how each country has its own local customs and laws. A team may agree to extend its process to a small degree (and thus temporarily lose some autonomy) in order to participate in a *Treaty* with one or more other teams. The enactment of a *multi-site task*, defined as any task that involves interaction among the several sites of a multi-site PCE, is called a *Summit*. Oz extends Marvel with Treaties, Summits, and an underlying inter-site communication and configuration infrastructure where each site corresponds roughly to an instantiated Marvel environment.

Only tasks specified in a Treaty may access data from other sites, and even then only in accordance with the privileges granted by the Treaty. For example, a site may agree to perform certain tasks requested by another site on its own local data; or a site may agree to allow another site to perform certain tasks on its local data; or a site may agree to perform certain tasks on data from several sites. However, each site (or team) is responsible for any prerequisites or consequences of such tasks with respect to its own data, following its own process, just as in preparations for and follow-ups of meetings among country leaders (the basis for our metaphor). Treaties may be dynamically defined while the process unfolds, i.e., while computation is in progress, permitting a degree of flexibility not found in most distributed systems.

One approach to heterogeneous PCE federation, where two or more distinct process systems are bound together into a multi-PCE, is taken by Heimbigner's ProcessWall [24]. Note that ProcessWall is the external glue supporting such binding, not itself a PCE. ProcessWall could of course be used to integrate multiple instances of the same PCE, say MARVEL, with different process models as in OZ, but in this paper we address only the more challenging case of using it to federate multiple distinct PCE systems.

Heimbigner refers to ProcessWall as a *process state server* because it enables interaction between the PCEs through a centralized representation of global process state that the teams agree to share. However, we believe it is more useful to treat the mechanism Heimbigner describes as a *process task server*: it may maintain the history of tasks that have already been completed, in aggregate representing the current process state, but more significantly from the viewpoint of federation the server posts those tasks that have been instantiated but not yet scheduled for enactment by one of the participating PCEs.

In particular, each participating PCE manages, schedules and enacts its own task descriptions, usually forwarding each description to ProcessWall only after that task has been completed, e.g., to allow users to exploit ProcessWall's process state inspection facilities (part of the glue). Thus the process remains primarily decentralized, since the actual process operation is performed by the separate PCEs without any interactions between them or with ProcessWall while the work is in progress. However, in some cases a PCE may send an instantiated (e.g., with data parameters) but unenacted task to ProcessWall intending it to be executed by some other PCE in the federation, because the sending PCE does not have the data, tool(s) or user(s) appropriate to conduct the work.

An intelligent scheduler might then be attached to ProcessWall to direct such posted tasks to particular sites, as described in [52], or alternatively ProcessWall might be treated as a "blackboard" (using artificial intelligence terminology [22]) from which the schedulers of the individual PCEs participating in the federation select those tasks they are suited to perform. Any sharing of software product artifacts, as opposed to process state, is implicit in the data information included with posted tasks. As in OZ, each site might autonomously devise its own process model.

Mentor [72] is similar to ProcessWall but divides the process state/task server into two components: a *worklist manager* acting as a pure task server and a *history manager* corresponding to a pure state server; data sharing is factored out as in ProcessWall. Note Mentor is a workflow management system intended for business applications, not a PCE oriented towards software engineering; whether there is any fundamental difference between workflow and process is a matter of some debate [63], but we blur the distinction in this paper. In any case, heterogeneous federation based on Mentor would probably be quite similar to the ProcessWall model.

Process interchange formats [47, 45] support translation of a logically single process model into the different representations of distinct process systems, but do not provide any means for collaboration and interoperability during the process enactment by those systems. Thus there is no true federation in the sense addressed by this paper. However, some kind of translation facilities are needed as part of any heterogeneous federation: Mentor transforms the heterogeneous process modeling formalisms into StateMate charts [31], but in the case of ProcessWall only process state is translated (or the participating PCEs might be implemented to use a common task format).

We mentioned above that process enactment by a federated PCE might involve movement of product artifacts among teams that could potentially be distributed across a WAN. Alternatively, all the sites might share a common centralized data repository, presumably located at one of the sites, or even a transparently distributed data repository. Globally shared data seems most appropriate for projects organized far in advance and involving only a single institution, perhaps with multiple campuses. In contrast, when different institutions work together, particularly when the federations are dynamically created and dissolved, most likely the institutions would prefer to maintain locally at least those product artifacts produced by their local process.

This paper discusses the architectural aspects of PCE federation and associated infrastructures, and then justifies our architectural choices for the fully implemented (and used in our day-to-day work since April 1995) Oz system in detail. We also explore a hypothetical Oz/ProcessWall interface. Our investigation of architecture is strongly influenced by the fact that the main purpose of PCE federation is to enact multi-site and global processes. For example, global processes devised using a top-down methodology, say intended for multiple campuses of a single institution, may require somewhat different architectural support than global processes constructed in a bottom-up manner, e.g., for temporary multi-institution collaborations. However, methodologies for developing global processes are outside the scope of this paper.

First we present architectural requirements and the alternative architectural models we considered for both homogeneous and heterogeneous PCE federations, the latter in contrast to the former (i.e., many of the requirements are shared and the architectures are analogous). We then elaborate the specific design decisions and tradeoffs that were made in developing the Oz architecture and infrastructure that extended our earlier Marvel PCE to a homogeneous PCE federation. We do not go into detail regarding ProcessWall, Mentor, or any other such heterogeneous federation glue, since that is properly left to their developers. Instead we briefly discuss how Marvel, or Oz, might be integrated into a heterogeneous federation based on ProcessWall's process state/task server model, to some extent synergizing the two federation mechanisms, i.e., allowing Oz to operate as a multi-site PCEs within a multi-PCE. In both sections, the range of architectures is explored specifically in the context of our choices for Oz. We conclude with the contributions of this work and outline some directions for future research.

2 Requirements and Alternative Architectures

In both the homogeneous and heterogeneous federated models, each local site runs a component of a multi-site PCE or multi-PCE. We refer to such a site component as a *sub-environment*, or just SubEnv, even though it may operate in stand-alone fashion as a full PCE. We refer to the "glue" that holds the SubEnvs together as the federation's *foundation*, or just Foundation. Database federation involves a similar foundational component or layer, e.g., to control global transactions, although many classes of distributed system do not include any foundational layer beyond a basic networking communication protocol. This section of the paper is concerned with the functionality (Section 2.2 for the homogeneous case and Section 2.5 for the heterogeneous case) and architectural design (Sections 2.3 and 2.6, respectively) of the Foundation. Recall that we are mainly concerned with multiple sites on a WAN, generally with independent administrative domains — although of course nothing prevents multiple sites from running on the same LAN, that is, a multi-site PCE or multi-PCE might operate entirely within a single organization or group and each "team" could conceivably consist of only one user (as in the Oz EmeraldCity environment we use to support our own software development [38]). We take as given the requirement that each site must be able to support an autonomously devised process model.

2.1 Local Environment Internal Architecture

Although the focus of this paper is on *federation* architecture, it is useful to begin the discussion with an overview of SubEnv *internal* architectures, since they have a substantial impact on the design of a homogeneous federation; internal architecture is less germane in the case of heterogeneous federation since, in general, each participating SubEnv may employ a different internal architecture. As we focus in this paper on process-centered SDEs and the impact of process on architecture, we characterize local PCE architectures based on the degree of centralization in process enactment, comprised of two aspects: process control and tool execution. The former refers to the function of deciding which task to enact, when, according to process constraints/context, whereas the latter corresponds to where and how the task gets executed, often but not necessarily via one or more specific tools. This separation is important in PCEs because it reflects the typical separation between the process itself and the tasks spawned by it, which may invoke external tools, take arbitrarily long to complete, involve one or more human (possible simultaneous) users, and so on.

Although our goal was to scale up our pre-existing Marvel PCE to support multiple teams each sharing a potentially different process, where the teams might be connected by either a LAN or a WAN, we identified four classes of internal PCE architecture — only one of which applies to the final Marvel version 3.1.1 we were concerned with. Note these are <u>not</u> the same classes suggested by Peuschel and Wolf [53] and we follow a different classification scheme: Peuschel and Wolf were concerned with the relationship between the process engine and the data repository, whereas we consider process control vs. tool (or task) execution.

1. Centralized process control and centralized tool execution. This is the simplest case, where both control and execution are carried out by the same component. An all-in-one single-user PCE such as Marvel 2.x [33] and some compiled process programs, e.g., written in APPL/A [68], would fit into this class. Even a client/server system might fall into this category if the client supported only the user interface and all process enactment was performed in the server. Given the multi-user multi-task nature of practical software engineering processes, this architecture is inherently unscalable, even for a single team.

2. Centralized process control and decentralized tool execution: A process server maintains the state of the process, controls its enactment, and synchronizes access to shared resources, but the tools themselves execute at process clients. Marvel 3.x [12], ProcessWEAVER [19], and Mentor fit this mold, albeit in different ways. Marvel 3.x relies on fixed user clients to fork tools, whereas ProcessWEAVER spawns user "work contexts" as needed by the process. Oz local sites are somewhere in between, with one server per site (i.e., per team), generally employing user clients as in Marvel 3.x but also supporting "proxy clients" that run tools

on behalf of one or more users under various circumstances, as explained in [70, 66]. Mentor is similar to Oz in that user clients can connect to multiple process servers in the federation.

3. Decentralized process control and centralized tool execution. Control is distributed among multiple process servers, where the tool execution function is supported by a single component. This model supports separate process engines for each user — as in Endeavors [14] or Merlin [61] — while sharing special computational or database resources used in tool invocation. One can easily imagine multiple workflows accessing the same tool management resource, particularly if only the tool broker is centralized, directing actual tool invocation to distributed hosts as in WebMake [48].

4. Decentralized process control and decentralized tool execution. Here the process itself is distributed across multiple nodes, where each node is responsible for the execution of its subprocess as well as corresponding tasks. Control flow and synchronization between the process segments is specified locally inside the nodes. Several transactional workflow systems, such as Exotica [1] and Meteor [32], operate in a fully distributed manner — by expressing the workflow implicitly in a network of task managers (which invoke the actual tools) that interact only with their predecessors and successors in the workflow routing.

Several issues influence the choice of single-site PCE architecture. A major factor is the level of data integration employed by the PCE for product artifacts. PCEs with extensive data integration facilities (e.g., SPADE [3], EPOS [16]) might choose a centralized control architecture to minimize communication between the data and process managers when disseminating tasks — unless the data management is itself distributed, and/or the data itself is physically distributed, in which case a distributed control architecture may be employed. PCEs with no data integration facilities might be fully distributed in an easier manner. Note, however, that full distribution of process enactment is not incompatible with sharing a centralized data repository; see [53].

Another characteristic that impacts the choice of local PCE architecture is whether the process modeling paradigm employed by the PCE is reactive or proactive (termed proscription vs. prescription by Heimbigner in [23]). Reactive enactment may be realized better in a centralized-control architecture, as requests for enactment are directed to a single server that dispatches the service to a client (perhaps the requester itself), whereas proactive enactment may be distributed by assigning a priori each task to a component, with the ordering and execution constraints inside each component — or implicit in their interconnection topology.

2.2 Requirements for Homogeneous Federation

We have identified the following additional requirements for the homogeneous model:

- The most fundamental functional requirement for multi-site process enactment is that the Foundation include an infrastructure whereby the SubEnvs communicate with each other regarding multi-site tasks. This might be constructed directly on top of TCP/IP sockets, or employ some higher level mechanism such as RPC or CORBA [51]. In any case, we are mostly concerned with the PCE-cognizant interconnectivity layer, i.e., the Foundation, not the underlying mechanism.

- On top of the basic interconnectivity support, the Foundation must supply means for local processes to interoperate, i.e., to model and enact tasks that in some way span multiple processes/sites and contribute to the global process. In particular, the Foundation must have facilities to (re)negotiate and (re)define (possibly dynamically) the specifics of process-interoperability for the relevant processes, e.g., via Oz-like Treaties.

Although a degenerate global process may involve only primitive operations (e.g., copy a data item), we in general assume some notation to define multi-site tasks whose enactment is controlled to some degree by the Foundation. In other words, we assume that multi-site tasks are themselves modeled in either top-down or bottom-up fashion as parts of a global process, with conceptually its own state and purpose. However, multi-site process modeling and enactment is the subject of another paper [10]; here we are concerned with structure and organization of components, i.e., architecture, that supports multi-site processes.

- As far as purely local work is concerned, i.e., work involving the local process operating only on local data, a SubEnv should operate autonomously and independently, and provide the same capabilities as would a single-site PCE. It should not in any way rely on communication with other SubEnvs, or with the Foundation, in order to perform its standard functions with regards to defining and executing the local process. The underlying assumption is that most of the work done by a site is local to that site, and therefore the multi-site PCE should still be optimized towards local work.

- A related issue is that the SubEnv should minimize the dependencies on uninvolved SubEnvs when executing part of a multi-site task. These two requirements are somewhat similar to control and execution autonomy, respectively, in multi-database transaction management [41]. The local site autonomy prized in the Oz approach to bottom-up process modeling has also been argued as necessary for top-down modeling: "A participant on a lower level [of the hierarchy] does not want his/her management to know how a task is performed" [62]. Thus we rationalize site autonomy as a critical requirement.

- The SubEnvs must somehow be aware a priori (statically), or become aware during the course of process enactment (dynamically), of each other's existence, i.e., the other members of the currently configured federation, if they are intended to directly communicate and, possibly, collaborate. In the case where a Foundation intermediary is the conduit for all communication and interactions among SubEnvs, the SubEnvs must at least be aware of that intermediary and vice versa.

- Since the lifetime of enacted processes is often long, months to years, the Foundation must allow for SubEnvs to join or leave a federation while a process is in-progress, that is, support configuration and reconfiguration of participants in the global process. It is of course also necessary for SubEnvs to determine or negotiate what services each can expect from other (perhaps anonymous) SubEnvs in terms of process control, tool execution, and data and other resources, and how to coordinate exploitation of those services, but again that is the subject of another paper [10].

- Since processes in general, and federated processes in particular, are enacted for long durations, they require facilities for persistent process state. In cases where each local PCE manages its own product-data repository, the Foundation must also provide mechanisms for transferring product artifacts, in addition to process state, among sites. This may involve the same or different inter-PCE communication channels for product vs. process data, but the two cases have to be handled separately because products typically involve significantly larger volumes of data. For example, in a multi-site `build` task one site may collect code

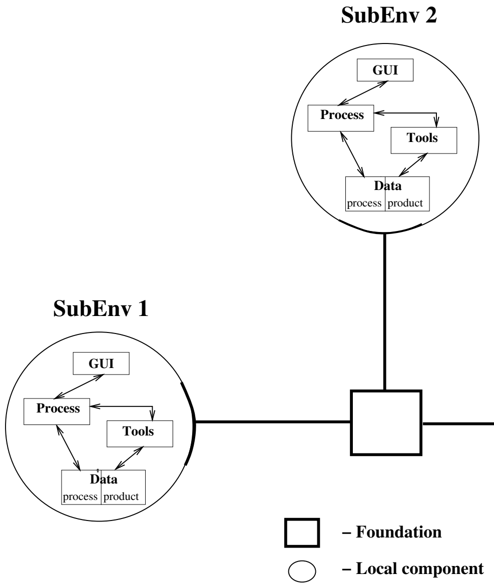

Figure 2: Centralized Architecture

modules from the other relevant sites and return to them copies of the resulting executables and/or libraries. Another example is a distributed groupware task such as multi-user editing, in which source code and/or documentation files stored at one site may need to be (simultaneously) transferred to several other sites. In general, bulk data may be temporarily cached, permanently copied, or migrated between sites.

- Another data-related requirement involves support for sophisticated and flexible concurrency control and failure recovery mechanisms due to the long duration of tasks and task segments, interactive control by users, and human-oriented collaboration among tasks and task segments while they are in progress [42]. The explicitness of the process in PCEs makes it possible to employ semantics-based transaction management [4, 29]. Multi-site tasks may modify data from multiple sites, and thus require some kind of global transactional support, such as two-phase commit that interfaces with local transaction managers. Investigation of this topic is beyond the scope of this paper, see [7, 26].

2.3 Homogeneous Federation Architectures

We identify five categories of architectures within the homogeneous (light grey) area of the "tightness of coupling" spectrum of Figure 1. Note relatively minimal (or no) translation services are needed in any of these categories: all the SubEnvs speak the same languages (including data formats, process modeling notation, and tool wrapper scripts). The Foundation may perform name mappings, since a common ontology is not assumed, but this is not its major function.

For the sake of the figures depicting the multi-site PCE architectures below we show one plausible set of components that may comprise a local SubEnv (process, data, tool, and user interface components), but we do not intend in any way to specify or constrain a SubEnv to follow the given structure, and any of the internal architectures discussed in Section 2.1 might be employed. Further, we do not specify any particular internal component for interfacing to the Foundation — interpret

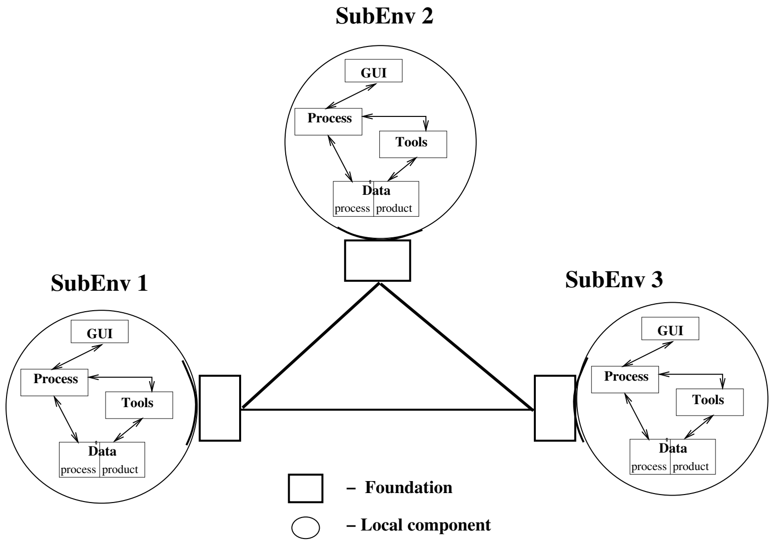

Figure 3: Decentralized Glue Architecture

the figures as if they *all* (potentially) do, to achieve federation of their various functionalities.

1. **Ad hoc:** Two or more instances of the single-site PCE are hardwired together in some ad hoc fashion for a particular purpose. There is generally no Foundation, per se. This model obviously does not scale, so is not addressed further.

2. **Centralized glue:** The SubEnvs communicate and interact through a single centralized glue component that constitutes the Foundation. As mentioned earlier, the Foundation, i.e., the federation glue, is intrinsically part of the multi-site PCE rather than imposed externally. However, each SubEnv necessarily includes code to interface to the Foundation, perhaps through RPC or TCP/IP socket calls originally part of native SubEnv if the PCE was designed as a multi-site PCE, or inserted later if not. The Foundation may perform brokerage or routing among SubEnvs, and maintain the state of multi-site process segments.

   Figure 2 illustrates this architecture. Centralized distributed systems do not scale beyond a certain level, since the centralized component becomes a performance bottleneck and single point of failure (i.e., if this one component fails multi-site tasks become impossible). The interface aspect of the centralized glue could be expanded in several different ways with respect to the SubEnvs, analogous to the intermediary, moderated and direct decentralized cases below. We do not discuss these options, since the variations between the cases are overwhelmed by Foundation centralization — although as the interfaces get "larger" the central component tends to get "smaller", as functionality is shifted, effectively achieving a hybrid between centralized and decentralized approaches.

3. **Decentralized glue:** The SubEnvs communicate and interact through *intermediaries*, with one intermediary attached to each SubEnv. These intermediaries collectively constitute the Foundation glue, and there is <u>no</u> centralized component. A Foundation intermediary may or may not be realized as a separate operating system process from its local SubEnv. If separate, it would usually reside "close" to the local SubEnv, e.g., on the same LAN, but not necessarily on the same host.

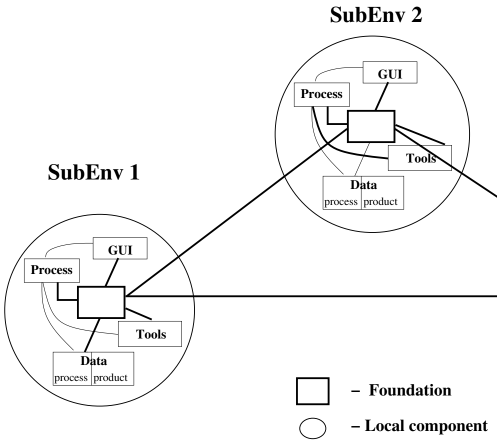

Figure 4: Moderated-peer-to-peer Architecture

However, this case is distinguished from the peer-to-peer cases below in that the intermediary has no special knowledge of the PCE's process-oriented functions and no access to its process model, nor any special knowledge of its tool execution facilities, data repository, user interface, etc. To the degree that these internal functionalities (whether or not distinguished as components) interoperate within the federation, and thus interact with the Foundation infrastructure, they must interface to the intermediary. The intermediaries are tightly coupled with each other, e.g., maintaining long-term connections which permit them to share the Foundation's global process state and work closely together to realize the Foundation's functionality (e.g., a distributed name service), but are loosely coupled with respect to their SubEnvs. See Figure 3. From a process perspective, the interaction between the global process and the local processes is quite limited because the Foundation has no access to the internals of the local processes.

Note a geographically distributed realization of the decentralized glue architecture is plausible — with the intermediaries acting as gateways to remote SubEnvs on a WAN; the two peer-to-peer architectures below also easily admit a geographically distributed implementation. Although a centralized architecture might also be geographically dispersed, this seems less likely from an administrative point of view — except possibly within a organizational intranet where the same organization owns and controls all the relevant sites including the machine hosting the central component.

4. **Moderated peer-to-peer:** The SubEnvs again communicate/interact through intermediaries, which we call *moderators* here, with one moderator attached to each SubEnv. These moderators collectively constitute the Foundation, and again there is <u>no</u> centralized component. Again there is no implication intended regarding physical realization, the moderator may or may not be realized as a separate operating system process from the rest of its local SubEnv. If separate, again it would necessarily reside "close" to the local SubEnv, most likely on the same host.

   Unlike the decentralized glue case, here each moderator is tightly coupled with its local SubEnv and has intimate knowledge of that SubEnv's process-oriented expectations regarding services from other SubEnvs. Similarly, the moderator is cognizant of the local process model and state, tool execution, data repository, user interface, etc., if relevant to federation. Again, to the degree that these internal functionalities (whether or not distinguished as components)

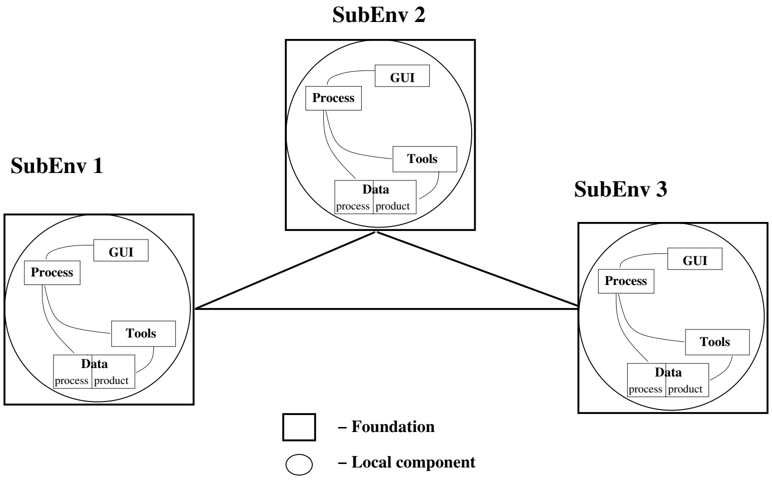

Figure 5: Direct Peer-to-peer Architecture

interoperate within the federation, and thus rely on the Foundation infrastructure, they must interface to the moderator. In contrast, the moderator is loosely coupled with respect to its peer moderators, e.g., making only short-term stateless connections. See Figure 4.

This approach again seems obviously more likely to scale than a centralized architecture, but more-or-less equivalent with respect to scaling as the decentralized glue case. However, in this case the architecture cannot assume any shared capabilities (e.g., name services) provided by the Foundation. In other words, it is a "shared nothing" architecture as far as the Foundation is concerned. (Note this does not preclude sharing among internal components of the SubEnv.) On the other hand, the interaction between the global process and the local processes is richer, because the Foundation has direct access to local processes.

That is, the primary distinction between the decentralized glue case and the moderated peer-to-peer case is that in the former the local Foundation components have no knowledge of the local processes and manage a multi-site process imposed on the local SubEnvs divorced from their local processes, whereas in the latter the local Foundation components have intimate knowledge of the local processes, but without any shared global process state or common control. This reflects the tradeoff between stronger coupling within the Foundation and weaker coupling between the local component of the Foundation and its local SubEnv, in the decentralized glue case, vs. weaker coupling within the Foundation and stronger coupling between the local Foundation and its SubEnv, in the moderated peer-to-peer case.

5. **Direct peer-to-peer:** The SubEnvs communicate/interact with each other directly, and the Foundation cannot easily be distinguished from the rest of the multi-site PCE. That is, the local component of the Foundation is *built into* one or more of the SubEnv's internal components, most likely the process engine; there is no specific component introduced *solely* to represent the Foundation infrastructure. See Figure 5.

   While this approach probably offers improved performance over the others described above, it is more challenging to realize for pre-existing single-site PCEs because it generally involves significant modification throughout the PCE code as opposed to "adding on" interfaces to a new component. Thus scaling is restricted for software engineering rather than distributed computing reasons.

2.4 Choice of Homogeneous Architecture

The choice of federation architecture for homogeneous multi-site PCEs depends largely on two concerns:

1. The paradigm chosen for modeling and enacting federated processes.
2. The style, design and implementation of the local SubEnv framework.

Regarding multi-site or global processes, we distinguish between two major paradigms, top-down and bottom-up, although of course hybrids are possible. *Top-down* refers to a process broken down through multiple levels of granularity each corresponding to subsequently smaller organization units, as in the enterprise-level to campus-level to department-level to group-level of the Corporation metaphor [65]; this is analogous to a global transaction in federated databases [57]. *Bottom-up* refers to interoperability among possibly pre-existing local processes, as in Oz's International Alliance metaphor, without a global overseer (unfortunately, the kind of distributed computing scenario where the Byzantine Generals problem arises — although consideration of fault tolerance in the face of malicious behavior is outside the scope of this work). We do not consider here which of the two paradigms is more appropriate for various applications (see [10] for such a discussion), but rather which *architecture* best supports each of the paradigms — particularly the bottom-up paradigm, since one of our major goals was to link pre-existing single-site MARVEL processes.

In order to support top-down global processes, the federation must support maintenance of global process state. This suggests a glue architecture, particularly centralized but also decentralized, where the Foundation manages the state. In contrast, bottom-up federation can naturally be realized on top of a peer-to-peer architectural style, again in one of two possible ways, namely the moderated or direct peer-to-peer architectures. In other words, we make a primary distinction between top-down vs. bottom-up process interoperability, and a secondary distinction between the architectural realization of each style. In general, bottom-up interoperability is more scalable than top-down, as in any other distributed system, but introduces process-related problems in our context such as lack of explicitness of the global process.

The association of top-down processes with glue and of bottom-up processes with peer-to-peer architectures is not exclusive, however. It is potentially feasible, for example, to realize a top-down process using a peer-to-peer architecture, but it is likely to be inefficient and harder to realize because of the needs to distribute the global process state among the loosely coupled intermediaries and to manage shared information over a shared-nothing architecture. It is probably easier to realize a bottom-up process using a glue architecture, provided that administrative barriers (regarding access to private process state at remote sites) can be relaxed or overridden.

Let us now consider the impact of the local SubEnv architecture on the choice of federated architecture. One factor stems from the degree of openness and extensibility of the process control and tool execution components (the data repository is also of concern, but those issues are not terribly different than in other federated database applications, so we concentrate here on PCE-specific matters). In particular, peer-to-peer architectures demand tighter integration at the process control and task/tool execution levels, between the local SubEnv and its Foundation component, than glue architectures. This is possible only if the local SubEnvs provide suitable application programming interfaces (APIs) for extending these functionalities — which would usually suffice for moderated peer-to-peer. Or, alternatively, if the SubEnv source code can be internally modified — which would by definition be required for the direct-peer-to-peer, assuming the PCE was not originally implemented as a multi-site system. (We know of none that were, e.g., multi-site Oz was realized by adapting the single-site MARVEL 3.x PCE.)

Another important factor that impacts mainly peer-to-peer architectures is the degree of centralization of the SubEnv internal architecture. SubEnvs with centralized (local) process control naturally lend themselves to a direct-peer-to-peer federation architecture, where the Foundation infrastructure is built into the local process engine — which becomes the conduit to communicate with other SubEnvs; communication via a centralized tool manager is also conceivable. Fully decentralized (local) process enactment, in contrast, seem better suited to a moderated-peer-to-peer architecture since there is no one component that stands out as the focal point. Instead, a new moderator component is attached to the SubEnv as a whole and communicates with each of the other local components as well as with its peer moderators. However, a direct-peer-to-peer architecture is not inconceivable for decentralized SubEnvs; see [71].

To summarize, the above categories represent different degrees of (de)centralization of the Foundation, ranging from a logically and physically centralized architecture, to several forms of logically and physically decentralized architectures with variations in the coupling between and within SubEnvs. Our key observation is that there is no one architectural style for federated PCEs that is inherently superior to all others. Instead, we argue that the choice of a proper architecture depends on the requirements of the system, and more specifically on the architecture of the local PCE and on the federated process paradigm. This is elaborated for the case of Oz scaling up MARVEL in Section 3.

2.5 Requirements for Heterogeneous Federation

Recall that in the heterogeneous model, each site (or team) runs a separate PCE that works together with other PCEs in a multi-PCE joined together via the Foundation. Each site may employ a *different* local PCE, selected to best fulfill its own needs or retained for historical reasons. A few may happen to use independent copies of the same system, or local PCEs with similar architectures and interfaces, but we cannot count on that and therefore treat each SubEnv as unique within its federation.

We have identified the following requirements for heterogeneous federation. In general these are *in addition to* homogeneous federation requirements, although in some cases we repeat the seemingly identical requirement followed by new discussion oriented towards the special circumstances of the heterogeneous case.

- The most basic function of the Foundation is to communicate with each SubEnv participating in the federation. In general, the SubEnvs cannot communicate directly with each other since (by definition) they were designed as independent PCE systems (although perhaps following a "standard" Foundation interface). Realization of multi-site tasks, or fulfillment of a request from one site for another site to undertake a task on its behalf, requires that some logically homogeneous component is added to each PCE so that it can bind into the federation. This component may involve quite diverse per-PCE physical implementations, e.g., to perform PCE-specific protocol conversions and data format translations. The conceptually common component is relatively limited, though, and in particular does not take over local process modeling and enactment functions — since otherwise we could consider it to effectively convert the federation to the homogeneous case.

- The SubEnvs (usually) must somehow be made aware of any federation(s) in which they participate, possibly more than one at a time, in order to interoperate and contribute to global process enactment. The SubEnvs need no direct knowledge of the other SubEnvs in the federation, per se, but there must be some means whereby the Foundation coordinates the global process, either by *notifying* a given SubEnv that it should or could perform specific tasks or by posting the request to some standard forum that each SubEnv *polls* to choose tasks it is able and willing to perform.

  Note this does not necessarily assume that SubEnvs have some means to inform the Foundation of pending tasks that they are unable or unwilling to do themselves: The Foundation could itself impose <u>all</u> tasks, perhaps through a special process modeling and enactment system intended to act as a "global hand" supporting some form of "superworkflow" [62], analogous to multi-part transactions submitted to heterogeneous multi-databases.

  In principle, it might be plausible for a SubEnv to perform work on behalf of a federation without ever noticing that the heterogeneous federation exists (which would not normally be the case for homogeneous federations). Thus an alternative is that only the Foundation is aware of the various SubEnvs, and picks up their results through some non-intrusive manner, such as understanding file formats of what the PCE considers internal process state information (as done, for example, in [27, 55]). This alternative model operates more in the vein of a broadcast message server, such as Field [59], where the only purpose of the Foundation is to forward notification messages that a particular SubEnv has *already* performed a particular task. This could be augmented with limited process support, as in Forest [20] and Provence [43], to transform notification messages into requests to perform various tasks triggered by process enactment in the Foundation.

- When process enactment at one SubEnv involves access to data "owned" by one or more other SubEnvs, the Foundation must provide mechanisms for transferring product artifacts and requisite process state among SubEnvs. As in homogeneous federations, bulk data may be temporarily cached, permanently copied, or migrated between sites. Note enactment of such tasks may not be frequent in a multi-PCE, e.g., data exchange may be limited to scheduled milestones, whereas collaborative tasks are expected to be more commonplace in a multi-site PCE.

  A homogeneous federation can assume a standard data repository, although perhaps with differing local schemas, whereas heterogeneous federations also incur the problems of incompatible data formats; this is basically a distributed computing issue attacked by OMG [50] and others through CORBA and similar layers, and not addressed further in this paper. Further, homogeneous federations assume compatible transaction management (concurrency control and failure recovery), generally supporting a two-phase commit protocol for distributed transactions, which may not be straightforward in the heterogeneous case. This issue has been addressed extensively in the database community, e.g., [15], and is also not discussed further here.

- Finally, there should be some means for configuring a federation. We anticipate this is considerably more difficult and heavyweight for heterogeneous than for homogeneous federations, and in the former case may involve substantial design and implementation to introduce a new PCE (i.e., if it was not previously integrated with the Foundation) rather than just invoking a pre-defined (re)configuration process. Substantial effort may be involved in introducing the conceptually homogeneous infrastructure component mentioned above into a given PCE, and the mechanism for doing so is necessarily ad hoc (i.e., PCE-specific).

2.6 Heterogeneous Federation Architectures

There are three main categories:

- **Ad hoc:** A handcrafted federation consisting of a very small number of distinct PCEs, e.g., Bull's ACME [2] integration of ConversationBuilder [40] and Marvel 3.x. While one might be able to find a special-purpose Foundation component in this or similar examples, we are concerned in this paper with general federation. There is no distinction between the ad hoc approach and peer-to-peer architectures in the heterogeneous case, because by definition there is no common means for introducing a tightly coupled Foundation moderator, or equivalent code inside the SubEnv's internal architecture, nor any general way for a moderating component to incorporate SubEnv-specific knowledge of process and other concerns.
- **Centralized glue:** The SubEnvs communicate/interact through a centralized component that implements the Foundation — the same architecture as shown in Figure 2, except that the SubEnvs may be different (originally) single-site PCEs rather than components of the same multi-site PCE. This approach is exemplified by ProcessWall. Mentor takes a similar tack, except that there may be *multiple* state servers and task servers, not just one (of each).
- **Decentralized glue:** The Foundation is divided into multiple distributed components, i.e., intermediaries analogous to those attached to each SubEnv as in Figure 3 and loosely coupled with their local SubEnv, except that each of the SubEnvs may be a different system. There is no centralized component.

When a wide range of internal architectures is exhibited among the PCEs of interest, there is usually no obvious preference exhibited for centralized vs. decentralized glue, except as in any distributed system a decentralized approach will generally scale better; however, the extra software engineering effort of separately retrofitting a large number of existing local PCEs is more likely to be the limiting factor than a central bottleneck. Thus the number of local PCEs is expected to be small. By analogy to the discussion of Section 2.4, when scaling is not an issue, top-down global processes would generally be more amenable to a centralized Foundation, and bottom-up to a decentralized Foundation. But the case is not so compelling for heterogeneous as for homogeneous federation, there considering glue vs. peer-to-peer, since construction of a global process using diverse process modeling languages and paradigms is so complex as to overwhelm all other concerns.

This may be why Heimbigner proposes a third model for his process task server, a hybrid of top-down and bottom-up: *constructor* PCEs post pending tasks to the shared process repository in a generally bottom-up fashion, whereas it makes more sense for the *constrainer* PCEs, which remove disallowed tasks from among those posted, to be organized top-down. That is, constructors create newly instantiated tasks according to local process workflow, but constrainers may disallow some tasks according to global process constraints. A distinguished constructor, effectively part of the Foundation, might post multi-site tasks implementing a top-down global process.

The discussion in Section 4 considers a heterogeneous federation architecture where Oz plays the role of a constructor (and employs its own constraints prior to instantiating a task to post); other PCEs integrated into the same federation via ProcessWall could, of course, act as constrainers on the posted tasks as well as additional constructors.

3 The Oz Homogeneous PCE Federation

Oz is the only fully implemented homogeneous PCE federation that we know of.[2] Oz originally (versions 1.1.1 and earlier) followed the direct peer-to-peer architectural model, where the majority of the Foundation functionality was built into the process engine (as elaborated in [6]). Oz was later reimplemented (versions 1.2 and later), using a new process engine as moderated peer-to-peer — with the Foundation moderator separated out into a component invoked via generic "callbacks" from the process engine. We are primarily concerned with the later form of Oz in this paper.

The overall choice of architecture for Oz follows the analysis given in Section 2.4. First, one of the major requirements for the Oz system is to support interoperability among autonomous, geographically distributed, and possibly pre-existing processes, e.g., the latter might have been designed for a single-site Marvel environment; we developed a utility that mechanically upgrades an instantiated Marvel environment to an Oz SubEnv. This requirement implies a bottom-up approach, which in turn suggests a peer-based architectural style. However, we have constructed both bottom-up (e.g., see [6], Appendix A) and top-down (e.g., see [38]) global processes for Oz.

Second, Oz was developed (among other reasons) to interconnect instances of the Marvel framework. Since Marvel's client/server architecture corresponds to the centralized-process-control, decentralized-tool-execution local architecture, and Marvel's process engine provided no API and the source code was handy, it was natural to adopt a direct peer-to-peer approach. Later on, Oz's native process engine adapted from Marvel was replaced with the Amber process server [54], which provides an API (and "callback" interface), and hardwires neither centralized-process control nor decentralized-tool-execution. Amber itself was not modified at all to produce Oz multi-site functionality, as discussed in [34].

3.1 Marvel and Oz Overview

Everything described here about Marvel is also true for Oz unless stated otherwise.

Marvel [37] provides a rule-based process modeling language in which a rule generally corresponds to a process step, or task. Each rule specifies the task's name as it would appear in a user menu or agenda; typed parameters and bindings of local variables from the project objectbase; a condition or prerequisite that must be satisfied before initiating the activity to be performed during the task; the tool script with in, inout and out arguments for the activity [21]; and a set of effects, one of which asserts the actual results or consequences of completing the activity (some activities have more than one possible result). Built-in operations like `add`, `delete`, etc. are modeled as rules, and can be overloaded, e.g., to introduce type-specific conditions and effects on those operations; built-in operations can also be used as assertions in the effects of other rules.

Marvel enforces that rule conditions are satisfied, and automates the process via forward and backward chaining. When a user requests to perform a task whose condition is not currently satisfied, the process engine backward chains to attempt to execute other rules whose effects may lead to satisfying the condition; if all possibilities are exhausted, the user is informed that the chosen task cannot be enacted at this time. When a rule completes, its asserted effect may trigger forward

[2]The Programming Systems Lab used a multi-site Oz environment to support all our day-to-day software development for about two years, since April 1995, although as of this writing we are transitioning to the OzWeb extension of Oz, which operates over the World Wide Web and supports a hypercode representation of product artifacts [35].

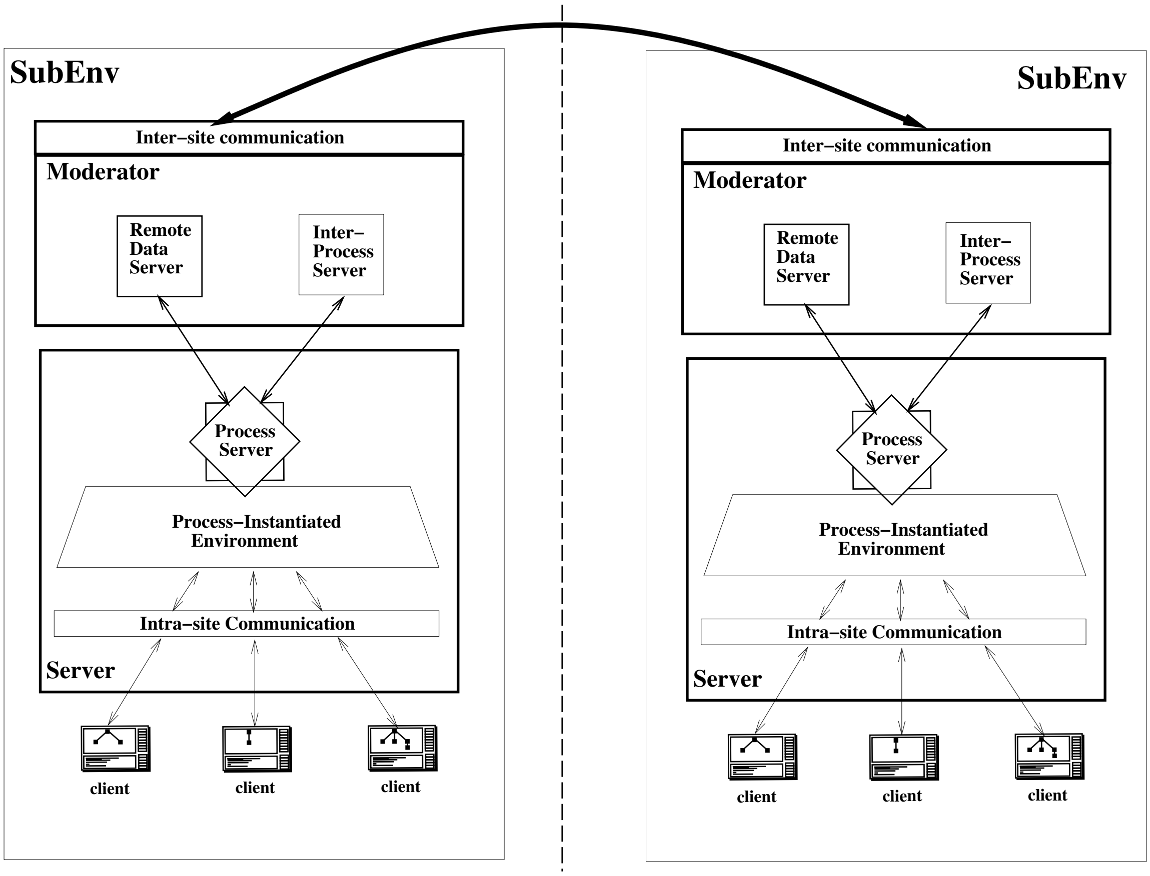

Figure 6: Oz External Architecture

chaining to automatically enact other rules whose conditions have become satisfied. Users usually control the process by selecting rules representing entry points into composite tasks consisting of one main rule and a small number of auxiliary rules (reached via chaining) for change propagation and automation of menial chores, but it is possible to define complete workflows as a single goal-driven backward chain or event-driven forward chain.

Marvel employs a client/server architecture [12]. Clients provide the user interface and execute tasks, usually by invoking external tools. The server context-switches among multiple clients, and includes the process engine, object management, and transaction management. Oz is essentially the same as Marvel, except that an Oz environment may consist of several servers, each with its own distinct process model, data schema, objectbase and tools [6]. Clients are always connected to one "local" server, and may also open and close connections to "remote" servers on demand. A server and its "local" clients constitute a SubEnv. The external view of the multi-site peer-to-peer Oz architecture is shown in Figure 6.

Oz servers communicate with each other mainly to establish and operate *alliances*, which involves (1) negotiation of *Treaties* — dynamically agreed-upon shared subprocesses that are automatically and incrementally added on to each affected local process on the fly when the Treaty is instituted (and automatically and incrementally removed when a site unilaterally revokes the Treaty); and (2) coordination of *Summits* — enactment of Treaty-defined process segments that involve data and/or local clients from multiple sites, with computation interleaved between shared and local computational models (i.e., the Treaty and the local processes). We stretch the International Alliance metaphor, since Treaties among sites precede and specify Summits rather than vice versa.

3.1.1 Treaties and Summits

The purpose of a Treaty is to establish a common subprocess. A Treaty consists primarily of a set of Oz rules. These rules define intentionally multi-site tasks, where the parameters are expected to be selected from multiple sites (i.e., distinct Oz objectbases chosen by a user via open connections to "remote" servers). Such tasks must be defined somewhere, in Oz's case within one of the participating SubEnvs. However, in general, Treaty rules are not an inherent component of any SubEnv's local process. Instead, the common subprocess is combined into each local participating process, in the sense that its tasks may be synchronized with other local tasks, depend on the outcome of their execution, and vice versa.

This is relatively easy to do with rules, the basis of Oz's process modeling formalism, since process enactment follows automatically determined forward and backward rule chains based on matching a predicate in one rule's condition to an assertion in another rule's effect [30]. It does not matter to the rule network construction algorithm whether the rules are included in the local process model or added later via a multi-site Treaty.

Treaties are defined pairwise between two Oz servers at a time, which allows local SubEnv administrators to form such agreements in a fully decentralized manner, without involving any global authority. Still, a Treaty among any number of sites can be created by forming all the relevant binary Treaties (and Oz provides commands to do this in one step, if the relevant administrator has appropriate privileges at each affected SubEnv). A Treaty between SubEnvs SE_1 and SE_2 over a subprocess SP is established when:

1. SE_1 issues an *export* operation of SP to SE_2. This operation assumes that SP already exists in SE_1 (either locally defined or imported from another SubEnv) and thus already integrated within its own local process. *export* also specifies execution privileges and general access control to the exported subprocess.

2. SE_2 issues an *import* command that fetches SP from SE_1 and tightly integrates it into its local process (the rules in a Treaty can be executed on purely local data, in which case they are not in any way distinguished from the local process).

In order to control execution privileges, i.e., which site(s) can initiate multi-site tasks (e.g., due to platform restrictions, security, etc.), both the *export* and *import* operation are parameterized with permissions to control on which site(s) those tasks can be executed and from which site(s) the relevant data can be fetched. That is, Treaties are <u>not</u> symmetric unless specified as such by both the *export* and *import* operations. To support decentralization, Treaties may be withdrawn unilaterally (except while multi-site tasks are actually being executed); note this requires dynamic Treaty validation, i.e., checking that none of the affected parties to the Treaty have revoked it. Thus, not only does the Treaty mechanism allow definition of decentralized multi-site processes, but the (meta-)process for establishing and maintaining Treaties is itself highly decentralized.

Summits are the process enactment counterpart of Treaty process models. When a multi-site rule (from some Treaty) is issued for enactment by a user client of its "local" Oz server, termed hereafter the *Summit coordinator*, that SubEnv performs the following main steps:

1. Verify that the corresponding Treaty is valid (i.e., it has not been retracted by one of the SubEnvs whose data was selected as parameters to the rule).

2. Evaluate the rule's condition to determine whether or not it is satisfied. This requires fetching all the parameters from their home sites and caching them locally.

3. If the condition is not satisfied, send to those participating SubEnvs whose data fail their condition predicates a request to issue *local pre-Summit* tasks, which involve local (hence private) process steps on local data with local tools determined via backward chaining from the requested multi-site rule.

4. Wait for all sites to return before continuing to the next phase. Note that if the original rule's condition is already completely satisfied, then the pre-Summit phase is null.

5. Execute the multi-site activity of the rule, usually but not necessarily involving data from multiple sites (it is possible that remote parameters appear only in the condition and/or an effect), and possibly multi-site tools (e.g., groupware).

6. Send to each participating SubEnv a request to update its own data affected by assertions of the rule's actual effect (determined according to the return code of the rule's activity).

7. Each such SubEnv issues corresponding *local post-Summit* tasks determining via forward chaining from the relevant assertions of the original rule's effect, again involving only local resources.

8. Wait for all affected sites to reply.

9. Enact further related Summits, if any, reached via forward chaining to other multi-site rules from the original Summit rule.

Thus, Summits alternate between execution of shared, global, and multi-site tasks, to execution of private, local and single-site tasks, and effectively enact multi-site processes with minimal interprocess dependencies beyond the explicitly defined shared subprocesses. Full details of Summits and Treaties are given in [10].

Treaties and Summits impose several requirements on the design of Oz's federated architecture. First, the tight process-level integration of an imported subprocess into the local process implies a strong coupling of the Foundation with the local PCE. The strict decentralization, even in the definition of the federated process (using Treaties), avoids the need for a global process state (except for the special configuration process, described in [8]), and further supports our choice of a peer-to-peer architecture according to the issues discussed in Section 2.4.

Finally, Summits do not require a global process controller, but do require functional extensions to local process engines to allow them to become Summit coordinators. Again, the peer-to-peer architecture is favored. However, none of the above aspects indicate any preference with respect to direct vs. moderated (peer-to-peer) approaches, assuming the process engine can be extended into a coordinator without internal code modifications — as is the case for the Amber process server but which would have been difficult for Oz's native process engine. Thus this decision is likely to be based on the lower-level architectural and implementation aspects of the local SubEnv — and, as noted above, we have tried both models and prefer the moderated approach on modularity and extensibility grounds.

Note that none of these concerns are specific to rules, as opposed to some other process paradigm, except in the sense that the rule network generation algorithm makes it trivial to tightly bind imported rules with the local process. Integration of imported Treaties into the local process for non-rule process paradigms is more complicated, but possible, as discussed in [10] and not addressed here.

3.2 Oz Architecture

The internal architecture of Oz is shown in Figure 7. We use the following graphical notations: squared boxes with the widest bold lines (e.g., the Server) represent operating system processes, or independent threads of control; squared boxes with lines with intermediate width (e.g., the Process component) represent top-level computational components that are part of the same operating system process as other components but are relatively independent from those components; squared boxes with narrow solid lines are computational subcomponents; dashed-line separators within subcomponents further modularize a (sub)component into its various functionalities; shaded rectangles within the above indicate "external" modules that extend the functionality of the basic component (as explained below); shaded ovals represent data repositories; and arrows represent data and/or control flow. The relative sizes of the various units are **not** intended to be meaningful.

Oz consists of three main runtime computational entities: the Environment Server (or simply, the *Server*), the *Connection Server*, and the *Client*. In addition, there are several utilities that convert the various project-specific definitions into an internal format that is understood and loaded by the server; they are of no concern in this paper.

There are three kinds of interconnections: client-to-local-server, client-to-remote-server, and server-to-server. The first connection is "permanent", in the sense that its existence is essential for the operation of the client. That is, a client is assumed to always be connected to its local server, and when such a connection becomes disconnected (either voluntarily on demand or involuntarily due to some failure) the client normally shuts down and is removed from the local server's state.[3] In contrast, the two other connections can be regarded as "temporary", since they are optional, and can be dynamically reconnected and disconnected over the course of a user session without disrupting the local operation of a SubEnv. This is a necessary feature to fulfill the independent-operation requirement, particularly when the servers are spread arbitrarily over multiple administrative domains.

An Oz (multi-site) environment consists of a set of instantiated SubEnvs, and at any point in time none, some, or all SubEnvs may be *active*. A SubEnv is considered active if exactly one server is executing "on the environment", meaning that it has loaded the SubEnv's process, and the SubEnv's objectbase (containing persistent product data and process state) is under the control of the server's data management subsystem (described in [44]). Typically an active environment also has at least one active (i.e., executing) client connected to its server, because the server automatically shuts itself down when there are no more active clients (and is automatically started up on demand by the Connection Server, as will be explained shortly).

[3] An extension of this model, in which clients can be disconnected from their server and continue to operate independently to enact a process segment until reconnection, has been investigated separately to support mobile computing [67].

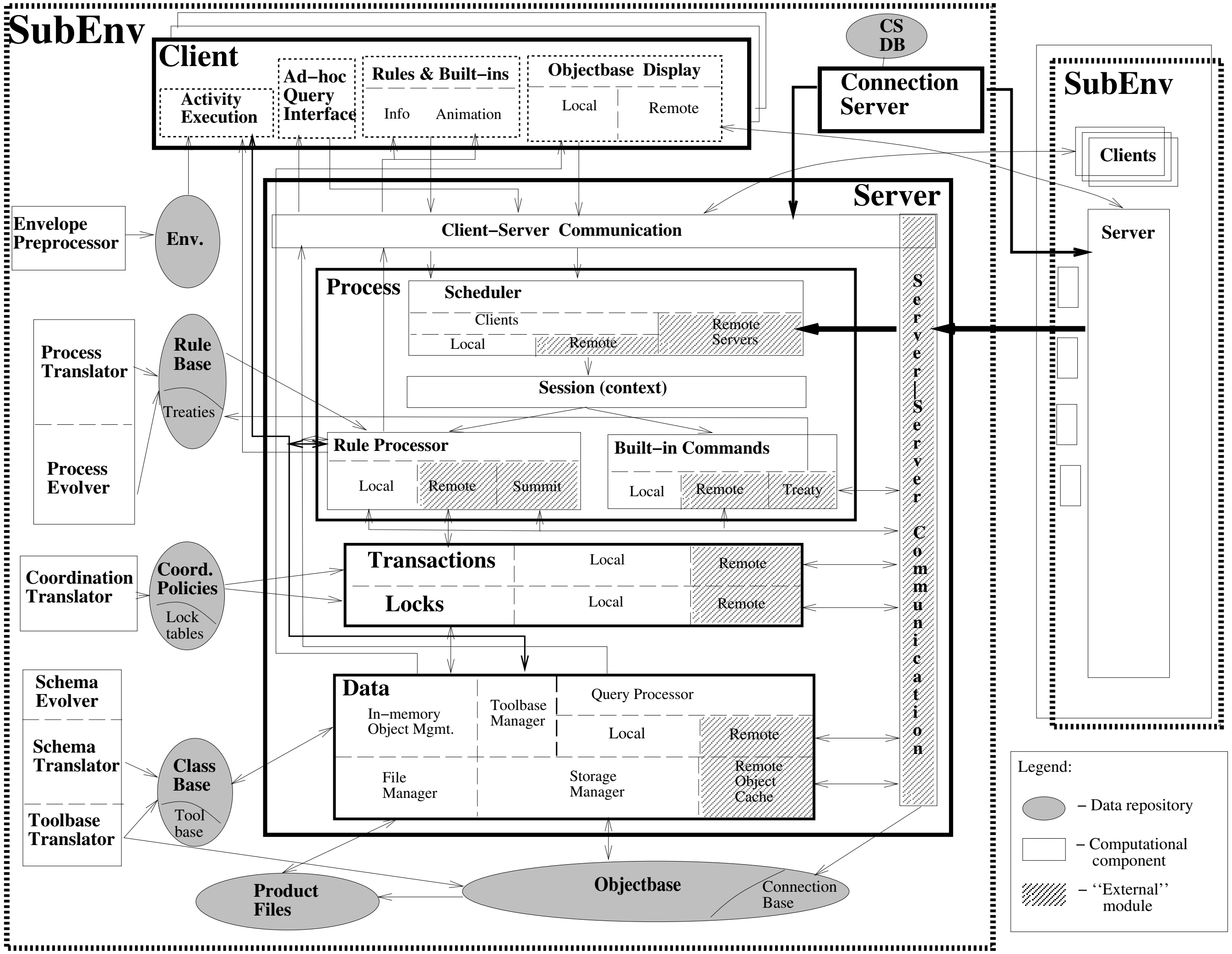

Figure 7: Oz Internal Architecture

3.2.1 The Environment Server

The server consists of three major components: process, transaction and data managers, each of which can be separately tailored by a combination of two facilities: declarative definitions loaded from a file and "external" code modules. The process manager loads the process model (including portions obtained through Treaty *import*), the transaction manager is parameterized by lock tables and concurrency control policies, and the data manager loads the schema for the product data and process state (currently imported rule sets must employ subschemas compatible with the local schema, although some conversion is supported). All of these tailorings are stored in environment-specific files; see [56] for details. The conceptually "external" code is hardwired into Oz's data manager [4], reasonably independent and invoked through a callback interface in the case of the process manager, and completely independent and dynamically loaded for the transaction manager. We do not consider the distinction between "external" vs. intrinsic code further in this paper, that is the subject of other papers ([34] and [28], respectively).

Process Manager

The process manager is the main component of the server. Its frontend subcomponent is the scheduler, which receives requests for service from three entities that correspond to the previously mentioned interconnections, namely local clients, remote clients, and remote servers. With few exceptions, notably to prevent deadlocks among mutual server communications, these requests are served on a first-come-first-served basis. The server is non-preemptive, i.e., it relinquishes control and context-switches only voluntarily.

The session layer encloses each interaction with a server in a context containing information that enables it to switch between and restore contexts. The context of locally executing tasks, including those that execute as part of a pre- or post- Summit, and the context of composite (multi-task) Summits, are represented in task data structures.

The rule processor consists of subcomponents for processing local tasks, local tasks spawned via pre-Summit or post-Summit processing from either local or remote Summits (denoted "Remote" in the figure), and Summit tasks. There are very few "system" built-in activities (notably parts of the configuration process), so the behavior of a particular instantiated SubEnv is mostly determined by the rule set that defines the process.

The built-in command processor handles all the kernel services that are available to every SubEnv. These include the primitive structural operations on the objectbase (e.g., `add` and `copy` object), several display options and image refresh commands, access control, and the various dynamic process loading and Treaty operations.

In the original direct peer-to-peer variant of Oz, all alliance support was hardwired. But in the newer moderated peer-to-peer versions, Summits, Treaties and related infrastructure has been culled out into the "external" code modules indicated in the figure.

Transaction Manager

All access to data is mediated by Oz's transaction manager. Due to the required decentralization, each transaction manager is inherently *local*, i.e., it is responsible only for its local objectbase. However, transaction managers attached to each server communicate among themselves to support concurrency control and failure recovery involving remote objects. Oz's transaction manager was

[4] Such code has been separated out in the later OzWeb.

developed separately and has been used independent from the rest of Oz. Further details are outside the scope of this paper, see [26].

Data Manager

This component includes an in-memory object manager that provides uniform object-based access to data from any system component. Objects can be looked up in one of three ways: by structural navigation, by class membership, and by their object-identifier (OID). Structural and by-class searches are requested by the query processor to service navigational and associative queries, respectively, and by-OID lookup is used for several purposes, among them to support direct user selection of objects (mouse clicking in the objectbase display) as parameters to rules.

The second major subcomponent is the query processor. It supports a declarative query language interface, and is called from both the rule processor for embedded queries and directly from the user client for servicing ad hoc queries. Queries on remote objects are handled at this level, by invoking a server-to-server service.

The rest of data management consists of an untyped storage manager (implemented on top of the `gdbm` package) that stores the objectbase contents; a file manager that handles access to file attributes (file attributes are paths to files resident in the environment's "hidden file system"); and an object cache that holds transient copies of remote objects during Summits.

The data manager is tailored by the project-specific schema tied to the instantiated objectbase, including both class- and composition-hierarchies. As in the case of rules and the process manager, without a schema the data manager is useless since it cannot instantiate any objects.

3.2.2 The Client

There are three main clients, supporting XView, Motif and tty (command line) user interfaces, as well as several auxiliary clients with no user interface intended primarily for tool execution. Each client consists of four major subcomponents: (1) access to information about rules and built-in commands, (2) objectbase representation, (3) activity execution, and (4) an ad hoc query interface.

The two graphical user interface clients are (conceptually) multi-threaded, i.e., a single client can support multiple concurrent interactions with local or remote servers. This enables a user to run in parallel several (possibly long) activities from the same client. The command interface includes a process-specific menu and utilities for displaying rule definitions and the rule network interconnections, all of which are stored in the client's address space and can be dynamically refreshed when a new process is (re)loaded or a Treaty is formed. A dynamic rule-chaining animator shows the control flow of enacted tasks as they execute, both local and Summits.

The objectbase display maintains an "image" of *structural* information, i.e., parent/child and reference relationships, for browsing and for selecting arguments to activities. The contents of primitive and file attributes are transmitted only when needed. Users can select the `open-remote` command to display the objectbase images from other sites and subsequently select objects from multiple sites, allowing invocation of a Summit rule. The client maintains multiple simultaneous connections to the remote servers, and is able to direct requests to appropriate servers.

3.2.3 The Connection Server

The Connection Server's main responsibility is to (re)establish connections to a local server from local clients, remote clients, and remote servers. However, it does not participate in the actual interactions between those entities; it serves only as a mediator for "handshaking" purposes. In some cases, the destination server to which a request for a connection is made may not be active, in which case the Connection Server is capable of automatically (re)activating the dormant server. In other cases the desired server may be active but its address (host IP address and port number) might be unknown to the requesting entity, in which case the Connection Server sends that information to the requesting entity for further communication.

Unlike the Environment Server, the Connection Server is (conceptually) always active, since it is implemented as a daemon invokable via the Unix `inetd` mechanism. Thus, each configured host has its own (logical) Connection Server that supports all SubEnvs (of the same or different multi-site environments) that reside on that host.

4 Oz/ProcessWall: A Hypothetical Heterogeneous Federation

A heterogeneous federation is inherently more general than a homogeneous federation. Thus it is desirable to consider how a multi-site PCE like Oz might "fit" into a multi-PCE organized via a process state/task server, the only specific model we know of for heterogeneous federation of PCEs. Note the federation would presumably also include various non-Oz SubEnvs.

One approach is to drop the homogeneous Foundation entirely and employ only the heterogeneous Foundation for multi-site tasks. Then the homogeneous SubEnvs — i.e., homogeneous with respect to system but heterogeneous with respect to process model — would be treated as if they were unrelated local PCEs rather than part of an Oz multi-site PCE. Assuming that some component is added to interface with the federation glue, this should work trivially if they fulfill the requirement of independent operation — that is, that they do not depend on each other in any way to perform entirely local work. In other words, in principle we could have used ProcessWall to scale up Marvel to support process interoperability across multiple teams, each with their own local Marvel environment instance. But then the main advantage of a homogeneous federation is lost, namely the relative ease with which SubEnvs can call on each other to perform specific agreed-upon services within the identical (and thus mutually understood) process modeling and enactment paradigm.

An alternative approach is to allow individual (or all) SubEnvs of a homogeneous federation to participate in one or more heterogeneous federations, while retaining the higher level of intimacy afforded by the system-level homogeneity when (intentionally) interacting with other local components of the same system.

Note that SubEnvs that happen to participate in the same homogeneous federation may happen to employ each other's services indirectly through the heterogeneous federation, without necessarily any knowledge that they have more direct means of interaction. In fact, through this "backdoor" one might, in an unusual circumstance, inadvertently arrive in a situation where a SubEnv indirectly requests services from itself without realizing that its doing so — which could potentially happen in a homogeneous federation as well, although not in the Oz realization because the server checks for this degenerate case.

4.1 Issues

Assuming a centralized Foundation in the style of the ProcessWall state/task server, the main questions to answer are:

1. How would an Oz SubEnv post to the Foundation those tasks it has instantiated but not initiated, and (generally speaking) would like some other PCE to perform? The local task descriptor must of course be converted to the Foundation's standard form. There are complications regarding representation of data arguments as part of the task specification, and later regarding data transfer when the task is enacted by some PCE participating in the federation. Even though the task may eventually be picked up by another Oz SubEnv, this cannot be assumed a priori (if it could, direct interaction through the homogeneous Foundation would almost certainly be more efficient).

2. How would the Foundation notify an Oz SubEnv of the completion of such a task? The Foundation's relevant state must be converted to a form understood by Oz, and either automatically transmitted to Oz by the Foundation or explicitly requested by Oz, e.g., via periodic polling or some kind of rendezvous. When data arguments are modified during the task, changed data must either be submitted back to the Oz objectbase, or Oz must be notified of its whereabouts and have some means to retrieve the data.

3. How would the Foundation inform a particular Oz SubEnv that it should perform a specific posted task? One approach involves some kind of scheduler or other entity that selects among enabled tasks for enactment, chooses the recipient SubEnv, and sends an appropriate request to that SubEnv. This Foundation-generated request model is compatible with Oz's current server-to-server communication mechanism. In contrast, a completely new interface would be needed to fit into a blackboard model that required each SubEnv to poll the Foundation for suitable enabled tasks. However, a hybrid might be achieved in the style of a broadcast message server like Field, where the SubEnv's *register* their interests in or abilities to perform certain kinds of tasks, perhaps by supplying a pattern that is matched by the Foundation against the enabled task specifications. The application of event subscription to workflow management system interoperability is suggested in [62].

4. How would an Oz SubEnv notify the Foundation of a task it had just completed? There are two cases: the task was previously posted to the Foundation by the same or a different SubEnv, or it arose entirely inside the given SubEnv (and thus is supplied only as historic information). In the former case, the Foundation might have requested that this SubEnv perform the task, or alternatively the SubEnv might have selected the task from among those enabled (via either polling or registration). Note that generally it is necessary for the Foundation to prevent multiple SubEnvs from concurrently agreeing to perform the same posted task. Again, data transfer is of concern here in both directions.

4.2 Integrating Oz Tasks and ProcessWall Tasks

There are three different "levels" of task-like units supported by Oz: rule activities, full rules, and rule chains, any or all of which could be mapped to ProcessWall's notion of tasks. The answers to the questions above will be somewhat different depending on which Oz unit is chosen for the mapping.

Oz's lowest task-like "level" is an individual rule activity, i.e., invocation of a tool script (and thence external tool). Oz already defines a client/server protocol whereby user interface clients tell the server to apply a selected rule to a list of objects and literal arguments; once the condition is deemed satisfied, the server supplies the client with corresponding file pathname and primitive arguments, and directs it to invoke the tool script specified in the rule activity; the client forks the tool script, and the tool script or the tool(s) it invokes are assumed to directly modify the contents of file arguments; finally, the client returns to the server with the return code from the tool script, which selects among the possible rule effects, as well as (optional) assignments to output variables. The encapsulating rule (and its pending chain) then continues.

It is simple to construct a special Oz client that receives the same message from the server identifying tool script and arguments but does something different than the typical user client; in fact, we have already introduced numerous such clients (see [66, 18, 70, 46]). Then, to implement points 1 and 2, the new client would be inserted into the multi-PCE architecture between the Oz server and the central Foundation. This client would convert the activity information provided into the Foundation's task representation and forward it to the centralized Foundation. Later, after the activity has been completed, the Foundation would notify this special client, which would then respond to the Oz server like any of its other clients. The special client would be responsible for data traffic in both directions.

The same special Oz client (or alternatively a distinct special client) can be used to implement points 3 and 4. The Foundation sends the task to the Oz client for that client to execute itself using the same tool invocation facilities as any other Oz client, without involving the Oz server. When the task completes, the special Oz client returns the results to the Foundation, again without interaction with the Oz server. Note that some special communication facilities will be needed in the Oz client, if the same client is used for both purposes, to avoid deadlock when the client happens to be forwarding an activity to the Foundation to be performed by some other PCE at the same time that the Foundation is sending a request to the client.

The intermediate task "level" corresponds to entire Oz rules, with condition and effect(s) as well as activity. Oz servers already transmit rule definitions between themselves as part of Treaty negotiation, and transmit the parameters and bound variables of instantiated rules as part of Summit enactment. The newer variants of Oz introduce a protocol for transferring instantiated rules between client and server to support delegation to and selection from user and group agendas ("to do" lists) [34]. Rules with and without already satisfied conditions may appear in an agenda. These facilities might be combined and extended, again through a translating client interposed between Oz and the Foundation, to support all four points.

A complication: In the lowest level case the condition or prerequisite is already satisfied, by definition, prior to posting the activity, but this would not in general be true in the intermediate case. Analogously, handling the effect or consequence of the activity is the concern of only the originally posting PCE, but again this cannot be assumed in the intermediate case. The ProcessWall and Mentor task representations allow for predecessors and successors, but not all the constraints embodied in Oz conditions are concerned with checking simple predecessor relationships (e.g., the local variable bindings might find all objects that match a complex associative query and then the condition checks that at least one of those objects satisfies a complex logical clause), nor are all assertions made in Oz effects concerned with triggering successors (e.g., objects can be created and deleted, reference links formed and removed, etc. through invocation of built-in operations).

One could argue for a simplification, whereby Oz's postings to the Foundation are limited to those tasks whose conditions and effects are solely concerned with predecessor/successor relations that can be directly represented by the Foundation's state and/or task model. Although Oz's process modeling language tends to obscure such relationships from a human-readability standpoint, they are visible in the internal rule network compiled from the process model. Since the Foundation can only represent such relations, by definition any requests sent by the Foundation to Oz would so restrict the implicit conditions and effects.

A better approach might be to extend the Foundation's task representation, or develop some additional control channel, for transmitting the conditions and/or effects from the Oz SubEnv to the (potentially) foreign SubEnv for evaluation within its paradigm, and vice versa regarding communicating any prerequisites and consequences that might be supported by the foreign paradigm to an Oz SubEnv (and of course both issues come up between pairs of non-Oz SubEnvs as well). If Oz were configured as a multi-site homogeneous federation where some (or all) sites happened to also belong to a heterogeneous federation, pending tasks posted through the Foundation to another Oz SubEnv (in the same federation) could include their conditions and effects in some *opaque* data stream understood only by Oz servers.

So difficulties arise only when pending tasks posted through the Foundation involve non-Oz SubEnvs. Fortuitously, we have already shown fairly straightforward mappings from most of the major PCE paradigms, including Petri nets [55], task graphs [25], and grammars [39], into Oz rules, and reverse mappings are not inconceivable. And as previously noted in Section 1, Mentor supports translation from one notation into another, as does the process interchange format standardization effort.

However, the general case requires substantial translation capabilities regarding data formats, and predicates and operations over those formats. The "universal data model" problem is a well-known unresolved, probably unresolvable issue in database research [49]. It may be possible to address a special case of this problem with respect to PCEs, e.g., if we assume the main data arguments are files and all attributes that might be referred to regarding task prerequisites and consequences (rule conditions and effects in Oz) are standard file system appendages supported by most operating systems, such as owner, read/write timestamps, access permissions, etc., or encoded predecessor/successor relationships modeled directly by the Foundation.

The third task "level" in Oz is a rule chain, i.e., all the rules emanating from some user-selected (or Foundation-requested) rule through backward and/or forward chaining. This seems easiest to handle by iterating the intermediate case as the rule chain unfolds, in the Oz to Foundation case, or incrementally sending each member of a sequence of predecessor/successor tasks, in the Foundation to Oz case.

Note we have so far ignored the issue of specifying which PCE *user* should perform the work in the case of an interactive task. Oz version 1.1.1 included process modeling and enactment facilities, which could be revived, to delegate a rule activity to a specific user, or to one or all members of a user group [11]. Later versions of Oz can delegate an entire rule or (rest of a) rule chain to a specific user or any group member via "guidance chaining", a form of forward chaining where the next rule in the chain is placed in an agenda rather than immediately enacted [69]. How *another* PCE might designate a user to perform tasks originally instantiated by Oz is of course open-ended.

5 Contributions and Future Directions

The main contributions of this work are:

- The elaboration of requirements and architectures for homogeneous and heterogeneous federations of process-centered environments. Both the homogeneous and heterogeneous federation architectures we present are in line with a proposed distributed workflow reference model [71].
- The design and realization of a specific homogeneous federation architecture for Oz.
- A presentation of the issues that must be addressed to integrate Oz into a heterogeneous federation based on the ProcessWall process state/task server (or Mentor worklist/history manager) approach.

The obvious next step is to complete an experimental integration between Oz and the realization of a process state/task server, assuming one becomes available. [5] It would be desirable to also include in the heterogeneous federation at least one other PCE, besides Oz. Evaluation against the heterogeneous federation requirements should prove interesting.

Finally, we would like to introduce greater flexibility into Oz alliances, i.e., homogeneous federations, including navigation and search among related SubEnvs both within and across alliances, easy movement of user clients from one SubEnv server to another, and lighter weight composition and destruction of SubEnv alliances. Note the second point presumes support for arbitrary geographical dispersion *within* a SubEnv, not just *among* SubEnvs. Thus Oz user clients could not continue to assume a shared network file system for accessing objectbase file attributes and communication bandwidth may become a concern, issues already addressed to a limited extent for our Oz "low bandwidth clients" in [66].

Acknowledgments

We would like to thank Steve Dossick, George Heineman, Wenyu Jiang, Wenke Lee, Steve Linde, Steve Popovich, Peter Skopp, Jack Jingshuang Yang and Sonny Xi Ye for their various work on Oz, Amber and the replacement of Oz's native process engine by Amber. We also thank Dennis Heimbigner, Alex Wolf and George Heineman for their first cut towards a hypothetical Oz/ProcessWall integration as part of a funding agency presentation. Oz is available for licensing at no cost for research and educational purposes; send email to MarvelUS@cs.columbia.edu for information. Connect to http://www.psl.cs.columbia.edu/ for up-to-date information about Oz and related projects.

This paper is based on work sponsored in part by the Defense Advanced Research Project Agency under [D]ARPA Order B128 monitored by Air Force Rome Lab F30602-94-C-0197, in part by National Science Foundation CCR-9301092, and in part by the New York State Science and Technology Foundation Center for Advanced Technology in High Performance Computing and Communications in Healthcare NYSSTF-CAT-95013. This work was conducted while Dr. Ben-Shaul was a student at Columbia University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US or NYS government, ARPA, Air Force, NSF or NYSSTF.

[5] We have asked Dennis Heimbigner to give us a copy of ProcessWall many times over the past few years, to no avail.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Gunthor, and Mohan U. Kamath. Exotica/FMQM: A persistent message-based architecture for distributed workflow management. In *IFIP WG 8.1 Workgroup Conference on Information Systems Development for Decentralized Organizations*, Trondheim, Norway, August 1995.

[2] John E. Arnold and Steven S. Popovich. Integrating, customizing and extending environments with a message-based architecture. Technical Report CUCS-008-95, Columbia University, Department of Computer Science, September 1994. The research described in this report was conducted at Bull HN Information Systems, Inc. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-008-95.ps.Z.

[3] Sergio Bandinelli and Alfonso Fuggetta. Computational reflection in software process modeling: the SLANG approach. In *15th International Conference on Software Engineering*, pages 144–154, Baltimore MD, May 1993. IEEE Computer Society Press.

[4] Naser S. Barghouti. Supporting cooperation in the MARVEL process-centered SDE. In Herbert Weber, editor, *5th ACM SIGSOFT Symposium on Software Development Environments*, pages 21–31, Tyson's Corner VA, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992. ftp://ftp.psl.cs.columbia.edu/pub/psl/sde92.ps.Z.

[5] Daniel J. Barrett, Lori A. Clarke, Peri L. Tarr, and Alexander E. Wise. A framework for event-based software integration. *ACM Transactions on Software Engineering and Methodology*, 5(4):378–421, October 1996.

[6] Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling*. Kluwer Academic Publishers, Boston, 1995.

[7] Israel Z. Ben-Shaul and George T. Heineman. A 3-level atomicity model for decentralized workflow management system. In Carlo Montangero, editor, *5th European Workshop on Software Process Technology*, volume 1149 of *Lecture Notes in Computer Science*, pages 61–64, Nancy, France, October 1996. Springer-Verlag.

[8] Israel Z. Ben-Shaul and Gail E. Kaiser. A configuration process for a distributed software development environment. In *2nd International Workshop on Configurable Distributed Systems*, pages 123–134, Pittsburgh PA, March 1994. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-022-93.ps.Z.

[9] Israel Z. Ben-Shaul and Gail E. Kaiser. A paradigm for decentralized process modeling and its realization in the Oz environment. In *16th International Conference on Software Engineering*, pages 179–188, Sorrento, Italy, May 1994. IEEE Computer Society Press. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-024-93.ps.Z.

[10] Israel Z. Ben-Shaul and Gail E. Kaiser. An interoperability model for process-centered software engineering environments and its implementation in Oz. Technical Report CUCS-034-95, Columbia University Department of Computer Science, December 1995. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-034-95.ps.Z.

[11] Israel Z. Ben-Shaul and Gail E. Kaiser. Integrating groupware activities into workflow management systems. In *7th Israeli Conference on Computer Systems and Software Engineering*, pages 140–149, Herzliya, Israel, June 1996. IEEE Computer Society Press. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-002-95.ps.Z.

[12] Israel Z. Ben-Shaul, Gail E. Kaiser, and George T. Heineman. An architecture for multi-user software development environments. *Computing Systems, The Journal of the USENIX Association*, 6(2):65–103, Spring 1993. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-012-92.ps.Z.

[13] Barry Boehm, editor. *10th International Software Process Workshop: Process Support of Software Product Lines*, Ventron, France, June 1996.

[14] Gregory Alan Bolcer and Richard N. Taylor. Endeavors: A process system integration infrastructure. In Wilhelm Schäfer, editor, *4th International Conference on the Software Process: Software Process – Improvement and Practice*, pages 76–89, Brighton, UK, December 1996.

[15] Jiansan Chen, Omran A. Bukhres, and Ahmed K. Elmagarmid. IPL: A multidatabaase transaction specification language. In *13th International Conference on Distributed Computing Systems*, pages 439–448, Pittsburgh PA, May 1993. IEEE Computer Society Press.

[16] Reidar Conradi, Espen Osjord, Per H. Westby, and Chunnian Liu. Initial software process management in EPOS. *Software Engineering Journal*, 6(5):275–284, September 1991.

[17] Michael Cusumano and Richard W. Selby. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. The Free Press, New York, 1995.

[18] Stephen E. Dossick and Gail E. Kaiser. WWW access to legacy client/server applications. In *5th International World Wide Web Conference*, pages 931–940, Paris, France, May 1996. Elsevier Science B.V. Special issue of *Computer Networks and ISDN Systems, The International Journal of Computer and Telecommunications Networking*, 28(7-11), May 1996. http://www.psl.cs.columbia.edu/papers/CUCS-003-96.html.

[19] Christer Fernström. PROCESS WEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, pages 12–26, Berlin, Germany, February 1993. IEEE Computer Society Press.

[20] David Garlan and Ehsan Ilias. Low-cost, adaptable tool integration policies for integrated environments. In Richard N. Taylor, editor, *4th ACM SIGSOFT Symposium on Software Development Environments*, pages 1–10, Irvine CA, December 1990. Special issue of *Software Engineering Notes*, 15(6), December 1990.

[21] Mark A. Gisi and Gail E. Kaiser. Extending a tool integration language. In Mark Dowson, editor, *1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 218–227, Redondo Beach CA, October 1991. IEEE Computer Society Press. ftp://ftp.psl.cs.columbia.edu/pub/psl/icsp91.ps.Z.

[22] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence Journal*, 26:251–321, 1985.

[23] Dennis Heimbigner. Proscription versus Prescription in process-centered environments. In Takuya Katayama, editor, *6th International Software Process Workshop: Support for the Software Process*, pages 99–102, Hakodate, Japan, October 1990. IEEE Computer Society Press.

[24] Dennis Heimbigner. The ProcessWall: A process state server approach to process programming. In Herbert Weber, editor, *5th ACM SIGSOFT Symposium on Software Development Environments*, pages 159–168, Tyson's Corner VA, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992.

[25] George T. Heineman. Automatic translation of process modeling formalisms. In *1994 Centre for Advanced Studies Conference (CASCON)*, pages 110–120, Toronto ON, Canada, November 1994. IBM Canada Ltd. Laboratory. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-036-93.ps.Z.

[26] George T. Heineman. *A Transaction Manager Component for Cooperative Transaction Models*. PhD thesis, Columbia University Department of Computer Science, June 1996. CUCS-010-96. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-010-96.ps.gz.

[27] George T. Heineman and Gail E. Kaiser. Integrating a transaction manager component with Process-WEAVER. Technical Report CUCS-012-94, Columbia University Department of Computer Science, May 1994. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-012-94.ps.Z.

[28] George T. Heineman and Gail E. Kaiser. An architecture for integrating concurrency control into environment frameworks. In *17th International Conference on Software Engineering*, pages 305–313, Seattle WA, April 1995. ACM Press. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-021-94.ps.Z.

[29] George T. Heineman and Gail E. Kaiser. The CORD approach to extensible concurrency control. In *Thirteenth International Conference on Data Engineering*, Birmingham, UK, April 1997. In press. Available as Columbia University Department of Computer Science CUCS-024-95 and Worcester Polytechnic Institute WPI-CS-TR-96-1. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-024-95.ps.gz.

[30] George T. Heineman, Gail E. Kaiser, Naser S. Barghouti, and Israel Z. Ben-Shaul. Rule chaining in MARVEL: Dynamic binding of parameters. *IEEE Expert*, 7(6):26–32, December 1992. ftp://ftp.psl.cs.columbia.edu/pub/psl/expert92.ps.Z.

[31] Watts Humphrey and Marc I. Kellner. Software process modeling: Principles of entity process models. In *11th Internation Conference on Software Engineering*, pages 331–342, Pittsburgh PA, May 1989. IEEE Computer Society Press.

[32] W. Jin, L. Ness, M. Rusinkiewicz, and A. Sheth. Concurrency control and recovery of multi-database work flows in telecommunication applications. In *ACM SIGMOD International Conference on Management of Data*, pages 456–459, Washington DC, May 1993. Special issue of *SIGMOD Record*, 22:2, June 1993.

[33] Gail E. Kaiser, Naser S. Barghouti, and Michael H. Sokolsky. Experience with process modeling in the MARVEL software development environment kernel. In Bruce Shriver, editor, *23rd Annual Hawaii International Conference on System Sciences*, volume II, pages 131–140, Kona HI, January 1990. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-446-89.ps.gz.

[34] Gail E. Kaiser, Israel Z. Ben-Shaul, Steven S. Popovich, and Stephen E. Dossick. A metalinguistic approach to process enactment extensibility. In Wilhelm Schäfer, editor, *4th International Conference on the Software Process: Improvement and Practice*, pages 90–101, Brighton, UK, December 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-016-96.ps.gz.

[35] Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, and Jack Jingshuang Yang. An architecture for WWW-based hypercode environments. In *1997 International Conference on Software Engineering: Pulling Together*, Boston MA, May 1997. In press. Available as Columbia University Department of Computer Science, CUCS-037-96, August 1997, ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-037-96.ps.gz.

[36] Gail E. Kaiser and Peter H. Feiler. An architecture for intelligent assistance in software development. In *9th International Conference on Software Engineering*, pages 180–188, Monterey CA, March 1987. IEEE Computer Society Press.

[37] Gail E. Kaiser, Peter H. Feiler, and Steven S. Popovich. Intelligent assistance for software development and maintenance. *IEEE Software*, 5(3):40–49, May 1988. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-401-88.tar.Z.

[38] Gail E. Kaiser, George T. Heineman, Peter D. Skopp, and Jack J. Yang. Incremental process support for code reengineering: An update (Experience Report). Technical Report CUCS-007-96, Columbia University Department of Computer Science, February 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-007-96.ps.Z.

[39] Gail E. Kaiser, Steven S. Popovich, and Israel Z. Ben-Shaul. A bi-level language for software process modeling. In Walter F. Tichy, editor, *Configuration Management*, number 2 in Trends in Software, chapter 2, pages 39–72. John Wiley & Sons, 1994. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-016-92.ps.Z.

[40] Simon M. Kaplan, William J. Tolone, Alan M. Carroll, Douglas P. Bogia, and Celsina Bignoli. Supporting collaborative software development with ConversationBuilder. In Herbert Weber, editor, *5th ACM SIGSOFT Symposium on Software Development Environments*, pages 11–20, Tyson's Corner VA, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992.

[41] Henry F. Korth. Extending the scope of relational languages. *IEEE Software*, 3(1):19–28, January 1986.

[42] Henry F. Korth. The double life of the transaction abstraction: Fundamental principles and evolving system concepts. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *21st International Conference on Very Large Data Bases*, pages 2–6, Zurich, Switzerland, September 1995. Invited paper.

[43] Balachander Krishnamurthy and Naser S. Barghouti. Provence: A process visualization and enactment environment. In Ian Sommerville and Manfred Paul, editors, *4th European Conference on Software Engineering*, volume 717 of *Lecture Notes in Computer Science*, pages 151–160. Springer-Verlag, Garmisch-Partenkirchen, Germany, September 1993.

[44] Programming Systems Lab. *Darkover Manual*, July 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/oz.1.2.manuals/IV.Darkover_API/.

[45] Jintae Lee, Gregg Yost, and the PIF Working Group. The PIF process interchange format and framework, December 1994. http://www-sloan.mit.edu/ccs/pifmain.html.

[46] Wenke Lee, Gail E. Kaiser, Paul D. Clayton, and Eric H. Sherman. OzCare: A workflow automation system for care plans. In James J. Cimino, editor, *1996 American Medical Informatics Association Annual Fall Symposium*, pages 577–581, Washington DC, October 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-012-96.ps.Z.

[47] Workflow Management Coalition Members. Coalition overview, September 1995. http://www.aiai.ed.ac.uk/WfMC/overview.html.

[48] Michael Baentsch, Georg Molter and Peter Sturm. WebMake: Integrating distributed software development in a structure-enhanced Web. In *3rd International World-Wide Web Conference*, Darmstadt, Germany, April 1995. Elsevier Science B.V. http://www.igd.fhg.de/www/www95/proceedings/papers/-51/WebMake/WebMake.html.

[49] Erich Neuhold and Michael Stonebraker (editors). Future directions in DBMS research. *SIGMOD Record*, 18(1):17–26, March 1989.

[50] Object Management Group. http://www.omg.org/.

[51] The common object request broker: Architecture specification revision 2.0. Technical Report ptc/96-08-04, Object Management Group, July 1996. http://www.omg.org/corba/corbiiop.htm.

[52] Leon J. Osterweil. Presentation at Software Process Architectures Workshop, March 1995.

[53] Burkhard Peuschel and Stefan Wolf. Architectural support for distributed process centered software development environments. In Wilhelm Schäfer, editor, *8th International Software Process Workshop: State of the Practice in Process Technology*, pages 126–128, Wadern, Germany, March 1993. Position paper.

[54] Steven S. Popovich. *An Architecture for Extensible Workflow Process Servers*. PhD thesis, Columbia University Department of Computer Science, January 1997. CUCS-014-96. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-014-96.ps.gz.

[55] Steven S. Popovich and Gail E. Kaiser. Integrating an existing environment with a rule-based process server. Technical Report CUCS-004-95, Columbia University Department of Computer Science, August 1995. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-004-95.ps.Z.

[56] Programming Systems Laboratory. *Oz 1.2 Manual Set*, July 1996. Columbia University Department of Computer Science. ftp.psl.cs.columbia.edu/pub/psl/oz.1.2.manuals.

[57] Calton Pu. Superdatabases for composition of heterogeneous databases. In Amar Gupta, editor, *Integration of Information Systems: Bridging Heterogeneous Databases*, pages 150–157. IEEE Press, 1989. Also appeared in *4th International Conference on Data Engineering*, Los Angeles CA, 1988.

[58] Sudha Ram, editor. *Special Issue on Heterogeneous Distributed Database Systems*, volume 24:12 of *Computer*. IEEE Computer Society Press, December 1991.

[59] Steven P. Reiss. *THE FIELD PROGRAMMING ENVIRONMENT: A Friendly Integrated Environment for Learning and Development*. Kluwer Academic Publishers, Boston, 1995.

[60] Wilhelm Schäfer, editor. *4th International Conference on the Software Process*, Brighton, UK, December 1996. IEEE Computer Society Press.

[61] Wilhelm Schäfer, Burkhard Peuschel, and Stefan Wolf. A knowledge-based software development environment supporting cooperative work. *International Journal on Software Engineering & Knowledge Engineering*, 2(1):79–106, March 1992.

[62] Friedemann Schwenkreis. Workflow for the German Federal Government. In *NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*, pages 64–68, Athens GA, May 1996. Position paper.

[63] Amit Sheth, Dimitrios Georgakopoulos, Stef M.M. Joosten, Marek Rusinkiewicz, Walt Scacchi, Jack Wileden, and Alexander Wolf. Report from the NSF workshop on workflow and process automation in information systems. Technical Report UGA-CS-TR-96-003, University of Georgia Department of Computer Science, October 1996. http://lsdis.cs.uga.edu/activities/NSF-workflow/final-report-cover.html/test.html.

[64] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[65] Izhar Shy, Richard Taylor, and Leon Osterweil. A metaphor and a conceptual framework for software development environments. In Fred Long, editor, *Software Engineering Environments International Workshop on Environments*, volume 467 of *Lecture Notes in Computer Science*, pages 77–97, Chinon, France, September 1989. Springer-Verlag.

[66] Peter D. Skopp. Low bandwidth operation in a multi-user software development environment. Master's thesis, Columbia University Department of Computer Science, December 1995. CUCS-035-95. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-035-95.ps.Z.

[67] Peter D. Skopp and Gail E. Kaiser. Disconnected operation in a multi-user software development environment. In Bharat Bhargava, editor, *IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 146–151, Princeton NJ, October 1993. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-026-93.ps.Z.

[68] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. APPL/A: A language for software process programming. *ACM Transactions on Software Engineering and Methodology*, 4(3):221–286, July 1995.

[69] Andrew Z. Tong, Gail E. Kaiser, and Steven S. Popovich. A flexible rule-chaining engine for process-based software engineering. In *9th Knowledge-Based Software Engineering Conference*, pages 79–88, Monterey CA, September 1994. IEEE Computer Society Press. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-005-94.ps.Z.

[70] Giuseppe Valetto and Gail E. Kaiser. Enveloping sophisticated tools into process-centered environments. *Journal of Automated Software Engineering*, 3:309–345, 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-022-95.ps.gz.

[71] Kurt Wallnau, Fred Long, and Anthony Earl. Toward a distributed, mediated architecture for workflow management. In *NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*, pages 74–84, Athens GA, May 1996. Position paper.

[72] Jeanine Weissenfels, Dirk Wodtke, Gerhard Weikum, and Angelika Kotz-Dittrich. The Mentor architecture for enterprise-wide workflow management. In *NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*, pages 69–73, Athens GA, May 1996. Position paper.