Processes + Transactions = Distributed Applications

Position Paper, HPTS'97

Gustavo Alonso
Institute of Information Systems, Swiss Federal Institute of Technology (ETH)
ETH Zentrum, Zürich CH-8092, Switzerland
E-mail: alonso@inf.ethz.ch

March 10, 1997

1 Introduction

In addition to the conventional operating system interpretation, the concept of *process* is starting to be used to refer to complex sequences of computer programs and data exchanges controlled by a meta-program. Thus, today one finds notions such as *process centered software engineering*, *business processes*, or *process based parallelism*. In fact, the idea seems to have widespread acceptance in many areas, in particular in those where computation is based on cluster of PC's and workstations or on environments involving heterogeneous platforms and applications. A careful analysis of areas such as business environments [Fry94], software engineering [BK94] or scientific data management [ILGP96, BSR96] reveals a surprising number of problems that are both pervasive and common to all of them. Such pervasiveness may explain the attention being devoted to *workflow* products, which are the current process support systems. It may also explain why some researchers consider workflow management to be just a reincarnation of job control languages.

Today's computing environments, however, have changed significantly since the conception of job control languages. In practice, the most widely available platforms for corporate computing are based on multiple stand alone computers linked by a network. Such clusters offer the possibility of implementing truly distributed systems, which can be done in two ways: *top-down*, the entire system is designed from the beginning as a distributed system, and *bottom-up*, where already existing applications are used as building blocks. The former approach offers many interesting research opportunities and considerable attention has been devoted to it. The latter approach is, above all, practical since in most cases both the hardware and software infrastructures are already in place and cannot be discarded. The notion of process described above, derived mainly from workflow management, targets precisely distributed systems built following a bottom-up strategy, which is likely to be the approach of choice for future distributed systems.

In this short paper, it is argued that the notion of process, augmented with transactional properties, can be a very powerful tool for designing and developing distributed applications over clusters of PCs and workstations. This is done by first defining the type of distributed applications being targeted (Section 2), showing how the notion of process supports such applications (Section 3), and finally pointing out the role transaction must play to complement the notion of process (Section 4). The paper concludes with an overview of how these systems may evolve (Section 5).

2 Distributed Applications

The distributed application addressed in this paper are those based on already existing, stand-alone tools located on clusters of PCs and workstations linked by a network. The idea is to provide a way to use these existing tools as building blocks of a higher level system in which the process acts as the blueprint for control and data flow. Typical examples are business processes in which several office tools such as spread-sheets, text editors, databases, and human decisions are combined into a higher level entity by encoding the business logic within the flow of control and data of the process. Among existing products, the ones most closely related to the notion of process used in this paper are workflow management systems. Existing workflow tools, however, target almost exclusively either business processes or imaging systems, suffer from severe limitations related to performance and functionality [AAEM97], and are not easy to apply in other areas [MVW96, BSR96]. Workflow management, however, is a first step towards supporting the development of complex applications over distributed systems using already existing tools. Its concepts can be generalized by extending the notion of process to any arbitrary sequence of tool invocations (a script or pipelined UNIX commands, for instance, are simple forms of processes). In this way, future workflow systems could act as a high level programming tools linking heterogeneous, stand-alone applications. Integrated with additional technology such as CORBA, queuing systems or TP-monitors, workflow management could also very well play the role of distributed operating systems in which to exploit the coarse parallelism and distributed characteristics of distributed processes.

3 Processes

A process can be seen as a description of an arbitrary sequence of application invocations along with the data flow between these applications. As such, the process acts as a the meta-program governing the interactions among existing applications. Each step within a process is an *activity*, which represents invocations of external applications. The flow of control within a process – what to execute next – is determined by control connectors labelled with *transition conditions*, usually boolean expressions based on data produced by the activities. Processes also include programming constructs that allow modular design and nesting, as well as the invocation of other processes.

During execution, the *process engine* navigates the description of the process determining which activities are to be invoked next. The procedure is very similar to that of executing any other program (more like an interpreted program than a compiled one). The description of the process is usually stored persistently in a database and the engine consults the database continuously to find out what is to be done next. Any changes to the process are also stored persistently (the status of executed activities, returned values, etc.), which opens up interesting possibilities in terms of recovery and overall reliability [KAGM96]. When an application is to be invoked, the workflow engine notifies an *application-agent* located at the same node where the program resides. The application agent then executes the program and returns the results of the program to the workflow engine.

These ideas hint at the possibility of considering the workflow engine as a distributed operating system executing programs that have been constructed using existing applications as programming

primitives. In practice, workflow engines do act as schedulers and resource allocators in distributed environments. But, unlike in the case of centralized operating systems, workflow engines have to contend with the network, which immediately brings up issues such as atomicity of invocations, interferences between concurrently executed processes, performance, and overall consistency. Here is where transactional notions should play an important part.

4 Transactions

Transactions are the conventional form of encapsulating database operations so as to provide *Atomicity*, *Consistency*, *Isolation*, and *Durability* [GR93]. They provide not only clean semantics to the interactions between concurrent executions but also a powerful abstraction in which to base optimizations to the architecture of a database system. This same ideas can be applied to processes so as to provide useful abstractions to reason about the correctness of the system, to express properties of the execution of processes, and to tune the overall architecture of process support systems. It is likely that transactional ideas applied to process support systems will be as successful as when applied to databases (which, to certain extent, has always been the goal of many advanced transaction models [Elm92]). However, in the case of processes, database-like transactions are too rigid. Some form of *light-weight* transactions should be used [BRS96]. Thus, within a process, instead of using a unique construct encompassing all transactional properties, e.g., *BOT/EOT*, several separate constructs should be used to group activities according to the desired semantics. Thus, one could consider *spheres of atomicity* (atomic units with the standard all or nothing semantics), *spheres of isolation* (isolation units, much like critical sections in traditional operating systems), and *spheres of persistence* (determining whether the activitys in the sphere are to be made persistent or not). Spheres can be naturally combined with the nested structures provided in most process specification languages. Such constructs could have the following semantics:

Spheres of Atomicity. Atomicity is the "most popular" property from transactions in that it addresses a common coordination problem. There have already been many suggestions regarding how to use atomicity in advanced transaction models [Elm92] and it has also been suggested as a basic mechanism for process navigation [Ley95]. In fact, process support systems may offer the only realistic scenario in which to use the many ideas related to compensation. For instance, activities could be declared to be *Basic* (non-atomic), *Semi-atomic*, *Atomic*, *Restartable*, or *Compensatable*. Basic activities are the default and correspond to non-atomic applications, i.e., those for which the system cannot guarantee atomicity. Semi-atomic activities are those providing enough information to implement a rollback method to be executed if the activity fails before completing its execution. Atomic activities are those that preserve atomicity by themselves, for instance, a transaction executed over an X/Open XA interface. Restartable activities [ELLR90] are those that can be invoked repeatedly until they eventually succeed. Compensatable activities are those that can be undone after they have finished using an user provided method attached to the activity interface[ELLR90]. Note that these categories can be applied at different levels, thereby allowing to group a set of activities into a semi-atomic block, for instance, or provide high level compensation for an entire sub-process by declaring it to be compensatable. The same navigation mechanisms used for processes could be used to drive operations related to atomicity properties.

Spheres of Isolation. 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. For applications that demand a database like approach both spheres of isolation and spheres of atomicity could be used in a manner similar to that of nested or multilevel transactions [Wei91, Mos81], which provide a powerful mechanism to reason about recovery in applications with a complex structure [GR93].

Spheres of Persistence. Spheres of persistence could be used to avoid the overhead incurred by storing all process information in a database. Using databases for storage provides significant advantages in terms of reliability, monitoring and audit control [AAEM97, KAGM96], but introduces a considerable overhead. Within a process, a programmer could specify whether a set of activities is to be executed persistently or not. If an activity or a group of activities are embedded within a sphere of persistence, every step of the execution is recorded in the underlying database. This guarantees forward recoverability in the event of failures. The information related to activities not included in a sphere of persistence about the execution is maintained only in main memory. Upon completion or after pre-determined time intervals, this information could be checkpointed to the database in an off-line fashion, thereby avoiding the I/O overhead.

5 Processes + Transactions = Distributed Applications

In practice, the equation that serves as the title for this position paper should include *Standarization* on its left side. The amount of efforts being devoted to CORBA, OLE, SOM, DSOM, and ODBC, to mention a few, show the growing interest in being able to link together stand alone tools. In terms of products, the same goal can be ascribed to TP-monitors, persistent queuing systems, CORBA implementations, and workflow management systems. These products are slowly converging towards a common point, as proven by the many ongoing attempts at combining their functionality. For instance, TP-monitors implementing the execution guarantees in CORBA, CORBA extensions to TP-Monitors, efforts to combine workflow standards and CORBA under the notion of business objects, or queuing systems being added to workflow management systems. Once applications follow a given standard interface and these interfaces have been widely accepted, the task of linking together stand alone systems will be greatly simplified. But in addition to the standards there must a way to express interactions between stand alone applications. A very powerful paradigm for this purpose is processes, with the added advantage that the technology for process support is almost already in place. Certainly it will not be in the form of current workflow management systems, but it is likely that it will be in the form of a combination of mainly TP-monitors (for transactional guarantees), queuing systems (to allow asynchronous interactions), CORBA (and/or any other standard providing a common way to interact with applications), and workflow management (for process support). In a way, this is not very different from early transactional operating systems [GR93], but it is an idea that offers very interesting research and commercial opportunities and may give transactional concepts a significant relevance in future information systems.

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 (to appear)*, 1997.

[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.

[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.

[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.

[Fry94] C. Frye. Move to Workflow Provokes Business Process Scrutiny. *Software Magazine*, pages 77–89, April 1994.

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

[ILGP96] Y.E. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: A desktop Experiment Management Environment. In *Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India*, September 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.

[Ley95] Frank Leymann. Supporting business transactions via partial backward recovery in workflow management systems. In *GI-Fachtagung Datenbanken in Büro Technik und Wissenschaft - BTW'95*, Dresden, Germany, March 1995. Springer Verlag.

[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.

[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.

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