Towards a Platform for Distributed Application Development

Gustavo Alonso, Claus Hagen, Hans-Jörg Schek, and Markus Tresch

Database Research Group
Institute of Information Systems
ETH Zentrum, Zürich CH-8092, Switzerland
E-mail: {alonso,hagen,schek,tresch}@inf.ethz.ch

Summary. This paper describes the architecture of a generic platform for building distributed systems over stand alone applications. The proposed platform integrates ideas and technology from areas such as distributed and parallel databases, transaction processing systems, and workflow management. The main contribution of this research effort is to propose a "kernel" system providing the "essentials" for distributed processing and to show the important role database technology may play in supporting such functionality. These include a powerful process management environment, created as a generalization of workflow ideas and incorporating transactional notions such as spheres of isolation, atomicity, and persistence and a transactional engine enforcing these "quality guarantees" based on the nested and multi-level models. It also includes a tool-kit providing externalized database functionality enabling physical database design over heterogeneous data repositories. The potential of the proposed platform is demonstrated by several concrete applications currently being developed.

1. Introduction

Hardware architectures for information systems seem to be evolving towards infrastructures based on multiple stand alone computers linked by a network. From a research point of view, such clusters are interesting as platforms for implementing truly distributed systems. From a practical standpoint, the problem is how to build something coherent out of systems that were not necessarily designed to work together [COR95a, Hol96, BN97]: existing products tend to be unsatisfactory because they are, in most cases, only partial solutions to a fairly general problem. For instance, CORBA [COR95a] needs the transactional services a TP-monitor provides [BN97, Obe94]; a TP-monitor could greatly benefit from the standard interface defined by CORBA; both TP-monitors and CORBA implementations need a workflow tool to help specifying complex sequences of interactions between the different system components [Hsu95]; distributed execution over the internet needs transactional guarantees [Le97]; and so forth. The problem is not the lack of solutions but the lack of integrated solutions. To address this issue, this paper describes a basic kernel integrating ideas and technology from distributed and parallel databases, transaction processing systems, and workflow management. While the paper describes ongoing work, a prototype being developed, and

some preliminary results, its main contribution is to propose a set of "essential features" in a distributed system and show the important role database technology may play in supporting such functionality.

The kernel features considered in here are *process management*, *execution guarantees*, and *exported database functionality*. Process management refers to the need to have an adequate environment in which to express and execute coarse distributed computations. To this end, OPERA, a kernel process management system, is being developed. OPERA generalizes many ideas from workflow management related to both transaction processing [BN97, GR93] and business re-engineering [AAEM97, AAEM97, AS96]. It provides a sophisticated exception handling mechanism, incorporating transactional notions such as spheres of isolation, atomicity, and persistence. Regarding execution guarantees, we believe with [GR93] and [BN97] that transactions should be one of the basic building blocks for distributed applications. Hence, as a first step, we have developed a reformulation of nested and multi-level transactions geared towards exploiting the inherent parallelism of distributed applications. These ideas are currently being incorporated into OPERA to form the core of the execution guarantees it provides. Additional database functionality is incorporated into OPERA in the form of query management and physical database design. In this point, OPERA has taken advantage of the work done within the CONCERT project. Both CONCERT [BRS96] and composite transactions [ABFS97, AVA+94] have been described elsewhere, thus, in this paper we concentrate on OPERA and show how it integrates all these ideas into a coherent system by providing three application examples, one of them used as the running example to motivate the discussion.

The paper is organized as follows. Section 2 motivates the paper with an example. Section 3 presents OPERA, its general goals, architecture, and most relevant aspects. Section 4 discusses a number of application areas in which OPERA is being used by extending and tailoring the basic system with additional functionality. These application areas include higher level object management, and a distributed object manager for heterogeneous data repositories. Section 5 provides additional information regarding the status of the project and concludes the paper.

2. Motivation

One of the basic platforms in which to implement generic multiprocessor systems is commodity hardware and software, usually in the form of clusters of workstations connected via a network. In such environments, scalability and reliability are ideally only limited by the number of elements in the system. They also have the advantage that most of the necessary infrastructure is already in place.

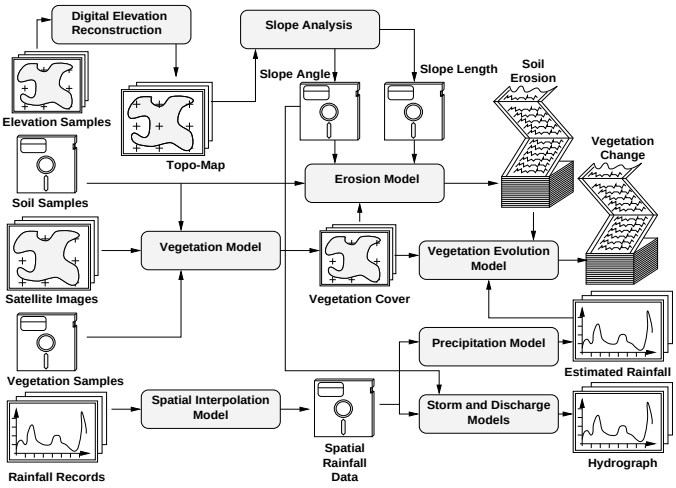

Fig. 2.1. Example of an earth sciences process

2.1 Application Example

As a running example of such environments, and to use one not directly related to traditional databases, consider earth sciences processes [SSE+95, SW93, HGM93], a concrete instance of which is shown in Figure 2.1 [AH97, AE94]. The interesting aspect of this example is that it shows how scientific data is handled, i.e., it is often subject to multiple transformations involving different applications and platforms. For instance, in the example shown, the purpose is to study the changes in the erosion patterns, vegetation and hydrographic characteristics of a given area. It is basically a series of transformations over base and derived data that, taken together, represent an interpretation of several interrelated geographic phenomena (erosion, rain storms, flooding). These processes are executed step-wise using a variety of GIS applications over a cluster of computers. Execution is controlled, at best, through scripting languages or, more often, manually. Any logging, accounting, indexing, classification, record keeping, or failure recovery is done, if at all, manually. These are, however, areas in which databases excel and that open up interesting opportunities for using databases in distributed environments. For instance, they could be used to support distributed computations in the same way that today's databases are used to support traditional data management. In other words, database technology could form the core of a "distributed operating system" facilitating the integration of independent systems into a single coherent whole. Some of this functionality (Figure 2.2) includes persistent execution with the database acting as a sophisticated data repository. This will certainly increase performance in environments in which many processes are run concurrently and in which it is necessary to provide sophisticated querying facilities to keep track of events in the system

and to analyze the characteristics of already executed processes. Basic functionality also includes transactional guarantees, not only regarding atomic commitment, but also issues like forward and backward navigation using the database as a sophisticated log, high level compensation, and process synchronization. This immediately raises the question of which language to use to express the desired behavior and execution semantics. Databases can also provide much needed availability and resilience to failures, as well as helping in providing reasonable performance in complex distributed environments by separating the execution data from automatic record keeping, logging and auditing data. All these ideas indicate the important role databases could play in executing distributed processes like the one in Figure 2.1.

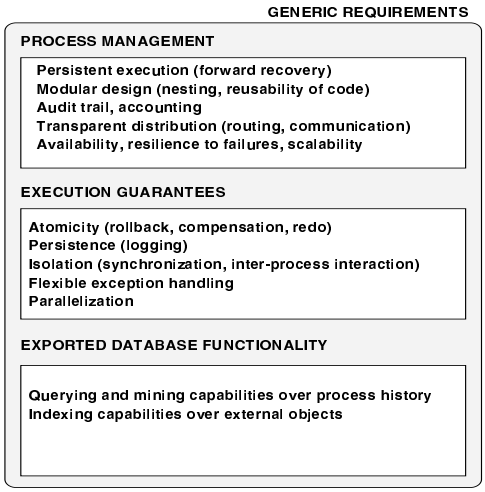

Fig. 2.2. Functionality necessary to support distributed process execution

2.2 Software Solutions

Existing solutions to distributed execution are commonly grouped under the name of *middleware*. Middleware products related to databases are, for instance, federated and multi-databases systems [SSW95, DSW94], TP-monitors [Obe94], persistent queuing systems [MH94], CORBA implementations [COR95a], workflow management systems [Hsu95, Hsu93], and process centered environments for software engineering [TKP94, BK94]. They can support the process in the example in a variety of ways. For instance, a CORBA implementation would impose an object oriented interface over each application, and the process would then be coded in C++. A TP-monitor would treat each separate step as a service provided by one or more servers and the process would be coded as a series of service invocations with some transactional semantics. Each of these approaches has advantages and

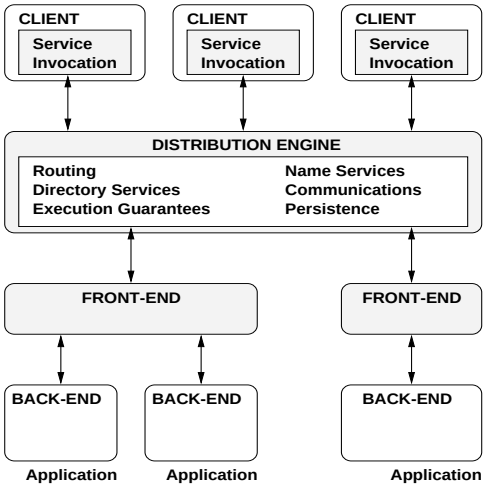

Fig. 2.3. Generic architecture of a middleware tool as a 3 -tier architecture

disadvantages, but none of them provides an integrated solution and there is always functionality missing. The main hypothesis behind this paper is that middleware systems have a lot in common. This shared functionality is mainly database related and, therefore, it should be possible to design a kernel system to provide it. As a first approach, the kernel can be based on the common aspects of existing middleware systems. An analysis of their functionality clearly shows a common generic architecture: the traditional 3-tier organization (Figure 2.3) and the typical configuration within a cluster of workstations (Figure 2.4) [Cor95c, COR95a, Cor95b, IBM95]. In both cases, there are four main components to consider: *Clients*, the *Distribution Engine*, *Front-Ends*, and *Back-Ends*.

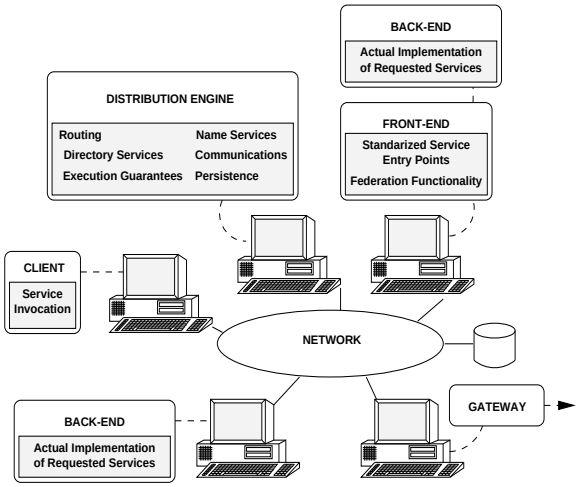

Fig. 2.4. Generic architecture of a middleware tool distributed in a cluster of workstations

Instance	Client	Distribution Engine	Front-End	Back-End
CORBA	Client	Object Request Broker	Object Adaptor	Server
Workflow Systems	Worklist Manager	Workflow Engine	Application Interface	Activity / Task
TP-Monitors	Client	TP-monitor	Server	Resource Manager
Queuing Systems	Client Application	Queue Server	Client Application	Client Application
Federated Databases	Client	Global Transaction Manager	Agent or Wrapper	Local Database

Fig. 2.5. Mapping between the generic components and the notations of several instances of middleware

The functionality of these components is as follows. The *client* interacts with the system through *service invocations*. The service invocations are handled by the *distribution engine*, which forms the core of the system. The distribution engine determines the nature of the system since it provides the main functionality: load balancing, routing, communication handling, protocol translation, name and directory services, system administration, client interaction, and so forth. The distribution engine passes the service requests to the *front-ends*, which provide a standarized entry point and generic interface to the *back-ends*, i.e., to the applications underlying the middleware system. It is a simple exercise to map these generic architecture to concrete products (Figure 2.5). With this generic architecture in place, the next step is to determine the functionality the kernel should provide.

2.3 Kernel Functionality for Distributed Systems

We are particularly interested in the aspects in a distributed system that can be supported by database technology. In particular, and as pointed out above, there are three aspects we consider "essential functionality": *process management*, *execution guarantees* and and *exported database functionality*. The reason to focus on these three aspects comes from our observations of current trends in existing products, which reveal a high degree of affinity among middleware systems. For instance, workflow systems and CORBA environments are slowly being merged under the notion of business objects (for process management), TP-monitors are being used to provide the execution guarantees in CORBA, and queuing systems are being added to workflow management systems to increase reliability (exported database functionality). By exploiting this affinity, we aim at designing a generic kernel providing the following "essentials":

Process management. Distributed computation is to be based on the concept of process. Processes are arbitrary sequences of application invocations over different locations and platforms. Hence, the main hardware infrastructure behind the system is a cluster of workstations used as a shared nothing multi-processor environment in which the kernel plays the roles of scheduler and resource allocator. The kernel should provide sufficient reliability, availability and scalability and, for this purpose, should take advantage of the underlying hardware platform to distribute its functionality. Ideally, all the components could be moved from node to node to enhance the overall robustness, availability, and scalability. For instance, in the example of Figure 2.1, if a node in which a step is usually executed is not available, it may be possible to run the same step at a different location. This should happen transparently so as to allow the system to dynamically adjust itself to configuration changes due to failures or scheduled shut-downs. Similarly, if a failure occurs midway during the execution of the process, it should not be necessary to restart executing from the beginning when the failure is repaired. Intermediate results should have been stored and execution resumed at the point where it was left off when the failure occurred. Moreover, in some cases it should even be possible to resume execution before the failure is repaired by using a reliable backup strategy. All this, of course, must be provided with reasonable performance and without introducing significant overheads.

Execution Guarantees. The kernel should guarantee correct results in spite of the fact that the execution is concurrent and involves autonomous and often uncooperative systems. Correctness includes concepts such as atomicity, "exactly once" semantics, concurrency control, recoverability, etc. Thus, the kernel should place a significant emphasis on the transactional aspects of distributed computations. It should provide not only language constructs to incorporate transactional notions into processes (spheres of isolation, atomicity, and persistence), but also support for composite transactional interactions [ABFS97] among the different components of the distributed system as a way of guaranteeing overall correctness. In addition, mechanisms are needed to exploit as much as possible the inherent parallelism of processes like the one in Figure 2.1.

Externalized Database Functionality. One of the main problems of today's databases is that they behave like black boxes. Their services are available only to the data that resides within the database. In practice, however, most data does not reside within databases and applications using this data must implement their own database services. This is certainly the case in many applications running over clusters of workstations. Part of the basic support the kernel should provide is database functionality such as indexing and query processing [BRS96]. The advantage of exporting database functionality is that it provides a very powerful mechanism to interact with data and applications residing outside of the system. It not only alleviates the task of writing new applications (since there is no need to incorporate this

database functionality in them) but it also establishes the basis for keeping track of the many elements involved in distributed environments (both data and applications).

3. The OPERA Kernel

OPERA is being designed as a kernel providing the core functionality described above. This kernel will be extended to build distributed solutions such as shared nothing parallel database systems, experiment management environments, or distributed object management engines (see below).

3.1 Architecture

The architecture of OPERA is organized around three service layers (Figure 3.1): *database services*, *process services* and *interface services*. The database service layer acts as the storage manager. It encompasses the storage layer (the actual databases used as repositories) and the database abstraction layer (which makes the rest of the system database independent). The storage layer is divided into five *spaces*: template, instance, object, history, and configuration, each of them dedicated to a different type of system data. Templates contain the structure of the processes. When a process is to be executed, a copy of the corresponding template is made and placed in the instance space. This copy is used to record the process' state as execution proceeds. For each running instance of a process the instance space contains a copy of the corresponding template. Storing instances persistently guarantees forward recoverability, i.e., execution can be resumed as soon as the failure is repaired, which solves the problem of dealing with failures of long lived processes [ST96, DHL91]. Instances also constitute the basic unit for operations related to process migration and backup facilities [KAGM96]. Objects are used to store information about externally defined data. They allow OPERA to interact with external applications by acting as a proxy containing the information indicating how to access external data [BRS96, AH97]. The history space is used to store information about already executed instances. It contains a detailed record of all the events that have taken place during the execution of processes, including already terminated processes. Finally, the configuration space is used to record system related information such as configuration, access permissions, registered users, internet addresses, program locations, and so forth.

The database abstraction layer implements the mechanisms necessary to make the system database independent. The experience with workflow systems shows that this is a crucial issue affecting scalability and the overall openness of the system [AAEM97]. Hence, OPERA uses internally a canonical representation [KAGM96] optimized for performance and expressibility.

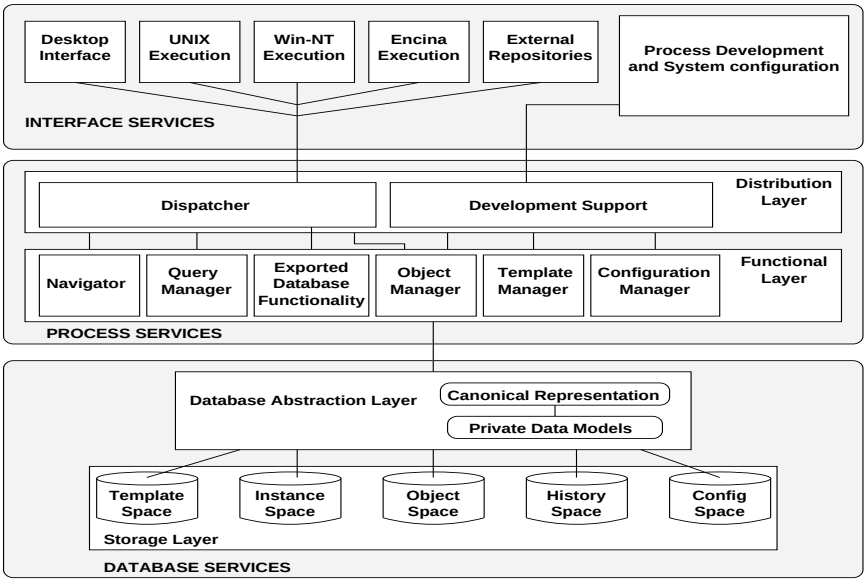

Fig. 3.1. System architecture of OPERA

This canonical representation is not suitable, however, for either commercial databases or user interaction. The database abstraction layer translates the canonical representation to the private representations of the underlying repositories (SQL, C++, system calls) as required by the physical implementation of the underlying database (Figure 3.2).

The process service layer contains all the components required for coordinating and monitoring the execution of processes. The most relevant components for the purposes of this paper are the *dispatcher*, the *navigator*, the *object manager*, the *query manager*, and the *exported database functionality* module. The dispatcher and the development support constitute the distribution layer, which deals with physical distribution. The dispatcher acts as resource allocator for process execution. It determines in which system the next step will execute, locates suitable nodes, checks the site's availability, performs load balancing, and manages the communication with remote system components. The development support performs similar tasks for process definition, forming the basis of OPERA's development environment. These two components are an example of the advantage of separating the database spaces. The development support module works mainly over the template space, the dispatcher mainly over the instance and configuration spaces. Separate spaces implies that these two components do not compete for the same resources.

The rest of the components form the functional layer, which implements most system's capabilities. The navigator acts as the overall scheduler: it "navigates" through the process description stored in the instance space,

establishing what to execute next, what needs to be delayed, and so forth. It enforces the transactional aspects of the execution. During navigation, the variables in the process instance are updated to keep track of every step taken in the execution of the process. When the process terminates, this information is moved to the history space to avoid interferences between process execution and history analysis. The system interacts with the data spaces through the query manager. The query manager provides a suitable interface for complex queries, which in many cases are standard. Instead of including them as part of all the modules where they are used, they are centralized in the query manager, which offers them as services to other components. We expect that most of these queries will be executed over the history space. In some applications this aspect of process support is heavily used, hence the advantage of separating the instance space, used to drive the execution of processes, and the history space, used for record keeping purposes.

Interaction with external objects takes place through the object manager and the exported database functionality module. The latter is is based on CONCERT [BRS96], a system designed to provide database functionality to data external to the database. The former is much more application dependent, acting as the repository for metadata information. Examples of this metadata include any dependency between external objects, lineages, versioning, etc.

Finally, the interface service layer encompasses all the mechanisms that allow OPERA to interact with applications in different hardware and software platforms. Users interact with the system via *desktop interfaces*, which are also used to inform the user of any activity that they need to execute as part of a process (like worklists in workflow engines). The interaction with external applications takes place through *execution interfaces* supporting several operating systems and specific tools. These execution interfaces communicate with the dispatcher, translating the commands sent by the dispatcher into the corresponding commands required to start an application, and notifying the dispatcher of the termination of the application.

3.2 Process Management: Model

The notion of process is central to OPERA. A process is a sequence of computer programs and data exchanges controlled by a meta-program (the process itself). Typical examples of processes are business processes, software processes, manufacturing processes, scientific experiments, and geographic modeling. The problem with existing process management systems is that they tend to focus on a particular type of process. The situation resembles in many ways that of databases before the adoption of the relational model. It is difficult to generalize individual solutions as proven by recent attempts to use commercial workflow products to support scientific applications [MVW96, BSR96]. Indeed, a generic notion of process can be compared

with SQL. SQL alleviates the task of data management by providing a universal interface to data management tools (the database). A generic notion of process would alleviate the task of defining the control flow between different applications by providing a common interface to process management tools. This is the idea, for instance, behind the joint efforts of the OMG (Object Management Group) and the Workflow Management Coalition to define a standard for process management in CORBA. Similarly, OPERA provides a generic way of representing arbitrary processes, as a first step towards becoming the engine behind many distributed applications. The crucial role process management could play in the design of server and client code is an example of the advantages of this approach. For the servers, it provides a mechanism to federate applications, as it happens today in workflow systems. This can greatly simplify the task of writing servers in the TP-monitor sense, which tends to be the most complex task from a designer's point of view. For the clients, process management opens up interesting new ways of interacting with the system. Processes can be seen as a sophisticated form of scripting, allowing the users to quickly build applications over distributed systems. Using the same concept for coding at both the client and the server blurs the distinction between them, allowing arbitrary nesting of systems, which we expect to become a very useful feature in practical applications.

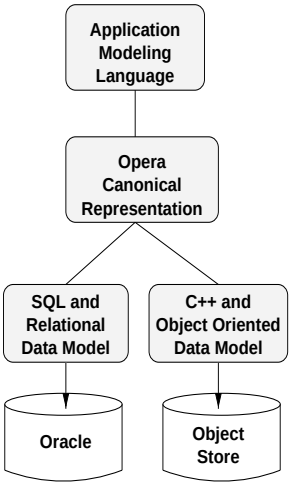

Fig. 3.2. The different language representations in OPERA

In order to provide a generic notion of process, OPERA contains a hierarchy of process representations rather than a single model (Figure 3.2). At the top of the hierarchy, and used at the interface service layer, is the application specific language. This is the language that can be customized to processes such as those shown in Figures 4.1 and 2.1. In most process support systems, this language has a strong graphic component. User representations, how-

ever, are not suitable for handling processes efficiently. Therefore, OPERA works internally using OCR (*Opera Canonical Representation*), which constitutes the second level of process representation. The third level appears when OCR is translated into the private representations of the underlying databases (currently ObjectStore and Oracle). The following are the most relevant components of OCR.

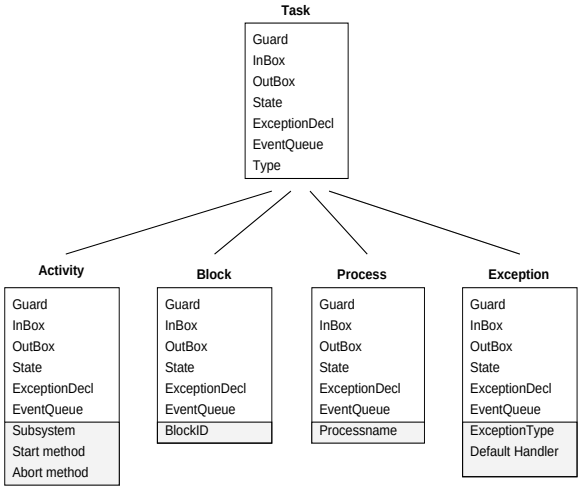

Fig. 3.3. Class hierarchy of tasks in OPERA

A *process* consists of a set of tasks and a set of data objects. Tasks can be activities, blocks, or processes. The task class hierarchy (Figure 3.3) also includes exceptions as a special type of internal tasks. The data objects store the input and output data of tasks and are used to pass information around. The connectors indicate the order in which the tasks are to be executed. To better understand how some of these notions fit together, Figure 3.4 shows how the example of Figure 2.1 as an OCR process (with some unavoidable simplifications for reasons of space, for instance, OCR does not implement connectors explicitly but implicitly).

Activities are the basic execution steps. In the example of Figure 2.1, each shaded box corresponds to an activity. An activity provides a *navigation interface* (Figure 3.3) to access information about its state, parameters, return values, raised events, possible exceptions, etc. This information is stored as part of the process template and is used during navigation. In addition, each activity has an *external binding*, which specifies the program(s) to be executed, users responsible for the execution, and/or resources to be allocated for its execution. This information is used by the dispatcher to execute external applications. The external binding is stored in the configuration space as several activities may use the same external program. The procedure to

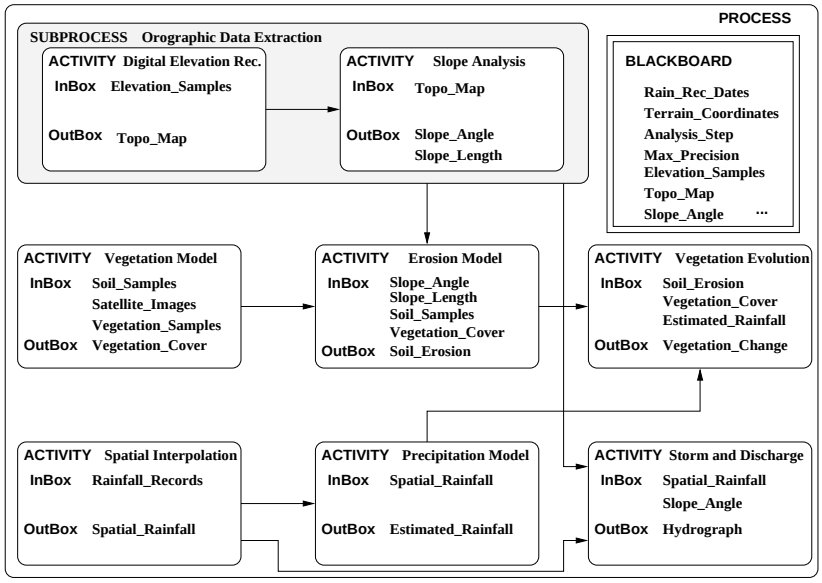

Fig. 3.4. The earth science process example in terms of OCR components

create the external binding is similar to that of registering programs in an operating system so they can be properly invoked.

Blocks are sub-processes defined only in the context of a process. They are used for two purposes, for modular design and as specialized language constructs such as loop blocks (for, do-until, while, fork), spheres of atomicity, spheres of isolation, or spheres of persistence.

Subprocesses are processes used as components of other processes. As an example, in Figure 3.4 two activities have been grouped into a subprocess called "Orographic Data Extraction". Subprocesses allow, like blocks, the hierarchical structuring of complex process structures. Late binding (the referenced process is read only when the sub-process is started) allows dynamic modifications of a running process by changing its sub-processes [IBM95].

Control flow inside a process is based on *guards* attached to each task. The guard concept borrows heavily from the ECA rule mechanism of active databases. A guard consists of an *event description* describing the process state(s) that activate the execution of a task. The task's execution can be restricted by an *activation condition* expressed as a predicate on output data of other tasks. Event descriptions can only refer to execution states and events raised by tasks within the same process. This helps to avoid many of the complexity problems associated to ECA rules. The reason to use guards comes from our experience in workflow systems. For instance, in the case of ObjectStore, it was determined that the cost of pointer chasing caused significant overheads and, therefore, an active mechanism was used (guards) instead of the traditional control connector approach (in which a control connector

implies several steps in a chain of pointers). In Figure 3.4 guards have been omitted for simplicity. Control flow is instead represented through connectors similar to those used in workflow engines.

Data flow is possible between tasks and between processes. Each task has an *input data structure* describing its input parameters and an *output data structure* to store any return values. The input parameters of a task can be linked to data items in the process' global data area (the blackboard) or in other task's output structures. When a task starts, these bindings are analyzed and the necessary values passed to the task. After the successful execution of a task, a *mapping phase* transfers data from its output structure to the global data area. The input and output data structures of a process are part of the *blackboard* which acts as the global data area for the process (Figure 3.4).

Events: OCR incorporates an event mechanism that allows the communication between processes as well as the externalization of intermediate results of activities. Processes and activities must declare the *events* they may signal during their execution. Processes subscribe to these events, with the process engine behaving as a broker that notifies subscribers of relevant. The guard of a task can also contain references to events, allowing to make the flow of control dependent on intermediate states of other processes or activities. Events are parameterized, which facilitates exchanging data between processes. For automatic activities, the OPERA API provides a call to signal events. Users may also raise events through the desktop interface.

Exceptions: Exceptions are raised by tasks when unexpected situations occur, or when external intervention is needed to either decide on the further flow of control or to change data values passed to a task [HA97]. Exceptions serve as a unique mechanism to parameterize the behavior of processes. The declaration of possibly exceptions is part of the navigation interface. A task defines default handlers for each exception it throws. A calling process can modify the behavior of a task by specifying *override handlers* that replace the predefined handlers. This mechanisms provides a very powerful way of coping with exceptions, one of the most pervasive problems of process management. Details of the exception handling mechanisms in OPERA have been published elsewhere [HA97] and they will not be discussed further in here.

3.3 Transactional Execution Guarantees

The transactional aspects of OPERA are embedded in the notion of *spheres* and in the scheduling performed by the navigator. The former addresses the problem of providing a way of bracketing operations as units with transactional properties. The latter forms the basis for actual transactional scheduling.

The concept of ACID transactions [GR93] has been very successful in databases. But once transactions are used outside the database, there is a

strong need for *light-weight* transactions in which some of the ACID properties are not enforced [BRS96]. In particular, in the case of OPERA processes, OCR tries to avoid a unique construct encompassing all properties, e.g., *BOT/EOT*. Instead, blocks are used to group tasks according to the desired semantics. From a transactional point of view, there are currently three possibilities (and combinations thereof): blocks as atomic units with the standard all or nothing semantics (sphere of atomicity), blocks as isolation units (sphere of isolation), or blocks as persistence units (spheres of persistence). In this regard, OPERA borrows heavily from existing work on advanced transaction models [Elm92, Kle91, CR91, BDG+94].

Regarding atomicity, the information about the properties of the corresponding application must be provided by the user when the activity is registered. There are several options when selecting a task interface: *Basic* (non-atomic), *Semi-atomic*, *Atomic*, *Restartable*, and *Compensatable*. Basic tasks are the default and correspond to non-atomic applications, i.e., those for which OPERA cannot guarantee atomicity. Semi-atomic tasks are those providing enough information to implement a rollback method to be executed if the task fails before completing its execution. Atomic tasks are those that preserve atomicity by themselves, for instance, a transaction executed over an X/Open XA interface. Restartable tasks [ELLR90] are those that can be invoked repeatedly until they eventually succeed. Compensatable tasks are those that can be undone after they have finished using an user provided method attached to the task interface[ELLR90]. Note that these categories apply to activities, blocks, and sub-processes. Thus, it is possible to group a set of tasks into a semi-atomic block, for instance, or provide high level compensation for an entire sub-process by declaring it to be compensatable.

Spheres of persistence are used to avoid the overhead incurred by storing all process information in the instance space. The size of a process and the significance of the delay due to I/O while accessing the instance space is largely application dependent. When a task or a group of tasks is embedded within a sphere of persistence, every step of the execution is recorded in the instance space. This guarantees forward recovery in the event of failures. By default, all tasks are embedded within a sphere of persistence. It is possible, however, to switch this option off, in which case the information about the execution is maintained only in main memory. Upon completion, this information is stored in the instance or the history space for record keeping purposes but this is done off-line, thereby avoiding the I/O overhead.

The semantics behind the notion of spheres of isolation follow the ideas suggested in [AAE96], which point out that processes may require more a notion of synchronization in the traditional operating systems sense than the traditional database concept of serializability. But there are also applications that clearly demand a database like approach (see below the description of HLOM). For these cases, spheres of isolation and spheres of atomicity are used in the scheduling performed by the navigator. Note that processes are essen-

tially nested structures. This is best captured by using a nested transaction or multilevel model [Wei91, Mos81], which provides a powerful mechanism to reason about recovery in applications with a complex structure [GR93]. A similar conclusion has been reached in several instances of middleware [BN97, Cor95c, CD96, SSW95]. Thus, the navigator in OPERA is based on recent work that extends the existing notions of nested and multilevel transactions and applies them to *composite systems* [ABFS97]. The goal is to exploit the parallelism inherent in processes, guaranteeing at the same time that the order of execution specified by the process designer is respected. For this purpose, OPERA allows to label the control flow between two tasks as a *strong ordering* or a *weak ordering*. The strong ordering forces OPERA to match the externally derived ordering when scheduling tasks. This order reflects external dependencies that OPERA cannot ignore. The weak order, however, is used to indicate a particular order that becomes meaningful when tasks actually conflict. When conflicts do not exists, the weak order can be ignored. The advantage of differentiating between the strong and weak orderings is that OPERA has yet another degree of freedom to parallelize the execution of processes.

From a practical point of view, the theory of composite systems has the following implications: conventional parallel programming gives the choice of executing operations sequentially or in parallel ("do in parallel"). The mechanisms described allow to add a third possibility, "do in parallel in a given order", which executes operations in parallel but preserves the order in terms of externally observable effects. The OPERA navigator then becomes yet another layer on a hierarchy of schedules enforcing *stack conflict consistency* as explained in [ABFS97].

3.4 Availability and Scalability

Unlike existing workflow management systems, OPERA provides support for generic processes. Business processes are a very special case in which issues like isolation or performance do not have significant role. In the same way that OPERA provides isolation capabilities for those processes that may need them, a significant effort has been made in OPERA to make it a robust and scalable system although this may not be a big issue in some cases [AAEM97, AS96]. Note that OPERA is to be interpreted more as a distributed operating system than as a workflow engine.

In terms of both scalability and availability, process support systems relying on a single database have a clear scalability limit and a single point of failure [KAGM96]. To avoid these problems, OPERA relies on OCR to make its operations database independent. This has the advantage that several databases can be used at the same time and there is no need for them to be the same. For instance, the current prototype can use Oracle and Object Store simultaneously. OCR also allows the separation of spaces, which is probable one of the most significant factors when performance and scalability

is considered. In a large system, it is not feasible to install the five spaces in a single machine. This would result in clear competition for resources. Since the five spaces are in practice orthogonal to each other, they can be located in different nodes. In this way, operations related to data mining and audit trail analysis (very significant in business and scientific processes) and operations related to execution use different databases. This reduces the system overhead but, as a result, certain non-key operations become more expensive. For instance, when a process is started, a copy has to be made from the template space into the instance space. If they are in different locations, this takes some time. For the cases in which this may become a problem, it is possible to prefetch process templates in advance.

As a further step towards enhancing scalability, OPERA allows to have several navigators working on different instance spaces. Such sub-systems share all the static information (template, history, object, and configuration) and may even share the same dispatcher (if the dispatcher becomes a bottleneck, several of them may also be run in parallel at the locations where the navigators are running). This opens up the possibility for OPERA to perform automatic load balancing by spawning new navigators and new instance spaces at other sites as the load increases. Several OPERA systems can also be interconnected to form a larger system.

In terms of availability, OPERA follows the suggestions of [KAGM96]. The user can chose among three levels of availability for each process. The highest level of availability (*critical*), guarantees a hot-standby, 2-safe backup. If the primary system fails, the backup can take over immediately. The intermediate level of availability (*important*) guarantees a cold-standby, 2-safe backup. After a failure at the primary, execution can be resumed at the backup once the state of the process is brought up to date. It is also possible to run processes as *normal* processes, in which case no backup mechanism is used (but the process is still persistent, thereby allowing to resume execution as soon as the failure is repaired). These same mechanisms can be used to implement dynamic process migration.

The mechanisms used for backup and process migration have already been implemented and are on the testing phase. These tests will serve as a sort of validation of the OPERA concept as its generality and validity depend heavily on its ability to provide reasonable performance, scalability and availability. They will also help to optimize the design and detect potential bottlenecks. In particular, we are specially interested on possibility of using the backup mechanism to dynamically migrate processes from node to node, a feature that will considerably enhance the load balancing capabilities of OPERA as well as its overall scalability. Preliminary results look promising. Figures 3.5, 3.6, 3.7, and 3.8 show the cost associated with starting an activity, terminate an activity, starting a process, and the relative cost of these three operations for processes executed with availability level of normal, respectively. As far as current experiments show, the cost is quite reasonable. The most expen-

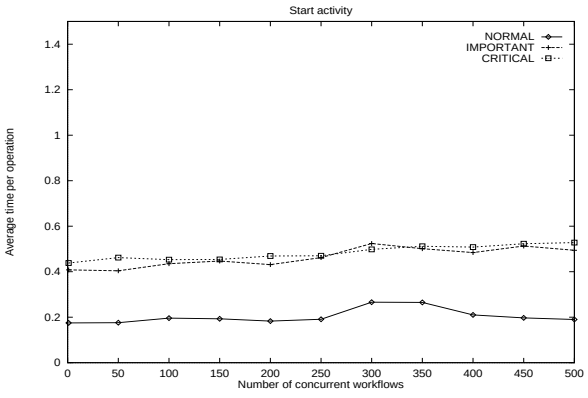

Fig. 3.5. Delay (in seconds) incurred when starting an activity.

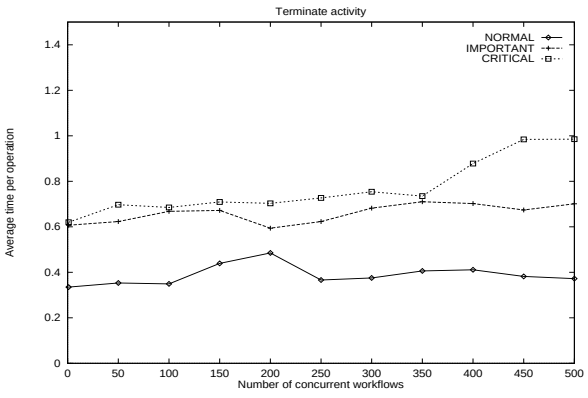

Fig. 3.6. Delay (in seconds) incurred when terminating an activity.

sive operation is starting a process, since this implies making a copy of the template and installing it in the instance space, in addition to performing the initial navigation steps. For applications in which the process duration is comparable to the figures shown, several optimizations are possible. For instance, the idea of providing spheres of persistence arises from the difficulty of providing reasonable response times if the process is kept in the instance space and each navigation step involves several access to the disk. The overhead incurred in these cases is not significant in business processes, but it will certainly have a considerable impact on applications like the ones described below. This will also reduce the time it takes to start a process. Another possible optimization is to keep the template in main memory to avoid having to copy it from the template space when the process is started. Similarly, the cost of terminating an activity can be reduced by performing navigation in main memory, without having to access the instance space. Note that terminating an activity involves determining what is to be executed next, a fairly complex operation requiring to evaluate several guards. A number of other

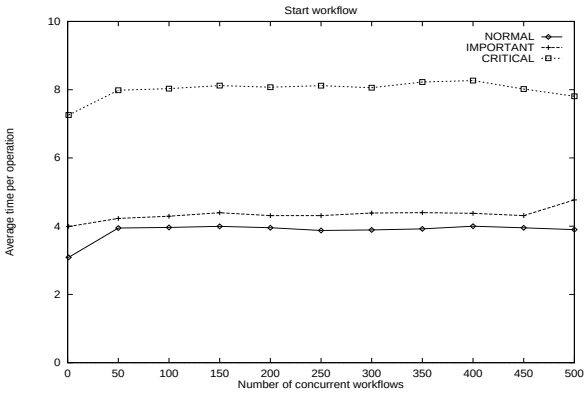

Fig. 3.7. Delay (in seconds) incurred when starting a process.

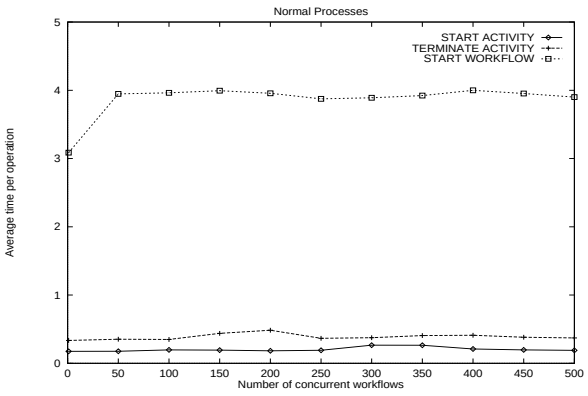

Fig. 3.8. Delay (in seconds) incurred when (a) starting an activity; (b) terminating an activity; (c) starting a process, shown for a process executed at availability level normal.

similar optimizations are possible, which gives us enough confidence in being able to improve those figures significantly.

3.5 Externalized Database Functionality

The advantages of the externalized database functionality provided by OPERA are best shown with a concrete application. Geo-Opera [AH97] is an extension of OPERA for earth sciences processes. The most relevant aspect of Geo-Opera is the use it makes of OPERA functionality to track data dependencies. A process like the one in the example produces derived data that cannot be interpreted without knowledge about the process itself [Arm88, Lan88, LV90]. This is related to the problems of lineage-tracking, change propagation and versioning [GG89, Rad91, SSAE93]. Queries such as "which models use algorithm X", "which results may change if dataset Y is updated", and "which data sets are used to derive result Z" are typical of such environments. The mechanisms provided to deal with such queries are implemented as extensions to the exported database functionality module and the object manager.

In the object space, each external dataset is represented as an object. As shown in Figure 3.9, a number of attributes in each object allow maintaining detailed information about existing versions, lineage, and usage [AH97, AE94]. This information is updated by the navigator every time one of the objects is used, thereby creating a record of all possible dependencies between objects. As we foresee systems handling a large number of objects and processes, the effective management of the object space requires to be able to build indexes over this information. This, as well as building indexes over information externally stored, can be done using the exported database functionality module as explained in [BRS96]. As an example of the functionality that can be provided, consider the active mechanisms implemented in Geo-Opera: *active objects* and *active tasks*. Active objects are automatically recomputed if some object they depend on has been modified. For instance, in the process of Figure 2.1, if the object "Hydrograph" is declared as active, it will be recomputed whenever a new version of the object "Rainfall Records" is available. To avoid expensive checks, the update takes place in a lazy manner. Instead of doing a search of all related objects every time an object is modified, the update takes place when an object is accessed. Detecting that an object needs to be recomputed is done by first calculating its lineage, which can be done by following the appropriate chain of pointers (Figure 3.9), and then looking for new versions of the objects found in the lineage, which is done by checking the corresponding attribute of each object in the lineage.

Similarly, active tasks are recomputed when some of their inputs are modified. This allows to get derived data automatically every time new input is available. As with active objects, the mechanism is recursive but, unlike active

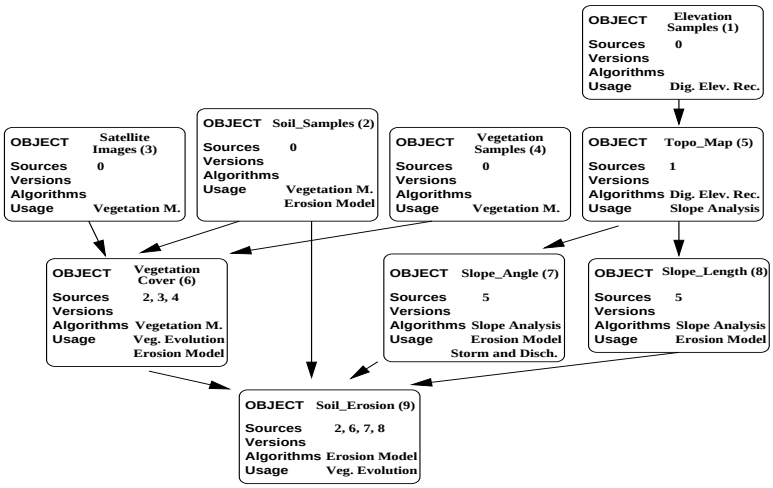

Fig. 3.9. Using the object space to keep track of dependencies among external datasets.

objects, is based on eager propagation, i.e., the task is recomputed every time one of the inputs changes. Thus, if the example of Figure 2.1 is defined as an active process, the entire process is recomputed whenever a new version of the object "Rainfall Records" is available, for instance. Note that the mechanism applies to tasks. This makes possible to apply the same idea to subprocesses and activities. Spheres of atomicity are used to avoid re-execution before the entire set of new input data is ready.

4. Application Examples

There are many examples in which functionality like the one provided by OPERA is needed. These include job scheduling in distributed systems, workflow applications, CORBA environments, OLAP and OLTP. The examples described below correspond to two research projects being carried out at ETH as extensions to the OPERA kernel.

4.1 High-Level Object Management (HLOM)

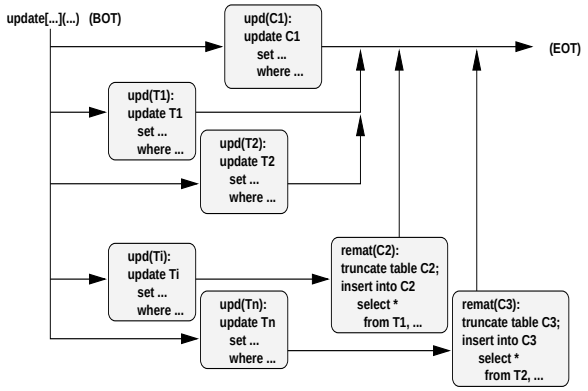

Fig. 4.1. Mapping of an object model operation to an OPERA process: execution dependencies on an update statement.

A first application of OPERA is within the HLOM project. OPERA treats clusters of workstations as heterogeneous shared-nothing multiprocessor platforms. HLOM is a research project in which higher-level parallelism is implemented over such clusters. Unlike other parallel database projects, the HLOM prototype is being built over stand-alone, off-the-shelf, commercial database systems which are treated as black boxes providing database services. Contrary to federated databases, HLOM is designed in a top-down manner, i.e., the designer has control over the data placement strategies, data partition, schema organization, etc. Some of the advantages of this approach have been demonstrated recently by implementing an object oriented

database on top of a relational system [RNS96]. In this preliminary work, the underlying database (Oracle) runs over a multiprocessor machine (Sun-Sparc Center 2000 with 10 processors). The operations specified in an object algebra (COOL) [TS94], are translated into SQL statements executable in the relational database. The available parallelism is exploited by, first, using a physical design in which tables are replicated, and, second, by transforming intra-transaction parallelism (within the object algebra operations) into inter-transaction parallelism (among the resulting SQL statements). Compensation and a multilevel scheduling is used to enhance parallelism and avoid retaining locks for the duration of a top level transaction.

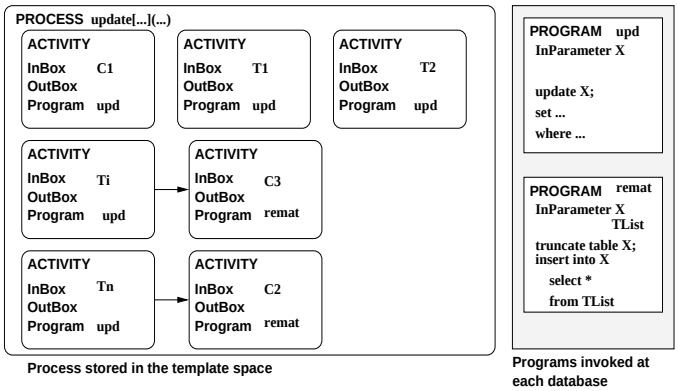

Fig. 4.2. Mapping of an object model operation to an OPERA process: the corresponding OPERA process.

HLOM is a generalization of this idea. It has been pointed out [Gra95], that one of the main performance limitations of parallel databases is the fact that the client interacting with the database is not a parallel application. Asynchronous SQL, for instance, is a step towards addressing such problem. An alternative approach is to consider the sequences of operation invocations as processes and let OPERA manage their execution. In other words, OPERA acts as the database scheduler and distribution engine for HLOM, which is built from a collection of stand-alone databases. Figures 4.1 and 4.2 show an example of how a generic update statement (borrowed from [RNS96]) can be expressed as an OPERA process (for simplicity the compensation mechanisms, also expressed in OCR have been omitted, for a more detailed explanation of how they can be used see [HA97]). Since the designer has control over the data placement and schema distribution, the execution plan for each COOL statement can be predetermined. Thus, in the same way that in [RNS96] COOL statements are translated into several Oracle transactions, in HLOM each COOL statement is mapped to a process. When a COOL statement is issued, it is parsed to determine the objects being ac-

cessed. This information is used as input parameters to the process, which will forward it to each individual task. Each task learns about which tables are to be accessed by consulting its InBox. The actual execution of each step takes place by invocating generic programs that can be parameterized. These can be implemented in a variety of ways, for instance, as stored procedures defined over each of the participating databases. A rough outline of how these programs look like is shown in Figure 4.2.

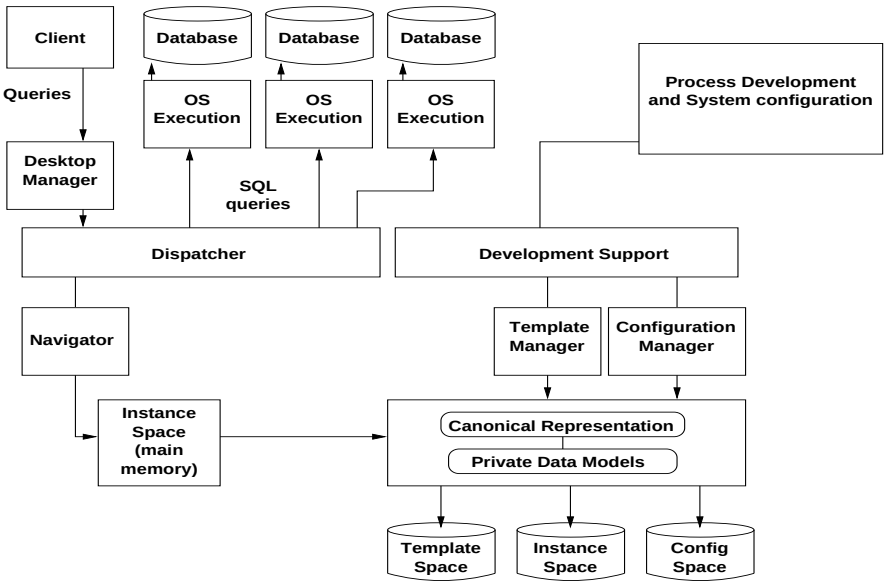

Fig. 4.3. Extensions and modifications to OPERA for HLOM

For the purposes of HLOM, OPERA is extended as indicated in Figure 4.3 (the OPERA components not represented are optional in HLOM). The desktop manager is converted into an interface for external clients (either as a console, or as an API). Clients submit queries through the desktop manager, who parses them and transforms them into process invocations, which are forwarded to the navigator. The query language (COOL in the case of [RNS96], asynchronous SQL in future versions of HLOM) becomes the Application Modeling Language. The translation of the Application Modeling Language into OCR is performed using the same compiler and scheduler as in the centralized case [RNS96]. Each resulting SQL statement is considered as a task, and each object statement is mapped to a process encapsulated within a single sphere of atomicity. Since SQL statements are treated as tasks (not activities) it is possible to further refine them into a subprocess to exploit the intra-query parallelism. HLOM uses as its concurrency control engine a multilevel scheduler [Wei91] augmented with the ideas described in

[ABFS97]. This engine is implemented as part of the navigator, using the transactional functionality of OPERA. In this model of correctness, locks are released before the parent transaction commits and compensation is used to undo any changes if needed. The functionality of OPERA is also useful in this regard. Every step of the execution plan (the process corresponding to the transaction) has a compensation action associated with it which is executed in the case of rollback. For the purposes of HLOM, the dispatcher needs additional information to be able to determine to which node a given SQL statement needs to be sent. This information is registered within OPERA using the standard interface provided. To help increase performance, a number of options provided by OPERA are turned off. The execution plans for object queries are not large processes. They can be easily kept in main memory to avoid the overhead of having to interact with the underlying repository after every activity terminates. One of the advantages of doing this is that the processes can be directly manipulated in OCR representation and there is no need to translate the operations into database operations. This, however, implies that the process execution is no longer persistent. Depending on the load at the navigator, it may be possible to checkpoint the state of the process by updating its image in the instance space. The decision about when to incur in this overhead is left to the user who can set the persistence level on a process (query) basis. It is here that the notion of spheres of persistence will play a major role. In addition, several navigators can be installed at different locations to help distribute the load. Although in principle OPERA supports databases of different type and residing in different networks (which can be exploited to build multimedia databases), the idea in HLOM is that the databases will be homogeneous and residing within the same LAN. This minimizes the cost of having to translate communication protocols and connecting different operating systems.

4.2 Distributed Object Management (DOM)

A similar idea to that of HLOM can be applied to heterogeneous data repositories. While HLOM is based on homogeneous components, the DOM project aims at integrating heterogeneous repositories by representing the information contained in them as virtual global objects. The idea is to provide database services to data residing outside the database. Such services include uniform object data modeling, view definition over multiple repositories, physical database design, query processing and optimization, and integrity constraint management. Some of these aspects have already been implemented as part of the Concert prototype, specially those related to physical database design [BRS96], in which the database is seen as a dataless repository exporting its services to data residing in external repositories. Of particular interest in DOM is the fact that external repositories do not need to be databases. Basic building block within DOM can be file-, email-, project management, or library information systems. Since the advantage of

using an object-oriented common data model is widely accepted [PBE95], DOM is based on the ODMG data model [Cat94], extended for a distributed environment [Rad86].

Integration of such repositories into a global system requires a translation mechanism between their internal representation and the DOM model and vice versa. As an example consider a Unix file system and a library information system over which we would like to perform cross queries. The *Unix File System* holds files organized in a hierarchical directory structure. Files may have many internal structures, often known to a particular application only. Consider, for example, the BibTEX format used by the TEX system for storing literature references. A BibTEX file can be seen as lists of entries with a well-defined structure for each entry. Thus, such files can be accessed through an iterator interface providing operations like getting the first/next object, deleting/updating the current object, or inserting a new object (object meaning files, directories, BibTEX entries, etc.). Typical queries over such an interface are "retrieve the n_{th} object" or "how many objects are available". On the other hand, the *Library Information System* consists of a database with record-oriented data about publications (books, papers, and journals) and an information retrieval system for querying purposes. Both act as a black box system, in the sense that no interface for direct access to data records is provided. Access is only gained through a WWW search interface with very limited capabilities: search is based on particular attributes and similarity, always producing a list of publications. Such an interface does not provide concepts like *current object*, or navigation following a prescribed order, and does not allow to count the number of objects stored in the library, or ask for the next object. It is even not possible to ask for all attributes of the objects.

To solve this and similar problems, (semi-) automatic integration methods have in most cases shown to be impractical. DOM, instead, pursues a different approach in which repositories can be integrated gradually. In contrast to related work, we do not focus on schema integration. Instead, only selected required parts of the repository need to be wrapped and integrated. This is done on demand, such that a more and more refined object-oriented view of external repository data is achieved over time in a stepwise manner. In the example above, for instance, the goal is to be able to query BibTEX files for specific entries and join these information with the book entries in the library system. Consider the following query, asking for the names of authors (stored in BibTEX-files), the ISBN number of books (stored in the library system), and the name of the corresponding BibTEX-file, for all publications of 1997. The resulting publications are ordered by their relevance to the keywords "objects" and "distributed".

```
select b.author, o.isbn, f.fname
from BibEntries b, Books o, Files f
where b.key = o.ident
and b.year = 1997
sort by o.abstract.about("objects","distributed")
```

In DOM, the objective is to optimize the execution of such query minimizing communication cost, I/O cost, global and local processing cost. For this purpose, first the object SQL query is parsed and translated into an object algebra expression [TS94]. These object algebra expressions (an example is shown in Figure 4.4a) serve as the starting point for query optimization. The result of this first step, an operator tree, is further optimized by a non-algebraic optimizer, that assigns concrete algorithms to the nodes of the operator tree and decides where to execute each of these nodes. The result is an execution plan such as that shown in Figure 4.4b. This execution plan is translated into an OPERA process and, henceforth, its completion is carried out as any other process within OPERA. The procedure is very similar to that described for HLOM.

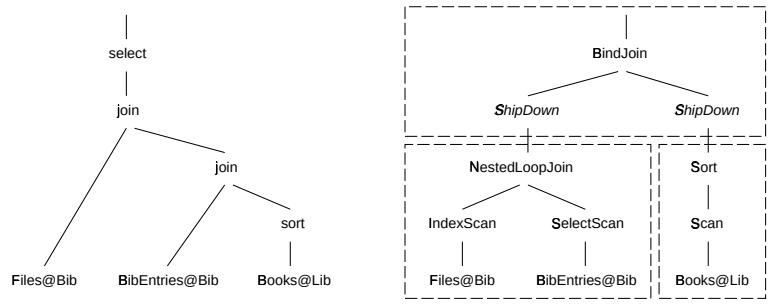

Fig. 4.4. (a) Algebraic Operator Tree and (b) Distributed Query Execution Plan

The extensions to OPERA necessary to implement DOM involve mainly those necessary to interact with heterogeneous data repositories. First, the desktop manager must incorporate the tools to translate queries into OPERA processes. This involves a language parser, algebraic optimizer, and non-algebraic optimizer. Unlike in HLOM, in DOM the mapping cannot be easily predetermined, hence the need to include an optimizer so as to avoid mapping a query into a process with too much overhead. In addition, since the external repositories may not provide any support for optimizing the execution plans, the object and query managers within OPERA are used to build indexes over external stored data. The mechanisms are similar to those described in the case of Geo-Opera. Similarly, the history space can be used to collect statistics and selectivity estimations to be used by the query optimizer. The interaction with external repositories takes place through the standard interface layer. In the case of DOM, these interfaces must be explicitly written to state the capabilities supported by the underlying system. This is a standard procedure when dealing with legacy applications since these interface can be seen as a wrapper. An example of such interface would be the iterator built over BibTEX files. If updates are considered, these interfaces need to be augmented to allow update operations. In this case, the same transactional

functionality discussed for HLOM can be used in DOM to guarantee correct execution of update operations.

5. Conclusions

Surprisingly, and to our knowledge, there is no real attempt at tackling the general problem of designing and building a general platform for distributed processing using stand alone systems and applications. Many partial solutions exist, but there is an urgent need to integrate these solutions. In this regard, the paper has tried to specifically motivate OPERA taking as starting point the current situation of middleware products. Any serious analysis of this situation points out the need to integrate all this functionality. This is not just a hypothesis, many commercial and research efforts are moving towards such a goal. But, to our knowledge, there is no in depth study of how such systems should be built and what functionality they should incorporate. The proposed system is a first step towards evaluating such questions.

References

[AAE96] G. Alonso, D. Agrawal, and A. El Abbadi. Process Synchronization in Workflow Management Systems. In *8th IEEE Symposium on Parallel and Distributed Processing (SPDS'97). New Orleans, Louisiana - October 23-26, 1996.*, October 1996.

[AAEM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert (Vol. 12, No. 5)*, 1997.

[ABFS97] G. Alonso, S. Blott, A. Fessler, and H.-J. Schek. Correctness and Parallelism of Composite Systems. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems, Tucson, Arizona, USA. May 12-15.*, May 1997.

[AE94] G. Alonso and A. El Abbadi. Cooperative Modeling in Applied Geographic Research. *International Journal of Intelligent and Cooperative Information Systems*, 3(1), May 1994.

[AM97] G. Alonso and C .Mohan. Workflow Management: the Next Generation of Distributed Processing Tools. In *Advanced Transaction Models and Architectures. S. Jajodia and L. Kerschberg (eds.). Kluwer Academic Publishers*, 1997.

[AH97] G. Alonso and C. Hagen. Geo-Opera: Workflow Concepts for Spatial Processes. In *Proceedings of the Fifth International Symposium on Spatial Databases, Berlin, Germany.* July 1997.

[Arm88] M.P. Armstrong. Temporality in spatial databases. In *Proceedings GIS/LIS*, pages 880–889, November 1988.

[AS96] G. Alonso and H.-J. Schek. Database Technology in Workflow Environments. *INFORMATIK/INFORMATIQUE (Journal of the Swiss Computer Science Society)*, April 1996.

[AVA+94] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A. El Abbadi, H.-J. Schek, and G. Weikum. Unifying concurrency control and recovery of transactions. 19(1):101–115, January 1994.

[BDG+94] A. Biliris, S. Dar, N. Gehani, H.V. Jagadish, and K. Ramamritham. ASSET: A System for Supporting Extended Transactions. In *Proc. 1994 SIGMOD International Conference on Management of Data*, pages 44–54, May 1994.

[BDS+93] Y Breitbart, A. Deacon, H.-J. Schek, A. Sheth, and G. Weikum. Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. *ACM Sigmod Record*, September 1993.

[BK94] I.Z. Ben-Shaul and G.E. Kaiser. A paradigm for decentralized process modeling and its realization in the oz environment. In *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, 1994.

[BMR94] D. Barbara, S. Mehrota, and M. Rusinkiewicz. INCAS: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments. Technical report, Matsushita Information Technology Laboratory, April 1994.

[BN97] P.A. Bernstein and E. Newcomer. *Principles of Transaction Processing for the Systems Professional.* Morgan Kaufmann, 1997.

[BRS96] Stephen Blott, Lukas Relly, and Hans-Jörg Schek. An open abstract-object storage system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

[BSR96] A. Bonner, A. Shrufi, and S. Rozen. LabFlow-1: A Database Benchmark for High Throughput Workflow Management. In *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT96), Avignon, France*, March 1996.

[Cat94] R.G.G. Cattell, editor. *The Object Database Standard: ODMG-93 (Release 1.2).* Morgan Kaufmann, San Francisco, 1994.

[CD96] Q. Chen and U. Dayal. A Transactional Nested Process Management System. In *Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA.*, February 1996.

[COR95a] *CORBA: The Common Object Request Broker Architecture, Revision 2.0.* The Object Management Group, July 1995.

[Cor95b] Transarc Corporation. *RQS System Administrator's Guide and Reference.* Transarc Corporation, 1995. ENC-D4003-02.

[Cor95c] Transarc Corporation. *Writing Encina Applications.* Transarc Corporation, 1995. ENC-D5012-00.

[CR91] Panos K. Chrysanthis and Krithi Ramamritham. A formalism for extended transaction models. In *Proceedings 17th Conference on Very Large Databases (VLDB)*, pages 103–112, Barcelona, Spain, September 1991.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-running Activities. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, pages 113–122, August 1991.

[DSW94] A. Deacon, H.J. Schek, and G. Weikum. Semantics-based Multilevel Transaction Management in Federated Systems. In *Proceedings of the 10th International Conference of Data Engineering, Houston, Texas, USA*, February 1994.

[EL96] J. Eder and W. Liebhart Workflow Recovery In *Proceedings of the 1st International Conference on Cooperative Information Systems (CoopIS96), Brussles, Belgium* June, 1996.

[ELLR90] A.K. Elmagarmid, Y. Leu, W. Litwin, and M.E. Rusinkiewicz. A Multidatabase Transaction Model for Interbase. In *Proc. of the 16th VLDB Conference*, August 1990.

[Elm92] A.K. Elmagarmid, editor. *Transaction Models for Advanced Database Applications.* Morgan-Kaufmann, 1992.

[GG89] M. Goodchild and S. Gopal. *Accuracy of Spatial Databases.* Taylor and Francis Ltd, 1989.

[GH94] D. Georgakopoulos and M. Hornick. *A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows.* International Journal of Intelligent and Cooperative Information Systems, 3(3). September, 1994.

[GHM96] D. Georgakopoulos, M. Hornick, and F. Manola. "Customizing Transaction Models and Mechanisms in a *Programmable Environment Supporting Reliable Workflow Automation.* IEEE Transactions on Knowledge and Data Engineering. April, 1996.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, April 1995.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufman, 1993.

[Gra95] J. Gray. Parallel Database Systems 101. In *Tutorial presented at the 15th ACM SIG International Conference on Management of Data (SIGMOD'95). Sanb Jose, CA, USA*, May 1995.

[HA97] C. Hagen and G. Alonso. Flexible Exception Handling in the OPERA Process Support System. *In preparation*, 1997.

[Has96] H. Hasse. Einheitliche Theorie für korrekte parallele und fehlertolerante Ausführung von Datenbanktransaktionen. Technical Report 11569, ETH Zürich, Department Informatik, 1996.

[HGM93] N.I. Hachem, M.A. Gennert, and Ward M.O. The Gaea System: A Spatio-Temporal Database System for Global Change Studies. In *AAAS Workshop on Advances in Data Management for the Scientist and Engineer, Boston, Massachussetts*, pages 84–89, February 1993.

[Hol96] D. Hollinsworth. The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition, December 1996. Accessible via: http://www.aiai.ed.ac.uk/WfMC/.

[Hsu93] M. Hsu. Special Issue on Workflow and Extended Transaction Systems. *Bulletin of the Technical Committee on Data Engineering, IEEE*, 16(2), June 1993.

[Hsu95] M. Hsu. Special Issue on Workflow Systems. *Bulletin of the Technical Committee on Data Engineering, IEEE*, 18(1), March 1995.

[IBM95] IBM. *FlowMark - Managing Your Workflow, Version 2.1.* IBM, March 1995. Document No. SH19-8243-00.

[JB96] S. Jablonski and C. Bussler. *Workflow Management.* International Thomson Computer Press, 1996.

[KAGM96] M. Kamath, G. Alonso, R. Günthör, and C. Mohan. Providing High Availability in Very Large Workflow Management Systems. In *In Proceedings of the Fifth International Conference on Extending Database Technology (EDBT'96)* , Avignon, France, March 1996. Also available as IBM Research Report RJ9967, IBM Almaden Research Center, July 1995.

[Kle91] J. Klein. Advanced Rule Driven Transaction Management. In *36th IEEE Computer Society International Conference CompCon Spring 1991*, pages 562–567, San Francisco, California, March 1991.

[KS96] H. Kaufmann and H.-J. Schek. Extending TP-Monitors for Intra-Transaction Parallelism. In *Proc. of the Fourth Int. Conf. on Parallel and Distributed Information Systems (PDIS'96), Miami Beach, Florida, USA*, December 1996.

[Lan88] G. Langram. Temporal GIS design tradeoffs. In *Proceedings GIS/LIS*, pages 890–899, November 1988.

[Le97] J. Lyon and K. Evans, J. Klein. Transaction Internet Protocol (TIP). Technical Report draft-lyon-tip-nodes.02.txt, Tandem and Microsoft, February 1997.

[LV90] D.P. Lanter and H. Veregin. A Lineage Meta-Database program for propagating error in Geographic Information Systems. In *Proceedings GIS/LIS*, pages 144–153, November 1990.

[MH94] C. Mohan and D. Haderle. Algorithms for flexible space management in transaction systems supporting fine granularity locks. In *Proceedings of the EDBT'94 Conference, Cambridge, UK*, pages 131–144, March 1994.

[Mos81] J. Eliot B. Moss. Nested transactions: An approach to reliable computing. M.i.t. report mit-lcs-tr-260, M.I.T., Laboratory of Computer Science, April 1981.

[MSKW96] J.A. Miller, A. Sheth, K.J. Kochut, and X. Wang. Corba-based runtime architectures for workflow management systems. *Journal of database Management*, 7, 1996.

[MVW96] J. Meidanis, G. Vossen, and M. Weske. Using Workflow Management in DNA Sequencing. In *Proceedings of the 1st International Conference on Cooperative Information Systems (CoopIS96), Brussles, Belgium*, June 1996.

[NSSW94] M.C. Norrie, W. Schaad, H.-J. Schek, and M. Wunderli. Exploiting Multidatabase Technology for CIM. Technical report, Computer Science Department, Database Research Group, ETH Zürich, July 1994.

[Obe94] R. Obermack. Special Issue on TP Monitors and Distributed Transaction Management. *Bulletin of the Technical Committee on Data Engineering, IEEE*, 17(1), March 1994.

[PBE95] E. Pitoura, O. Bukhres, and A. Elmagarmid. Object orientation in multidatabase systems. *ACM Computing Surveys*, 27(3), June 1995.

[Rad86] E. Radeke. Extending odmg for federated database systems. In *Proc. 7th Int'l Conf. on Database and Expert Systems Applications*, Zurich, Switzerland, September 1986.

[Rad91] F.J. Radermacher. The Importance of Metaknowledge for Environmental Information Systems. In *Proceedings of the 2nd Symposium on the Design and Implementation of Large Spatial Databases, Springer Verlag*, volume 1, pages 35–44, August 1991.

[RNS96] M. Rys, M.C. Norrie, and H.-J. Schek. Intra-Transaction Parallelism in the Mapping of an Object Model to a Relational Multi-Processor System. In *Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India*, September 1996.

[Sch96] H.-J. Schek. Improving the Role of Future Database Systems. *ACM Computing Surveys*, 28(4), December 1996.

[SSAE93] T.R. Smith, J. Su, D. Agrawal, and A. El Abbadi. Database and Modeling Systems for the Earth Sciences. *IEEE Bulletin of the Technical Committee on Data Engineering*, 16(1):33–37, March 1993.

[SSE+95] T. Smith, J. Su, A. El Abbadi, D. Agrawal, G. Alonso, and A. Saran. Computational Modeling Systems. *Information Systems*, 20(2), 1995.

[SSW95] W. Schaad, H.-J. Schek, and G. Weikum. Implementation and performance of multi-level transaction management in a multidatabase environment. In *Proc. of the 5. Int. Workshop on Research Issues on Data Engineering, Distributed Object Management, Taipei, Taiwan*, 1995.

[ST96] B. Salzberg and D. Tombroff. DSDT: Durable Scripts Containing Database Transactions. In *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, Louisiana, USA, February 1996.

[SW93] H.J. Schek and A. Wolf. From Extensible Databases to Interoperability between Multiple Databases and GIS Applications. In *Proceedings of the 3rd Int. Symposium on Large Spatial Databases*, Singapore, June 1993.

[SWY93] H.-J. Schek, G. Weikum, and H. Ye. Towards a unified theory of concurrency control and recovery. In *Proceedings of the ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 300–311, June 1993.

[SZ96] A. Silberschatz and S. Zdonik. Database Systems, Breaking out of the Box. *http://www.cs.brown.edu/people/sbz/cra/paper.ps*, September 1996.

[TKP94] A.Z. Tong, G.E. Kaiser, and S.S. Popovich. A flexible rule-chaining engine for process-based software engineering. In *Proceedings of the Ninth Knowledge-Based Software Engineering Conference*, Monterey, CA, USA, 1994.

[Tre96] M. Tresch. Principles of distributed object database languages. Technical Report 248, ETH Zürich, Dept. of Computer Science, July 1996.

[TS94] M. Tresch and M. H. Scholl. A classification of multi-database languages. In *Proc. 3rd Int'l Conf. on Parallel and Distributed Information Systems (PDIS)*, Austin, Texas, September 1994. IEEE Computer Society Press.

[Wei91] G. Weikum. Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems*, 16(1), March 1991.

[WH93] G. Weikum and C. Hasse. Multi-level transaction management for complex objects: Implementation, performance, parallelism. *The VLDB Journal*, 2(4), 1993.