Processes in Electronic Commerce

G. Alonso C. Hagen A. Lazcano
Information and Communication Systems Research Group
Swiss Federal Institute of Technology (ETH), ETH Zentrum, Zürich CH-8092, Switzerland
E-mail: {*alonso, hagen, lazcano*}*@inf.ethz.ch*
http://www.inf.ethz.ch/department/IS/iks/

Abstract

Trading communities, virtual business processes and virtual enterprises are key elements in today's Electronic Commerce. In this paper, we argue that supporting these notions is similar to supporting the development of coarse grained distributed programs and that these programs can be abstracted as processes. Following this idea, we have developed a language for the enactment of processes that incorporates several important features like exception handling, event management, and transactional guarantees. The paper describes these features and discusses the use of processes in E-Comm.

1 Introduction

Electronic commerce practices are not new. The EDI (Electronic Data Interchange) standard, for instance, is widely used for business to business electronic commerce activities, mainly stock and supply management. In this highly automated form of electronic commerce, transactions are typically carried out between the mainframes of two companies. When a consumer company runs low on certain supplies, a purchase order is triggered automatically. A mainframe contacts predefined suppliers to place a request using predefined catalogs with the available items and prices. The mainframe at the supplier receives the request, processes it, generates the adequate responses, confirmations, billing information and accounting records, and the supplies are then delivered to the requesting company. The advantages of the approach are clear: speed (transactions are carried out electronically) and accuracy (no human intervention prevents clerical and data entry errors). In the long term and for a sufficiently high transaction volume, this approach also reduces the cost of purchase order management. Successful as it has been, this technology has two significant drawbacks: The high costs involved and the lack of suitable application software.

The advent of affordable computing facilities and the Internet has opened up the field to many more participants. Today, the investment in hardware and communication necessary to deploy an basic E-Comm application is almost negligible compared with what it used to be. Unfortunately, adequate application software is still missing.

As a first step towards addressing this lack of suitable software, we are currently developing a framework –one of the many possible– for E-Comm applications. The core of our approach is the notion of *process.* In addition to the conventional operating system interpretation, the concept of process is starting to be used to refer to complex sequences of application invocations and data exchanges controlled by a meta-program [1]. Thus, today one can find, for instance, *process centered* software engineering, *business processes,* or *process based parallelism*, where coarse granularity distributed applications are modeled and implemented based on processes. We believe this same idea can be successfully applied to some forms of E-Comm. In particular, we see the individual tasks to be executed by each partner in an E-Comm endeavor (e.g., "check inventory", "compare prices" or "accept purchase order") as the basic building blocks of a distributed application. These tasks are implemented separately at each one of the participating companies and we can use them (conveniently wrapped if necessary) as plug and play components of a more sophisticated application, the process, deployed across organizational boundaries.

It is this notion of process, and the supporting technology, that we see as the framework which existing E-Comm solutions are lacking today. In this paper, we briefly describe our framework, point out its advantages, and discuss its implementation as part of WISE (Workflow based Internet SErvices), an ongoing research project in which we are developing several E-Comm solutions. The results obtained so far allow us to be quite optimistic about the approach we have taken. When applied in practice, our ideas have proven to be not only feasible but very useful in a variety of E-Comm environments.

In the following section, we define the notions of *trading communities*, *virtual business processes* and *virtual enterprises*, which are central to our approach. In Section 3, we discuss the framework and its implementation within WISE. Finally in Section 4, we briefly summarize the status of our implementation efforts and discuss future work.

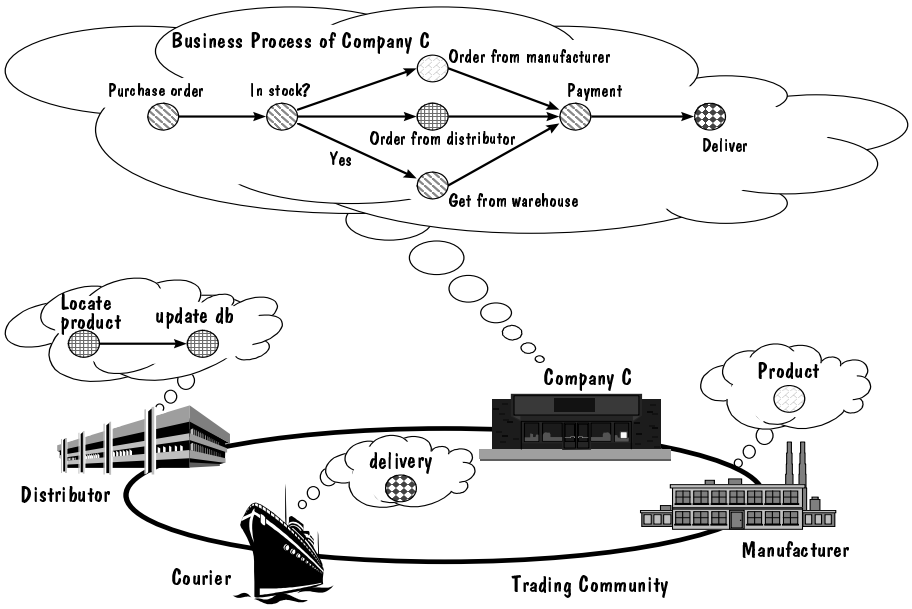

Figure 1: A company incorporating a virtual process as part of its own business processes

2 Problem Statement

2.1 Virtual Enterprises

The most relevant activities within a corporation are often described in the form of business processes. This is not surprising since business processes model the procedures and rules followed in order to accomplish a concrete goal (open a new bank account, obtain a credit, purchase a computer, find out the current location of a parcel, resupply shops, etc.). Following this idea, we see electronic commerce as the incorporation of information and communication systems technology into the business process to expand it beyond the corporation boundaries. In this context, we define a *virtual business process* as a business process whose definition and enactment cannot be directly tied to a single organizational entity (be it a department or a company). From here, we define *virtual enterprises* as those whose business processes are virtual business processes. Given the trend towards decentralization, we also consider that virtual business processes linking together several departments of a single organization define as well a virtual enterprise. Finally, we refer to the set of companies participating in a virtual enterprise as a *trading community*. Each member of the trading community provides a number of services to be used as building blocks for the virtual process. Based on these services, the virtual enterprise can be created by defining a virtual process in which each individual activity corresponds to one of the services provided by the participants.

We believe trading communities, virtual enterprises and virtual processes are a very powerful approach to interpret and identify the needs of a wide range of electronic commerce practices. For instance, in the case of retailing, a company can provide a much more sophisticated product by outsourcing aspects of the operation which are not central to its activities. A common example are companies offering a product (books, CD, flowers) without actually handling (producing, storing or delivering) the product themselves. Most of the handling is left to companies providing specialized services, which allows to significantly reduce the operational costs. The virtual enterprise model naturally captures such scenarios by simply having the distribution and delivery services incorporated as activities within the business processes of the company selling the product as shown in Figure 1.

2.2 E-Comm Processes

In the example of Figure 1, independently of whether it involves mainframes and leased lines or a few PCs linked via Internet providers, an E-Comm application has many of the characteristics of a distributed computing environment. While the notions of trading community and virtual enterprise are conceptually useful, the real challenge is to use them in a software solution. Here is where the idea of process becomes relevant: the virtual business process can be seen as a distributed program running on some form of middleware linking together the resources of the trading community [4]. These resources are the concrete applications or

services offered to the virtual enterprise by the trading community and are used as the basic building blocks for the distributed program (the virtual business process). From here, the type of software to develop is the type of software that would be needed to support the definition and execution of such a coarse grained distributed program.

The analogy between an E-Comm process and a distributed program can be taken a step further. Any realistic solution to electronic commerce must take into account the true complexity of the problem. We see E-Comm processes as valuable assets needing to be properly specified, designed, developed, tested, debugged, and maintained in an effort not unlike software life-cycles. In order to do this, the language used to describe the processes must provide the necessary primitives, otherwise these tasks become extremely difficult and largely ad-hoc endeavors (as it is today).

2.3 A Software Platform for E-Comm

From these ideas, we can derive a sketch of the specification of a possible software platform for E-Comm applications of the type described above. Since the main abstraction we use is the process, the specification revolves around how to support processes. Thus, we consider a three step *process management cycle* for virtual enterprises:

- *Process definition*, which includes a process language incorporating features such as exception handling, interprocess communication, event management, and execution guarantees (in the database transactional sense). We strongly believe this functionality is sorely needed since a business process, virtual or not, is just a description of an evolving practice. Applying known software engineering techniques will certainly be a practical advantage.

- *Process enactment*, which includes compiling the description of the virtual process so as to transform it into a representation suitable for enactment, and the actual enactment by invoking the services of the trading community making up the virtual process. The process enactment engine should also support features such as back-up, replication, load balancing, process migration, persistence, and recovery.

- *Process monitoring and analysis*, that is, keeping track and recording every step of the virtual process for load balancing and quality of service purposes as well as, later on, for analysis of the behavior of the process. The idea is to capture the entire execution history for both on-line use (load balancing, system administration) and off-line analysis (mining of the process data for business re-engineering, provisioning, and optimization purposes).

3 WISE: A Platform for E-Comm Processes

In many ways, this specification can be seen as that of a high level operating system designed to work over a heterogeneous, geographically separated cluster of computers and conceptually based on the notion of process. Instead of the traditional system level calls, our building blocks are already existing applications. Instead of conventional programs, we work with processes. This is the idea behind the OPERA process support system, a system developed in the Information and Communication Systems Research Group of ETH Zürich with the objective of providing a generic process tool which can be tailored to different applications [3]. As part of the WISE project we have extended OPERA to provide the necessary functionality for E-Comm purposes. This functionality is related to the notions of trading community, virtual enterprise and virtual business process as explained above.

Within WISE, we emphasize the fact that the three steps of the process management cycle are only different aspects of the same problem. In this sense, the best way to describe WISE may be as an integration effort, i.e., an attempt to combine several different technologies into a coherent whole. In what follows, we briefly discuss some of these technologies and how they have been implemented in WISE.

3.1 A Programming Language for Processes

One of the most important aspects of our approach is the language used to define and execute processes. Our research efforts have not been focused on a particular abstraction (such as Petri-Nets, State-Activity charts, or transaction models) but on the functionality the language should support. Moreover, our experience with users shows that it is not sensible to impose a language on the users. Most companies already use their own business process modeling tools and will not be willing to change this tool for another.

As a result, what we support in terms of language is an internal language into which we compile many other languages [10]. This internal language is called OCR (Opera Canonical Representation) and it is optimized for enactment purposes (minimize overhead since the language is persistent, optimize flow to speed up the navigation through a process, and incorporate enough information for recovery and monitoring). OCR is not intended to be used directly by users. In WISE, we use the business modeling tool (Structware, which is commercially available) of one of our industrial partners (IvyTeam) as the front end [12, 14]. Users define processes using this tool and we compile the representation of this tool into OCR.

From our point of view, the interesting aspect is the functionality supported by OCR. This includes exception handling [10], event management, inter-process communication mechanisms [11], and atomicity guarantees. It is

this functionality, separately and as a whole, that differentiates WISE from any, to our knowledge, of the existing approaches to both E-Comm and process management.

Exception Handling. In environments like E-Comm where a large number of processes will be executed, having exceptions is bound to be a normal occurrence. Any programming tool intended for large, complex applications has to face this problem. These tools usually incorporate exception handling mechanisms to separate the failure semantics from the program logic and thus facilitate the design of readable, comprehensible code. First in OPERA and then in WISE we have incorporated mechanisms for failure handling which are based on well-known programming language concepts [10]. Among these, we take advantage of the language extensions for structured exception handling first presented by Goodenough [8] and subsequently adopted in a number of programming languages (CommonLisp [18], C++ [19] and Java [6]). The key aspect of our exception model is separating exception detection from exception handling. The process description is decomposed into the process itself, which contains only the business logic, and the exception handlers, which are activities or subprocesses triggered when another activity reaches an exception. Exceptions can be generated by external programs or as semantic exceptions internal to WISE. The latter are like ordinary activities and, therefore, can occur anywhere inside a process. The former are based on user-provided predicates that define under which circumstances a given exception is raised. Like in structured programming languages, exception are propagated upwards along the invocation hierarchy until an appropriate handler is encountered, a very useful feature when dealing with nested processes. It is also possible to define different behaviors after the exception has been dealt with: return to the signaler, abort the signaler and return to the invoker, etc. With this idea, the process language of WISE adds a significant degree of flexibility when defining virtual business processes and greatly increases the overall fault tolerance of the system.

Event management and inter-process communication. Current process specification languages usually support the definition of nested processes, which helps to achieve some modularity by defining blocks of activities as subprocesses. In general, the data and behavior of a process in encapsulated within its own execution scope, preventing other processes from accessing this information until the process terminates. As a result, each subprocess appears necessarily as a black-box, and structuring a process by means of subprocesses greatly decreases the degree of parallelism since progress in the top level process is not possible until the complete subprocess has finished. We have tried to solve this problem in WISE by means of interprocess communications based on event- and rule-based (ECA rules) mechanisms [15, 21, 7]. Our approach differs from standard ECA rule management practice in that we do not use an active database system or a distributed event engine as the underlying platform. We opt instead for an architecture in which interprocess communication is implemented as an additional module of the execution engine [11]. In this architecture, interprocess communication is based on the exchange of events between concurrently running processes. Events are typed and parameterized signals which can be raised by a running activity to inform other processes of certain situations reached or produced during its execution. The parameterization allows to pass arbitrary context data with the event. At runtime, an activity can signal only the events previously registered with the system as part of the activity declaration. In principle, the visibility of an event is limited to the block or process an activity belongs to (similar to local variables in a programming language). To enable inter-process communication, a subscription mechanism is used. This mechanism supports the propagation of events over several levels of nesting. Within its implementation of interprocess communication, WISE also incorporates recovery and exception handling features related to events, in order to deal with the effects of aborted processes and consequently revoked events. These new language primitives enhance the flow-oriented features of process languages with event mechanisms, giving the process designer the option to arbitrarily combine both paradigms. We believe that the interprocess communication facilities implemented in WISE can lead to better designed virtual business processes, since designers do not have to use workarounds like using flat, unstructured process models in order to accommodate the complex information flow between the different parts of a process.

Transactional Processes. Transactions are used to encapsulate database operations so as to provide *Atomicity*, *Consistency*, *Isolation*, and *Durability* [9]. Transactions provide clean semantics to concurrent executions and a powerful abstraction for optimization purposes [1]. We have applied these same ideas within WISE 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 up the overall architecture. However, since we work with processes some form of *light-weight* transactions should be used. Thus, within a process, instead of using a unique construct encompassing all transactional properties, e.g., *BOT*/*EOT*, several separate constructs are used to group activities according to the desired semantics. We 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 [17]), and *spheres of persistence* (determining whether the activities in the sphere are to be made persistent or not). A process is divided into spheres by the user and the system automatically checks whether the specification is cor-

rect. This is specially useful in regard to atomicity since whether the entire process is atomic or can be guaranteed to reach a consistent state depends on the application with which it interacts. These spheres are then used during execution to decide what to do in case of failures, conflicts with other processes, rollback already executed parts of the process, and determine what parts of the process are to be made persistent. Like with exceptions, we rely on user-provided information to decide the atomicity status of each activity within the process and whether two processes or activities within different processes conflict. Unless the user specifies otherwise, the process is assumed to be persistent in its entirety.

3.2 Execution of Processes

Processes are executed in WISE using the WISE internal engine, which is an extension to the OPERA Process Support System [3]. We have invested a great deal of effort in making the WISE engine as robust and reliable as possible and paid attention to functionality without which an E-Comm platform would not be credible: security, execution guarantees and quality of service. In terms of security, we do not expect to come up with new ideas but to use already available tools. The innovation within WISE is the incorporation of execution and quality of service guarantees.

The need for execution guarantees is mostly in terms of atomicity as some other authors have pointed out [5, 20]. Using the notion of spheres of isolation explained above, we can formally analyze the structure of a process and determine whether we will be able to reach a consistent point at any time or there are some execution paths that, once started, are irrecoverable (we cannot rollback, we cannot compensate, and we may not even be able to go forward). This analysis is likely to be very useful for process designers who can then identify problematic steps and try to solve them in a different form.

The quality of service guarantees are based on execution statistics and network characteristics gathered in near-real-time fashion (see below). To avoid unnecessary overhead, we distinguish between three process categories: *critical*, *important*, and *normal*, each one of them with different quality of service guarantees. A critical process uses a RSVP protocol (resource reservation protocol) to guarantee the bandwidth necessary to maintain a given throughput and response time. We will take advantage of the structure of processes and information about the past history to make predictions about the future load in the system and reserve resources as needed. For an *important* process, the engine uses different techniques to guarantee certain maximum delays in the execution. These include increasing the execution priority, attending to load balancing parameters, and providing early warnings if necessary. While these guarantees are not as strong as for critical processes, they still allow to bound the time a process will be delayed. Finally, *normal* processes are executed in a best effort mode, that is, without guarantees. The necessary information for providing guarantees is based on an analysis of previous history and an assessment of the current situation in the system.

WISE incorporates additional functionality to back-up the servers in charge of the execution so that execution can be resumed immediately after a failure, and the ability to dynamically migrate process as they execute from one server to another.

3.3 Using the Process History

The process history (a log with information about the execution of processes) is used within WISE for two purposes. The first one is to create a system model for using in on-line decision making regarding the configuration and status of the system. The second is as the basis of a process history data warehouse which coupled with a process data mining tool will allow to analyze the data off-line.

The intuition behind the system model is that WISE needs to have enough information about what is taking place in the system in order to make informed decisions about configuration, quality of service, resource reservation, load balancing, and so forth. This functionality goes beyond what conventional workflow systems provide, which limit themselves to keep track the status of any process and store the resulting data. The goal here is to generate near-real-time information to be able to asses the state of the system at any point in time. At this stage, we have the necessary functionality to generate this information and gather it in a centralized location. We are currently developing the algorithms implementing load balancing, resource reservation, and quality of service guarantees, which are the ones making more use of this information.

The process data warehouse is based on current ideas regarding business process modeling. Process design is a difficult task and in virtual enterprise environments it will be difficult to foresee all possible eventualities until some sample runs are available. Process design is an iterative procedure where WISE can be of great help by providing accurate measurements of all the characteristics affecting the execution of a process: overall duration, bottlenecks, relative duration of each task with respect to the duration of the entire process, loads at each participant site, deadlines missed, and so forth. For this purpose, we use a *history space* where information about all already executed processes is stored and organized in a way that facilitates its analysis. We refer to this part of WISE as the process data mining tool since it will act as a repository where designers can explore existing data to find out relevant information about process execution.

4 Conclusions

Trading communities, virtual enterprises and virtual business processes are important notions in electronic commerce. The question is how to support these abstractions in a coherent manner. In this paper we have argued that virtual business processes can be interpreted and treated as persistent, coarse grained programming languages for distributed applications. Doing so allows to discuss the necessary functionality based on a common abstraction instead of looking at isolated problems. To prove our argument we have develop an specification for a system supporting E-Comm processes and show how this specification can be implemented in practice using the WISE project as an example. While we do not expect all forms of electronic commerce to be expressed in the form of processes, we believe there are many which could greatly benefit from the ideas here described.

Regarding the status of WISE, the first prototype [2] was finished in 1998. We will proceed with user trials during the second half of 1999. The system, in its current status, allows to define and enact a virtual process across multiple platforms (UNIX, Windows, and OS/2) and geographically separated applications (FlowMark [13], SAP-R/3 [16], OPERA [3], and Structware [12]) linked via Internet. The prototype uses a commercial business modeling tool, Structware, as front end, and an extension to OPERA as the main execution engine. We are currently working on improving the interfaces for general administration of the system. Future work includes incorporating features exploiting Virtual Private Networks (for quality of service purposes), linking the system with a multimedia conferencing tool that will allow the participants in the trading community to communicate with each other using context information about the processes, optimizing several aspects of the execution, and incorporating catalogue services to use a WWW interface for the trading community [14, 2].

Acknowledgments

We would like to thank all the partners in the WISE project for their helpful comments and ideas. More information about WISE can be found in the following URL:
http://www.inf.ethz.ch/department/IS/iks/research/wise.html

References

[1] G. Alonso. Processes + Transactions = Distributed Applications. In *Proceedings of the High Performance Transaction Processing (HPTS) Workshop at Asilomar, California, September 14-17th, 1997.*, September 1997. Also available in MiddlewareSpectra, vol. 11, no. 4, Spectrum Reports Inc./Ltd. and Financial MiddlewareSpectra, vol. 11, no. 4., Spectrum Reports Inc./Ltd (www.middlewarespectra.com).

[2] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. Wise: Business to Business E-Commerce. In *Proceedings of the IEEE 9th International Workshop on Research Issues on Data Engineering. INFORMATION TECHNOLOGY FOR VIRTUAL ENTERPRISES (RIDE-VE'99). Sydney, Australia, March 23-24, 1999.*, March 1999.

[3] G. Alonso, C. Hagen, H.J. Schek, and M. Tresch. Distributed Processing over Stand-alone Systems and Applications. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97)*, Athens, Greece, August 1997.

[4] G. Alonso and C. Mohan. Workflow management: The next generation of distributed processing tools. In Sushil Jajodia and Larry Kerschberg, editors, *Advanced Transaction Models and Architectures*, chapter 2. Kluwer Academic Publishers, 1997.

[5] L. Camp, M. Harkavy, D. Tygar, and B. Yee. Anonymous Atomic Transactions. In *In Proceedings of the 2nd Usenix Workshop on Electronic Commerce, pages 123 - 133.*, November 1996.

[6] D. Flanagan. *Java in a Nutshell*. O'Reilly & Associates, 1996.

[7] A. Geppert and D. Tombros. Event-based distributed workflow execution with EVE. Technical Report 96.5, University of Zurich, 1996.

[8] J.B. Goodenough. Exception handling: Issues and a proposed notation. *Communications of the ACM*, 18(12):683–695, December 1975.

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

[10] C. Hagen and G. Alonso. Flexible exception handling in the OPERA process support system. In *Proceedings os the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, Amsterdam, The Netherlands, May 1998.

[11] C. Hagen and G. Alonso. Beyond the black box: Event-based interprocess communication in process support systems. In *Proc. of the 19th Intl. Conference on Distributed Computing Systems*, Austin, Texas, USA, May 1999.

[12] IvyTeam. Structware'98 Process Manager. Available through http://www.ivyteam.com, 1998.

[13] F. Leymann and D. Roller. Business Processes Management with FlowMark. In *Proc. 39th IEEE Computer Society Int'l Conference (CompCon), Digest of Papers*, pages 230–233, San Francisco, California, February 28 – March 4 1994. IEEE.

[14] H. Lienhard. IvyBeans - Bridge to VSH and the project WISE. In *Proceedings of the Conference of the Swiss Priority Programme Information and Communication Structures, Zürich, Switzerland*, July 1998.

[15] N.W. Paton, O. Diaz, M.H. Williams, J. Campin, A. Dinn, and A. .Jaime. Dimensions of active behaviour. In *Proc. 1st Int. Wshp. on Rules In Database Systems*, pages 40–57. Springer-Verlag, 1994.

[16] SAP AG. SAP business workflow. Available through http://www.sap.com, 1997.

[17] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency control and recovery in transactional process management. In *Proc. of the Conference on Principles of Database Systems (PODS)*, Philadelphia, USA, jun 1999.

[18] G.L. Steele. *Common Lisp: The Language*. Digital Press, 2 edition, 1990.

[19] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 2 edition, 1991.

[20] D. Tygar. Atomicity in Electronic Commerce. *Internet Beseiged, ACM Press and Addison-Wesley. P. Denning and D. Denning, editors.*, pages 389 – 406, 1997.

[21] J. Widom and S. Ceri. *Active Database Systems*. Morgan Kaufmann Publishers, 1996.