Automating Workflows for Service Provisioning: Integrating AI and Database Technologies

Michael N. Huhns and Munindar P. Singh
Microelectronics and Computer Technology Corporation
Information Systems Division
3500 West Balcones Center Drive
Austin, TX, U.S.A. 78759-5398

Abstract

Workflows are the structured activities that take place in information systems in typical business environments. These activities frequently involve several database systems, user interfaces, and application programs. Traditional database systems do not support workflows to any reasonable extent. Usually human beings must intervene to ensure their proper execution. We have developed an architecture based on AI technology that automatically manages workflows. This architecture executes on top of a distributed computing environment. It has been applied to automating service provisioning workflows; an implementation that operates on one such workflow has been developed. This work advances the Carnot Project's goal of developing technologies for integrating heterogeneous database systems. It is notable in its marriage of AI approaches with standard distributed database techniques.

1 Introduction

The Carnot Project at MCC seeks to develop a variety of technologies that enable the integration of heterogeneous data and information resources. Project deliverables include an environment for the development of complex multisystem applications that access information stored in preexisting heterogeneous systems and maintain consistency constraints across them [5, 8]. An important component of this effort is a facility for *workflow management* [2, 10, 11]. Briefly, workflows consist of *tasks*, appropriately structured. A task is any unit of computation that performs some useful function in a system. The tasks that are of particular interest are database transactions, but other computations, e.g., those that generate visualizations, can be presented in the same framework.

Integrating preexisting systems is in general a harder problem than designing distributed systems afresh. Many systems, especially those based on older mainframe architectures, allow data to be accessed only through arcane interfaces of limited functionality. The systems and their interfaces cannot be easily modified, and our work assumes they cannot. This is because of two main reasons: (1) the complexity of the programming effort that would be required to achieve any modifications, and (2) the constraint that older applications continue to run as before, since they typically have a wide user base that relies heavily upon them. Thus, the integration must permit newly developed applications to coexist with previous applications.

The major goal of the Carnot Project is to create general principles and approaches for integration of heterogeneous information resources. The Carnot Project is distinguished from other database research projects not only in terms of its goals, but also in having a larger and more significant AI component than is perhaps typical. We also undertake various *application partnerships* with our sponsors in order to develop prototype systems that address their specific problems. This not only serves to test our research ideas, but also suggests important research problems to work on. We did one such application partnership with one of our clients, a telecommunications company. In this paper, we describe the key ideas of our ongoing research, as well as how they were applied to the problems of this client. In Section 2, we describe how workflows and AI fit into heterogeneous information systems. In Section 3, we describe the specific problem we addressed and, in Section 4, our solution to it.

1043-0989/94 $03.00 © 1994 IEEE

2 Background

Classical transaction processing in databases deals with executing access and update tasks on a single database. Such tasks are traditionally assumed to have the so-called ACID properties: *atomicity*, where each task happens either fully or not at all; *consistency*, where each task takes the database from a consistent state to a consistent state; *isolation*, where the intermediate results of a task are not visible to another task; and *durability*, where the changes caused by a task are persistent.

These assumptions help simplify transaction management considerably. However, they prove to be overly restrictive in loosely coupled heterogeneous environments. For example, one of the ways in which ACID tasks may be coordinated is through mutual commit protocols, which ensure that either all of a given set of tasks commit or none do. Such protocols, the classical example of which is the two-phase commit protocol, are notoriously inefficient when executed over networks. Further, to execute such a protocol, one requires access to the internal states of transactions, such as their precommit states. A transaction is in its precommit state when it is internally ready to commit, but is awaiting permission from the transaction manager to do so. Most commercial database systems do not provide access to such internal states, thereby making direct implementations of commit protocols extremely difficult.

The ACID properties are naturally realized when the correctness of database transactions is characterized through some purely syntactic or structural criterion, such as serializability [3]. However, serializability cannot be efficiently implemented in distributed systems whose component systems are autonomous. Instead of attempting to characterize correctness criteria purely syntactically, following [7], we attempt to characterize them semantically. This allows us to specialize the correctness criteria to the given application at the cost of building a deeper model of the application domain. This helps simplify several coordination requirements. For example, instead of executing mutual commit protocols, we can optimistically commit different tasks. If this action should prove erroneous, we undo the effects of incorrectly committed tasks. This is achieved by means of *compensating* transactions, whose definition depends on the semantics of the underlying domain.

Consequently, in heterogeneous environments, the unit of relevant activity is not a single database transaction, but rather a *workflow* that executes over a set of database and information resources. The constituent tasks of a workflow may be individually ACID, but the overall workflow usually is not. The problem is to ensure that no semantic constraint of the information model is violated despite this.

The activities that comprise a workflow of interest are typically already being carried out in the given organization. However, they are usually carried out by hand, with people intervening in several crucial stages to ensure that the necessary tasks are done and that organization-wide consistency constraints are enforced. The semantics that we alluded to above is supplied by the people or is implicitly encoded in different business procedures. The canonical examples of workflows are document flows through organizations. For instance, when an order is received, it must be entered into the system and several decisions must be taken to process it properly. These decisions would typically involve access to several information resources within an enterprise and possibly some outside of it. For example, a request to transfer money from one account to another requires that the authorization be verified, the account numbers be validated, and the source account be tested to have the required balance. External sources of information would be accessed for other requests, such as loan applications, where a credit bureau's databases may be consulted to determine the credit worthiness of an applicant.

It is of great importance to be able to handle the myriad error conditions that may arise in different workflows. The exception conditions in workflows are ones that are the hardest to automate. It is in identifying and resolving such conditions and managing control and data flow appropriately that AI technology can contribute substantially.

3 The Problem: Workflows for Service Provisioning

One of our clients provides a variety of telecommunication services. We studied the workflow for provisioning one such service that establishes a telecommunication link between two specified points. In the extant workflow, a set of paper forms is received that gives a number of relevant details about the service being ordered. These forms are entered into the system. A test is then performed to determine if certain essential telecommunication equipment is already in place. If it is, the service can be provided relatively quickly; otherwise, the processing must be delayed until the equipment is added.

Service provisioning typically takes several weeks

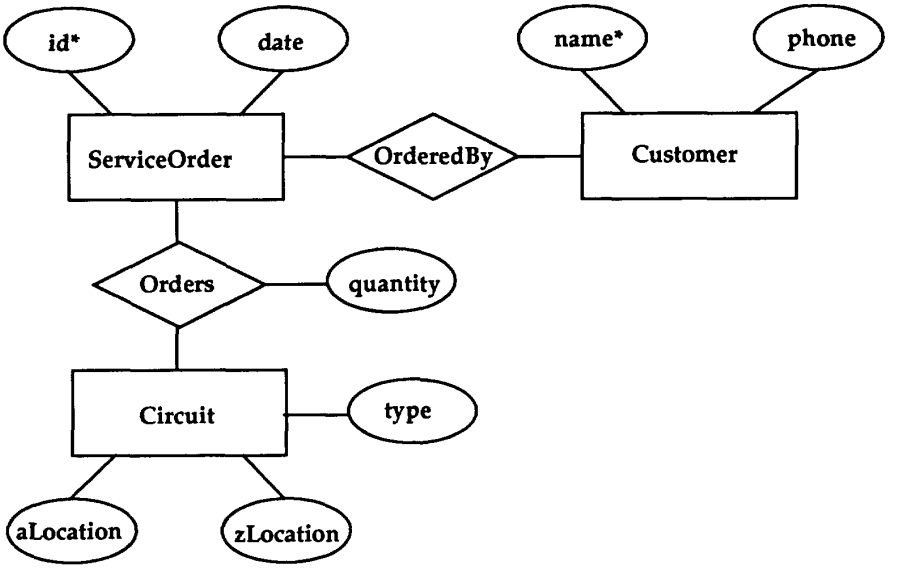

Figure 1: Abbreviated Semantic Model of the Provisioning Environment

and requires coordination among many operation-support systems and network elements. Configuring the operation-support systems so that they can perform such a task often takes several months to complete. This proves to be of competitive significance in the business environment in which our client operates. Many of its competitors were formed in the last decade or so. Unlike our client, these companies are not encumbered with legacy systems and typically have more modern computational facilities.

We investigated ways to improve the provisioning of one type of communication facility—digital services (DS-1). Provisioning DS-1 takes more than two weeks and involves 48 separate operations—23 of which are manual—against 16 different database systems. Our goals were to reduce this time to less than two hours and to provide a way in which new services could be introduced more easily. Our strategy for accomplishing these goals was to (1) interconnect and interoperate among the previously independent systems, (2) replace serial operations by parallel ones by making appropriate use of relaxed transaction processing [4, 6, 1], and (3) automate previously manual operations, thereby reducing the incidence of errors and delays.

An important goal of our project was to exhibit the feasibility of a workflow management approach that applies to workflows in general. Our specific challenge was to automate the DS-1 workflow as a test-case to achieve the efficiencies of our client's competitors without discarding its legacy systems. We should note, however, our implementation is not meant at this stage for production use, but as a proof-of-concept exercise.

Figure 1 presents an entity-relationship diagram that shows the most relevant components of the semantic model of the provisioning problem. Figure 2 presents the basic structure of the workflow we studied—it shows the admissible executions when everything works correctly. Each node denotes a task. The partial order reflects the dependencies among the different tasks. Tasks cannot be initiated until all their dependencies are met; ordinarily, they must be initiated if those dependencies are satisfied.

4 The Carnot Solution

We defined a distributed agent architecture, shown in Figure 3, for intelligent workflow management that functions on top of Carnot's distributed execution environment. An important constraint on our design was to use existing procedures as much as possible so as to ensure that other applications were not adversely affected by our system. This turned out to be easily accommodated by our architecture; indeed, we welcomed not having to worry about the details of

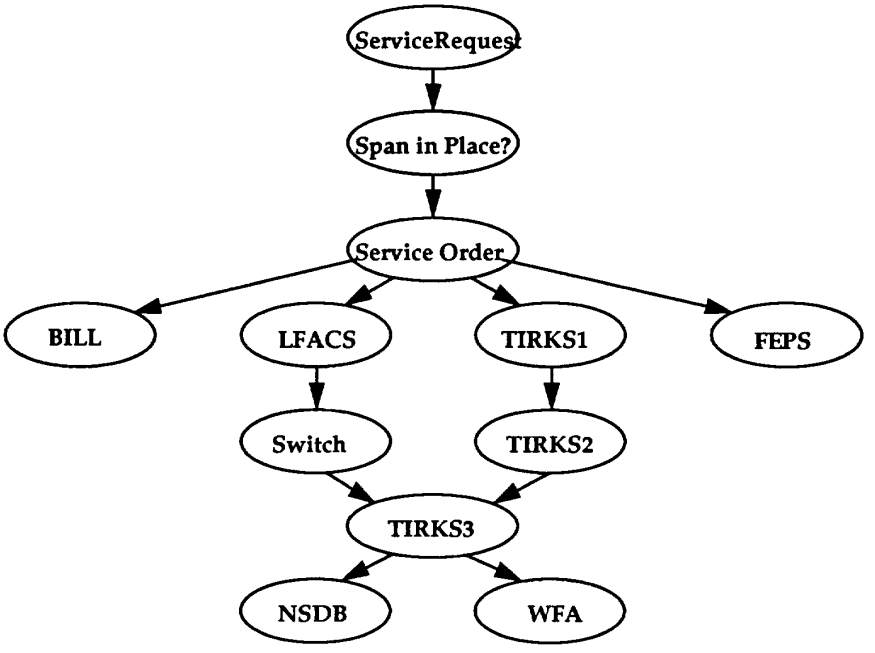

Figure 2: The Provisioning Workflow Automated

the mainframe systems on which we ran various tasks. Since the actual applications executed by the workflow were assumed to be defined already, our goal was to manage the overall structure of the applications in as domain-independent a manner as possible.

Our multiagent system consists of four agents that interact to produce the desired behavior. Figure 3 shows the key components of our architecture. The databases mentioned on the figure are assumed to include the relevant data and application programs that execute on them. The necessary applications are executed by the schedule processing agent; the user interface agent queries the systems to help a user fill in an order form completely and correctly, and to provide feedback about progress. This enables the detection of data inconsistencies. It is highly desirable to resolve inconsistencies early in the process.

The present architecture is enabled by our previous integration of an expert system shell, which has forward and backward chaining capabilities, a type system, and truth maintenance, into Carnot's distributed execution environment. This environment provides the basic message passing facility that our agents use to interact with other agents anywhere on the network. We used this facility to implement a scheme by which agents can exchange assertions, thereby triggering or disabling rules in each other. We augmented our scheme so that agents that are not expert systems can also participate in interactions, provided they satisfy a simple protocol. This enabled us to integrate transparently a graphical interaction agent, which is not an expert system shell, into the multiagent system.

Figure 4 describes our implementation at a high-level as an entity-relationship diagram. A key point to note is that the different tasks that correspond to the nodes of Figure 2 are modeled as database transactions. Each such transaction is initiated by an agent. Each task has associated with it a message type. The message type essentially encodes the computation that the underlying IMS databases must execute. When an agent executes a task, it does so by passing along the relevant message, i.e., the name of the file that contains it.

The agents operate as follows. The graphical-interaction agent helps a user fill in an order form correctly and completely, and checks inventories to give the user an estimate of when the order will be completed. It also informs the user about the progress of the order. The scheduling agent constructs the initial schedule for the given request, doing so on the as-

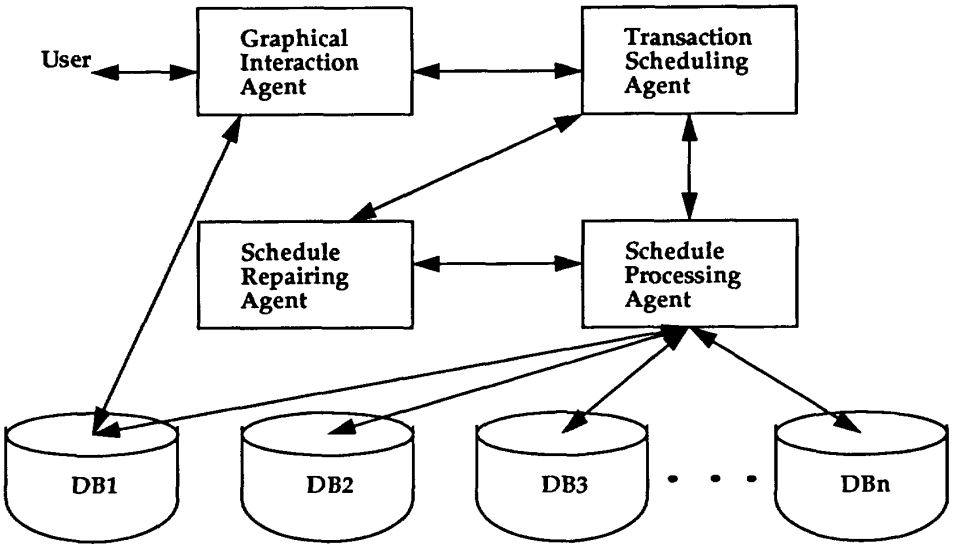

Figure 3: A Distributed AI System for Workflow Management

sumption that the relevant subtasks will succeed. The tasks are scheduled with the maximum concurrency possible, while still satisfying all required precedence constraints.

The schedule processing agent executes the schedule by invoking different tasks as necessary. It maintains connections to the databases involved in telecommunication provisioning, and implements transactions on them. The schedule processor also ensures that different workflows do not interact spuriously. This is akin to the problem of concurrency control in traditional database systems. Concurrency control has to do with ensuring that different transactions that access the same data items do not access them in relative orders for which there are no equivalent serial executions. With a workflow, we need to ensure that subtasks on each database can be serialized in the same order. This may require delaying, or aborting and retrying, different subtasks.

If the schedule processor encounters an unexpected condition, e.g., the failure of a task, it notifies the scheduling agent, which communicates with the schedule repairing agent for advice on how to fix the problem. The advice can be information on how to restart a transaction, how to abort a transaction, how to compensate for a previously committed transaction, or how to clean-up a failed transaction. These actions are meant to restore semantic consistency across the system. For example, if the system is unable to allocate a span to a given service request, it aborts it. The billing task, if already begun, is aborted. On the other hand, if the billing task fails, while the span allocation succeeds, the service order is allowed to proceed and the billing task is retried later. A conceptual model for the knowledge of the schedule-repairing agent is shown in Figure 5.

In our approach, the initial schedule is constructed on the assumption that things will succeed as expected. This leads to a small, easily executable, schedule. If error conditions should arise, they are accommodated at run-time by repairing the initial schedule as appropriate. Some of this is automatic, since the undesirable and unexecuted parts of the schedule are disabled by the truth maintenance system of the scheduling agent when their preconditions fail to hold.

The basic structure of this system is domain-independent. The details of the messages are clearly domain-dependent. Certain parameters, e.g., the identifier of the service request, are known to the scheduler, but most of the data is passed through the file system. The files are uniquely named using the known identifier, thereby allowing different requests to execute concurrently. The other domain-dependent components of the system are the procedures required to convert data formats from those produced by one application to those expected by the next. These translation routines were written using the tools Lex and Yacc. They are invoked as neces-

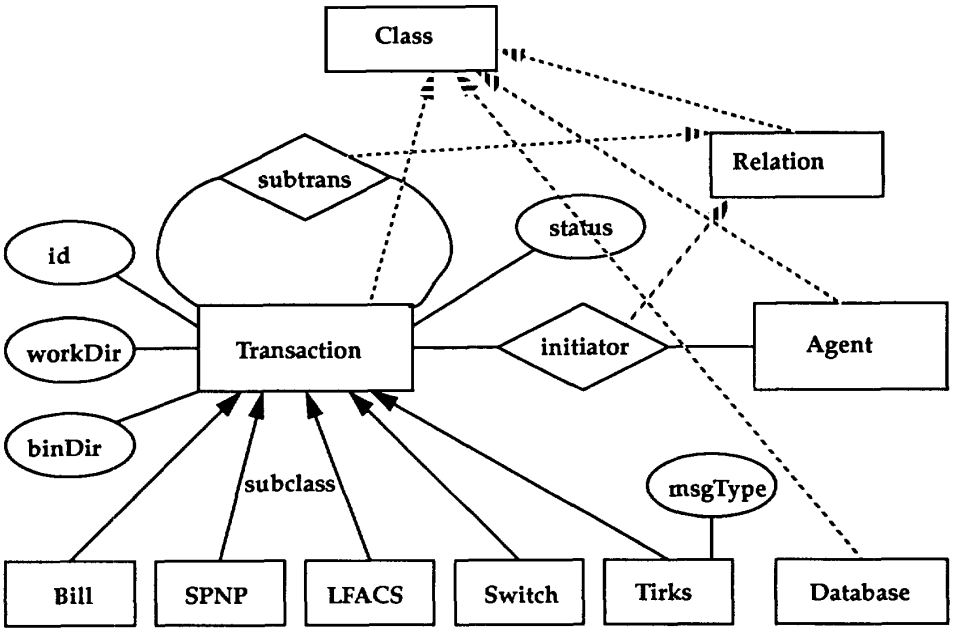

Figure 4: Description of the Transaction-Scheduling Agent's Implementation

sary by the schedule-processing agent. The remaining domain-dependent aspect of the provisioning workflow is in the resource constraints, which guide the scheduling and repairing processes.

5 Conclusions

We have a prototype implementation, which we treat as a proof-of-concept exercise, rather than a deployable system. The prototype is being reimplemented for installation in a restricted production environment (one switching center). If it is successful, it will be deployed in all switching centers by our client.

Certain desired features will call for AI technology in the final implementation: these include schedule repair and other semantical aspects of the domain. Because of business constraints, we do not expect to use our present Lisp-based system for these, although the ideas will be reimplemented in a C++ or Rosette-based constraint processor. Certain other features, notably those to do with schedule processing, do not really require AI approaches, even though AI approaches are useful for rapidly prototyping them.

It is safe to conclude that AI technology helped us sort out various issues and easily build a working system that could be tested. Having an implementation will help us understand the needed components and the interfaces among them. This will aid in the design and testing of industrial-strength modules.

The benefits realized from automatic workflow processing include

- Improved turnaround time.
- Error checking of the initial input; validation of fields with respect to other fields and information in customer databases.
- Streamlining of the present procedures by removing redundant data gathering and processing.
- Ability to modify the structure of the overall procedure easily.

We believe that as information systems become more complex, there will be an increasing demand for AI technologies to manage them. It is likely, however, that AI technologies will have to take a somewhat different, possibly more mundane, form in applications, than might have been initially envisioned by the people who developed them.

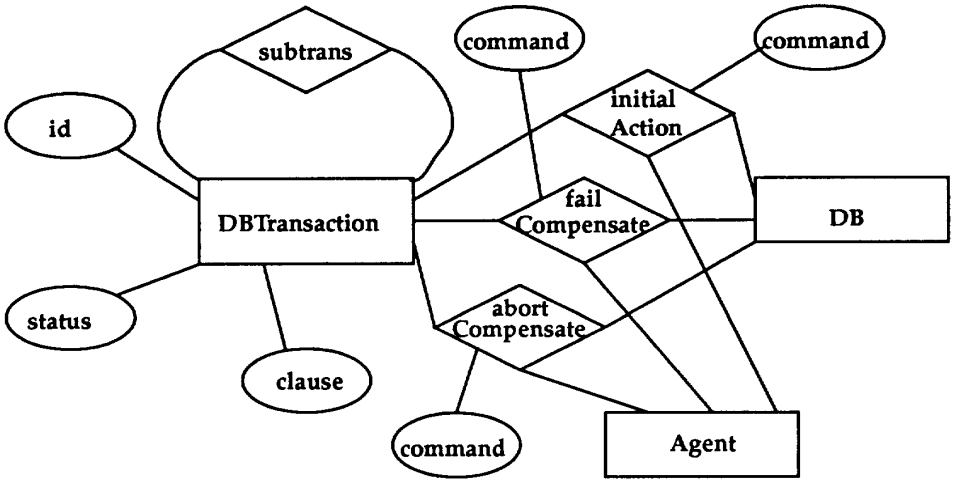

Figure 5: Conceptual Model for Schedule-Repairing Agent

References

[1] Mansoor Ansari, Marek Rusinkiewicz, Linda Ness, and Amit Sheth, "Executing Multidatabase Transactions," *Proceedings 25th Hawaii Int'l. Conf. on Systems Sciences*, January 1992.

[2] Paul C. Attie, Munindar P. Singh, Amait P. Sheth, and Marek Rusinkiewicz, "Specifying and Enforcing Intertask Dependencies," *Proceedings of the 19th VLDB Conference*, 1993.

[3] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Co., 1987.

[4] Omran A. Bukhres, Jiansan Chen, Weimin Du, Ahmed K. Elmagarmid, and Robert Pezzoli, "InterBase: An Execution Environment for Heterogeneous Software Systems," *IEEE Computer*, Vol. 26, No. 8, Aug. 1993, pp. 57–69.

[5] Philip E. Cannata, "The Irresistible Move towards Interoperable Database Systems," *First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.

[6] Ahmed K. Elmagarmid, ed., *Database Transaction Models for Advanced Applications*, Morgan Kaufmann Publishers Inc., San Mateo, CA, 1992.

[7] Hector Garcia-Molina and K. Salem, "Sagas," *Proceedings of ACM SIGMOD Conference on Management of Data*, 1987.

[8] Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Wei-Min Shen, Munindar Singh, and Philip Cannata, "Integrating Enterprise Information Models in Carnot," *International Conference on Intelligent and Cooperative Information Systems (ICICIS)*, Rotterdam, The Netherlands, June 1993.

[9] W. Woody Jin, L. Ness, M. Rusinkiewicz, and A. Sheth, "Executing Service Provisioning Applications as Multidatabase Flexible Transactions," Bellcore Tech. Report (unpublished), 1993.

[10] Christine Tomlinson, Paul Atie, Philip Cannata, Greg Meredith, Amit Sheth, Munindar Singh, and Darrell Woelk, "Workflow Support in Carnot," *IEEE Data Engineering*, 1993.

[11] Darrell Woelk, Paul Attie, Philip Cannata, Greg Meredith, Munindar Singh, and Christine Tomlinson, "Task Scheduling Using Intertask Dependencies in Carnot," *ACM SIGMOD*, 1993.