Distributed Processing over Stand-alone Systems and Applications

Gustavo Alonso Claus Hagen Hans-Jörg Schek Markus Tresch
Institute of Information Systems
Swiss Federal Institute of Technology (ETH)
Zürich, CH-8092, Switzerland
{alonso,hagen,schek,tresch}@inf.ethz.ch

Abstract

This paper describes the architecture of OPERA, a generic platform for building distributed systems over stand alone applications. 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 correctness 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

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, with the added advantage that most of the necessary infrastructure is already in place. Unfortunately, there is also the problem of building something coherent out of systems that were not necessarily designed to work together. A great deal of effort has been devoted to address this issue, specially in the form of *middleware*: federated and multi-databases systems, TP-monitors, persistent queuing systems, CORBA implementations, workflow management systems, or process centered environments for software engineering. These efforts seem to indicate that the difficulty may not lay on the lack of solutions but on the lack of *integrated solutions*. As a matter of fact, the need to integrate the different approaches in practical systems is very clear: CORBA needs the transactional services a TP-monitor provides; 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; distributed execution over the internet needs transactional guarantees; and so forth. The synergy is not surprising since most middleware systems have a lot in common: logging, directory and name services, accounting, indexing, classification, record keeping, or failure recovery, to mention a few issues. This shared functionality is used to provide properties such as persistent execution, transactional guarantees, forward and backward navigation, high level compensation, and process synchronization, which are at the core of any distributed system. The main hypothesis behind this paper is that it should be possible to design a kernel system capable of providing such core functionality.

Following this idea, the paper describes OPERA, a basic kernel for distributed processing over clusters of workstations. OPERA incorporates ideas and technology from areas such as distributed and parallel databases, transaction processing systems, and workflow management. It also takes advantage of previous results in exporting database functionality [BRS96] and transactional middleware systems [ABFS97].

**Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997**

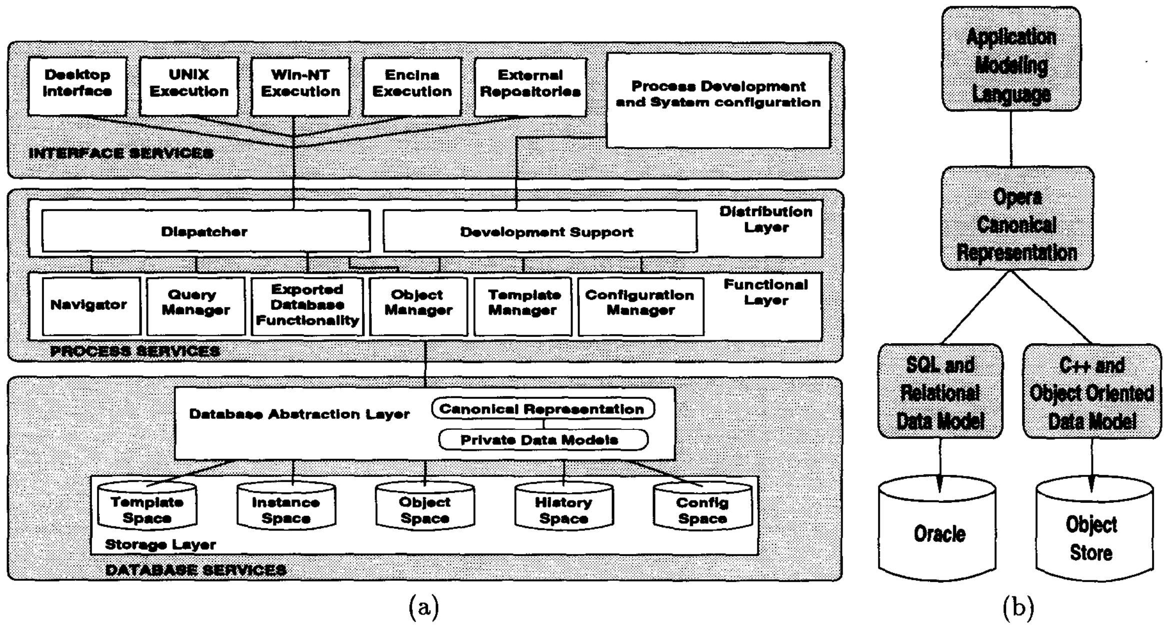

Figure 1: *(a) System architecture of OPERA; (b) The different language representations in OPERA*

The paper is organized as follows. Section 2 presents OPERA, its general goals, architecture, and most relevant aspects. Section 3 discusses a number of application areas in which OPERA is being used by extending and tailoring the basic system with additional functionality. Section 4 provides additional information regarding the status of the project and concludes the paper.

2 The OPERA Kernel

2.1 Functionality

We are particularly interested in the aspects in a distributed system that can be supported by database technology, among these:

Process management. Processes are arbitrary sequences of application invocations over different locations and platforms. Hence, the kernel should play the roles of scheduler and resource allocator for such processes. The kernel should also 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.

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 (spheres of isolation, atomicity, and persistence).

Externalized Database Functionality. In many applications running over clusters of workstations, most data does not reside within databases, forcing the applications to implement their own database services. 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 but it also establishes the basis for keeping track of the many elements involved in distributed environments [AH97, AHST96].

2.2 Architecture of OPERA

OPERA is being designed as a kernel providing the core functionality described above [AH97, AHST96]. Its architecture is organized around three service layers (Figure 1.a): *database services*, *process services* and *interface services*. The database service layer acts as the storage manager. It encompasses the actual databases used as repositories and the database abstraction layer. 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. For each running instance of a process, the instance space contains a copy of the corresponding template. Objects are used to store information about externally defined data. The history space is used to store information about already executed instances. The configuration space is used to record system related information.

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]. This canonical representation is not suitable, however, for either commercial databases or user interaction. Thus, 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.

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 acts as resource allocator for process execution. The navigator acts as the overall scheduler. It also enforces the transactional aspects of the execution. The system interacts with the data spaces through the query manager. The query manager provides a suitable interface for complex queries about the status of the processes, which in many cases are standard. We expect that most of these queries will be executed over the history space, 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 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 [AH97]. Finally, the interface service layer encompasses all the mechanisms that allow OPERA to interact with applications in different hardware and software platforms.

2.3 Process Management

The notion of process is central to OPERA. 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 and it is is difficult to use them in areas other than the ones for which they were designed. 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.

In order to provide a generic notion of process, OPERA contains a hierarchy of process representations rather than a single model (Figure 1.b). At the top of the hierarchy, and used at the interface service layer, is the application specific language. 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.

A *process* consists of a set of tasks and a set of data objects. Tasks can be activities, blocks, or processes. The data objects store the input and output data of tasks and are used to pass information around. *Activities* are the basic execution steps. An activity provides a *navigation interface* to access information about its state. 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.

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. Subprocesses allow, like blocks, the hierarchical structuring of complex process structures.

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. *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. All input and output data structures are mapped to the *blackboard*, which acts as the global data area for each process.

Events are used to allow processes to communicate 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 events. *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.

2.4 Transactional Execution Guarantees

The transactional aspects of OPERA are embedded in the notion of *spheres* and in the scheduling performed by the navigator. Currently, there are three types of spheres in OPERA, spheres of atomicity, spheres of isolation and spheres of persistence. In all cases, the spheres must be defined by the user when the process is created. Spheres of atomicity generalize the standard all or nothing semantics in that they guarantee controlled termination including alternative execution paths in case of failures or exceptions [HA97]. OPERA distinguishes among several classes of these spheres depending on how atomicity is maintained (an issue also important when considering both atomicity and isolation): *Basic* (non-atomic), *Semi-atomic*, *Atomic*, *Restartable*, or *Compensatable*. Spheres of persistence are used to avoid the overhead incurred by storing all process information in the instance space. 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. The semantics behind the notion of spheres of isolation generalize the ideas suggested in [AAE96], which point out the need for a notion of synchronization in the traditional operating systems sense, and recent work that extends the existing notions of nested and multilevel transactions and applies them to *composite systems* [ABFS97]. The relation between spheres of atomicity and spheres of isolation is treated following the concepts of the unified theory of concurrency control and recovery [AVA+94].

2.5 Availability and Scalability

Availability and scalability are crucial aspects of distributed processing environments [AAEM97]. To increase scalability, OPERA uses several mechanisms. OPERA can use several databases as the underlying repositories (the current prototype can use Oracle and Object Store simultaneously) and allows the separation of spaces across these databases, which is probable one of the most significant factors when performance and scalability is considered. This also 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. As in [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.

3 Applications of OPERA

OPERA is currently being used in three research projects: Geo-OPERA, HLOM, and DOM.

Geo-OPERA is a process support system tailored to spatial modeling and GIS engineering [AH97]. It facilitates the task of coordinating and managing the development and execution of large, computer-based geographic models. It also integrates a flexible environment for experiment management, incorporating many characteristics of workflow management systems as well as a simple but expressive process modeling language, exception handling, and data and metadata indexing and querying capabilities, which are provided by OPERA.

HLOM (High Level Object Management) is a research project exploring parallelism in systems built out of stand-alone, off-the-shelf, commercial database systems which are treated as black boxes providing database services. Contrary to federated databases, HLOM is designed top-down, 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] and a document management system using a TP-Monitor on top of a relational database [KS96]. In HLOM, OPERA acts as the global database scheduler and distribution engine.

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 which can be manipulated using an object algebra [TS94, Tre96]. 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 data-less repository exporting its services to data residing in external repositories. Thus, the extensions to OPERA necessary to implement DOM involve mainly those necessary to interact with heterogeneous data repositories.

4 Project Status and 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. The ideas described in this paper are to be seen as a work of synthesis and integration trying to explore the potential uses of databases as basic components of distributed systems. Many of the ideas discussed in the paper have already been implemented and tested. Currently we are in the process of completing the first prototype, and using it in the Geo-Opera [AH97], HLOM and DOM projects.

What is new and interesting about OPERA is that it brings together ideas from many areas within one single system. 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, also hoping to stimulate the discussion about the role of databases in the future.

Acknowledgements

Part of this work has been funded by ETH Zürich and by the Swiss Science Foundation within the TRAMS project. For reasons of space, references not directly related to the project have not been included. These references can be found in the extended version of the paper [AHST96] also available in our WEB server: *http://www-dbs.inf.ethz.ch/*.

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'96). New Orleans, USA.*, October 1996.

[AAEM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert*, 12(5), September-October 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.

[AH97] G. Alonso and C. Hagen. Geo-Opera: Workflow Concepts for Spatial Processes. In *Proceedings of the 5th International Symposium on Spatial Databases, Berlin, Germany,* July 1997.

[AHST96] G. Alonso, C. Hagen, H.J. Schek, and M. Tresch. *Towards a Platform for Distributed Application Development.* 1997 NATO Advance Studies Institute (ASI). A. Dogac, L. Kalinichenko, T. Ozsu, A. Sheth (editors). August 12 -21, 1997, Istanbul, Turkey, 1996.

[AVA$^+$94] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A. El Abbadi, H.J. Schek, and G. Weikum. A Unified Approach to Concurrency Control and Transaction Recovery. In *Proceedings EDBT'94. Springer-Verlag LNCS 779*, pages 123–130, March 1994.

[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.

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

[KAGM96] M. Kamath, G. Alonso, R. Günthör, and C. Mohan. Providing High Availability in Very Large Workflow Management Systems. In *In Proceedings of EDBT'96* , Avignon, France, March 1996. Springer-Verlag, LNCS 1057.

[KS96] H. Kaufmann and H.-J. Schek. Extending TP-Monitors for Intra-Transaction Parallelism. In *Proc. of the 4th Int. Conf. on Parallel and Distributed Information Systems (PDIS'96), Miami Beach, Florida, USA*, December 1996.

[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.

[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.