Bulletin of the Technical Committee on

Data Engineering

March, 1995 Vol. 18 No. 1 **IEEE Computer Society**

Letters

Special Issue on Workflow Systems

Conference and Journal Notices

Letter from the Editor-in-Chief

Status of the Bulletin

The current issue of the Bulletin is a late because of rather large changes in the lives of editors of the Bulletin. Mei Hsu, the issue editor, assembled the issue under tight deadlines and in a completely new work environment as she has recently moved to EDS. I too changed jobs recently, moving to Microsoft in January, and preparing the mechanisms for continued production of the Bulletin here involved substantial systems efforts. Kendall Martin of Microsoft Research's technical support group deserves special thanks for finding the relevant LaTex, dvi, and postscript software and helping with its installation. The current issue of the Bulletin was prepared entirely on a PC workstation running Windows/NT. The result of these major changes is that the Bulletin is about a month late, or perhaps I should say, only a month late.

The Bulletin continues publication under the auspices of the IEEE Computer Society. And the Computer Society has on several occasions taken steps to see that hardcopy publication continued. Unfortunately, no long term structural arrangement for publishing the Bulletin has been found. So it is a sad fact that each issue might be the last, at least in hardcopy form. Electronic distribution of the Bulletin can continue, though the precise details are changing. In the future, "ftp" will be the technique used to receive the Bulletin. Announcements will continue to be sent via email. But issue distribution is switching away from Digital's CRL tech report server.

To receive issues of the Bulletin beginning with this March, 1995 issue (vol. 18, no. 1), log on to

```
ftp.research.microsoft.com
```

as "anonymous" and give as your password your email address. Change directories as follows-

```
cd  pub
cd debull
```

You can then do a "dir" or "ls" to determine what files are present. Read the README file in the debull directory for more complete information.

Having selected the issue you wish, and the format you desire, select the appropriate file with the ftp "get" command, e.g.

```
get mar95-letfinal
```

to have the file delivered to your system. The files are all in postscript so you will need a postscript viewer and/or printer to be able to read them.

The March issue returns to the theme of workflow, an area that should be of particular interest to database folks. It presents an interesting area that can benefit from the introduction of database technology. One of the strengths of the current issue is geographic spread, range of institutions, and where on the research to product spectrum the articles sit. The articles thus present an interesting cross-section of the state of the art. Mei Hsu deserves extra thanks for succeeding in assembling this interesting issue while the world changed around us so dramatically.

David Lomet
Editor-in-Chief

Letter from the Special Issue Editor

A number of business and technical trends of the last few years have attracted researchers and developers to the area of workflow systems. As businesses feel competitive and economic pressures to automate and streamline their operations, they examine their business processes and look to business process reengineering (BPR) practices to provide solutions. Increased adoption of and migration to open systems have created an opportunity for businesses to examine their Information Technology (IT) practices and look for ways to coordinate and automate the flow of work between people and groups in an organization. Workflow systems have surfaced as a foundation on which to build solutions supporting the execution and management of business processes.

Workflow systems have a complementary relationship with database systems. The database research community has long pursued the notion of extended transaction models and is exploring the applicability of such models to business processes (also known as "business transactions"). Workflow systems require an infrastructure for reliably capturing the states of business processes and tying together distributed applications, and DBMSs have been found by many to be the most natural choice for this infrastructure . As workflow systems scale to the enterprise level, the requirements on robustness, scalability and distributability are likened to those required and developed in TP Monitors and distributed DBMSs.

This Special Issue examines some current research and development activities in workflow systems, with a focus on infrastructure requirements and leverage. It also examines trends in business process design technology to bridge the "gap" between business process design and workflow automation.

Schwenkreis and Reuter concisely summarize their perspective on workflow system functional requirements (and as implemented in their prototype APRICOTS) into two critical points: a rich data model for representing process context data, and an invariant-based model to characterize correctness of state transitions in a process. They argue succinctly why the isolation property in the classic transaction model cannot be realized in a business process (i.e., a business transaction), and why new models must be d efined for workflow systems.

The article on TriGS_{flow}, by Kappel, Lang, Rausch-Schott, and Retschitzegger, presents a novel approach which utilizes an active object-oriented DBMS to implement a workflow system. The GemStone-based active DBMS provides the object-oriented environment for organizing workflow descriptions with classes and inheritance (e.g., agents, workflows, activities), and it provides the triggering mechanism needed during workflow execution.

The article by Mohan, Alonso, Gunthor, and Kamath outlines the on-going research at IBM's Almaden Research Center that focuses on robustness and scalability of workflow systems. They approach the problem by adapting concepts and techniques developed in database and distributed systems research to workflow systems. They use FlowMark, an IBM workflow system product, as their test bed.

The article by Davis, Du and Shan describes the architecture and design principles of the Open Process Manager (OpenPM) under development at HP Labs. OpenPM uses HP's Work Manager product as a base line, and aims at providing an open platform to integrate different kinds of business activities and to automate activity execution.

Chan and Harty presents an overview of the Teknekron Information Bus (TIB) and its associated toolkit, and how this messaging infrastructure has been used to build production workflow applications that require real-time communication among applications. TIB is one of the few commercially available, high performance event- and data-flow infrastructures based on the publish-subscribe model of communication paradigm. Teknekron is yet to offer a busines process definition and execution model such as those off ered by workflow systems. However, the TIB-enabled workflow applications have unique and powerful features that are likely to influence future directions of workflow systems.

The article by Bernstein, Dellarocas, Malone and Quimby describes the Process Handbook project which focuses primarily on supporting business processes modeling from the point of view of a business process designer. Its approach combines inheritance hierarchy, decomposition hierarchy, and explicit dependency modeling. The authors consider the Process Handbook to be a CAD tool for business process design. To streamline workflow

application development, it is crucial to allow the output of a business proces s CAD tool, such as the Process Handbook, to be systemantically refined into the busines process model executable in workflow systems.

This Special Issue brings forward a range of approaches to building enterprise-wide workflow systems, as well as perspectives on tools and models for business process design and re-engineering. Advanced workflow systems are focused on system attributes such as programmability, performance, robustness, scalability, and infrastructure leverage, while advanced business process design systems are focused on business attributes of processes, alternative process structures, and their presentations which facilita te intelligent design decisions. We believe that a close link between application system design and business process design, and automating the business processes through the use of robust, scalable, infrastructure-based workflow systems, hold a great deal of promise in shaping the business application development of the future.

I wish to thank all the authors for their generous contribution. I would like to espeically thank Dave Lomet for his patience and support throughout the editorial process without which this Issue would not have been possible.

Mei Hsu
EDS Management Consulting Services
3945 Freedom Circle
Santa Clara CA 95054

ConTracts - A Low-Level Mechanism for Building General-Purpose Workflow Management-Systems

Andreas Reuter *Friedemann Schwenkreis*
Institute of Parallel and Distributed High-Performance Systems (IPVR)
University of Stuttgart
Breitwiesenstr. 20-22
D-70565 Stuttgart, Germany
email: {Andreas.Reuter, Schwenkreis}@informatik.uni-stuttgart.de

1 Introduction

The ConTracts project was started to investigate the possibilities of handling long-lived computations in a database context. Note that the emphasis on databases is not a restriction. It simply acknowledges the fact that all long-lived computations - by their very nature - will accumulate a certain amount of state, will access shared data and will interact with the environment and with other long-lived computations. In other words, they perform a large number of activities that are properly supported by database systems. Hence, the discussion of long-lived computations and the definition of the role of databases in that context cannot be separated.

At the time the project started, there were some approaches we could refer to. The first line of suggestions tried to generalize classical ACID transactions by adding more features, giving them more flexibility, etc. Proposals of this sort are sagas, flex transactions, split transactions, to name a few [1][4]. Another group of proposals took a much more pragmatic perspective by defining extra functions in existing TP-monitors that allow to chain two subsequent transaction programs, for example. Some database systems support chain transactions, there are mini batches [3], and a number of other ideas to try and work around the problem.

Our approach, called ConTracts, which will be described in this paper, is characterized by three very simple insights:

- Extending the transaction model does not solve the problem, because a long-lived computation is much more complex than an ACID transaction. On the other hand, nobody is helped by a type of transaction, which is neither A, nor C, nor I, nor D.
- Managing long-lived computations requires more systematic approaches than pragmatic extensions to existing operating systems and TP-monitors.
- Apart from its close relationship to databases, handling long-lived computations is a problem of proper language abstractions. Whereas simple modules (and transaction programs, for that matter) have a system residence time of 10^{-1} sec, a long-lived computation can be around for a number of years; i.e. something in the order of 10^{8} sec. Quite obviously, phenomena which are 9 orders of magnitude away from simple modules, need different programming techniques and different styles of talking about consistency, for example.

Starting from these considerations, we have developed a programming model called ConTracts [10] and a prototype run-time system to go with it, called APRICOTS. The functionality provided covers the whole range of requirements that are meanwhile discussed under the heading of “workflow management”.

2 Critical points

In this article, we will focus on the concepts which are not yet addressed by other approaches.

First the usage of (nested) transactions in ConTracts has a direct impact on the following discussion and it is necessary to say a few words on this topic. It is an important feature of ConTracts that every *Step* of an execution is running under the protection of a classical transaction and that several of these *Steps* can be grouped to run inside the same top-level transaction. Additionally, there is a guaranteed forward recovery which ensures that once started, a ConTract never gets lost or stuck in an inconsistent state. Instead, it will either be executed to its "normal" end or reset to an acceptable state by a compensating mechanism (if requested by the user).

2.1 Requirements for the data model

The data model for workflow systems has not yet been discussed in the literature, though it is an accepted fact that there are special requirements for a data model which supports long-running processes [2]. We believe the following are among the requirements for workflow systems:

- Users are very interested in examining the history of workflows. That means that they want to be able to get information of executed processes regarding, for example, information about "which path was taken and why?" or "who has done what?" (support of history keeping and examination).

- Since *forward* recovery is a very needed feature in workflow systems the information needed to continue an execution after a failure must be easily accessible and as persistent and recoverable as the process itself (recovery support).

- At the definition or programming level, nobody wants to use the "low-level" notation of the (workflow) engine. Instead, a high-level programming mechanism is needed (either graphical or textual) which offers well-known programming constructs like oops, branches etc. Most of these classical programming expressions are using "program variables", i.e. loop counters or variables to define predicates of branches, which have to be easily depictable on the data model too.

  Another problem of using "high-level" programming constructs appears in combination with the need to support the history-keeping of a process (either for examination by the user or for a compensating mechanism). Since the user usually defines a single variable to be used, for example, as a loop counter which will be overwritten every time the loop is executed, the system is unable to examine the variable's state at an arbitrary point of the execution. This problem can also be solved by a proper data model (support of high-level programming).

- Especially in case of parallel executions it is very important to have the capability to define an explicit data flow or "data piping" from one *Step* to another *Step*. That means that a proper data model has to provide an explicit addressing method which can be used at the user level (explicit addressing).

2.2 Isolation requirements

Similar to the requirements for the data model, the long duration of workflow processes causes special requirements for isolation. The classical transactional approach of establishing locks on touched objects is inapplicable for processes which are possibly running for months. Furthermore, it is in most cases unacceptable to abort or simply delay an execution if a concurrency problem occurs. Considering applications of the real world, the following requirements can be identified:

- Total isolation is neither required nor meaningful for a workflow process.

- Real applications need a flexible mechanism to express their isolation needs properly.
- Conflicts caused by isolation problems (in most cases) cannot be solved by an automated system mechanism. Instead, explicit conflict handling by the application itself has to be supported.
- Due to the fact that so-called "groupware" systems should be integrated with workflow systems concurrency control mechanisms for workflow systems must be powerful enough to support cooperative work.
- As a result of the points above, the correctness criterion for workflow processes cannot be serializability. Instead, a new criterion has to be defined on which proper mechanisms can be built.

3 Ideas and approaches

Motivated by the requirements introduced in the section before, a lot of work has been done to define a proper data model and powerful concurrency control mechanisms for the ConTract model and its prototypical implementation APRICOTS. A short overview of the concepts is presented.

3.1 Context - the data model of ConTracts

As introduced in [10] the definition of a *Script* contains the declaration of "ConTract-local" variables - *context elements* which built up the *context*. These variables are kept in stable transactional storage and are only accessible from the ConTract in which they are defined.

One of the major characteristics of *context elements* is the fact that their value can only be affected by *Steps*. This means that only *Steps* of a ConTract can read or modify the value of *context elements*. Another important feature is the transactional protection of *context elements*. Since the *Steps* are running under the protection of a transaction, their modifications of the *context* are protected by this transaction. This results in the effect that changes to *context elements* can always be associated with committed *Step-instances*.

To support the examination of the history, *context elements* are stored in an update-free way, i.e. every modification of a *context element* does not overwrite the old value of the *context element*. Instead, a new *version* is generated which represents the actual value of the variable. This approach can also be found in database systems supporting the so-called *time domain addressing* [9][8][6]. Since the examination of the history of a workflow process is usually done based on the process definition and not on a time basis, the access of the different versions of *context elements* on a time basis is not very helpful. Furthermore, the support of high-level programming constructs cannot be solved by a time-based access mechanism.

During the investigation of the requirements it was found out that the different ways of addressing *context elements* are independent from each other. From an abstract point of view this means that a multi-dimensional addressing schema is needed. Since the *time domain addressing* approach represents only a two-dimensional approach (identification and time) it is not sufficient.

Our investigation of high-level programming constructs has concluded that the following dimensions are needed:

- *Sequential Steps*: New variable versions are created by sequential execution of *Steps*.
- *Loops*: Another dimension can be found when loops are executed.
- *Parallel Branches*: The third identified dimension considers the case of parallel branches generated by the run-time system to realize, for example, a *parforeach* statement.

Since we store the *context* in a transactional storage which is used by several ConTracts two more dimensions are needed to address a *context element*: the identification of the running ConTract instance and the name of the *context element*.

The five dimensions introduced above are the basis for the addressing schema which is used to store and access elements of the *context*. That means to access a *context element*, a 5-tuple (variable-name, contract-id, step-id, loop-counter, parallel index) in which we assume that the contract-id identifies the ConTract instance in the system and the step-id identifies the creating *Step-instance* as defined in the *Script* of a ConTract.

Since the last four elements can be seen as the unique identification of the step-instance which created the version of the *context element*, we can compress the schema to (variable-name, step-instance-id). However, the user will access the context indirectly providing the same information.

A short example will show the usefulness of the introduced concept:

- A programmer defines (implicitly) a parallel branch in a *Script*:
  `parforeach person of S step_send_call_for_papers(receiver = person)`
  where person is a string variable and S is a set of strings defined in the *context*.
- At execution time a so-called *Pseudo-Step* is executed which generates as many versions of the *context-element* (person) as there are elements in S. These versions can be identified by the *parallel index*.
- Then as many versions of the *Step* as elements in S are generated which will be visualized on the user interface. They are also identified by the *parallel index*.
- Even if all of the visualized *Steps* have the same name and all of the input variables have the same name the semantics are unique because of the assigned *parallel index* which can be used from now on for recovery or history examination.

The addressing schema of *context-elements* is also used at the programming level, i.e. programmers assign *context-elements* to the parameters of *Steps*, for example, parameter1 = *context element*. Several defaults are provided to make the programming of the *data flow* easier:

- The current parallel index is set automatically and can be used by a *special name*.
- The current loop-counter can be used by a *special name*.
- Other *Steps* can be referenced by using their identification (also supported by the graphical user interface).
- If only the variable name is given the newest version is used.

3.2 Invariant based synchronization

As said before, serializability is not a good correctness criterion for long-running processes like workflow. Hence, the correctness of ConTracts had to be newly defined [7]. The first step to define a new correctness criterion was to investigate the users' view of correctness which resulted in the observation that there are at least three states of a process which are correct:

- The state when a process has never been started: *initial state*
- A state which is defined equivalent to the *initial state*: *compensated state*
- The state of the process when it is successfully executed: *final state*

Since it is necessary to avoid total isolation in workflow processes, preliminary results will become visible outside of an executing process. Hence, the transactional approach of atomicity (all or nothing) cannot be realized because a *rollback* mechanism requires the isolation property. Therfore, other mechanisms have to be used to "undo" the work done by a process if it is necessary to cancel the execution of a process: we call these mechanisms *compensation*. To *compensate* in the case of ConTracts means that for every committed *Step* a so-called

Compensation-Step has to be executed which represents the (semantic) counter-action of the original *Step*. If all necessary *Compensation-Steps* have been executed successfully the *compensated state* is reached.

A correctness criterion for a ConTract can then be defined as follows:

Definition 1: The execution state of a ConTract is correct iff:

1. The ConTract is in an *initial state* or
2. The ConTract is in a *final state* or
3. If neither 1 nor 2, then for every committed *Step* the execution of its *Compensation-Step* is ensured.

To ensure the correctness of ConTracts according to the definition above, it is assumed that the prerequisites to execute a *Compensation-Step* are known when the corresponding *Step* has been executed. This means that after the execution of a *Step*, constraints can be established to ensure a state of concurrently accessed objects which in result guarantees the executability of the *Compensation-Step*. The constraints defined after *Steps* are called *Invariants*.

In detail, the mechanism works as follows:

- The programmer assigns predicates with *Steps* in the *Script* which represent the *Invariants*
- At execution time the run-time system checks at the end of the execution of a *Step* if the predicates are valid. If they are valid, the constraint is established and the transaction which protects the *Step* is allowed to commit.
- If the constraint cannot be established the ConTract-Manager is informed. That means the ConTract-Manger gets information about an isolation conflict and is able to initiate actions to resolve the conflict.

As introduced in the original model, *Invariants* can also be used to support the executability of a ConTract. For example, a *Script* can contain two *Steps*: checking a flight and the reservation of a flight. Instead of grouping those two *Steps* under one transaction and restricting the access on the flight database more than needed, a predicate can be established after the first *Step* which expresses the pre-reservation of a certain amount of flight seats. This ensures the executability of the second step without restricting the accessibility of other seats. On the other hand, this is not a real *Invariant* of the whole ConTract because the constraint can be removed after the second *Step* has committed. Hence, another "class" of *Invariants* has been defined which supports the executability of a ConTract instead of ensuring correctness.

Both ways to use *Invariants*, to ensure correctness and to support the executability, provide a powerful mechanism to express the isolation requirements of a process. Furthermore, they allow us to release unnecessary access restrictions as soon as possible. For example, it was found out that there are many cases where only the existence of an object has to be guaranteed, while other arbitrary accesses can be allowed.

Another important feature of our isolation concept is the explicit handling of problems caused by a conflict on shared resources. There is built-in direct support for defining special parts in the *Script* which should be executed if the evaluation of a predicate fails. Additionally, in case of a conflict with an *Invariant* defined to support the executability of another process, negotiations can be supported (e.g. to grant access temporarily).

4 Summary and prospect

Two remarkable features supporting workflow in the ConTracts project have been presented in this paper: advanced data models and advanced concurrency control mechanisms.

The data model of ConTracts supports the usage of high-level constructs and the examination of a process' history as requested by users. It is not sufficient to provide them only with transactional storage. Since the data

objects used by workflow processes are getting more and more complex (e.g. multi-media documents), the evolution of such an object is very important and the information about which activity has modified the object in which way must be provided by workflow systems.

The isolation concept used in ConTracts is a powerful means to express the isolation needs of workflow processes. It was built to allow processes to release access restrictions on shared objects as soon as possible without violating their correctness. Therefore a new correctness criterion has been introduced which allows parallelism far beyond serializability. Furthermore, the design allows us to handle conflicts explicitly instead of reacting to a conflict by simply aborting the current transaction.

Although the prototypical implementation of a ConTract system (APRICOTS) [5] has been under development for more than three years now, not all features have been completely implemented yet. We are currently working on the efficient implementation of the concurrency control concepts. Additionally, we are looking at the problem on how existing transactional environments such as Encina from Transarc can be enhanced to support transactional workflows like ConTracts directly. Likewise, we are looking at database systems and other "resource managers" to find out on how they can be leveraged to join a ConTract environment.

References

[1] Ahmed K. Elmagarmid (ed.), Database Transaction Models for Advanced Applications, Morgan Kaufmann Pub., 1992

[2] U. Dayal, M. Hsu, C. Mohan, M. Rusinkiewicz, F. Schwenkreis, Panel Session: Workflow Automation, International Conference on Data Engineering, Taipei Taiwan, 1995

[3] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, San Mateo, Calif. 1993

[4] M. Stonebraker (ed.), Readings in Database Systems 2nd edition, Morgan Kaufman Pub., 1994

[5] F. Schwenkreis, APRICOTS- A Prototype Implementation of a ConTract System: Management of the Control Flow and the Communication System, Proc. of the 12th Symposium on Reliable Distributed Systems, 1993

[6] B. Salzberg, D. Lomet, Access methods for multiversion data, Proc. of the ACM Intern. Conf. on Management of Data SIGMOD, 1989

[7] F. Schwenkreis, A Formal Approach to Synchronize Long-Lived Computations, Proc. of the 5th Australasian Conference on Information Systems, 1994

[8] M. Stonebraker, The Design of the Postgres System, Proc. of the 13th Intern. Conf. on Very Large Data Bases, 1987

[9] L. Svobodova, A Reliable Object-Oriented Data Repository for a Distributed Computer System, Proc. of the 8th Symp. on Operating Systems Principles, 1981

[10] Helmut Wdchter, Andreas Reuter: The ConTract Model, chap. 7 in [1]

Workflow Management Based on Objects, Rules, and Roles

G. Kappel P. Lang S. Rausch-Schott W. Retschitzegger
Department of Computer Science
University of Linz, A-4040 Linz, Austria
email: {gerti,peter,stefan,werner}@ifs.uni-linz.ac.at

Abstract

Workflow management systems are becoming important not only as key technology in their own right but they are gaining leverage as a key application of database management systems. One of the main issues of workflow management systems, and thus implicitly of database management systems, is their support for flexibility to cope with frequently changing requirements in an organization. This has been the driving force behind the development of the workflow management system TriGS_{flow}*. It is based on an object-oriented database management system, enhanced with active concepts in terms of ECA rules and object evolution in terms of roles. In this way, flexible modeling and enactment of business processes is supported allowing changes even during workflow execution. The architecture of* TriGS_{flow} *and its key concepts for workflow modeling are presented.*

1 Introduction

Workflow management systems (WFMS) have been introduced to support the design, execution and monitoring of business processes. A business process a.k.a. workflow consists of various activities which have to be executed in some (partial) order involving multiple collaborating persons to fulfill a certain task of an organization. Reimbursement of business trips and reorder of goods are just two examples of workflows in some business organization.

A WFMS has to support both the modeling and the enactment of workflows [6]. The modeling phase is concerned with the specification of a workflow, i.e., the specification of activities to be executed, data to be processed, and agents to be involved in activities. Agents are both persons and automatically executing software processes. The enactment phase is responsible for scheduling, executing, and monitoring activities to ensure correct workflow execution.

One of the most challenging issues of WFMS is to provide concepts allowing flexible reaction to changes during workflow enactment. To achieve this, we have integrated three basic technologies. Firstly, we use *object-oriented database technology* [2] to build a WFMS providing both database functionality and possibilities for modeling and reusing complex business domain objects [7]. Secondly, to cope with changes of the personnel, *roles* have been integrated into the object-oriented environment separating activities from particular persons executing the activities. Thirdly, those parts of a workflow subject to frequent changes, such as the ordering of activities, are modeled by using *ECA rules*, which may be adapted dynamically.

The remaining sections describe the architecture and implementation aspects of TriGS_{flow} and emphasize on the use of roles and rules for workflow modeling. Future directions are outlined in the conclusion.

2 System Architecture and Model

In the following we describe the architectural components of the TriGS_{flow} prototype. Figure 1 illustrates the overall architecture of TriGS_{flow}. It has been implemented as a layer on top of the object-oriented database system $\mathrm{GemStone}^{TM}$ extended by the component TriGS providing ECA rules and a module providing roles. TriGS_{flow} is realized as a class library consisting of only 18 base classes. These classes devise a generic workflow model illustrated in Figure 2 and explained step by step in the next section.

Currently, two frontend tools named MasterTOOL and AgentTOOL are being developed providing a graphical user interface for the modeling phase and the enactment phase, respectively. We will now describe each layer of the architecture in more detail.

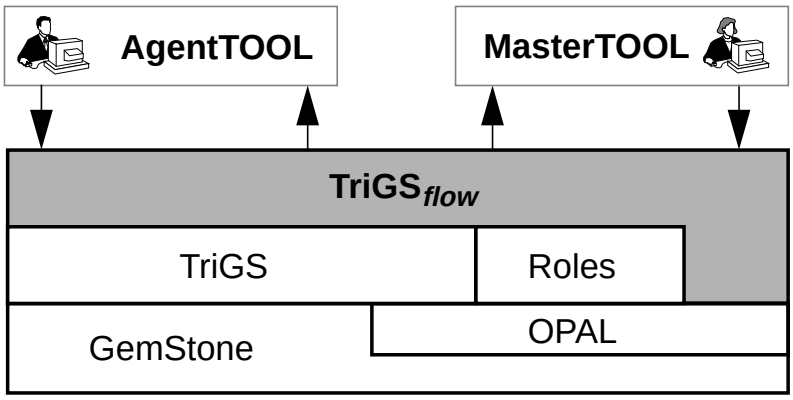

Figure 1: Architecture of TriGS_{flow}

GemStone [3] is a commercially available object-oriented database system. Its data model is based on the $OPAL^{TM}$ programming language, which is a variant of $\mathrm{Smalltalk80}^{TM}$. In addition to Smalltalk, OPAL provides some predefined classes in order to realize database functionality. Since TriGS_{flow} is implemented on top of GemStone database functionality such as concurrency control and recovery comes for free.

The *Roles* module used in our prototype is based on the work of [8]. Traditional class-based object-oriented systems are characterized by an immutable linkage between an object and its object class, i.e., an object always belongs to exactly one class. In order to extend or change the behavior of an object, a new instance of the corresponding class has to be created, and the attribute values have to be transformed from the old to the new instance. The concept of roles (e.g., [1, 8, 12]) eliminates these drawbacks by allowing object evolution during runtime permitting objects to learn and forget roles dynamically and to play several roles at the same time. In TriGS_{flow}, an agent and its roles are represented as instances of several distinct object classes (cf. subclasses of class `ObjectWithRoles` in Figure 2). These instances are linked to each other hierarchically via the *roleOf* relationship (instance variable `roleOf` in class `Role`). This hierarchy is called the role hierarchy and can be specified orthogonal to the class hierarchy. In addition to inheritance at the class level via the class hierarchy, the role hierarchy supports inheritance at the instance level. A more specific role instance inherits the attributes *and* their values from the roleOf related more general objects. Similarly, it inherits the methods defined for a roleOf related more general object. Since agents may play the same role simultaneously several times, roles can be further qualified by objects of a specified class (cf. class `QualifiedRole` with attribute `qualifyingObj`). The application of the role model in TriGS_{flow} will become apparent in subsection 3.1.

The active component of TriGS_{flow} is based on *TriGS* (=Trigger System for GemStone) [10], which implements Event/Condition/Action rules (*ECA rules*) [5] on top of the object-oriented database system GemStone. TriGS rules monitor the behavior of objects and can be attached to specific classes or defined independently of any class hierarchy. Rules can be (de)activated at different levels of granularity ranging from the object instance level to the object class level. Moreover, since rules and its components are first-class objects they can be dynam-

ically defined, modified, and extended independently of any application. TriGS_{flow} employs rules for activity ordering (subsection 3.2), agent selection (subsection 3.3), and worklist management (subsection 3.4).[1]

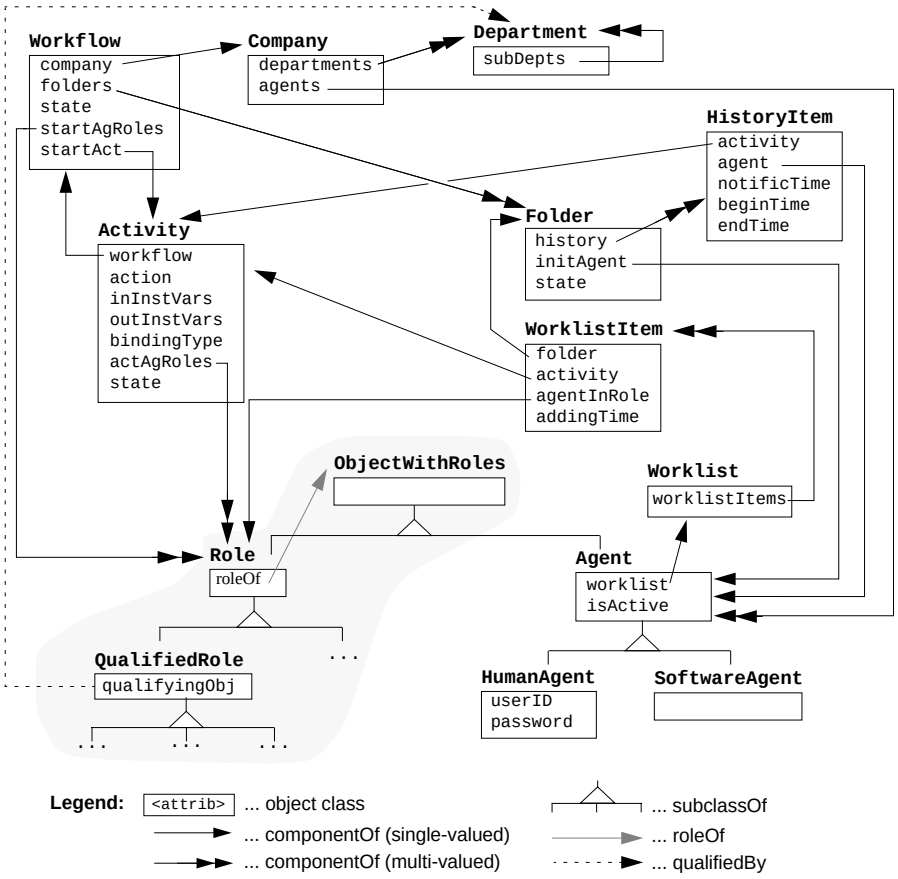

Figure 2: Generic Workflow Model (selected parts)

[1] Part of the work on TriGS has been carried out while S. Rausch-Schott and W. Retschitzegger have been with the Research Institute of Applied Knowledge Processing (FAW Linz, Austria).

3 Workflow Modeling

The workflow model of TriGS_{flow} is generic in the sense that different kinds of workflows (cf. class `Workflow` in Figure 2) such as reimbursement of business trips or reordering of goods are modeled by simply specialising and instantiating the corresponding classes of the generic workflow model. Workflow modeling comprises the following steps:

- modeling the organizational structure consisting of departments, agents, and roles
- specifying the ordering of activities (control flow) by using activity nets
- assigning agents to activities by employing agent selection policies
- specifying agent communication (data flow) through the activity net by using folders and worklists

These modeling steps will be explained in detail in the next subsections.

3.1 Organizational Structure

The first step of workflow modeling is to specify the organizational structure of the business organization under consideration. For this task, the generic workflow model provides the predefined object classes `Department`, `Agent`, `Role`, and subclasses thereof.

A business organization is subdivided into several *departments*, which again may consist of subdepartments. An *agent* represents an autonomous unit able to perform different activities. Agents are classified into human agents (`HumanAgent`) and software agents (`SoftwareAgent`). The former represent persons working in the business organization. Activities performed by human agents are processed in an interactive manner. The latter represent software artifacts, which have a pendant in the real world, like a fax machine and its driver software, or which solely exist as software processes, like a calculator. In contrast to human agents, software agents are able to perform their activities without user interaction. Every agent is assigned to one or more *roles* describing his/her context-dependent behavior, thus representing context-dependent duties and competences of the agent in the business organization. Note, in TriGS_{flow} roles may be played not only by human agents but also by software agents.

Let us illustrate the modeling of a particular organizational structure by an example (Figure 3). A company is subdivided into the three departments `purchasing`, `production`, and `sales`. `production` is further divided into the departments `cutting` and `assembling`. `Employee` is a subclass of `HumanAgent` and the root of a role hierarchy. An employee may play different roles, i.e., he/she may work as system administrator (`SysAdmin`), as clerk (`Clerk`), and as foreman (`Foreman`). In our example, Jeff and Joe are working as clerks. As stated in Section 2, the behavior of roles is determined by their methods *and* the methods of the roleOf related more general objects. Thus, both Jeff and Joe are able to get offers and select suppliers as clerks, and to quit their work as employees (cf. method specifications in the respective classes). `Clerk` and `Foreman` are qualified roles with the qualifying class `Department`, i.e., whenever they are instantiated the instances have to be attached to one of the departments. Mary has different positions at the same time - she is working both as system administrator and as foreman. In addition, she works as foreman both in the cutting department and in the assembling department. Finally, a software agent is used as a fax driver.

3.2 Activity Ordering

Activities (cf. class `Activity`) have to be executed in a certain order determined by their relationships within an activity net. These relationships comprise basic control structures, namely sequencing, branching, and joining. More precisely, we distinguish three different kinds of branching, namely, exOR-Branching, AND-Branching,

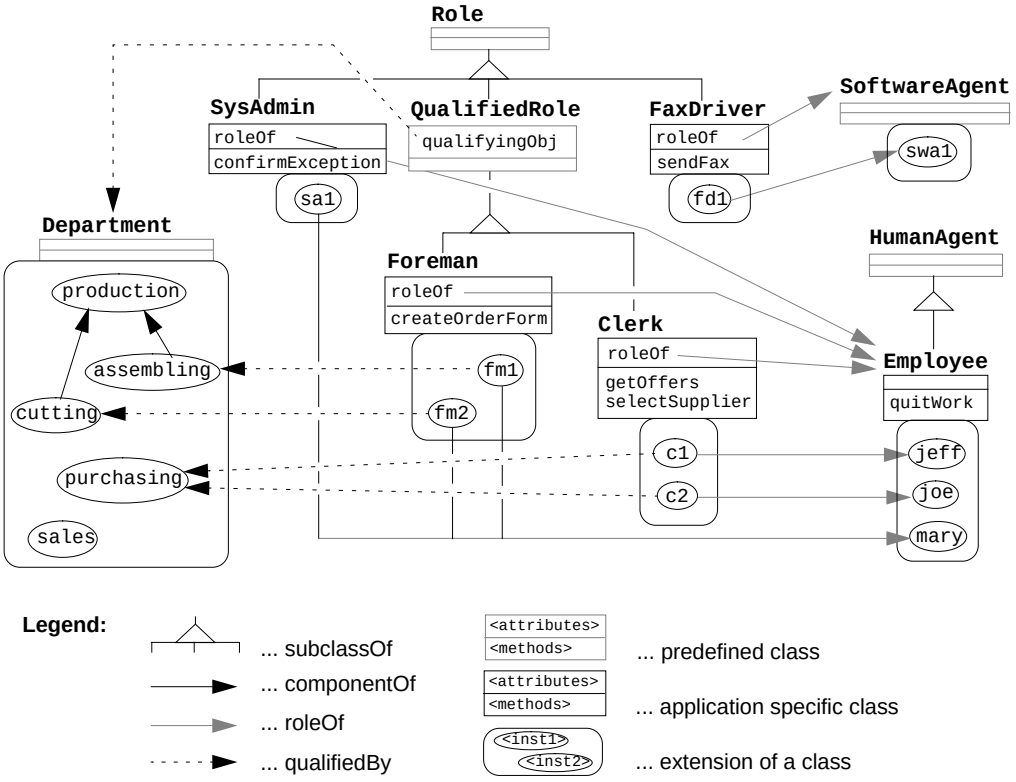

Figure 3: Organizational Model and Sample Instantiation

and inOR-Branching. If two or more parallel branches have the same successor activity joining has to be supplied. This can be specified by an AND-Join, an exOR-Join, and an inOR-Join, respectively. More complex control structures, e.g., iteration can be built by combining these basic control structures. For a detailed discussion of these control structures see [11].

Considering the realization of an activity net, all relationships between activities are mapped to ECA rules according to the basic control structures. The *event* of an activity ordering rule always constitutes the end of a preceding activity, or - in case of an AND-Join and of an inOR-Join - of all participating preceding activities, respectively. The *condition* checks whether the succeeding activity has to be performed or not. This is done by evaluating queries against the actual folder and/or further information concerning the workflow up to that point, e.g., which agent(s) performed former activities. In case of an inOR-Branching the result sets of the conditions may overlap, whereas an exOR-Branching is reflected by mutual exclusive conditions. AND-Branching is realized by using the same condition in all branches. Note that an exception handling mechanism has to handle the case that all conditions evaluate to false implying that no succeeding activity can be executed [6]. The *action* of an activity ordering rule has to notify the agent responsible for performing the succeeding activity. The method `notifyAgent` puts a corresponding worklist item for the succeeding activity into the worklist of the agent chosen to perform this activity (for selecting agents see Section 3.3). The method `perform` starts the execution of the activity no matter whether a human agent or a software agent is engaged. In case of a human agent it waits until the human agent signals the end of the execution. Every rule responsible for activity ordering contains the methods `perform` and `notifyAgent` in its event part and action part, respectively.

Consider the following two rules, realizing an exOR-Branching between the activities “get offers”, “select supplier”, and “create order form” (cf. Figure 4 depicting part of a reordering workflow done in the purchasing department). The two rules named `R_GetOffers` and `R_SelectSupplier` are activated for the same instance of class `Activity`, namely `a_create_order_form`. That is, after the method `perform` has been executed on that

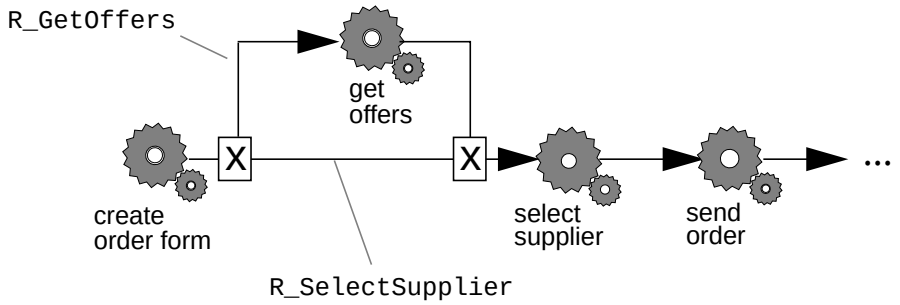

Figure 4: Part of an Activity Net

instance (denoted by the keyword POST), both rules are triggered. The condition of the rule named R_GetOffers checks if offers are missing or no longer up-to-date. If the condition evaluates to true the action is executed by sending the message notifyAgent to the instance a_get_offers of class Activity representing one of the mutual exclusive successors of the activity a_create_order_form within the activity net (cf. Figure 4).

```
DEFINE RULE R_GetOffers
     ON POST (Activity, perform: aFolder) DO
     IF Offers notUpToDateFor: ((aFolder at: 'Order') positions) THEN
     EXECUTE a_get_offers notifyAgent
ACTIVATED FOR (a_create_order_form).
```

The condition of the rule named R_SelectSupplier is complementary to the condition of R_GetOffers. If it evaluates to true no more offers are required. Therefore the supplier can be selected immediately by sending the message notifyAgent to the instance a_select_supplier of class Activity representing the second successor of the activity a_create_order_form.

```
DEFINE RULE R_SelectSupplier
     ON POST (Activity, perform: aFolder) DO
     IF Offers areUpToDateFor: ((rule_trgO aFolder at: 'Order') positions)
THEN
     EXECUTE a_select_supplier notifyAgent
ACTIVATED FOR (a_create_order_form).
```

3.3 Agent Selection Policies

During workflow enactment, the assignment of activities to agents is done on the basis of agent selection policies [4] using actual data. An example of an agent selection policy is that agents on holidays are not allowed to be selected. Agents actually performing a certain activity are called *actual agents* (instance variable actAgRoles in class Activity). They are selected out of a set of agents able to perform the activity called *possible agents*. Possible agents are implicitly specified by mapping activities to capabilities of roles realized by the instance variable action of class Activity. For sake of simplicity the discussion in this paper is focused on a single actual agent. Agent selection policies are implemented by means of ECA rules allowing immediate reaction to state changes of some agent, e.g., a machine breakdown. Furthermore, it is possible to realize different selection policies and to switch between them dynamically by simply activating and deactivating the corresponding rules at activity instance level. For example, the following rule named R_MinWorkload realizes a minimal workload policy.

```
DEFINE RULE R_MinWorkload
     ON PRE (Activity, notifyAgent) DO
```

```
     IF (rule_trgO possibleAgents) selMinWorkload THEN
     EXECUTE (rule_trgO actAgRoles) removeAll; add: QUERY_RESULT
ACTIVATED FOR (a_select_supplier).
```

The event of `R_MinWorkload` is signaled before the method `notifyAgent` of class `Activity` is executed (denoted by the keyword `PRE`). The condition determines the set of actual agents on the basis of possible agents (`possibleAgents`) and according to a specific selection policy, e.g., choose the agent with the lowest workload (`selMinWorkload`). Analogous to the condition of activity ordering rules this is done by means of queries against the actual folder and/or other data concerning the actual workflow. The selected agent is passed to the action of the rule (cf. keyword `QUERY_RESULT`). The action of the rule assigns the query result to the instance variable `actAgRoles` of the triggering object.

3.4 Worklist Management

Besides the specification of the execution order of activities, data exchange between agents is necessary for realizing cooperation. Data exchange is realized by worklists (`Worklist`) associated with agents. If an agent is requested to perform an activity a new worklist item (`WorklistItem`) is inserted into his/her/its worklist. This worklist item specifies the requested activity and includes a reference to the folder (`Folder`) containing the data necessary for performing the activity. A new folder is created on every start of a particular workflow execution. This can be done by any agent explicitly authorized to start the workflow (cf. instance variable `startAgRoles` of class `Workflow`in Figure 2). The flow of this folder from one agent's worklist to another is controlled by the activity net. If more than one successor activity is performed concurrently, a reference to the folder is transferred to each responsible agent. Concurrent folder access of concurrently working agents is handled by the transaction mechanism of GemStone. Note, that duplicating folders for concurrent access might also be useful in some cases. However, duplication of folders involves version management, which is not treated in this paper.

The folders waiting inside the worklist of a specific agent are processed on the basis of ECA rules. They monitor, for example, the worklists of software agents and start processing the next folder every time a software agent becomes inactive. This state is indicated after a folder is removed from the worklist. The following rule named `R_StartOnRemove` is triggered every time the message `remove` is sent to an instance of class `Worklist` that belongs to a software agent (cf. `ACTIVATED FOR` clause). The condition checks whether there are remaining folders to be processed. If the worklist is not empty, i.e., there are remaining folders, the action selects the next folder in turn (method `nextWlItem`) and sends the message `perform` to the corresponding activity with the folder as parameter.

```
DEFINE RULE R_StartOnRemove
     ON POST (Worklist, remove) DO
     IF ruleTrgO isEmpty isFalse THEN
     EXECUTE ((ruleTrgO nextWlItem) activity) perform: ((ruleTrgO
nextWlItem) folder)
ACTIVATED FOR (SoftwareAgent collect:[:x|x worklist]).
```

The processing of a folder which is inserted into an empty worklist is triggered by another rule whose event part is defined on the insertion of a folder. Similar rules exist for human agents.

4 Future Directions

Further research directions are twofold. Firstly, business processes need to access distributed information sources [13]. These information sources will not only supply passive data but also applications and potentially other WFMS which have to interoperate. In this respect, external isolated applications have to be integrated with new

applications already built into the WFMS. Since these distributed information sources have to comply with common protocols OMG's CORBA protocol will become very important. And secondly, business processes need correctness, reliability, and failure criteria to be enforced similar to transaction systems, since they consist of interrelated activities with data flow constraints and control flow constraints. Due to their long-lasting asynchronous nature business processes require extended transaction models incorporating long transactions and multi-level transactions [9]. The extension of GemStone's transaction model along these lines will be investigated.

References

[1] M. Ader et al., "WooRKS, an Object Oriented Workflow System for Offices," Technical Report, Bull S.A. (Paris), Technics en Automatitzacio d'Officines S.A. (Barcelona), Dep. of Computer Science (University of Milan), Communication and Management Systems Unit (Athens), 1994.

[2] E. Bertino and L. Martino, *Object-Oriented Database Management Systems: Concepts and Architectures*, Addison-Wesley, 1993.

[3] P. Butterworth, A. Otis and J. Stein, "The GemStone Object Database Management System," in *Communications of the ACM*, vol. 34, no. 10, pp. 64-77, Oct. 1991.

[4] C. Bußler and S. Jablonski, "Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems," in [5].

[5] S. Chakravarthy and S. Urban (eds.), *IEEE Proceedings Research Interests in Data Engineering: Active Database Systems (RIDE'94)*, Houston, 1994.

[6] C.A. Ellis and G.J. Nutt, "Modeling and Enactment of Workflow Systems," Technical Report, Dep. of Computer Science, University of Colorado at Boulder, 1993.

[7] G. Engels and G. Kappel, "Object-Oriented Systems Development: Will the New Approach Solve Old Problems?," in *Proceedings of the IFIP94 World Congress, Vol.III*, ed. K. Duncan and K. Krueger, North-Holland, 1994.

[8] G. Gottlob, B. Röck and M. Schrefl, "Extending Information Systems with Roles," in *ACM Transactions on Information Systems (accepted for publication)*, 1994.

[9] M. Hsu (ed.), *IEEE Bulletin of the Technical Committee on Data Engineering: Special Issue on Transactions and Workflow Management*, vol. 16, no. 2, June 1993.

[10] G. Kappel, S. Rausch-Schott and W. Retschitzegger, "Beyond Coupling Modes - Implementing Active Concepts on Top of a Commercial OODBMS," in *International Symposium on Object-Oriented Methodologies and Systems (ISOOMS)*, ed. E.Bertino and S.Urban, Springer LNCS 858, 1994.

[11] G. Kappel, B. Pröll, S. Rausch-Schott and W. Retschitzegger, "TriGSflow Active Object-Oriented Workflow Management," in *Hawaii International Conference on System Sciences (HICSS'95)*, 1995.

[12] B. Pernici, "Objects with Roles," in *Proceedings of the ACM/IEEE Conf. of Office Information Systems*, pp. 205-215, Cambridge, MA, April 1990.

[13] B. Reinwald, *Workflow-Management in Distributed Systems*, Teubner Verlag, 1993 (in german).

Exotica: A Research Perspective on Workflow Management Systems [†]

C. Mohan G. Alonso R. Günthör M. Kamath[‡]

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
{mohan, gustavoa, rgunther, kamath}@almaden.ibm.com

1 Introduction

Workflow is probably one of the most exciting areas of research that has emerged in the past few years. The concepts and ideas have been around in one form or another for a long time: *computer supported cooperative work*, *form processing*, *cooperative systems*, *office automation*, etc. However, only recently the technology and know-how required to implement commercial systems have been available.

In general, workflow management systems (WFMS) are used to coordinate and streamline *business processes*. These business processes are represented as *workflows*, i.e., computerized models of the business process [11], which specify the individual *activity* steps, the order and the conditions in which the activities must be executed, the flow of data between activities, the users responsible for the execution of the activities, the tools to use with each activity, etc. A WFMS is the set of tools that allow the design and definition of workflows, their instantiation and controlled execution, and the coordination and integration of heterogeneous applications within the same workflow [11]. Users interact with the WFMS by accessing their individual *worklists*, where they can find the activities for which they are responsible without necessarily being aware of the higher level process to which the activities belong. A crucial point to understand workflow management systems is their dependency on a variety of technologies, from databases to distributed processing. When designing a WFMS, the challenge is to build a feasible and common working environment in which all these technologies are integrated in a flexible and easy to use fashion. This integration aspect is precisely what has made workflow management systems so elusive to date and explains the limited success of earlier attempts to provide support for cooperative work. With the first generation of commercial workflow management systems [10], there are still many integration issues that remain to be solved, but there is no doubt that in the future workflow management systems will be pervasive in large corporations.

In this paper we present the Exotica Research Project, currently in progress at the IBM Almaden Research Center. One of the goals of the project is to bring together industrial trends and research issues in the workflow area. It is for this reason that we have focused on a particular commercial product, *FlowMark*, IBM's workflow product. However, our results are easily generalized to general workflow management systems since FlowMark's

[†]This work is partially supported by funds from IBM Hursley (Networking Software Division) and IBM Vienna (Software Solutions Division). Even though we refer to specific IBM products in this paper, no conclusions should be drawn about future IBM product plans based on this paper's contents. The opinions expressed here are our own.

[‡]Currently visiting IBM Almaden from the Computer Science Department at the University of Massachusetts, Amherst, MA 01003, USA.

model is similar to that proposed by the Workflow Management Coalition [11]. In particular, the paper contains a high-level overview of our research in six specific areas that are not product specific. The list of these areas is not, by any means, exhaustive. There are still many issues that remain open. For reasons of space, the paper does not discuss workflow related concepts [11, 26, 12, 24, 27] or the particular details of the architecture and functioning of FlowMark [19, 18, 14, 15]. The interested reader can find more information in the references.

2 Large Scale Workflow Systems

Before discussing workflow management, it is necessary to put in perspective the applications addressed by such systems. Some of the common goals of a WFMS are to achieve better performance of business processes, better quality, enhanced effectiveness, enterprise wide coordination and monitoring, etc. As we believe it is the case with most workflow management systems, FlowMark addresses the coordination of *large* scale business processes. In this there are certain similarities with databases. Small databases for personal computers have an undeniable value, however interesting research and industrial strength products relate to issues such as query optimization, performance, data caching, data mining, schema evolution, or triggers, which only become interesting in large database scenarios. For small applications, it is possible to obtain similar or even more benefits by using a less sophisticated cooperative tool rather than a full-fledged workflow management system. However, when the size of the application grows, and the number of users, sites, and processes increases, only a workflow management system is suitable for the task.

It remains to be determined how "large" is a large application. To give an idea of the magnitude of the problem, consider some of the studies that have been done with FlowMark. In one instance, the number of business processes to be executed concurrently in a period of a month is 300,000. In another case, the problem is to link together more than 4,000 sites geographically distributed over an entire country and coordinate the work taking place over heterogeneous platforms at those sites. A third example involves more than 100,000 users working concurrently on the completion of processes. With these figures, issues that were not previously considered as related to workflow become key aspects to the success of a commercial system. Failure handling, continuous availability, navigational flexibility, and replication, to name a few, are no longer nice features but crucial components of the overall system. It is within this framework that our research is inscribed.

3 Research Issues in Workflow Management

The Exotica project is currently focused on six major research areas. Each of them reflects a distinctive need for workflow management systems but, in most cases, they are related to each other and successful solutions will have to integrate aspects from all of them. The list of these areas is by no means exhaustive, and there are many other issues that remain open.

3.1 Failure Resilience in Distributed WFMSs

Given its goals, the architecture of a workflow management system is necessarily distributed in the sense that its components will reside in different and heterogeneous machines. Furthermore, given its relevance in the control of the business processes of a corporation, the system must be continuously available. Hence, each component must be able to deal with local and communication failures, and the design must be robust enough to avoid stopping the execution of processes even in the event of failures. Such requirements define a wide range of failure handling capabilities that the system must support to be commercially viable. Part of our research has been to analyze the possible failure scenarios and design methods to handle them [1, 2, 17].

In the first place, the execution of an activity within a process involves several components: the database server where the workflow process related data is stored, the workflow server that determines where the activity

is going to be executed, the client interacting with the user, and the client interacting with the actual application performing the activity. All of these components interchange information before, during and after the execution of the activity. This information should not be lost, since this may imply that the results of the execution are not recorded, or that the command to start executing the activity never reaches its destination. Hence, some form of *coordination protocol* must be used amongst all these components to guarantee that all agree on the status of a particular activity. In general, the database where the data is stored is transaction based. In FlowMark, for instance, the data is stored in ObjectStore, an object-oriented DBMS. ObjectStore does not provide either *hot standby* support or a *prepare-to-commit* interface, which makes it complicated to reach a point in the execution where data is guaranteed not to be lost. Moreover, the applications interacting with the workflow management system may not be aware of it, thereby becoming uncooperative and committing work regardless of the state of the workflow management system. We are studying several approaches to these problems: using a persistent message mechanism, providing stable storage in each of the components, and designing a handshake protocol that ensures that information is almost never lost.

A second aspect of failure resilience is the possibility that a component fails while it is executing part of a process. If the failed component is the database, replication can be used to minimize the impact of the failure, as discussed below. If the failed component is part of the workflow management system, means must be devised to allow other components to reroute their operations to components that are still available. This implies using multiple connections among components. We have designed a new architecture for workflow management systems in which the notion of clusters of servers is used to provide enhanced availability of the system [2]. By using several clusters, each of them attached to a separate database, the impact of failures on the overall performance can be reduced. Since each cluster's database contains the same schema information such as process templates, role and staff assignments, and organization definition, process instances can be run at any cluster. Thus, failures will affect only a subset of the process instances and will not prevent the users from starting new instances. The problem that still remains is how to handle the instances that were running in a failed cluster. This particular issue is addressed below when discussing replication in workflow management systems.

3.2 Compensation and Navigation in Workflow Networks

A workflow network, or a process, is the representation of business processes. Its instantiation for execution is a process instance. Any process instance usually encompasses a wide range of activities and involve several participants in the form of computer programs and individuals. At any given moment in time, there may be several instances of the same process being executed simultaneously, with possibly each one being at a different stage of progress. Since processes are defined in advance, it is not possible to evaluate all possible exceptions and error conditions beforehand. A workflow management system must provide means to cope with such situations. Similar ideas are found in advanced transaction models [6] such as Sagas [8] for backward recovery, i.e., how to undo the effects of certain operations, or in Flex transactions [5, 21, 28] for forward recovery, i.e., how to select different paths of execution. However, in workflow systems there are certain errors that may force the user to abort the execution of a process instance, and therefore to lose work that has to be performed again when the process is restarted.

Forward recovery is guaranteed in FlowMark, in the sense that if there are failures, the process is guaranteed to make progress. However, semantic failures are more difficult to address, unless the designer of the process was able to foresee them and introduce the appropriate checks in the flow of control. For backward recovery, FlowMark is being enhanced to provide a form of compensation that allows the user to specify *spheres of compensation* [20] to determine the scope and extent of compensation in case of failures.

Our current research is focused on how to provide a flexible mechanism to navigate through the flow of control of a process in either direction, i.e., forward progress or compensation, and allowing to switch directions several times as necessary during the execution of a process.

3.3 High Availability through Replication

High availability is a key requirement of a workflow management system. Failures should be transparent to the users and should have minimal impact on the normal functioning of the organization. This can only be achieved through replication of the process instance information. As in databases, we use a primary/backup architecture in which process instances run on a primary server, while all actions are also recorded at the backup so that if the primary were to fail, then the backup server can take over the execution of those same instances. As in databases, replication has a high cost in terms of synchronization and lower throughput. Given the large number of processes involved in the application, replication of every single process instance may not be cost-effective. Hence, we define three priority levels with respect to replication. A *hot stand-by* process is fully replicated, i.e., all changes to its data are performed both in the primary and in the backup databases. A *cold stand-by* process is also replicated, but the backup contains only the messages with the information regarding the different steps taken in the execution of the process. When the backup needs to take over the execution of the process, it must first replay those messages to bring the process state up-to-date. This will delay resuming the execution but it reduces the overhead of maintaining the replicated copies. Finally, a *normal* process is not replicated at all. The only guarantee FlowMark provides for these processes is forward recovery, i.e., when the server comes up again, execution will be resumed from where it was left off.

As in DBMSs, besides the replication scheme, it is also necessary to deal with additional problems such as dynamic configuration of the system, incorporation and exclusion of new servers without interrupting the system, message duplication, and so forth. We have addressed all these issues and proposed a replication mechanism for workflow management systems that greatly improves the availability of the system avoiding excessive overhead [17].

3.4 Mobile Computing

Workflow management systems must coordinate users distributed over a wide geographic area. The basic idea being that the processes to be executed are defined and controlled in a centralized server, while the users can perform parts of these processes at remote clients. Each step of a process can be executed anywhere in the system, but after its completion, the results are communicated to the server which in turn prepares the next steps for execution. This is the most common approach in existing systems, and simplifies many problems such as synchronization, concurrency and monitoring, but forces the users to remain *connected to the server* while performing multiple tasks. An increasingly large set of users may not be satisfied with this mode of operation, given the widespread use of portable computers and home computers for office work.

For these users, the common way of operation is to load data in their laptops or desktops by briefly connecting with a server in the office. Then they disconnect from the server, work on that data and, after a few hours or few days, reconnect to the server to transfer the results of their work. This has been identified as one of the most common modes of computing in the future [16]. Note that this includes not only mobile terminals but also desktops or any other computer type connected to the server only occasionally via a modem. Mobile or nomadic computing greatly expands the scope of an organization's distributed computing infrastructure. However, from the workflow management point of view, it becomes more difficult to coordinate the work of many users. Also, note that while workflow systems are tools for cooperation, portable computers are generally viewed as tools for individual work.

To address this problem, we have proposed a design for disconnected client operation based on FlowMark [3]. Among other constraints, the design had to provide support for disconnected clients with minimal changes to the semantics of a business process and its implementation, effectively allowing users distributed over a wide geographic area and working with heterogeneous resources to cooperate, while preserving their mobility and independence. In particular, we define the semantics of loading work to a mobile, disconnectable computer by introducing the notion of *locked activities* and the *user's commitment* to eventually execute them. Locked activities

stay within the user's machine unless they are unlocked. The feasibility of our approach is proven by outlining the implementation issues for FlowMark. We are currently working on a prototype based on these ideas.

3.5 Distributed Coordination

To further enhance the availability and failure resiliency of a workflow management system, it is possible to design it entirely as a distributed system. This implies that processes are transferred from site to site as their execution progresses, without a centralized server keeping all the relevant information. There are two possible approaches. One is to make a large package that contains all the information pertaining to the process and its execution and circulate this package across the different sites. This approach has several problems, mainly the size of the message and the need to maintain multiple copies of it in the system to allow concurrent activities to take place. The other solution is to precompile the process definition to determine at which sites the different activities are to be executed. Once this has been done, the only information that gets transferred from site to site are the results of previous computations and the process state. Using this approach, failures can be made transparent by rerouting a process to different nodes. It also eliminates the potential bottleneck created by a centralized database and server.

Exotica/FMQM [1] proposes a similar architecture in which persistent storage is replaced by persistent messaging. For persistent messaging, IBM has defined an application programming interface (API) standard called Message Queue Interface (MQI) [13], and a family of products called MQSeries that supports MQI [22]. MQSeries products operate on IBM and non-IBM platforms and they support the architected MQI. Communication takes place through named queues that do not require all participating programs to be available, i.e., up and running, simultaneously. Moreover, MQI is not sensitive to network transport protocol differences. The system we propose, *Exotica/FMQM*, FlowMark on Message Queue Manager, is a distributed workflow system in which a set of autonomous nodes cooperate to complete the execution of a process. Each site functions independently of the rest of the system, the only interactions between nodes being conducted through persistent messages which inform about activity completions. This approach, while enhancing the availability and resilience of the system, has the problem of needing more complex mechanisms to monitor and audit the overall execution since there is no centralized server where this information resides.

3.6 Advanced Transaction Models

The goal of most advanced transaction models is to eliminate the constraints imposed by conventional DBMSs on new applications. The requirements of such new applications are completely different from the traditional data processing applications targeted by standard DBMSs. Hence, conventional DBMSs are unsuitable in these new environments. Several advanced transaction models have been proposed [6] but, to this date, most remain as theoretical constructs which have not been implemented. A reason for this state of affairs could be that advanced transaction models are ahead of the available technology and their time is yet to come. We believe, however, that the reason why these models are not being implemented has to do more with their inadequacy to operate in real working environments than with the available technology or applications demands. Advanced transaction models are too database-centric, which provides a nice theoretical framework to work with but limits the possibilities and flexibility of the models. Since they tend to remain theoretical models with no implementation, there are a number of important design issues that are generally not discussed in the literature [23].

Paradoxically, workflow systems are tools to support distributed, heterogeneous environments very similar to those targeted by transaction models. However, the models being used are *workflow* models instead of transaction models. Compare the more than 70 vendors who claim to have workflow systems [10] with the almost absolute lack of commercial products supporting advanced transaction models. Workflow systems bear a strong resemblance to advanced transaction models both in their goals and modeling approach, yet they are quite different in that they address a much richer set of requirements than existing transaction models. As part of our research, we

have analyzed the characteristics of workflow models and the notion of business processes by comparing them with existing transaction models [2]. It is possible to show that the semantics of workflow models are, in general, richer than those of advanced transaction models and more apt to be used in commercial products. Hence, workflow models can be used to implement advanced transaction models. For instance, we have used FlowMark to implement Sagas [7] and Flexible Transactions [28]. Our basic goal was to provide synergy between advanced transaction models and workflow models, an interaction from which both sides may benefit. In doing so, we have developed a better understanding of the inherent limitations of the so-called "advanced models" and identified many points for improvement of workflow systems.

Our approach is not another attempt to merge different transaction models into one or to provide a general framework to program advanced transactions [4]. Transactions are used as the accepted currency in the database community, but our goal is to show the expressibility and power of workflow models compared with what is currently in the research literature. From a purely advanced transactions point of view, our proposal differs from previous ones in that we use an existing commercial product that runs across different platforms to implement different transaction models. This allows us to draw conclusions from direct experience implementing such models in real, working environments. We see workflow models as the the next step after advanced transaction models. They complement many aspects of the latter and address an entirely new range of issues (role and staff assignment and resolution, worklist management, interaction with manual activities, etc.) that make them more suitable for building applications. As has been widely recognized, there is a lack of tools to scale from individual transactions to complex applications [25]. One reason being that advanced transaction models will never reach their technological maturity until they are interpreted in a much broader context. As we show in our research, business processes constitute such a context and workflow models are the tools required to support complex applications.

4 Conclusions

The importance of workflow management systems in large corporations cannot be stressed sufficiently. There is a high demand for such systems, but successful solutions will have to integrate many technologies and approaches. The challenge lies in providing systems capable of dealing with very large, heterogeneous, distributed and legacy applications, while providing an acceptable degree of reliability and availability. The research areas briefly discussed in this paper point towards some of the issues that need special attention in the design of workflow system. These issues apply to the vast majority of existing systems, since they are not constraints imposed by a particular architecture but generic demands of the working environment targeted by these systems. In our case, we have focused on FlowMark since this is an IBM product and we have access to its code and developers. However, the solutions we proposed can be easily extended to other systems given that FlowMark's model and architecture closely resembles the generic model proposed as part of the Workflow Management Coalition standarization effort [11].

Acknowledgements

Part of this work has been done in collaboration with Amr El Abbadi and Divyakant Agrawal, of the University of California at Santa Barbara.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Günthör, M. Kamath, *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management*, **Research Report, RJ 9912**, IBM Almaden Research Center, November 1994.

[2] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, C. Mohan, *Failure Handling in Large Scale Workflow Management Systems*, **Research Report, RJ 9913**, IBM Almaden Research Center, November 1994.

[3] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi, C. Mohan, *Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System*, **Research Report**, IBM Almaden Research Center, *in preparation*, January 1995.

[4] A. Biliris, S. Dar, N. Gehani, H.V. Jagadish, K. Ramamritham, *ASSET: A System for Supporting Extended Transactions*, **Proc. 1994 SIGMOD International Conference on Management of Data**, May 1994, pp. 44-54.

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

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

[7] H. García-Molina, K. Salem, *Sagas*, **Proc. 1987 SIGMOD International Conference on Management of Data**, May 1987, pp. 249-259.

[8] H. García-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem, *Coordinating Multi-transaction Activities*, **Proceedings IEEE Spring Compcon**, 1991.

[9] H. García-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem, *Modeling Long-Running Activities as Nested Sagas*, **Bulletin of the Technical Committe on Data Engineering, IEEE**, vol. 14, no. 1, pp. 18-22, March 1991.

[10] C. Frye ,*Move to Workflow Provokes Business Process Scrutiny*, **Software Magazine**, April 1994, pp. 77-89.

[11] D. Hollinsworth, *The Workflow Reference Model*, Workflow Management Coalition, TC00-1003, December 1994.

[12] M. Hsu, *Special Issue on Workflow and Extended Transaction Systems*, **Bulletin of the IEEE Technical Committee on Data Engineering**, vol. 16, no. 2, June 1993.

[13] IBM,*Message Queue Interface: Technical Reference*, Document No. SC33-0850-01, April 1993.

[14] IBM ,*FlowMark: Managing Your Workflow*, Document No. SH19-8176-01, September 1994.

[15] IBM ,*FlowMark: Programming Guide*, Document No. SH19-8177-01, September 1994.

[16] T. Imielinski, B.R. Badrinath, *Mobile Wireless Computing: Solutions and Challenges in Data Management*, **Communications of the ACM**, vol 37, no. 10, October 1994.

[17] M. Kamath, G. Alonso, R. Günthör, C. Mohan, *Providing High Availability in Very Large Workflow Management Systems*, **Research Report**, IBM Almaden Research Center, *in preparation*, January 1995.

[18] F. Leymann, W. Altenhuber, *Managing Business Processes as an Information Resource*, **IBM Systems Journal**, vol. 33, no. 2, pp. 326-348, 1994.

[19] F. Leymann, D. Roller, *Business Processes Management with FlowMark*, **Proc. 39th IEEE Computer Society International Conference (CompCon)**, February 1994, pp. 230-233.

[20] F. Leymann, Supporting Business Transactions Via Partial Backward Recovery in Workflow Management Systems, **Proceedings BTW95**, Dresden, Germany, March 1995.

[21] S. Mehrotra, R. Rastogi, A. Silberschatz, H.F. Korth, *A Transaction Model for Multidatabase Systems*, **Proceedings of the International Conference on Distributed Computing Systems**, June 1992, pp. 56-63.

[22] C. Mohan, R. Dievendorff, *Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing*, **Bulletin of the IEEE Technical Committee on Data Engineering**, vol. 17, no. 1, March 1994, pp. 22-28.

[23] C. Mohan, *Advanced Transaction Models - Survey and Critique*, Tutorial presented at the **ACM SIGMOD International Conference on Management of Data**, 1994.

[24] B. Reinwald, *Workflow Management in verteilten Systemen, Teubner-Texte zur Informatik*, **B.G. Teubner Verlagsgesellschaft**, *Stuttgart, Leipzig, 1993*.

[25] B. Salzberg, D. Tombroff, *A Programming Tool to Support Long-Running Activities*, **Technical Report NU-CCS-94-10**, College of Computer Science, Northeastern University, Boston, 1994.

[26] A.P. Sheth, *On Multi-system Applications and Transactional Workflows*, collection of papers from Bellcore, 1994.

[27] K. Walk, *Workflow Management Concepts, a White Paper*, IBM Vienna Software Development Laboratory, Austria, November 1993.

[28] A. Zhang, M. Nodine, B. Bhargava, O. Bukhres, *Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems*, **Proc. 1994 SIGMOD International Conference on Management of Data**, 1994, pp. 67-78.

OpenPM: An Enterprise Process Management System

Jim Davis, Weimin Du and Ming-Chien Shan
Hewlett-Packard Laboratories
Palo Alto, CA 94304
{davis, du, shan}@hpl.hp.com

1 Introduction

The pressure of increasing competition is forcing major corporations to reengineer their business processes. In order to stay viable, companies must examine all of their business processes, streamlining and improving them to reflect the changing nature of doing business. HP OpenPM is designed to meet this requirement by providing a platform integrating different kinds of business activities and a runtime environment automating activities execution.

This short paper gives an overview of HP OpenPM describing the design principles, process model, and overall architecture.

1.1 Overview

OpenPM is a middleware service that enables the automation of business activities supporting complex enterprise processes in a distributed heterogeneous computing environment.

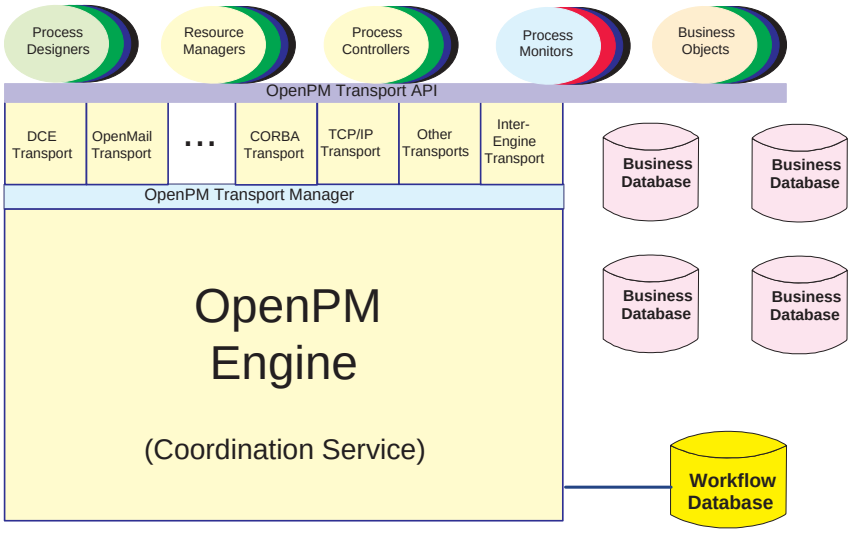

Figure 1: OpenPM Application Architecture

Figure 1 shows the OpenPM application architecture. The OpenPM environment includes the **OpenPM engine** and a set of **business objects**, **resources managers**, **process designers**, **process controllers**, and **process monitors**.

Business activities are building blocks to construct an OpenPM process. A business activity is a representation of an activity in a business domain. Business activities are mapped to business objects during the execution

of a business process. This mapping provides an abstraction allowing multiple possible implementations of a business activity. It also protects the business process designer from involvement in the details of the implementation of the programs and transports which will implement the business activity. A business object may represent a manual operation by a human or the execution of a program to integrate a legacy application, access databases, control instrumentation, sense activity in the external world, or even effect physical changes.

A process is defined using a process designer. A process can then be started, stopped, or intervened using a process controller. A process monitor keeps status information about each individual process and load information of the entire system.

The OpenPM engine coordinates the overall business process flow. It functions as a highly reliable, log based state machine. OpenPM engine interacts with other components through a uniform transport interface, independent of the actual physical dispatch of the requests.

The Resource Management Services contain all of the information about physical, user, and electronic resources usable by the business processes. At business activity runtime the specified Resource Management Service assigns an agent to the task. Agents can be an individual person, a group of people, an application on a particular machine, etc.. This mapping is performed according to the policies provided by the resource administrator. A resource manager may consult data sources in fulfilling its requests, for example "who is in which department," "who is on vacation."

1.2 Design Principles

There are three major design principles: simplicity, openness, and flexibility.

- OpenPM should avoid introducing unnecessary complexity because process flow management must deal with very complex environments, integrating heterogeneous legacy applications while taking advantage of newer technologies. OpenPM uses a simple, graph based model for business processes and their definition. All components (including both business objects and resource managers) of OpenPM are connected via a uniform transport interface.

- OpenPM also provides an open interface to heterogeneous application environments. There are two implications. First, the system should be able to easily integrate open components that are flexible, customizable, and replaceable. Second, it should also be able to integrate legacy applications that do not conform to the open standards.

- OpenPM business processes are usually long-running activities, lasting hours, days, or even years. It is important for OpenPM to provide a flexible process execution environment: allowing not only flexible integration and execution of business objects, but also dynamic change to both the business process and the underlying business objects.

2 OpenPM Process Model

The OpenPM system executes business processes in a flexible automated environment. A business process is a directed graph with edges (arcs) and nodes. Nodes may represent business activities, business processes (subprocesses), routing decisions about the paths over which execution shall flow, and timers triggered by epoch, interval, or external event.

Business objects are the process flow management encapsulation of units of work. Both new and legacy applications may be encapsulated into business objects. The mapping between the business activity specified in the business process and the business object to be invoked is performed by one of the Resource Management Services as part of running the business activity.

Work nodes are an abstract definition of a business activity together with some abstract resource request, optional timeout or deadline information, and some business process information. At each evaluation the resource request is bound to a business object for execution. Work nodes have at most one inward and one outward **forward arc**, they are separated from the decisions about what tasks should next be performed in the business process.

To define process flows more complex than a simple sequence **Route nodes** are used. When an inward arc is traversed, a route node decides — based upon the status passed along its inward arcs, the timing of each inward arc's firing, and the data associated with the running process instance — which outward arcs to fire (traverse). Route nodes may have many inward and many outward arcs. A rule language (**BTR**) is used to program the route nodes decisions.

At most a single occurrence of each node can be executing within a single instance of a business process instance. One can think of the work nodes as being used up once they have been triggered into action. But how then can one create loops or explore alternatives?

Arcs provide the answer. There are two kinds of arcs: **forward arcs** and **invalidation arcs**. If we define the **scope** of a node in a business process definition to be the set of nodes reachable by following forward arcs through its graph. Each node in a business process graph must be within the scope of one of its start nodes. Invalidation arcs are used to create loops or explore alternatives. These arcs are restricted to have their source within the scope of their destination(i.e. the arc must point "backwards"). When traversed, an invalidation arcs causes all nodes within the scope of the destination but not within the scope of their source, as well as their source and destination to be reset to allow their reuse. In figure 2 below the invalidation arcs are labeled **IA**.

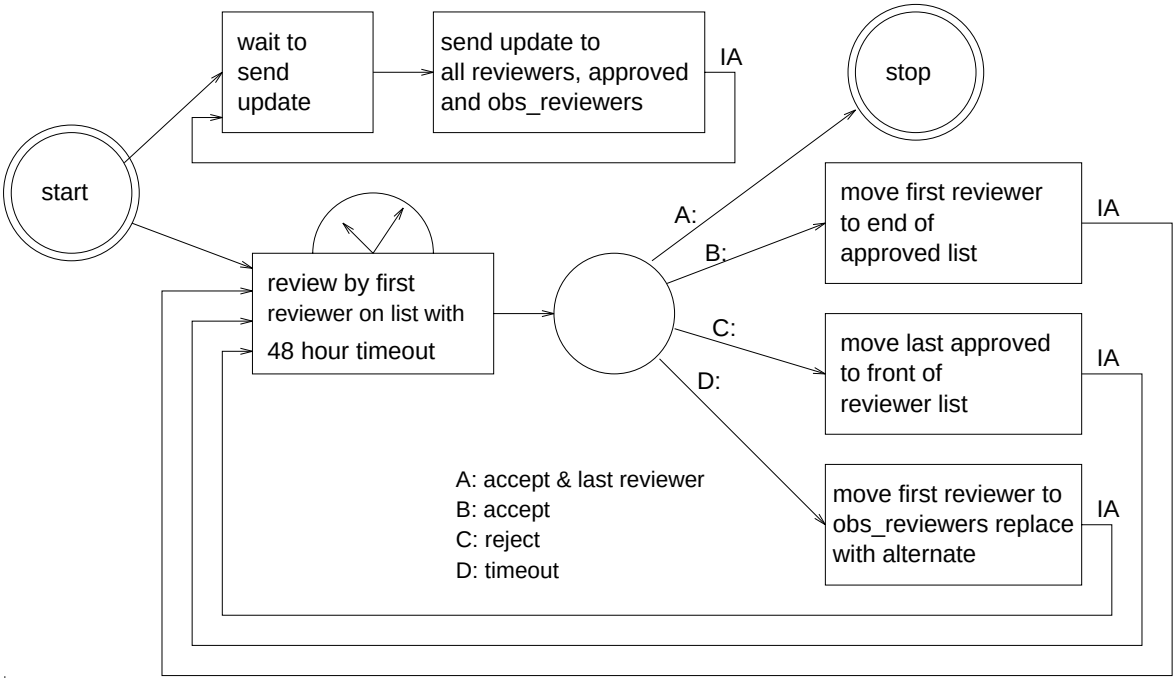

Figure 2: Wage Review Business Process

Figure 2 shows a very simple business process. The wages of employees should be reviewed by their effective first and second level manager (e.g. managers may delegate such authority, the **CEO**'s wages will be reviewed by the Board of Directors). Each review must be finished within a time limit or it should be given to an appropriate authority to be completed. Each process instance has data associated with it. The wage review example has the standard data (e.g. process id, process version id, process instance id) as well as some process specific data:

reviewer	list	initially manager,manager's manager
approved	list	initially empty
obs_reviewers	list	initially empty

Note that the details of authority, and delegation need not be visible at the time of business process design. This allows for the independent evolution of the business process, and the business authorization and work dispatch model.

3 OpenPM Architecture

The **Coordination Service** (also called **OpenPM engine**) represents the heart of OpenPM. It handles all of the routing, synchronization, and transportation of tasks within the enterprise process management system. Figures 1 and 3 show the Coordination Service in relation to the other processes in the OpenPM system. The APIs for process design, management, and business object encapsulation are layered upon the transport service. A business object may be moved to a new transport, or served by several transports yet need not take that into account within its code.

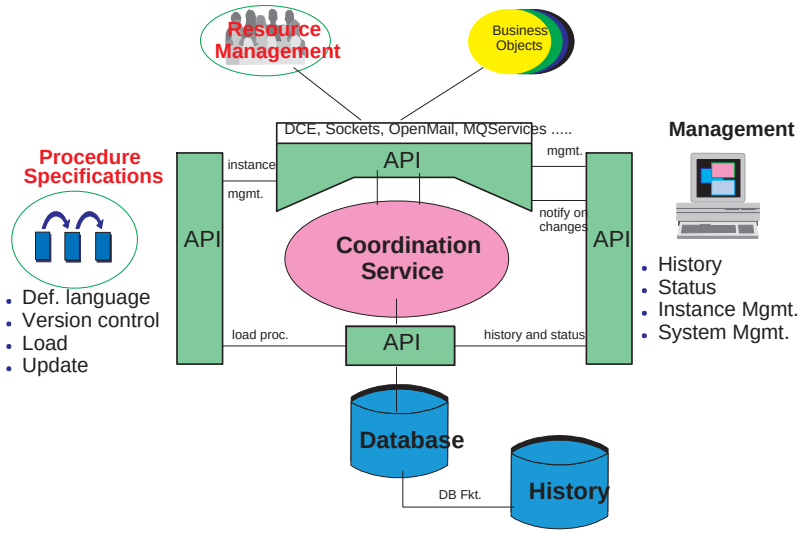

Figure 3: OpenPM Architecture and **API**s

3.1 OpenPM Internal Structure

Figure 4 shows the main modules within the OpenPM Engine. The darker lines represent the critical paths through the engine of which the highest levels of throughput must be demanded.

In many ways this engine functions as a robust, recoverable, log based state machine. Each node transition must be durably logged so that it can be rolled forward in the event of system failure or used for compensation in the event of business activity failure.

Figure 5 shows a simplified version of the work node state transition machine (compensation has been omitted). Note that an activity may be invalidated at any time, this may occur through the traversal of an invalidation arc, due to intervention, or because compensation requires that this activity not complete.

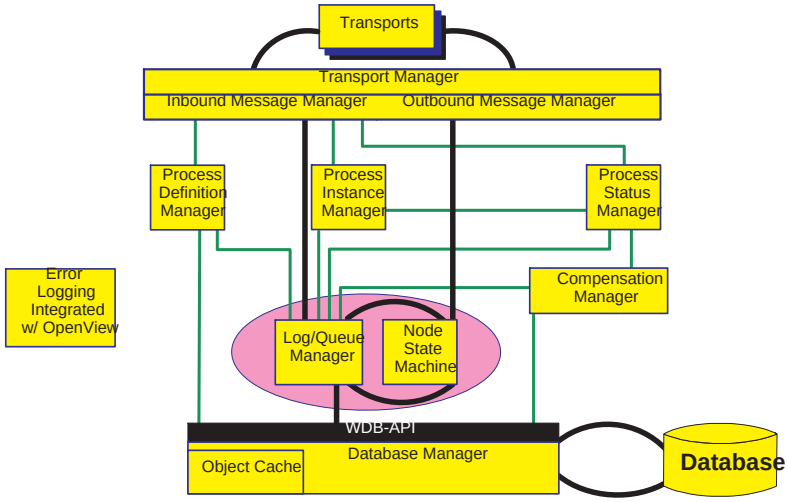

Figure 4: Internal Structure of OpenPM Coordination Service

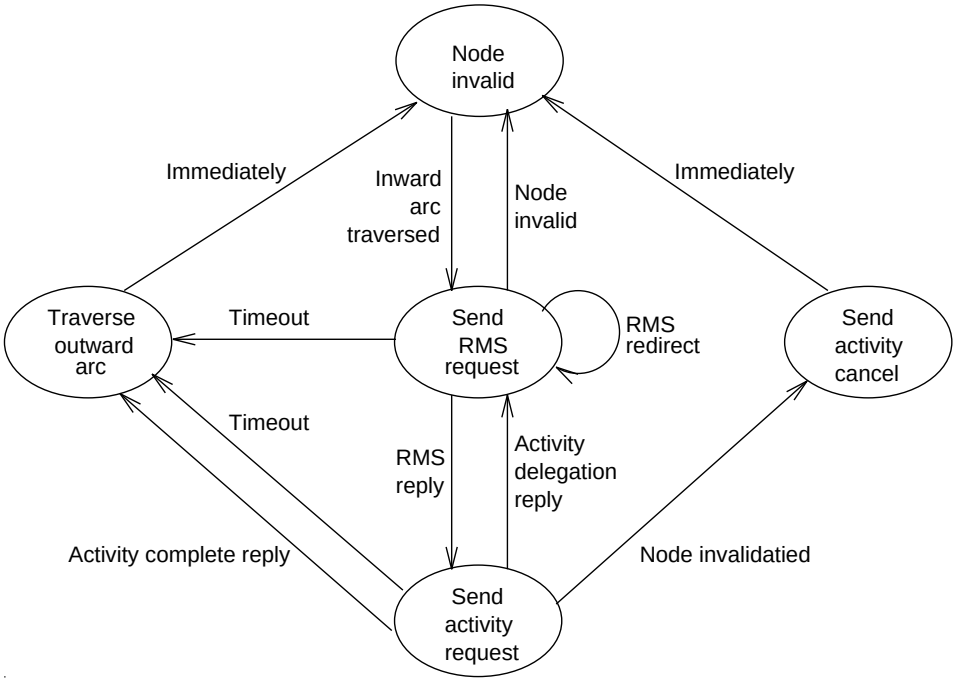

Figure 5: Simplified Work Node State Diagram

4 Failure and Compensation in OpenPM

A work node specifies information about the business activity and may also specify the compensation needed to undo the results of that activity. From a database perspective each business activity represents one or more transactions in one or more databases. Once an activity has completed it cannot be simply rolled back or undone. However, a business activity may be able to be compensated for. An example is that after an automated teller machine has dispensed money there is no way to undo that effect, it cannot vacuum the money back into the machine. But when viewed as a business activity we see that we can compensate by attaching to assets of the beneficiary of the undesired activity. We can deduct from an account of theirs, or even send them a bill.

Business processes may set compensation points in much the same way as database processes set savepoints. When an activity requests compensation it may specify a compensation point or the "nearest" one will be computed. The entire business process instance is frozen, the scope of the required compensation is computed, and the log is used to run appropriate compensation activities. After that the business process instance is unfrozen and execution can proceed again from the compensation point (as well as points outside the compensation scope).

For each invocation of each instance of each activity whose forward execution is to be compensated, the business compensation activity is provided with the post-image of the process instance data and the delta (or pre-image). The business compensation activity performs whatever actions are required to logically undo the effects of the forward execution.

Not all work nodes need to be compensatible. A node may specify that it needs no compensation, or that it cannot be compensated for. In the former case compensation is skipped, while a request for compensation the latter case would cause the business process instance to fail. If the failing business process instance is a subprocess in some other business process instance than the calling instance may provide compensation for the entire subprocess.

5 Conclusions

The OpenPM architecture and model provide a simple, open, and flexible system for the execution of enterprise-wide processes. Our prototype is currently being used to research further into compensation, failure recovery, and migration of running business process instances as processes are changed.

6 Acknowledgments

The OpenPM project acknowledges the contributions of

- Umesh Dayal for research direction and management support;
- Clemens Pfeiffer, Chip Vanek and Elisabeth Caloyannis for their support of OpenPM;
- Qiming Chen for his work on compensation in structured process management.
- The Hewlett-Packard Work Manager Operation (WMO) for providing us with a firm grounding in process data management systems.

Building Flexible Distributed Applications With The Teknekron Enterprise Toolkit

Arvola Chan Kieran Harty
Teknekron Software Systems
525 Lytton
Palo Alto, CA 94040, USA
{arvola, kieran }@tss.com

1 Introduction

Abstract

The Teknekron Enterprise Toolkit is an infrastructure for building large-scale, loosely-coupled, distributed applications that can easily be maintained and enhanced. At the core of the toolkit is the Teknekron Information Bus (TIB), an unlimited area messaging facility that supports a novel publish-subscribe communication paradigm, as well as the more traditional request-reply, using an anonymous communication principle called subject-based addressing. As a software platform, the TIB hides from users and programmers the boundaries of various systems and services, network configurations, and the heterogeneity of machine environments. We describe the communication mechanisms supported by the TIB, and illustrate how the TIB and other components of the Teknekron Enterprise Toolkit can be used to build loosely-coupled workflow style applications. We also discuss two financial applications that are built using this toolkit. Both applications are in production use.

2 Introduction

Today's enterprise has to think globally, react quickly and decisively to changes in the market, and be flexible enough to tailor its products and services to the changing requirements of its customers. At the same time, an enterprise needs to preserve its existing capital investment and infrastructure - it is not economically feasible to re-build an enterprise from scratch. It is therefore desirable to take an incremental approach to making the transition from a traditional relatively static enterprise to one that is dynamic, flexible and responsive.

The Teknekron Enterprise Toolkit provides the middleware for connecting all the applications of a business in a globally distributed environment. It consists of a set of software tools that builds upon existing hardware, software and networking platforms to provide a computing infrastructure that is global, flexible and responsive. It includes tools to build highly flexible applications, allowing them to be developed, deployed and modified rapidly and with minimal effort. It also includes tools that allow applications to inter-operate with new or existing applications, wherever they may be within the enterprise. Moreover, the Teknekron Enterprise Toolkit provides support for integrating legacy applications into the enterprise infrastructure or for gracefully migrating away from these applications.

This paper explains the communication infrastructure that forms the core of the Teknekron Enterprise Toolkit. Section 2 identifies two communication paradigms, both supported in the Teknekron Enterprise Toolkit, that are crucial in data-intensive application environments. Section 3 describes the key characteristics of the Teknekron

Information Bus (TIB) that contribute to the loose coupling of resulting applications. Section 4 discusses the major protocols that are supported by an object-oriented interface called the Common Information Bus Interface (CIBI). Section 5 describes two financial applications that are built using the Teknekron Enterprise Toolkit: a program trading system and a global order routing system. Both applications are in production use..

3 Two Distributed Communication Paradigms

In data-intensive application environments, two distinct kinds of data flows are evident. Both are supported in the Teknekron Enterprise Toolkit.

The first kind of data flow is related to data capture and dissemination. In a semiconductor fabrication facility, "work-in-progress" information concerning the movement of silicon wafers through the various fabrication processes must be captured and disseminated to process engineers as well as monitoring applications. In a bank, loan applications must flow through several distinct departments during the approval process. In a retail store, data captured in the inventory system must be disseminated to other departments such as accounting, purchasing, and marketing. This kind of data flow is critical for real-time decision support.

The second kind of data flow is concerned with queries, transactions, and control commands. Marketing personnel must access demographic information on consumers and match that against long-term selling trends. Bank customers enter transactions against their accounts. Station controllers issue commands to devices on factory floors. This kind of data flow is used traditional decision support, as well as in online transaction processing systems.

In data-intensive industries, the first kind of flow dominates, often constituting 60 to 70 percent of the total data flow. This is a data-driven or event-driven flow: it is initiated when a new data object is generated, or when a new event is detected. The dissemination of stock price data and the detection and dissemination of a critical temperature being reached are two examples of this kind of flow. The second kind of flow is more familiar to most people. This is a demand-driven flow: it is initiated when an application or a user either explicitly requests data or demands the execution of a transaction or command.

The two kinds of flow have very different interaction properties. In a data-driven / event-driven flow, the data exchange is typically "one-to-many." In network communication, this type of exchange is called a "multicast." Such an interaction paradigm suggests the use of multicast network communication protocols. In the demand-driven flow, the client initiates the data flow with a request in the form of a query, transaction, or command. This is the traditional client-server interaction model. Interaction is bilateral and "one-to-one." Such an interaction paradigm fits well with current communication protocols, which support "point-to-point" communication efficiently and reliably.

These two kinds of data flow also exhibit very different technical properties. The two kinds of data flows suggest the use of different underlying communication protocols. They imply differences in many other technical factors as well: they require different "session management" protocols, they require different failure and recovery protocols, and they imply different load-balancing algorithms. In fact, such differences must be factored into all levels of a system's architecture from low-level communication protocols to high-level session management protocols.

The communication paradigm used for demand-driven communication is remote procedure call. To support event-driven communication, the TIB provides **publish-subscribe**. Event-driven communication is typically asynchronous, with the publisher being able to continue processing without having to wait on the processing of the published event by subscribers. In contrast, request-reply interactions are inherently more synchronous. Publish-subscribe is anonymous, neither participant needs to know the identity of the other.

The event-driven interaction paradigm has an additional benefit: it encourages more loosely coupled components, i.e. components need to have minimal knowledge about one another. The most important couplings among modules are "semantic couplings," which are the processing and data assumptions that one module makes about

another. For publish-subscribe interactions, typically the only assumptions required are an agreed upon common set of data types that can be published / subscribed. On the other hand, for request-reply processing, a different kind of agreement is required-the server must specify a well defined set of operations that it supports. This includes an agreement on the request messages that invoke these operations, an agreement on the reply messages that return the results of these operations, and, most importantly, an agreement on the behavior of these operations.

The event-driven paradigm also leads to savings in network bandwidth. If only demand-driven communication primitives are available, applications requiring information about events would have to periodically "poll" other applications for information about these events. The more timely they need this information to be, the more frequently they have to poll. This introduces an unnecessary load on the network. With event-driven communication primitives, information is communicated to applications when it is produced, with no network overhead being incurred at other times.

Almost all software platforms today are tailored exclusively for one interaction paradigm or the other. In particular, the client-server paradigm is rapidly becoming the paradigm of choice in a new generation of distributed object-oriented applications. For example, typical implementations of Object Request Brokers based on the CORBA architecture only support request-reply interactions. However, as the above analysis reveals, most of the data flow in data-intensive industries is actually concerned with information dissemination, which is poorly supported by the client-server paradigm.

As the trend to migrate and reengineer monolithic applications from mainframes to open systems accelerates, and as the level of automation increases in business, data exchange will be increasingly initiated by a system generating an event, as opposed to by a human making a request, as is typical today. The support of both communication paradigms in the Teknekron Enterprise Toolkit makes it a powerful environment for building distributed workflow style applications.

4 The Teknekron Information Bus

The Teknekron Information Bus (TIB) platform is a body of software that facilitates the intelligent dissemination, organization, and integration of data in a distributed environment. The distributed design of the TIB platform facilitates reliable, high-performance exchange of data and services among applications.

As a software platform, the TIB platform implements a high-level system environment, making it much easier to develop and maintain applications. The TIB environment hides the boundaries of various systems and services from programmers, thereby enabling programmers to hide these boundaries from users. The environment also hides network configuration and heterogeneity among machine environments. Programming in the TIB environment enables the building of large software systems that, though complex, are still easy to develop and easy to use.

The TIB platform permits users and application developers to request information by subject through its patented *Subject-Based Addressing* capability. This is a simple, intuitive model where applications and services interact indirectly and anonymously through self-describing data objects which are labelled with user-meaningful subjects. The services generating the data objects and the breakdown of functionality among those services are transparent to the applications. Data objects are sent and received based on a subject, independent of the identities and location of data producers and data consumers.

The semantics of publish-subscribe communication depends on the requirements of the application. One semantics provided by the TIB is *reliable* message delivery. Under normal operation, if a sender and receiver do not crash and the network does not suffer a long-term partition, messages are delivered exactly once in the order sent by the same sender. If the sender or receiver crashes, or there is a network partition, messages may be lost.

A stronger semantics is *guaranteed* message delivery. In this case, the message is logged to non-volatile storage before it is sent. The message is guaranteed to be delivered regardless of failures. The publisher will

retransmit the message at appropriate times until a reply is received. Guaranteed delivery is particularly useful when sending transactions to a database over a network that may be disconnected for periods of time (the typical case in a globally connected enterprise).

For local area networks, reliable publication is implemented with Ethernet broadcast. This allows the same data to be delivered to a large number of destinations without a performance penalty. Moreover, Ethernet broadcast eliminates the need for a central communication server. The current implementation uses UDP packets in combination with a retransmission protocol to implement reliable delivery semantics.

In the implementation of subject-based addressing, a daemon is used on every host. Each application registers with its local daemon, and tells the daemon to which subjects it has subscribed. The daemon forwards each message to each application that has subscribed. It uses the subject contained in the message to decide which application receives which message.

For each requested subject, the TIB software determines the services that provide data on that subject and sends appropriately formatted data requests to those services. Any number of services can provide data on a particular subject. Therefore, through a single requested subject, data from multiple sources may be returned. The Subject-Based Addressing capability enables users and application developers to fully exploit the TIB platform's high-level information model.

Subjects are hierarchically structured, allowing data to be organized in user-meaningful ways. Subjects are set-up and controlled by the system administrator, and can be customized on a per site basis. Subjects can be partially specified or "wildcarded" by the consumer, which permits access to a large collection of data from multiple producers with a single request. The TIB itself enforces no policy on the interpretation of subjects. Instead, the system designers and developers have the freedom and responsibility to establish conventions on the use of subjects.

The resulting anonymous communication is a powerful mechanism for adapting to software changes that occur at run-time. A new subscriber can be introduced at any time and will start receiving immediately new objects that are being published under the subjects to which it has subscribed. Similarly, a new publisher can be introduced into the system, and existing subscribers will receive objects from it.

The TIB system model is an information-oriented model, in contrast to the connection-oriented model supported by most existing systems. The latter model is lower-level and yields rigid, tightly-coupled systems. The TIB model is high-level and yields systems that are flexible and de-coupled. Applications developed in the TIB platform exhibit the following kinds of decoupling:

- Source-Service Decoupling: Users should not have to specify from which service or database data is to be retrieved, nor should developers be forced to hard-code into their software the source of requested data. The system should support an "information-model" (rather than a traditional "service-model") and should allow the user or application to request information by "subject" (rather than by "service").

- Service Protocol Decoupling: This next level of decoupling insulates the application programmer from the differing access protocols from the various services. Instead, the programmer accesses services through a single generic interface, and the TIB directs each service request to the appropriate protocol for the referenced service. The various service access protocols are linked into the TIB in a modular fashion.

  Encapsulated service protocols not only insulate the programmer from the multiplicity of access methods, but it also insulates him/her from the failure-recovery procedures and the distribution management procedures used by a service. Such procedures are complex and contribute greatly to the inherent complexity in programming distributed applications.

- Data Model Decoupling: This next level of decoupling is provided through the use of *self-describing data objects*. Each data object contains enough descriptive information to allow receiving applications to interpret the format, organization, and simple semantics of the information inside. The advantages are: auto-

matic data conversions, non-programmatic evolution of existing data classes, and support of complex data structures (e.g., nesting and lists).

- Configuration Decoupling: This fourth level of decoupling removes application dependency on network topology and system configuration. It also removes location dependency concerning where services or applications are executed. This is achieved by "location transparent" naming. All software components, including internal TIB components, refer to services and applications in a location independent fashion. Network addresses and host names are never "hard-coded." Nor is there a necessity for a centralized configuration database that defines the location of all system components. Instead system components can be "discovered" by using the communication facilities of the TIB platform. In our model, the new implementation needs only use the same subjects as the old implementation; neither publishers nor subscribers must be aware of the change. Subject names can be rebound at *any* time to a new address, a facility that is more general than traditional late-binding.

5 Key Components

The Common Information Bus Interface (CIBI^{TM})

The Common Information Bus Interface (CIBI) is a uniform interface to the TIB's publish/subscribe facilities. CIBI provides a transport-level interface to the TIB. The Subject-Based Addressing capability supported in CIBI provides a simple rendezvous mechanism for publishers and subscribers. Currently the major service disciplines supported in the CIBI interface include:

- ESA - Extended Subject-Addressed,
- GDSS - Guaranteed Delivery Subject Service,
- TIBQ - TIB Queue.

All of the service disciplines support the asynchronous publish and callback for subscribed message communication paradigm; they all deliver messages in the order in which the messages are published. The ESA service discipline is a simple discipline with low protocol overhead. It provides *reliable* communication between publishers and subscribers but no application-level fault-tolerance. It uses a straightforward data dissemination algorithm. The advantage of ESA is that it is fast, computationally cheap and imposes no system administration overhead. It is ideal for non-critical services and end-user applications.

The GDSS service discipline provides *guaranteed* message delivery. Unlike the ESA discipline, subscriptions are not implicitly cancelled when a subscriber process exits; they must be explicitly cancelled. As long as a subscriber has not cancelled its subscription, it is able to receive all the messages published since the beginning of its subscription despite crashes it or the publisher may have.

The TIBQ discipline also provides guaranteed message delivery. However, under TIBQ, each message published is delivered to one subscriber, rather than to all subscribers. Therefore, the TIBQ discipline implements distributed queues. One usage of this service is in implementing services using redundant servers to achieve fault-tolerance and load balancing.

Super*Cell Router

The Super*Cell Router transparently integrates multiple local TIB environments into a global one. It is intended to meet the needs of enterprise-wide and inter-enterprise-wide information dissemination and integration.

The architecture is based on the fundamental notion of *Cells, SuperCells, and Super*Cells*. The *Cell* represents the smallest unit allowing independent system administration. It represents a single TIB environment, including a single subject naming context.

The Super*Cell Router uses daemons to interconnect cells. Within the cell, the daemon acts as a normal TIB application. It publishes and subscribes to data using the standard TIB communication environment. However, the daemons can also communicate with each other directly. In this way, one daemon can receive a message in one cell and have the message re-published in another cell using another daemon. Each daemon acts on behalf of all other applications in all other cells.

Used in conjunction with the Super*Cell Router, the CIBI interface provides an unlimited area messaging facility that works transparently over LANs and WANs. Thus, a ESA / GDSS / TIBQ message published from New York can be delivered to a subscriber at London, for example.

RMI

The TIB RMI package provides remote method invocation for use in a distributed object-oriented system. It can be used for seamless communication within a process, between processes on one machine, or between machines, to support the request/reply communication paradigm. Object name references are location independent and anonymous so that the application does not need to explicitly locate a remote object. Despite this flexibility, no central server is required to deliver messages. Hidden optimizations are built into the protocol so that repeated references to one object are streamlined.

The TIB RMI package is built on top of the TIB platform, utilizing its high-speed, highly reliable communication facilities. The TIB RMI package also uses the TIB platform's Subject-Based Addressing capability, and can optionally use its self-describing data facility. The Subject-Based Addressing capability provides the mechanism for initially communicating with objects, and it does not impose any system administration overhead. The self-describing data facility ensures that messages exchanged across the TIB platform will be understood by all parties, and it can be used as part of the larger object model.

Used in conjunction with the Super*Cell Router, the RMI package provides an unlimited area client/server rendezvous mechanism that works transparently over LANs and WANs.

Remote Object Framework (ROF)

The Remote Object Framework (ROF) facility brings the advantages of typed object-oriented interfaces to distributed workflow computing. Interfaces to the TIB can be defined using CORBA's Interface Definition Language (IDL) or Teknekron's TDL. A feature that distinguishes the ROF facility from other remote object systems is its level of dynamism. Interfaces can be defined and evaluated at run time, thereby allowing new types of services to be introduced into a running system. It is also possible to generate new types of senders and receivers at run time. So the ROF facility not only provides the means for building an extensible distributed system, it also permits extensions to be made to a running system.

6 Example Applications

In this section we illustrate the use of Teknekron Enterprise Toolkit in two different financial application. Both make use of different publish-subscribe service protocols. Both applications are used in production today. One of these applications is a globally distributed system that seamlessly extends the TIB beyond a LAN environment using the Super*Cell Router.

An Order Routing Application

TORUS (*T*eknekron *O*rder *Rou*ting *S*ystem) is a program trading facility widely used by major Japanese brokerage firms. It is designed to enable brokers to take advantage of market volatility and achieve optimum trading results. It can be used for such diverse purposes as portfolio insurance, stock index arbitrage, or simply for equity trading (with or without internal crossing-matching of orders that can be satisfied without going through the exchanges).

Architecturally, TORUS is composed of three layers:

- The first layer is an order routing and execution system. Built as a distributed system, it consists of a suite

of independent modules that exchange information over the Teknekron Information Bus (TIB) using the TIBQ guaranteed delivery service discipline.

- The second layer is an API which allows application programmers to create and submit orders for execution and monitoring of status changes.

- The third layer is the TORUS front-end, a program trading user interface which provides a comprehensive set of "push-button" trading operations, including basket/wave creation and editing, and basket order /execution matching.

The order routing and execution system consists of three primary components: order managers which submit orders to the exchange, execution managers which receive execution confirmations from the exchange, and the database manager which receives order and execution information from the order and execution managers and records them in a relational database.

The TORUS front-end provides traders with real-time updates on orders executed. Traders also have access to real-time market information from the TSE64K feed handler. The latter is a Teknekron-provided feed handler that interfaces to the Tokyo Stock Exchange (TSE64K) data feed and republishes the information in a canonical format, using the ESA service discipline.

Since orders and executions must be delivered to the trading venues and stored in the database reliably, TORUS uses TIBQ, the TIB protocol that stores items in reliable, disk-resident queues before delivering them. When a process, machine, or disk failure occurs, system state can be recovered from the disk-resident queues. No loss of orders, order status, or executions will be incurred.

Orders originate from the graphical TORUS front-end. They are submitted by applications written using the TORUS API. Orders consist of symbol, size, instructions, account number, basket id, and other fields. When a trader initiates a submit, reissue, or reverse operation, all the stocks in the wave are deposited into a submission queue. All the available order managers (one per relay station) take the orders from this queue, validate them, and transmit them to the TSE. If validation fails (for example, if a null limit price is specified) the order manager does not transmit the order, instead it deposits the order into the database queue and publishes it on the network with the status set to `REJECTED BY ORDER MANAGER`. If an order is sent to the TSE and then rejected, the order goes to the database queue with status set to `REJECTED BY TSE`. Accepted orders are deposited to the database queue and published on the network with the status set to `OK`. An accepted order contains the acceptance time, TSE acceptance sequence number, etc. The speed of order submission is limited only by the relay station's ability to accept orders and by the number of relay stations.

All orders, executions, and order status information are sent to the database asynchronously with respect to the delivery of orders or receipt of execution information to/from the trading venues. All order execution and order status information is logged in the database and published on the TIB, using the ESA service discipline. Applications that wish to show real-time order matching, and basket and order status can do so by monitoring the published information. Applications with less critical real-time requirements can query the database directly.

A Global Order Routing Application

This application is developed for one of the leading brokers in the United Kingdom. The firm has both trading and sales operations in tens of locations worldwide. It specializes in emerging market equity products, and its business is essentially cross-border.

Already a user of the TORUS system in Tokyo, this firm initially investigated traditional database replication schemes to address its needs but found the development estimates excessive and the administrative requirements overly complex. It subsequently contracted with Teknekron to solve its problem of distributing data across multiple locations. The resulting infrastructure component from Teknekron is TSM, or *Transaction Services Manager*, which functions as an application server for database access and which takes care of the asynchronous replication of data between multiple sites.

Each TSM transaction is encapsulated in a TDL (Teknekron Design Language) object. These TDL objects, when submitted into the TSM system, are routed and delivered to the appropriate services for processing. TSM provides full or selective transaction replication across wide or local area networks. With full replication, mirrored databases can be maintained in multiple locations. The TSM server is event driven and uses the store-and-forward approach to replicate transactions, providing high availability and low data communication overhead.

TSM is structured as a Remote Object Framework server that replicates databases and other transaction resources at local and remote sites. It runs either in a sender or a receiver mode. In sender mode, it serves as an application server that provides transaction services. It updates the local database and then publishes the updates on the TIB using the GDSS service discipline. In receiver mode, it subscribes to the updates published by other replication sites and reapply those updates to the local database copy. The implementation of TSM utilizes the following components from the Teknekron Enterprise Toolkit:

- DBIS, a generic relational database interface (based on the X/Open CLI standard), for access to the underlying relational database.
- GDSS for guaranteed delivery of messages.
- Super*Cell for the transparent routing of messages over a wide area network.
- ROF for the communication between client applications and the TSM server.
- Object Sheet, a graphical application builder, for the administration console.

With TSM handling the data distribution requirement, this firm has implemented a global order and position management system that provides automated order entry, order routing, and execution systems for the sales desks; position monitoring, deal capture, and real-time analytical functions for the trading desks; a system to support a global risk control function, and a global client information system.

Software Tools for a Process Handbook

Abraham Bernstein Chrysanthos Dellarocas Thomas. W. Malone
John Quimby
MIT-Sloan School of Management
Center for Coordination Science
1 Amherst Street
Cambridge, MA 02139
E-mail: {avi, dell, malone, quimby}@mit.edu

Abstract

This paper provides a progress report on the development of software tools in the Process Handbook project currently underway at the MIT Center for Coordination Science. We begin with a brief overview of the project as a whole. Then we focus on software tools emphasizing aspects that relate to workflow control. Finally, we conclude with a brief description of future avenues of research. The process handbook tools help (a) redesign *existing organizational processes, (b)*invent *new organizational processes that take advantage of information technology, and finally (c) automatically* generate software *to support organizational processes. An important related goal is the ability to (d)* import *and export process descriptions from and to other process modeling architectures. The approach combines in a novel way the ideas of process* decomposition*, process* specialization*, and the* coordination *of* dependencies *between activities. The paper presents an overview of findings from multiple implementations of this approach.*

1 The MIT Process Handbook Project

The goal of the Process Handbook project [8, 4] is to provide a firmer theoretical and empirical foundation for such tasks as enterprise modeling, enterprise integration [3], and process re-engineering [9, 6]. The project includes (1) *collecting* examples of how different organizations perform similar processes, and (2) *representing* these examples in an on-line "Process Handbook" that includes the relative advantages of the alternatives. The Process Handbook can be thought of as an Organizational-CAD tool. It is intended to help (a) *redesign* existing organizational processes, (b) *invent* new organizational processes that take advantage of information technology, and finally (c) *automatically generate software* to support organizational processes. An important related goal is the ability to (d) *import and export* process descriptions from and to other process modeling architectures.

The work in this project can be divided into three main categories, each of which is briefly discussed below: *theory development, field studies* and *tool building*.

1.1 Theory Development

A key element of the work is a novel approach to representing processes at various levels of abstraction. This approach uses ideas from computer science about inheritance and from coordination theory [10, 11] about managing dependencies. Its primary advantage is that it allows users to explicitly represent the similarities (and differences) among related processes and to easily find or generate sensible alternatives for how a given process could be performed. Section 2 provides a brief introduction to the major ideas behind our representation.

1.2 Field Studies

In parallel with developing theories and software to support the process handbook concepts, we are also conducting a series of field studies to gather data for inclusion in the handbook and to refine our concepts and methodologies for representing processes. Example studies we have conducted include:

- study of supply chain processes in athletic footwear companies
- study of software development management processes at a major computer manufacturer
- study of healthcare delivery processes at a community hospital
- comparative study of supply chain processes in (1) supplying pharmaceuticals in a hospital (2) telephone equipment installation in a regional telephone company, and (3) third-party software acquisition in a computer manufacturer.
- study of several processes in corporate banking

1.3 Tool building

In order to refine and test our ideas about how a process handbook should work, we have developed a series of prototype systems embodying various aspects of the functionality envisioned in our proposal. Sections 3 and 4 provide a detailed review of where we are and what our vision is.

2 Theoretical Background

A key to this project is developing novel techniques for representing organizational processes. Our goal is to use advanced process representation techniques in order to:

- assist process analysis and identification of inefficiencies
- facilitate generation and comparative evaluation of alternative processes

Space precludes a full description here of our approach to representing processes. Briefly, however, the representation used in the handbook combines three basic concepts in a novel way to create a taxonomy of processes (see [8] for a complete description).

1. *Decomposition*. Processes are decomposed into activities, which may in turn be further decomposed into subactivities. Decomposition allows the nesting of processes within processes, and allows the handbook to share and re-use process descriptions throughout the taxonomy. Decomposition itself is, of course, not a novel element in process representation, but it is combined here in new ways with the other two elements.
2. *Specialization*. Processes (and activities) are specialized in a manner similar to a traditional type hierarchy. Unlike a simple object hierarchy however, each node is itself a complex entity that inherits a decomposition and the associated dependencies from its parents. Loosely speaking, our notion of process specialization can be thought of as a ”dual” of traditional object-oriented programming. Instead of inheriting down a hierarchy of objects with associated actions (methods), we inherit down a hierarchy of actions (processes) with associated objects.

Dependency	*Examples of coordination processes for managing dependency*
Shared resources	”First come/first serve”, priority order, budgets, managerial decision, market-like bidding
Task assignments	(same as for ”Shared resources”)
Flows	
Prerequisite constraints	Notification, sequencing, tracking
Inventory	Inventory management (e.g., ”Just In Time”, ”Economic Order Quantity”)
Usability	Standardization, ask users, participatory design
Design for manufacturability	Concurrent engineering
Simultaneity constraints	Scheduling, synchronization
Task / subtask	Goal selection, task decomposition

Figure 1: Examples of common dependencies between activities and alternative coordination processes for managing them. (Indentations in left column indicate more specialized or decomposed versions of general dependency types.)

3. *Dependencies.* The third key concept is the notion from coordination theory that coordination processes can be thought of as ways of managing dependencies among activities [10, 8]. Organizational processes can be viewed as containing both *production* and *coordination* components. The production component includes the process activities that are physically or logically necessary to achieve the stated goals of the process. The coordination component consists of the activities necessary to properly manage the dependencies among production activities.

We are testing the hypothesis that most coordination processes encountered in practice can be viewed as alternative ways of managing a relatively small set of elementary dependency types. From this perspective, we can build a taxonomy of basic dependency types and associate each dependency type with a specialization hierarchy of alternative coordination processes for managing it. Furthermore, it seems that many coordination processes are applicable across a large range of functional domains. For example, a coordination process used to manage a flow dependency in a manufacturing process, may be used intact or with minor modifications to manage a similar dependency in a finance process.

Table 1 [8] suggests the beginnings of such an analysis. The table shows a set of common dependencies between activities together with examples of alternative coordination processes used to manage them.

Note that dependency types themselves form a specialization hierarchy, with more specific types inheriting (and possibly specializing) the set of managing processes of more general dependency types. For instance, *task assignment* can be seen as a special case of allocating shared resources. In this case, the ”resource” being allocated is the time of people who can do the tasks. This implies that the coordination processes for allocating resources in general can be specialized to apply to task assignment.

2.1 Advantages of activity-dependency representation

Abstraction enhances understanding. Coordination activities are often relatively ”low-level” (e.g., ”Complete requisition form”, ”Ship packet by courier”, etc.), compared to the production activities whose dependency they manage. Traditional process representations, which mix production and coordination activities in the same picture, very fast become cluttered with detail. This hinders understanding the essence of the process, as well as the purpose of coordination activities. Separating coordination activities and ”hiding” them behind dependencies,

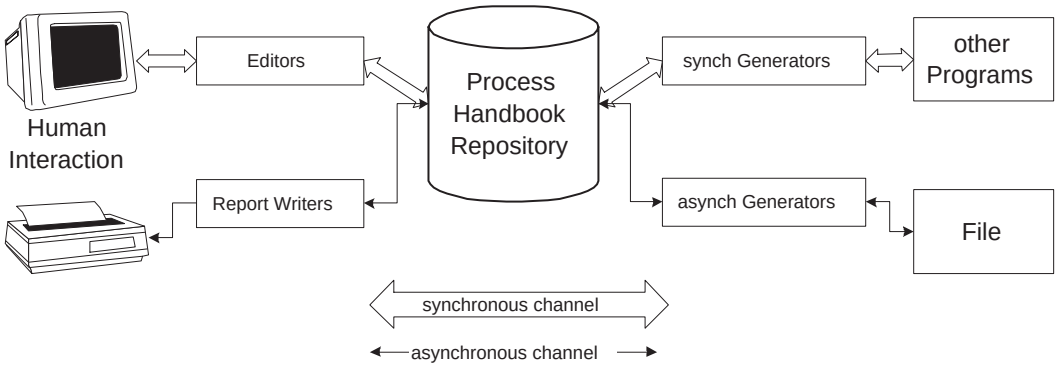

Figure 2: The Process Handbook Architecture

results in simpler representations, which highlight the essence of a process. In addition, it associates coordination activities with their underlying dependencies. Thus, the purpose (or lack of it) for each coordination activity is made clear.

Dependencies enhance generativity. As we mentioned earlier, each dependency type is connected to a specialization hierarchy of alternative coordination processes for managing it. The association of an existing coordination process with its underlying dependency, not only improves understanding of the business process, but also provides immediate access to a large number of alternative ways of performing that coordination, extracted from business processes in many different domains. This helps produce ideas for generating improved alternatives for the studied process.

3 The Process Handbook Tools

3.1 The Process Handbook Architecture

The Process Handbook usage scenarios propose several different types of usage by humans and machines. In general, there are two types of interaction with the process handbook repository: *synchronous* and *asynchronous*.

The synchronous channel tools are directly connected to the process handbook repository throughout their usage. In most cases they change the process handbook repository. We call our synchronous tools that enable human computer interaction *editors*. Synchronous tools that enable inter program communication are synchronous generators. Synchronous channels usually use function libraries to access the process handbook repository API directly in combination with other standard inter process communication protocols like sockets, DDE, RPC, and OLE.

Report writers are asynchronous tools that interact with people through some other medium. Asynchronous generators extract certain parts of the process handbook repository and generate some output file.

3.2 The Repository

The process handbook repository is the ”memory” of the process handbook. It stores the process descriptions using the representation techniques described in section 2 and ensures the data integrity. The span of the usage scenarios asks for a scaleable solution that can be used on different systems ranging from laptops to large server based process repositories. To solve this scalability problem we chose a relational database system as our storage engine, which is wrapped by two layers that simplify the interaction of the different tools with the repository.

The inheritance layer provides the process inheritance functionality needed by the process handbook. For instance, it propagates changes made to a process at one level to all the specializations of that process at lower

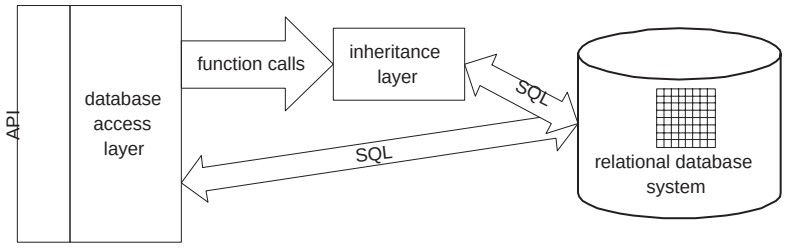

Figure 3: The Repository Architecture

levels. It would presumably have been easier to use an object-oriented database system for this purpose. However we were unable to find a suitable object-oriented database available for the PC/Windows environment in which our system is implemented.

Our database access layer [2] encapsulates all database-operations. This middleware-layer introduces an abstraction layer between the object-oriented data and the relational database system. Through its API, it offers different means of navigation through the object- like Process Handbook space. This abstraction layer is also being designed to integrate an additional repository based on Lotus Notes. This alternate repository is restricted in the sense that it does not have the full richness of the relational database-structure. However, it is optimized to store rich text descriptions of processes, doing full text retrieval, and support the collaborative dialogues of field researchers working remotely in the field. One of our goals for the abstraction layer (not yet fully implemented), is that all programs accessing the Process Handbook repository can access the information in both storage engines.

3.3 The Editors

One of the major aspects of the process handbook project is to help its users in their task of browsing through process descriptions, changing them, simulating, and executing them. The way users interact bi-directionally with the process handbook repository is through Editors. To test the hypothesis that the process handbook captures a superset of the information in other systems, a number of editors for different types of notations have been implemented.

The first group of editors are tree editors. They are used to browse and edit the specialization and decomposition trees of processes. They are implemented as graphical, direct manipulation programs, and therefore provide very direct access to structural information in the process handbook repository.

The second type of editors are additional (non tree-) graphical editors. They represent process information in different notational formats. These editors so far include an IDEF0 editor, a Dataflow editor, and a Flow-chart like editor that allows the exact representation of dependencies between activities. This latter editor also supports users in choosing the appropriate managing activities for dependency and can be used to activate a special software generator (see section 3.5)

The third group of editors are text based. They allow navigation through the detail sheets of processes, and dependencies with hypertext-like links. These editors are mainly used to enrich the process handbook repository with detailed information about its objects rather than to change structural information. These details include full text descriptions and statistical information about the processes which can be used to drive a process animation viewer, and can also be exported for use by external applications. An example of such an external application would be a commercial process simulator.

3.4 The Report Writers

Report writers are used to produce paper representations of the information stored in the process handbook repository. This includes printing the graphical notations, printing textual information about processes and exporting statistical data. In some cases, data is exported to a specialized drawing program for users to further edit manually.

3.5 Synchronous Generators

A whole set of programs focuses on direct communication between the process handbook repository and outside applications. We call these tools synchronous generators, because they are in bi-directional communication with the repository.

One of the tools with which we are experimenting takes process handbook process descriptions and executes it as open nested transactions [12]. This particular program extracts the process descriptions from the process handbook repository and acts as coordination process for those processes. By observing the rules of open nested transaction execution the system ensures the correct execution of the processes. The current prototype based on [1], communicates with independent interfaces, supporting actions in a distributed environment. In addition to ensuring the execution of the transactional workflows, the system has a management interface that graphically shows the current state of a particular, focused activity.

3.6 Asynchronous Generators

In addition to direct communication with other programs, it is also possible to communicate indirectly with other programs through asynchronous file exchange. One of the efforts related to the process handbook project has been the development of a Process Interchange Format (PIF see [7]). The PIF was developed by a working group consisting of representatives from several universities and companies. The goal of the PIF project is to automatically exchange process descriptions between different systems.

To achieve this goal, each system will only need to have a translator to convert data between its native format and the common PIF format. The format itself is a representation of process entities, such as activities and resources, along with their attributes and relationships. In addition to the core set of object types, the PIF provides a framework for extending this set of types to include additional information needed in specific applications. Currently, our PIF translators are the primary method of communication for handbook tools operating on different databases or platforms. In the larger scope of the PIF Working Group, we have also performed experimental translations among systems developed by different research groups. Our team is writing a platform-independent C++ toolkit [5] which will simplify the task of writing future translators, not only for our group but also for anyone who would like to share business process descriptions using this format.

4 Future Vision

The wide range of user profiles of the Process Handbook from business school students to organizational theorists to management consultants presents an interesting set of user interface challenges. Advancing the usability of the Process Handbook based on a wide range of usage scenarios is one line of ongoing work. This includes more advanced process representations and richer manipulation metaphors that can be tailored according to usage patterns of the tool users. Even though we are encouraged by our progress so far in process representation, we believe that process representation is still in its infancy. By analogy to the history of musical notation, our current work can be seen as analogous to the use of a G Clef. Imagine what it will be like to add measure bars, time signatures, and dynamic markings!

A particularly rich area for ongoing research is the automatic and semi-automatic generation of operational systems and the control of those systems from the Process Handbook. Can the imagining, specification, implementation, deployment, operational control, and evolution of a production system be supported from within a single integrated framework? What are the changes in tool sets and in the design practices required to support such a vision? These are the kinds of questions our work so far on the Process Handbook raise for us.

5 Acknowledgments

The Process Handbook is a team project. A number of researchers are working on and have made significant contributions in the various areas of the project. We would especially like to acknowledge the work of Erphan Ahmed, Steven Buckley, Frank Chan, Kevin Crowston, Naved Khan, Jintae Lee, Greg Pal, Brian Pentland, Narasihma Rao, George Wyner, Greg Yost, and Gilad Zlotkin.

References

[1] Bernstein A. Specification and implementation of workflows in banking environments. Master's thesis, Diploma Thesis at the Computer Science Division of the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1993.

[2] Erfanuddin Ahmed. Thesis for master of engineering degree in electrical engineering and computer science. Master's thesis, Massachusetts Institute of Technology, June 1995.

[3] Petrie C. Enterprise integration modeling. 1992.

[4] T. W. Malone K. Crowston C. Dellarocas, J. Lee and B. Pentland. Using a process handbook to design organizational processes. *Proceedings of the AAAI Spring Symposium on Computational Organ. Design*, pages 50–56, 1994,.

[5] Frank Yeean Chan. The round trip problem: A solution for the process handbook. thesis for master of engineering degree in electrical engineering and computer science. Master's thesis, Massachusetts Institute ofTechnology, June 1995.

[6] Champy J. Hammer M. *Reengineering the Corporation*. Harper Business, 1993.

[7] Yost G. Lee J. and the PIF Working Group. The pif process interchange format and framework, version 1.0. 1994.

[8] Lee J. Malone T.W., Crowston K. and Pentland B. Tools for inventing organizations: Toward a handbook of organizational processes. *Proceedings of the 2nd IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises*, 1993.

[9] Davenport T.H. *Process Innovation.* Harvard Business School Press, Boston, Massachusetts, 1992.

[10] Malone T.W. and Crowston K.G. Toward an interdisciplinary theory of coordination. 1991.

[11] Malone T.W. and Crowston K.G. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26, 1994.

[12] Schaad W. Weikum G., Deacon A. and Scheck H. Open nested transactions in federated database systems. *IEE Data Engineering No. 2, Special Issue on Workflow and Extended Transaction Systems*, 16, 1993.

CALL FOR PAPERS

DATA ENGINEERING

12th International Conference on Data Engineering

February 26 - March 1, 1996
New Orleans, Louisiana, USA
Sponsored by the IEEE Computer Society

IEEE

SCOPE

Data Engineering deals with the use of engineering disciplines in the design, development and evaluation of information systems for different computing platforms and system architectures. The 12th Data Engineering Conference will provide a forum for the sharing of original research results and practical engineering experiences among researchers and practitioners interested in all aspects of data and knowledge management. The purpose of the conference is to share solutions to problems that face our information-oriented society and to identify new challenges and directions for future research.

TOPICS OF INTEREST

The topics of interest include but are not limited to:

- Data Engineering and Information Superhighways
- Mobile Computing
- Network Databases and Security
- Knowledge Mining and Discovery
- Interoperability of Heterogeneous Information Systems
- Distributed Transaction Management
- Query processing and Optimization
- Virtual Enterprise Management
- Work Management and Simulation
- Multimedia Systems and Applications
- Open Architecture and Extensible Systems
- Visualization and User Interface
- Active Database/Knowledge Base Systems
- Intelligent and Deductive Systems
- Parallel and Distributed Database Systems
- Object-oriented Databases and Software Engineering
- Real-time, Temporal and Spatial Systems
- Industrial Challenges in Data Engineering
- Scientific and Statistical Databases
- Information Systems for Engineering Application
- Databases and Information Retrieval

PAPER SUBMISSION

Six copies of original papers not exceeding 6000 words (25 double spaced pages) should be submitted by May 31, 1995 to program chair:

Stanley Y. W. Su
Database Systems R&D Center
470 CSE Building
University of Florida
P. O. Box 116125
Gainesville, FL 32611-6125
E-mail: icde96@cis.ufl.edu
Tel. (904) 392-2680, FAX: (904) 392-1220

Authors should explicitly specify in the cover page one or two topics of interest most relevant to the submission. Submissions for the industrial program should also be specified in the cover page.

ORGANIZING COMMITTEE

General Chair: Marek Rusinkiewicz, University of Houston
Program Chair: Stanley Y. W. Su, University of Florida
Industrial Program Chair: Umesh Dayal, Hewlett-Packard Laboratories
Panel Program Chair: Arie Segev, University of California at Berkeley
Tutorial Program Chair: Witold Litwin, University of Paris

PROGRAM VICE CHAIRS

Data Engineering and Information Super Highways
Gunter Schlageter, Univ. of Hagen

Extensible and Active Database/Knowledge Base Systems
Sharma Chakravarthy, Univ. of Florida

Interoperability of Heterogeneous Information Systems
Dennis McLeod, Univ. of Southern California

Parallel and Distributed Database Systems
Chaitanya K. Baru, IBM Toronto Labs

Scientific and Statistical Databases and Applications
Arie Shoshani, Lawrence Berkeley Laboratory

Object-oriented Databases and Software Engineering
Luqi, Naval Postgraduate School

Multimedia Systems and Applications
William Grosky, Wayne State Univ.

Real-time, Temporal and/or Spatial Systems
Alex Buchmann, Technical Univ. Darmstadt

INDUSTRIAL, PANEL &TUTORIAL PROGRAMS

There will be a series of sessions focused on issues relevant to practitioners of information technology. Proposals for the industrial program, panels and the tutorial program are being sought. Proposers should submit a short write-up to the persons in charge of the Industrial, Panel and Tutorial programs.

PUBLICATIONS & AWARDS

All accepted papers will appear in the Proceedings published by IEEE Computer Society. The authors of selected papers will be invited to submit extended versions for possible publication in the *IEEE Transactions on Data and Knowledge Engineering* and in the *Journal of Distributed and Parallel Databases*. An award will be given to the best paper. A separate award honoring K.S. Fu will be given to the best student paper (authored solely by students).

IMPORTANT DATES

- Paper, Panel, and Tutorial submissions: May 31, 1995
- Notification of acceptance: October 2, 1995
- Tutorials: February 26-27, 1996
- Conference: February 28 - March 1, 1996

EUROPEAN COORDINATOR

- Keith G. Jeffery, Rutherford Appleton Lab

FAR EAST COORDINATOR

- Ron Sacks-Davis, CITRI

EXHIBITS PROGRAM

- Wei Sun, Florida International Univ.

PUBLICITY CHAIR

- Dimitrios Georgakopoulos, GTE Laboratories

FINANCIAL CHAIR

- Stephen Huang, Univ. of Houston

REGISTRATION

- to be named

LOCAL ARRANGEMENTS

- William Buckles, Tulane Univ.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398