Synchronization of Multiple Agents in Rule-Based Development Environments
-- Thesis Proposal --

Naser S. Barghouti[1]

Department of Computer Science
Columbia University
New York, NY 10027

Technical Report CUCS-504-89

8 December 1989

Abstract

The Rule-Based Development Environment (RBDE) is a recently-developed approach for providing intelligent assistance to developers working on a large-scale software project. RBDEs model the development process in terms of rules, and then enact this model by automatically firing rules at the appropriate time. The RBDE approach has been used to develop single-user environments, but support for multiple developers cooperating on the same project is still not available because of the lack of mechanisms that can synchronize the efforts of multiple developers, who concurrently select commands, causing the firing of multiple rules (either directly or via chaining) that concurrently access shared data. Conflicts between different rules and concurrent access to shared data may cause the violation of consistency in the project database, and thus necessitate the synchronization of concurrent activities. The conjecture of this proposal is that an RBDE can provide the required synchronization if it is provided with knowledge about what it means for the data of a specific project to be in a consistent state, and about the semantics of operations that developers perform on the data. The research that this paper proposes will formulate a framework for specifying consistency of data in an RBDE, and formulate a mechanism for synchronizing the actions of concurrent rules fired on behalf of multiple developers cooperating on a common or different tasks.

Copyright © 1989 Naser S. Barghouti

[1]Barghouti is supported in part by CTR. Research on Programming Systems is supported by National Science Foundation grants CCR-8858029 and CCR-8802741, by grants from AT&T, Citicorp, IBM, Siemens, Sun and Xerox, by the Center for Advanced Technology, and by the Center for Telecommunications Research.

Table of Contents

List of Figures

1. Introduction

1.1. The Problem

Rule-based development environments (RBDEs) provide intelligent assistance to developers of large-scale projects by modeling the development process of a project in terms of rules, and enacting that process model. Existing RBDEs, however, are either single-user environments or support minimal synchronization mechanisms such as locking the entire database[2] whenever a developer is accessing it. They do not support the needs of multiple developers working concurrently on related subtasks of the same project. These needs include maintaining the project database in a consistent state, but at the same time not obstructing the need for multiple developers to cooperate by sharing data among them. More specifically, existing RBDEs lack appropriate mechanisms for providing them with:

1. A specification of what is meant exactly by maintaining the data of a particular project in a consistent state.
2. A definition of what level of concurrent access to the shared project database should be allowed between different developers? Specifically, should the RBDE tolerate any inconsistency in the database in order to allow multiple developers cooperating on a common task more concurrent access to shared data than developers working on disjoint tasks?
3. A method of synchronizing concurrent actions triggered by multiple developers that would use the specification of both consistency and concurrency in order to maintain whatever consistency is desired in the particular project and still be able to allow cooperation among developers.

A more precise statement of the problem in terms of the assumed RBDE model is made in the next section.

1.2. Importance of Solving This Problem

RBDEs model the development process of a project, such as developing a software system, in terms of a set of rules that, when fired automatically (to enact the process model), perform some of the development and data management tasks that would otherwise be done manually by the developers or not done at all. This idea of integrating rule-based process modeling with database capabilities within the framework of a development environment has gained popularity recently. In fact at the Fifth International Workshop on Software Specification and Design (ISPW-5), about half of the position papers were concerned with rule-based process modeling [Perry 89]. Thus, it seems that pursuing the RBDE approach to support "real" large-scale projects is a worthwhile effort.

These "real" projects typically involve many developers, a complicated process model that may require hundreds of rules to describe it, and a significant amount of highly structured data that is stored in a shared project database. Existing RBDEs, however, do not scale up to all of these requirements. Research in AI has already produced rule-based systems that consist of thousands of rules, and it seems that the problem of supporting a large amount of data is also being solved. The proposed research aims at

[2] It is assumed that the RBDE stores all the data belonging to a project in a database. We do not assume any particular database model, but use the term in the generic sense.

providing the mechanisms necessary for RBDEs to scale up to support multiple developers by providing them with a synchronization mechanism, which aims at maintaining the consistency of data while providing the highest possible level of concurrent access to this data. Such a mechanism would enable RBDEs to support the consistency maintenance needs of multiple developers without obstructing their cooperation, and thus make the RBDE model a more powerful one in terms of providing intelligent assistance in the development process of large-scale projects.

1.3. Approach to a Solution

Synchronization in an RBDE does not involve only the human developers but also rules that automatically perform operations on their behalf, since those rules also access the shared database. The term *agent* is used in this proposal to refer to both human developers and rules that are fired automatically by the RBDE on their behalf. What is lacking, then, in existing RBDEs is the ability to synchronize concurrent accesses by multiple agents to shared data while still providing an environment that supports cooperation among its users (called cooperative environment hereafter).

Synchronization of multiple agents involves: (1) conflict resolution in order to guarantee that rules that directly conflict with each other are not allowed to be fired concurrently on behalf of the same or different developers; and (2) concurrency control in order to ensure that consistency of data is not violated because of concurrent access to the same data items. An RBDE may be able to support the required synchronization if it is provided with knowledge about what it means for the data of a specific project to be in a consistent state, and about the semantics of operations performed by agents on the database. These two pieces of information are different for different projects and thus they have to be provided to the RBDE rather than built into it. Given this knowledge, an RBDE can actively participate in both automating some tasks and maintaining consistency in the sense that it could monitor the activities of multiple developers and be able to automatically perform some operations, which would either retain consistency of the data in case of violation or automatically perform some task, in response to changes made to the data. A passive environment, alternatively, would provide only an interface to the data and to the tools used by the developers.

The research that this paper proposes will use the following approach to formulate a solution to the problem:

1. In order to solve the conflict resolution problem, the tasks performed by both the developers and the RBDE on their behalf have to be encapsulated in logical units, which resemble transactions in database systems. The similarities and differences between the logical units used here and transactions are clarified later.

2. In order to solve the consistency specification problem, there is a need to design a framework in which it is possible to: (1) define exactly what is meant by maintaining a specific project in a consistent state; and (2) specify allowable levels of concurrency between different developers' transactions.

3. Develop a synchronization mechanism that uses this framework to achieve a reasonable compromise between two conflicting goals:
    a. Maximizing concurrency of access by multiple agents to the shared project database, and thus maximizing the number of operations that can proceed concurrently. This has the effect of minimizing the environment's response time to user commands, and thus improving the overall productivity of cooperating developers.

b. Maintaining the project database in a consistent state as specified in the consistency framework, thus reducing the time wasted on fixing up inconsistent data, and thereby facilitating the successful completion of the development task.

1.4. Requirements on the Solution

The synchronization mechanism should allow multiple developers to share knowledge among themselves, to cooperate on a common task, and to be able to control the progress of their tasks without much obstruction[3]. Specifically, the concurrency control requirements in an RBDE model include:

- Supporting both long-lived database operations that last for an arbitrarily long period of time (e.g., editing a source file in a software project), as well as short database operations that are similar to operations in traditional databases.
- Supporting a spectrum of allowable interaction between multiple agents, ranging from synergistic cooperation among them to their isolation from each other, depending on the project specification. For example, if multiple developers are working on the same part of a project, they should be able to look at each other's modifications, even if these modifications are not complete, rather than having to wait until the modifications are committed. Looking at partial modifications might help developers discover inconsistencies early on rather than at the end when they have to integrate their work. At the same time, these cooperating developers should not be allowed to modify the same object at the same time in conflicting ways if that violates the consistency of data required in the particular project.
- Supporting interactive user control over transactions, which means that the users make up the operations of the transaction as they go rather than programming all operations that they want to perform in batch transactions before hand.

The remainder of this proposal is structured as follows. Section 2 describes the rule-based development environment model assumed in the proposed thesis, and presents a statement of the problem in terms of the model. Section 3 presents the approach that will be followed to solve the problem and describes how the success of the solution can be measured. Section 4 presents MARVEL, the development environment kernel that will serve as the experimental framework for the proposed work. Section 5 describes the steps that will be followed to formulate a solution to the problem, and a tentative schedule for achieving these steps. Section 6 outlines the organization of the proposed dissertation.

2. Definition of the Model

This section describes the development model assumed in the proposed research, and more precise statement of the problem. We state what is assumed as a given and what is part of the problem.

2.1. Background: The RBDE Model

Some of the most well-known development environments are rule-based. For example, the Common-Lisp Framework (CLF) [CLF Project 88] supports rule-based process modeling through what it calls consistency and automation rules [Cohen 86]. Refine [Smith et al. 85], an automatic transformation system for the purpose of program synthesis, also provides a limited form of controlled automation in the

[3] It should be noted that we are assuming a cooperative environment in which the developers do not compete for resources with each other.

style of CLF. Darwin [Minsky and Rozenshtein 88] is a rule-based system implemented in Prolog that restricts what programmers can do by treating rules as constraints, and automates checking and enforcement of these constraints. Grapple [Huff and Lesser 88] is a system that uses rules to do planning and plan recognition in order to enact and monitor the process model. Marvel [Kaiser et al. 88a] enacts the process model by providing controlled automation via forward and backward chaining among rules.

We assume a general RBDE model, which includes all of the systems mentioned above, that enacts a model of the development process of a particular project by automatically firing rules that encapsulate operations on the project database. These rules are not built-in but are specified to the RBDE and can be changed dynamically during a session. The commands that a developer can select from in an RBDE either correspond to rules or are provided by the environment as built-in commands, such as adding and deleting objects. Each rule has a *condition* (also called the *left hand side* or *precondition*) that must be satisfied before the second part, which is the development activity the rule encapsulates, is executed on the database. The activity is modeled as a black box whose inputs and outputs are known, but in order to know which output it will produce, the black box has to be invoked. The third part is a set of mutually exclusive *actions* (also called the *right hand side* or *postconditions*), which change values in the database. Which action to assert depends on the results of the rule's activity.

If the actions of a rule change the objects in the database in such a way that the condition of other rules become satisfied, those rules are fired automatically. This behavior is termed *forward chaining* and it is the model that production systems implement. Alternatively, if a condition of a rule is not satisfied, *backward chaining* is performed to satisfy it. The backward chaining model (also called *backtracking*) is implemented in theorem provers, constraint systems, and some production systems. Forward and backward chaining are two mechanisms for enacting the process model.

RBDEs in general provide either a forward chaining model or a backward chaining model or both in order to do one or more of the following:

1. Maintain consistency as defined by the conditions and actions of rules. Thus, no operation will be performed until its condition is satisfied; and if an operation is performed, all implications (in terms of performing other operations) are taken care of.
2. Automate the performance of some operations that would otherwise be performed manually by developers. Thus, by requesting one database operation, several other operations might be performed automatically by the RBDE on behalf of the developer who requested the first operation.
3. Monitor the development process and collect information about any violations of it. The RBDE might only warn developers about these violations rather than enforcing the model.

Some existing RBDEs distinguish between automation rules and consistency rules (CLF), some provide only consistency rules (Darwin), while others provide only automation rules (Marvel). In this proposal, we assume a general model in which: (1) both forward and backward chaining are supported; and (2) rules may serve either the purpose of automation or consistency maintenance.

2.2. A More Specific Statement of the Problem

When multiple developers cooperate on a project within an RBDE, they share a common database that contains all the objects of the project. These developers start concurrent sessions in order to complete their specific tasks. During their sessions, the developers concurrently request operations that access objects in the shared project database. These concurrent operations might violate the consistency of the objects they access if they concurrently change either the same attribute or dependent attributes of the same object in conflicting ways.

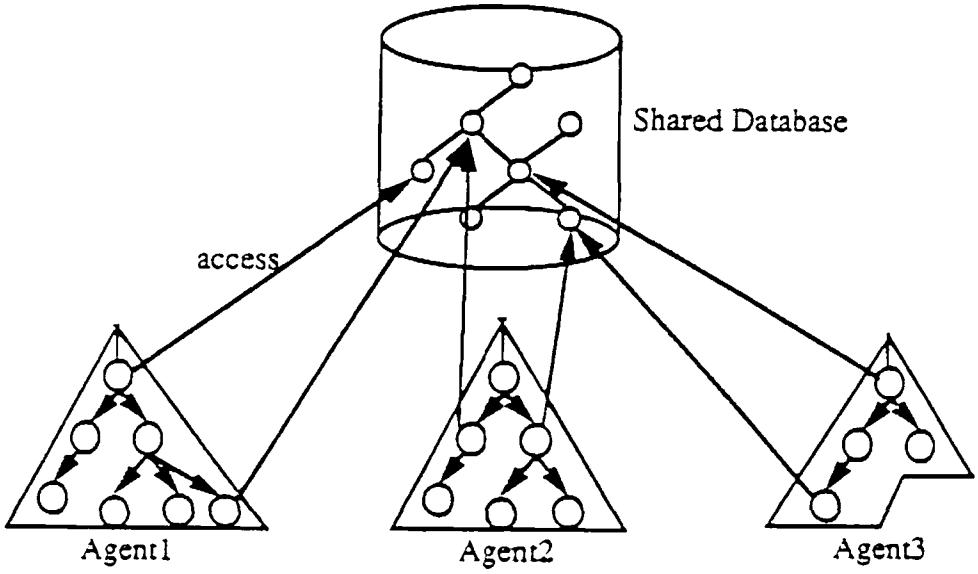

Figure 2-1: The Multi-Agent problem in RBDEs

Since most operations correspond to rules and since chaining might lead to firing other rules that perform more conflicting operations on the database, more inconsistencies might be introduced in the database. More generally, the overall behavior of cooperating developers in an RBDE can be modeled as multiple sets of rules, where multiple rules from each set are fired concurrently to perform operations on the shared project database. This situation is depicted in figure 2-1. Based on this, the synchronization problem that the proposed research aims to solve can be divided into three subproblems:

- The *conflict resolution* problem: deciding whether or not two or more rules can logically be fired within concurrent chaining cycles (i.e., making sure that the conditions and actions of these rules do not negate each other).
- The *consistency specification* problem: specifying what kind of consistency has to be maintained in the database of the particular project, and what kind of concurrency is allowable between developers' tasks?
- The *concurrency control* problem: given that several rules have been fired concurrently after passing the conflict resolution phase, how can we make sure that their access to data is consistent with the specification of the project.

We now precisely formulate each of these subproblems, describe the related work that has been done to provide partial solutions to each subproblem, and present proposed approach to solving it.

3. The Proposed Research

The proposed research addresses three problems associated with synchronizing multiple concurrent agents in an RBDE: (1) the conflict resolution problem, which involves choosing which rules to fire automatically in response to changes in the project database; (2) the knowledge-based consistency problem, which involves formulating a specification framework in which it is possible to provide an RBDE with the data consistency desired for, and the level of inconsistency that can be tolerated by, a specific project.; and (3) the concurrency control problem, which requires finding a mechanism that enables the RBDE to prevent concurrent access to data if that access violates the data consistency defined for the particular project. In this section, we formulate each problem, present related work that addresses it, and describe the proposed approach to solving it.

3.1. Conflict Resolution

3.1.1. Problem Formulation

Assume that two developers Bob and Mary are working on a common task within the context of an RBDE. The objects[4] that Bob and Mary need to access in order to complete their task are stored in a shared database. All the commands that access the database (e.g., read, write, edit, compile, format, etc.) are implemented in terms of rules as described in section 2. Each command that either Bob or Mary selects causes a chain of rules, each of which may access objects in the database.

The problem is that when Bob and Mary request commands concurrently, Bob's command might trigger a chain of rules, one or more of which might conflict with on of the rules on the in-progress chain that Mary's command has triggered. Alternatively, the conflict might have occurred even in a single-user context if, for example, Bob had requested two commands concurrently (e.g., Bob is executing two commands in two different windows concurrently). This problem arises when the RBDE allows multiple rules chains to execute concurrently. The trivial solution of simply serializing chaining cycles is not satisfactory since chaining might invoke long-lived operations.

Let us illustrate the problem by pursuing the example above. Assume that Bob requested a command *c1* at time *t1*, which triggered the forward chaining cycle shown in figure 3-1. Before the chaining resulting from *c1* is finished, Mary requests a command *c2*, which corresponds to the rule *r2*, at time *t2*. The condition of *r2* is not satisfied, which causes a backward chaining cycle in order to satisfy it. The conditions and actions of the rules involved in both chains are shown in figure 3-1. From the definition of the rules, it should be noted that the condition of rule *r4* is negated by the actions of *r5*.

Now assume that the operations involved in both chaining cycles occur in the order depicted in the figure. If the RBDE allowed *r5* to be executed, it would invalidate the backward chaining cycle that Mary's command triggered. The two possible actions to take in this situation are: (1) do not fire *r5* and thus end Bob's forward chaining cycle; or (2) fire *r5* and then invalidate Mary's chaining cycle by

[4] the term *object* is used in the generic sense here to mean a data item. For example, the source code of a procedure can be stored as an object as well as a chapter in the user's manual of a system. In the next subsection we define data objects more formally.

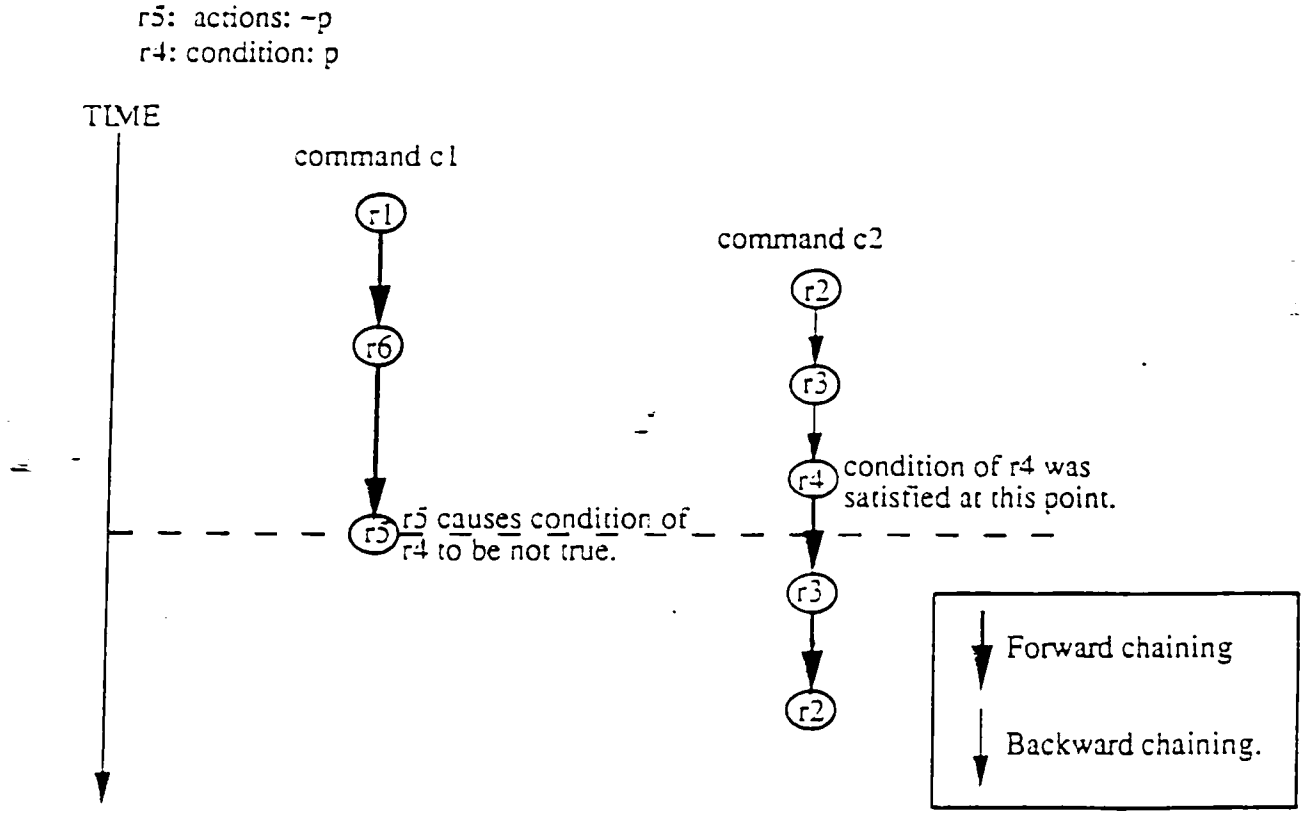

Figure 3-1: Example of conflict resolution problem

backtracking (i.e., from the point when *r4* was fired). The appropriate action to take, it seems, depends on the specification of the automation model of the project.

3.1.2. Related Work

The conflict resolution problem has been addressed in AI rule-based systems, such as OPS5 production systems [Forgy 81], which are typically composed of a set of rules that operate on a shared database of facts called the working memory. Each rule has a condition part, which is a partial description of working memory; if the facts in the working memory satisfy this description, the rule can be fired. The production system interpreter executes a cycle that has three phases: (1) a match phase in which the left hand sides of all productions are matched against the working memory, resulting in a set of instantiations of productions that are ready to be fired; (2) the conflict-resolution phase in which one of the instantiations is chosen for execution; and (3) the act phase in which the actions of the selected production are executed, resulting in changes to working memory. This cycle is repeated until it quiesces. In a SOAR production system [Laird 86], the conflict resolution step is skipped and all the production instantiations selected in the match step are executed. The proposed RBDE model in this paper is closer to the SOAR model in this respect.

The match step is the most time consuming step in the cycle since it involves searching the knowledge base (the working memory and the rules) in order to find information relevant to solving a problem [Gupta 86]. Match algorithms attempt to decrease the number of search operations in a match step by saving the results of executing the previous recognize-act cycle so that only the changes to working memory need to be considered for the following cycle. The Rete algorithm [Forgy 82] is an example of a state-saving match algorithm used in OPS5 production systems. This solution, however, is not appropriate for our model because we assume an environment that is interactive, which means that the developer might at any point suspend his session, consistency preserving, which might require saving checkpoints, and dynamic, which means that rules can be added and deleted dynamically. These characteristics would require either saving the state of the match network, which consumes a very significant amount of space, or reconfiguring the match network, which consumes a considerable amount of time.

Many researchers have attempted to significantly speed up the execution of production systems through the use of parallelism. Both Gupta [Gupta 86] and Miranker [Miranker 86] have concluded in their respective theses that the speed up that can be expected from parallelism is quite limited in the context of OPS5 production systems that are implemented using the Rete algorithm.

In the proposed research, we address a different problem that sounds superficially similar. Multiple developers working to complete a project, concurrently perform their tasks, which are made up of sets of operations, each of which is implemented by a rule, causing multiple rules to be fired concurrently. Since each rule has a precondition that needs to be satisfied, the match steps for these rules might be performed in parallel. Thus, parallelism in this case is a given and not a mechanism to improving performance. However, any proposed solution will require the use of the best algorithm possible for synchronizing parallel rule executions. It is thus of interest to look at the results of parallel production system implementations.

3.1.3. Proposed Approach to the solution

The approach that we propose to solve this problem is to use a concept similar to transactions in database systems. The transaction concept was developed in traditional database research in order to solve the consistency maintenance problem in these databases. There is a lack of knowledge in traditional database systems about the application-specific semantics of database operations, and a need to design general mechanisms that cut across many potential applications. Thus, the best a DBMS can do is to abstract all operations on a database to be either a read operation or a write operation, irrespective of the particular computation. All computations are then programmed as transactions that consist of a sequence of read operations, write operations, and conditional statements. The DBMS can guarantee that the database is always in a consistent state with respect to reads and writes by executing transactions atomically; i.e., either all the operations in a transaction are performed in order or none are.

In addition to being consistency (atomicity) units, transactions are also logical units that group sets of operations that comprise a logical task. It is in this sense that I propose to use transactions to solve the conflict resolution problem by grouping the set of rules that are fired by the RBDE automatically in response to a developer's command either directly (i.e., the rule corresponding to the command) or indirectly (i.e., through chaining) during a session into logical units. These units would be most similar to

interactive transactions, where the code of the transaction is made up as rules are fired and their actions are asserted. In other words, each chaining cycle that is triggered in response to a developer's command is encapsulated in a transaction.

Given that each rule now occurs within a transaction, the match step of the rule should take into consideration not only the condition of the rule, but also how the actions of the rule might affect the truth of conditions of other rules that have already been fired in other in-progress transactions (chaining units).

3.2. Knowledge-Based Consistency

3.2.1. Problem Formulation

Consistency of a database is maintained if each data item in the database satisfies some consistency constraints that depend on the application. The database is said to be in a consistent state if every data item in it satisfies all the consistency constraints. In RBDEs, some inconsistency can be tolerated as a price for expanding the set of allowable interleavings between concurrent transactions, thus increasing concurrency. The level of tolerable inconsistency depends on the operations in a particular application.

For example, if two programmers John and Mary are working on coding related parts of the same subsystem, their respective tasks might last for several days or weeks. It is important to have a notion of a task that needs to be completed (i.e., all the coding has to be finished in order to reach a satisfactory state of the project), but that task certainly does not need to be atomic. If Mary is modifying a module that somehow depends on a module in John's part, she would usually rather be able to look at the partial modifications done to John's module than have to wait until the modifications are completed and made available to others. The reason is that it might be that Mary wanted to look at John's module in order to look up the type of a variable that she knows John has changed recently, but that will most likely not be changed by John in the future. In this case, Mary will not be concerned about John's part not being in a "consistent" state before she can access it.

The problem thus is finding a framework for specifying which interleavings are allowable in a particular environment. The framework should specify the granularity at which different database operations can be interleaved. This specification framework can then be used by a concurrency control algorithm to provide maximum concurrency while maintaining consistency.

3.2.2. Related Work

The idea of defining a framework for specifying the consistency unit in database transactions was proposed by Lynch [Lynch 83], Garcia-Molina [Garcia-Molina 83, Garcia-Molina and Salem 87], and the CAD group at MCC [Bancilhon et al. 85]. Garcia-Molina's semantic atomicity scheme statically divides transactions into atomic steps, and specifies compatibility sets that define the allowable interleavings with respect to those steps. Thus if transactions of type X are compatible with transactions of types Y and Z, then any two transactions T_i, of type Y, and T_j, of type Z, can arbitrarily interleave their steps with a transaction T_k, of type X. There is thus no distinction between interleaving with respect to Y and interleaving with respect to Z.

Lynch observed that it might be more appropriate to have different sets of interleavings (in the form of

specific breakpoints) with respect to different transaction types [Lynch 83]. This observation seems to be valid for RBDEs in which activities tend to be hierarchical in nature, for example, software development environments. Transactions in such systems can often be nested into levels, where at each level, transactions that have something in common, in terms of access to data items, are grouped. Level one groups all the transactions in the system while subsequent levels group transactions that are more strongly related to each other. A strong relation between two transactions might be that they often need to access the same objects at the same time in a non-conflicting way. A set of breakpoints (defining interleavings) is then described for each level of the nesting where the higher order sets (for the higher levels) always includes the lower order sets. This results in a total ordering of all sets of breakpoints. This means that the breakpoints that specify interleavings at a level cannot be more restrictive than those that define interleavings at a higher level.

Let us illustrate this concept by an example from the software development domain. Let us suppose that Bob, John and Mary are cooperatively developing a software project. In their development effort, they need to modify objects (code and documentation) as well as get information about the current status of development (e.g., the latest cross-reference information between procedures in two modules A and B). Let us suppose that Mary starts two transactions (in two different windows, for example) T_{Mary1} and T_{Mary2} to modify a procedure in module A, and get cross-reference information, respectively; Bob starts a transaction T_{Bob1} to update a procedure in module B; John starts two transactions T_{John1} to modify module A, and T_{John2} to get cross-reference information.

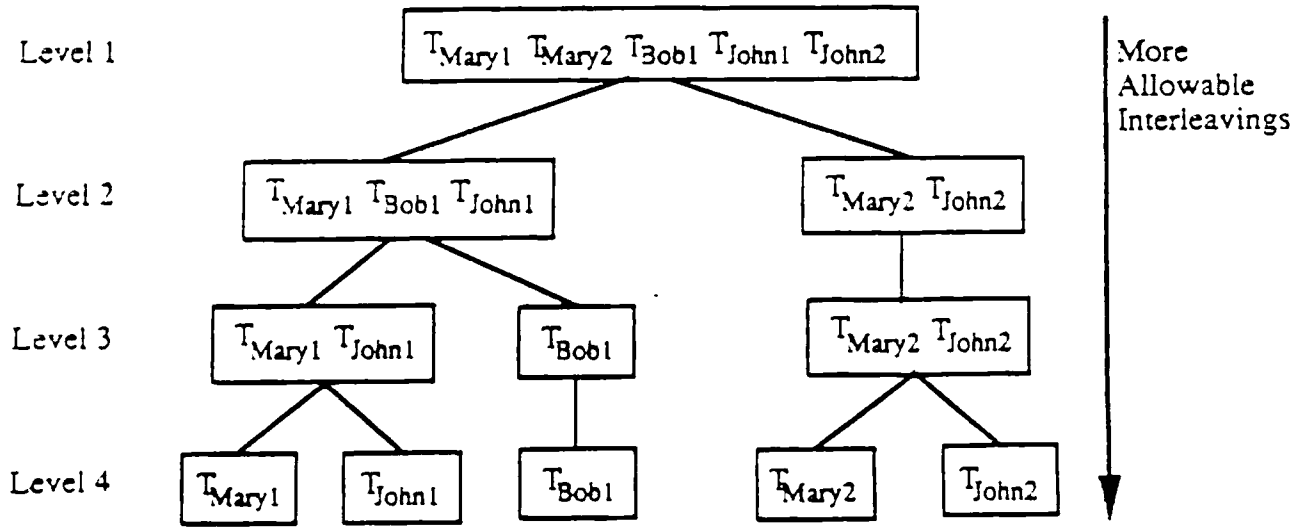

Figure 3-2: Multilevel transaction

A nested transaction system can be set up as shown in figure 3-2. The top level includes all transactions. Level 2 groups all modification transactions (T_{Mary1}, T_{Bob1} and T_{John1}) together and all cross-reference transactions (T_{Mary2} and T_{John2}) together. Level 3 separates the transactions according to which modules they affect; for example, it separates the transactions that modify module A (T_{Mary1} and T_{John1}) from those modifying module B (T_{Bob1}). Level four contains all the singleton transactions.

Then, the sets of breakpoints are specified by describing the transaction segments between the breakpoints. For example, the top level set might specify that no interleaving is allowed; the second-level set might specify that all modification transactions might interleave at some granularity, and that cross-reference transactions might similarly interleave, but that modification and cross-reference transactions cannot interleave (to guarantee that cross-reference information does not change while the transaction is in progress).

Using the sets of breakpoints, the concurrency control mechanism can provide as much concurrency as defined by the multilevel atomicity scheme. Thus, the mechanism in our example might allow transactions T_{Mary1} and T_{John1} to interleave their steps while modifying module A (i.e., allow some level of cooperation so as not to block out module A for a long time by one of them), but it will not allow T_{Mary1} and T_{John2} to interleave their operations.

Both Garcia-Molina's and Lynch's schemes assume that the body of each transaction is known, and thus they can statically define the breakpoints and allowed interleavings. However, as was noted in the statement of the problem, developers operate on the project database interactively. Thus, their long transactions will not be programmed (i.e., it will not be known what operations might be executed in them). Also, the results of database operations are nondeterministic, which necessitates having multiple, possibly mutually exclusive, actions in the rules that implement the operations. Given the interactive nature of developers' sessions and the nondeterminism of database operations, the assumptions made above become inappropriate. Specifically, it would be impossible to classify transactions *apriori* into different types and then specify the interleavings between these types. Thus, we need to develop a framework that builds on the semantics-based consistency work, but that takes into consideration the interactive and open-ended nature of transactions in RBDEs.

3.2.3. Approach to a solution

The approach to solve the problem of defining consistency for a specific project is three-pronged: (1) define the smallest consistency unit in RBDE tasks; (2) define developers' tasks, that may last for several sessions, in terms of transactions; (3) develop a framework for specifying consistency for a particular project and allowable concurrency levels.

Defining the Smallest Consistency Unit: In an RBDE, each rule has a condition that ensures that the database operation it implements happens in the correct context (i.e., in a consistent state), and since the actions explicitly describe all the effects of each operation, each rule, like an atomic transaction, transforms the database from one consistent state to another. Thus, it might be reasonable to define the rule as the smallest consistency unit. This unit might be too small for some projects, and thus the actual project-specific consistency units will have to be specified in terms of a set of rules.

Defining Tasks as Transactions: Each developer's task involve a non-deterministic set of chaining units, and can be modeled as a nested transaction made up of subtransactions which are the operations performed during the session. Each operation is itself a nested transaction that includes the subtransactions representing the operations that are performed due to chaining. Each developer's long transaction starts with a "begin task", ends with a "end task", and is interactive.

Knowledge-Based Consistency: Finally, we need to develop a notion of a project-specific consistency

unit that is more flexible than the traditional serializability-based atomicity unit. The desired consistency can be specified in terms of breakpoints in the long transaction, where the unit of consistency is the operations between two breakpoints. Other transactions can look at the partial results of a long transactions only at the breakpoints. Thus, the breakpoints specify points at which transactions can be interleaved. In other words, they specify which transactions (or more likely, what types of transactions) can be interleaved at specific breakpoints. Thus, different developers will be able to see the state of the the same developers' transactions at different breakpoints. This results in a flexible specification of concurrency levels between different kinds of tasks.

3.3. Concurrency Control

3.3.1. Problem Formulation

As outlined in section 1, one of the problems associated with synchronization of multiple agents is concurrency control, which stems from having a shared project database. The reason for storing objects belonging to a project in a database in the first place is that RBDEs, like other development environment models, utilize database technology to uniformly manage data (e.g., design documents, circuit layouts, source files, *etc.*) manipulated by cooperating developers. Utilizing database technology in development environments is desirable for several reasons including data integration, data integrity, convenience of access, and data independence [Bernstein 87, Dittrich et al. 87, Nestor 86, Rowe and Wensel 89].

Given the nested transaction model developed above to represent developers' long transactions, the problem then becomes how to control the concurrent access to the shared database by operations within the developers' concurrent nested transactions.

3.3.2. Related Work

It might occur to the reader that an appropriate concurrency control mechanism to use is Moss's nested transactions mechanism [Moss 85]. This mechanism, however, like all other concurrency control mechanisms developed for traditional databases would require that the developers' long transactions be atomic and serializable. Serializable executions of transactions with respect to reads and updates on the database are enforced in conventional concurrency control mechanisms because of: (1) the lack of semantic information about the database operations, and (2) the desire to design general-purpose mechanisms that do not depend on application-specific information [Bernstein and Goodman 81]. But there is nothing that makes a non-serializable schedule inherently inconsistent. And although serializability is acceptable in traditional applications, which involve transactions that last for a short period of time and that are programmed (i.e., the code that the transaction executes is known before the transaction starts), it is too restrictive for advanced applications, such as multi-agent RBDEs.

Even in the simplified case of programmed long transactions, the performance of such an algorithm would not be acceptable because the operations of the transactions might take an arbitrarily long time to complete. Some of these long transactions need to access the partial results of concurrent transactions in order to achieve cooperation among designs. Thus, isolating long transactions not only constrains concurrency, but it also prevents desired cooperation among developers and prevents transformation of the project database to a new consistent state if that transformation requires concerted actions by multiple developers.

The concurrency control requirements for development environments include supporting long transactions, cooperative transactions, user control, and complex objects [Bancilhon et al. 85, Barghouti 89, Yeh et al. 87]. Researchers have recently addressed these requirements and have designed several unconventional concurrency control mechanisms. The problem with these mechanisms is that most of them only address one requirement, and that none of them completely define the semantics of database operations and the user tasks in which these operations occur. It is important, however, to understand these mechanisms because a more satisfactory solution to the concurrency control problem will definitely build on many of the concepts and ideas they explore.

Semantics-Based Consistency: One of the key ideas that the proposed research will rely on is that of utilizing application-specific semantics of database operations and user tasks in order to increase concurrency. This idea has been explored in two main approaches that have been pursued to support long transactions (LTs): (1) extending serializability-based mechanisms while still maintaining serializable schedules; and (2) relaxing serializability of schedules containing LTs. The first approach makes use of any additional information that can be extracted about transactions, and uses that information with one of the traditional techniques, while maintaining the same traditional scheme in case the additional information is not available. Mechanisms such as altruistic locking [Salem et al. 87], commutativity-based concurrency control [Weihl 88], and snapshot validation [Pradel et al. 86] are examples of the extension approach. The problem with this approach is that it assumes the availability of specific semantic information to extract, such as information about access patterns of transactions, which might not be appropriate for all applications. Another problem is that they assume programmed transactions and do static analysis using the semantic information. The solution we desire cannot assume either of these two things.

In addition to long transactions, it is important to address the issue of cooperation among concurrent agents in the proposed research because even in a relatively small project in which developers work independently most of the time on the parts of the project they are responsible for, they need to interact at various points to integrate their work. In a larger project, which involves a large group of developers divided into several groups, each responsible for a part of the development task, members of each group usually cooperate to complete the part they are responsible for. In this case, there is a need to support cooperation among members of the same group, as well as coordination between multiple groups. Thus, isolating long transactions that encapsulate individual developers' tasks not only constrains concurrency, but it also prevents desired cooperation among developers and prevents transformation of the project database to a new consistent state if that transformation requires concerted actions by multiple developers. There are several mechanisms and concepts that were developed to address the issue of cooperation among concurrent transactions. We present these concepts and mechanisms and explain why they do or do not help us in formulating our approach to a complete solution.

Dynamic Restructuring: One concept that has been proposed is that of dynamically restructuring transactions as more information about the usage of resources becomes available. The concept was proposed by Pu *et. al.* [Pu et al. 88] and used in a mechanism called commit-serializability, which, in addition to supporting long transactions, supports user control over transactions by allowing users to split and join their transactions dynamically in order to restructure the in-progress transactions as new information becomes available.

Coordinated Access: Another approach that has been used to coordinate access to shared data (in the form of individual files) is the reserve/deposit mechanism [Tichy 85], which has been extended to work on collections of files by providing a two-level database hierarchy consisting of a public shared database and private databases for each developer [Lorie and Plouffe 83, Katz and Weiss 84, Kaiser and Feiler 87]. Other mechanisms provide a multilevel database hierarchy capable of providing more concurrency [Kaiser and Perry 87, Honda 88].

User Groups: The concept of coordinated access, however, only addresses the issue of accessing the database from the viewpoint of data in the sense that it does not associate collections of data with specific developers. In order to do that two new facilities were introduced to be used by concurrency control mechanisms. These are the concepts of groups [Abbadi and Toueg 88, Skarra and Zdonik 89], and notification [Yeh et al. 87, Leblang and Chase, Jr. 87]. Several mechanisms use these concepts to implement concurrency control policies that support synergistic cooperation among multiple developers [Klahold et al. 85, Bancilhon et al. 85, Kaiser 90].

The third requirement that has also been recently addressed by some researchers is that of complex objects. In advanced database applications, data is often defined in multiple levels of granularity. For example, a data object that represents a program in a software project might consist of modules, each of which contains procedures and documentation. If a user wants to gain exclusive access to the whole program (perhaps to build the executable of the program), he has to make sure that every subobject is made unavailable to other users. In this case, it is convenient to be able to lock the entire nested object in one operation rather than requiring a separate operation for each subobject. This, however, should not prevent the user from locking a subobject exclusively without having to lock the whole complex object. There is thus a need for a concurrency control mechanism that operates on complex objects.

To support complex objects, the hierarchical data model of advanced applications has to be incorporated into the concurrency control mechanism. The Orion object-oriented database system incorporates such a multiple granularity data model into serializability-based 2PL protocols [Kim et al. 88, Garza and Kim 88]. Another approach is to model hierarchical access to data as a nested object system [Martin 87]. Each object in the system exists at a particular level of data abstraction. Operations are specified for objects at all levels where operations at level i are specified in terms of operations at level i-1.

The mechanisms presented above follow the trend of using more semantics that is now available in order to improve concurrency. The semantics used includes information about interleavings between transactions, about the division of tasks among teams of developers, about the structure of complex objects, and the access patterns of transactions. In a multi-agent RBDEs, all these kinds of information are available and can thus be used to integrate the mechanisms for supporting LTs with those supporting synergistic cooperation and complex objects. What is missing in the above mechanisms is the ability to define the exact semantics of database operations; i.e., what each operation changes in the database and what are the implications of those changes. If this information were available in the form of the knowledge-based consistency framework as proposed in the previous subsection, it can be used to provide more flexible concurrency control mechanisms.

3.3.3. Approach to a Solution

The approach to solving this problem builds on the solution that was suggested by the CAD group at MCC [Bancilhon et al. 85], which allows the operations of developers' long transactions to be interleaved. The mechanism that the CAD group proposed is to treat all the operations that are performed within the long transactions of developers in the same team as if they were requested by the same transaction. Thus, the mechanism would only insist that the leaf operations (not the nested transactions) be serializable. This mechanism cannot be used "as is", however, in a multi-agent RBDE because of interactive and open-ended nature of developers' sessions and operations.

An appropriate mechanism would have to take into account the consistency framework as described above, as well the notion of teams (or groups) of developers working on the same task. The idea would be that the mechanism would provide more concurrency for members of the same team at the expense of sacrificing some consistency as long as the sources of the inconsistencies are known to the system. This would enable developers to query the system about remaining inconsistencies and remove them.

One remaining problem is the integration of the conflict resolution mechanism and the concurrency control mechanism. One approach that shows some promise at this point is the use of an attribute grammar to represent the rule network, where the conditions and actions of the rules are coded as attributes to the nodes representing the rules in the grammar. Concurrency control can then be integrated with the change propagation algorithm that controls the re-evaluation of attributes (i.e., preconditions and postconditions) in the attributed graph. A big advantage of pursuing this approach is the existence of multi-agent change propagation algorithms for attribute grammars (Josephine Micallef's thesis). Another advantage is that the algorithm (or at least simplified version of it) has already been implemented in the Mercury system here at Columbia, and it might be possible to use the ideas in Mercury.

3.4. Contributions of the Proposed Research

The main contributions of the proposed research will be: (1) to provide a framework for specifying consistency in a project database underlying an RBDE; and (2) to provide an efficient synchronization mechanism for multi-agent RBDEs that uses the framework to allow the highest level of concurrency while maintaining the specified consistency. We expect to be able to demonstrate the feasibility of such a concurrency control framework by implementing it in an existing RBDE called MARVEL, which is described in the next section. Additional contributions include the formalization of the concurrency control problem in the context of RBDEs. This formalization will be useful for object-oriented databases, all of which lack a flexible concurrency control mechanism. The deliverables that are to be completed in the proposed work are:

1. A framework for specifying consistency in a multi-agent RBDE in terms of interleavings between transactions that encapsulate developers' tasks.
2. A corresponding concurrency control algorithm that supports cooperative, long and interactive transactions.
3. A synchronization mechanism that integrates the concurrency control mechanism with a conflict resolution mechanism.
4. A set of analytical metrics to measure the relative productivity of multiple agents.

5. Implementing the synchronization mechanism in MARVEL.

3.5. Measuring the Success of the Solution

The purpose of developing a flexible synchronization mechanism is to be able to maximize concurrent access to data so that concurrent developers will not need to be delayed (by being locked out) longer than necessary before accessing data. The purpose is to improve the productivity of developers working concurrently on the same project. Two problems that should be addressed in the thesis are: (1) how to model the relative productivity of developers, and (2) how to measure this relative productivity.

In order to solve these problems, we need to develop metrics for analyzing the relative productivity. At this time, the approach of doing this is not clear.

4. MARVEL

A software development environment kernel, MARVEL, that combines ideas from rule-based systems and object-oriented databases has been implemented here at Columbia. Currently, MARVEL is a single user system. A multi-agent implementation of MARVEL using the algorithm that is proposed as part of this thesis would provide a proof-of-concept implementation of the solution. In the rest of this section, MARVEL is described, and some ideas about how to provide it with a multi-agent capability are outlined.

MARVEL is a kernel for software development environments based on rule-based process modeling and controlled automation [Kaiser et al. 88b, Kaiser et al. 88a]. The kernel is tailored by specifications written by designated *MARVEL administrators* in an object-oriented language called MARVEL Strategy Language (MSL). MSL specifications are divided into modules called *strategies* that describe the organization of data in the particular project, as well as the process of development of the project. Organization is described in terms of object classes, where each object in the database is an instance of one of the specified classes. Multiple inheritance and complex objects are supported in MARVEL. The process model is described in terms of rules.

MARVEL administrators typically develop a library of strategies that are appropriate for the particular project, which are then *loaded* on request by into the MARVEL kernel by end-users (i.e., developers working on a project), who need not be directly involved in writing strategies. In fact there is no need for developers to be aware that the environment is driven by rules.

The definition of classes in the strategies that are loaded by MARVEL describe the organization of the database in which MARVEL stores all the objects that are created in the lifetime of the project. Each object is an instance of a particular class which defines the attributes associated with the object. The values of the object's attributes are manipulated by the rules defined in the strategies. MARVEL's rule model is similar to the general model described in section 2. It supports both forward and backward chaining. MARVEL's rules, however, are automation rules and not consistency maintenance rules.

The most significant limitation of MARVEL's current implementation is its lack of support for multiple users working on the same project. Whenever a user starts a MARVEL session on a project database, the whole database is locked so that other users will not be able to invoke MARVEL on it. Within the same MARVEL session, multiple activities cannot be performed concurrently even on behalf of the same user.

In order to use MARVEL as a framework for implementing the research in the proposed thesis, we need to transform MARVEL into a multi-agent system by implementing the proposed synchronization mechanism.

5. Plan of Action

This section describes the steps that will completed in pursuit of the deliverables described in section 3. The goal of achieving the contributions can be decomposed into three subgoals: (1) completing the design of a generalized mechanism for synchronizing multiple agents in an RBDE; (2) implementing the mechanism in MARVEL: and (3) developing metrics for measuring difference in productivity as a result of applying the synchronization mechanism.

To achieve the first subgoal, the following steps need to be taken:

1. Design and implement a parallel match network for rule execution. As mentioned in section 3, this might be based on attribute grammars.
2. Design and implement a framework to specify interleavings between sets of rules, taking into account forward and backward chaining.
3. Design a concurrency control mechanism along the lines described in section 3
4. Integrate the concurrency control mechanism with the match network.

To achieve the second subgoal, the following needs to be done:

1. Analyze the differences between MARVEL and other RBDEs as far as concurrency control requirements are concerned.
2. Taking these differences into account, design a MARVEL-specific version of the interleavings specification framework for RBDEs.
3. Implement the concurrency control algorithm in MARVEL.
4. If time and other logistics permit, apply the algorithm to the CLF environment.

6. Proposed Outline of the Thesis

The plan of action provided in the previous section should give an insight into the organization of the dissertation. The tentative organization is as follows:

1. Introduction: stating the problem and context.
2. Related Work
   a. Concurrency Control in Advanced Database Applications: much of it would be from my area paper [Barghouti 89].
   b. Parallel Rule-Based Systems: basically Gupta's thesis [Gupta 86], Miranker's thesis [Miranker 86], van Biema's thesis [van Biema ar], and backward chaining rule-based systems, as well as a quick look at expert databases and concurrent Prolog.
   c. Distributed Artificial Intelligence.
3. Conflict Resolution
   a. Transaction Model
   b. Categorization of Conflicts
4. Knowledge-Based Consistency
   a. Semantics-Based Atomicity
   b. Framework for Specifying Interleavings Among Transactions
   c. Specifying Consistency in Interactive Transactions

5. Concurrency Control in RBDEs
    a. General Concurrency Control Requirements
    b. Long Transactions
    c. Cooperative Transactions
    d. Interactive Transactions
    e. Complex Objects
6. MARVEL
    a. Single-User Model
    b. Objectbase and Object Management System
    c. Controlled Automation
    d. Multi-agent MARVEL
7. Conclusions and Future Work

References

[Abbadi and Toueg 88]
Abbadi, A., and Toueg, S.
The Group paradigm for Concurrency Control Protocols.
In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 126-134. ACM Press, Chicago, IL, June, 1988.

[Bancilhon et al. 85]
Bancilhon, F., Kim, W., and Korth, H.
A Model of CAD Transactions.
In *Proceedings of the 11th International Conference on Very Large Data Bases*, pages 25-33. Morgan Kaufmann, Stockholm, August, 1985.

[Barghouti 89] Barghouti, N. S.
Concurrency Control in Advanced Database Applications.
Technical Report CUCS-425-89, Columbia University Department of Computer Science, New York, NY, March, 1989.

[Bernstein 87] Bernstein, P.
Database System Support for Software Engineering -- An Extended Abstract.
In *Proceedings of the 9th International Conference on Software Engineering*, pages 166-178. Monterey, CA, March, 1987.

[Bernstein and Goodman 81]
Bernstein, P., and Goodman, N.
Concurrency Control in Distributed Database Systems.
ACM Computing Surveys 13(2):185-221, June, 1981.

[CLF Project 88] *CLF Manual*
University of Southern California, Information Sciences Institute, Marina Del Rey, CA, 1988.

[Cohen 86] Cohen, D.
Automatic Compilation of Logical Specifications into Efficient Programs.
In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 20-25. AAAI, Philadelphia, PA, August, 1986.

[Dittrich et al. 87] Dittrich, K., Gotthard, W., and Lockemann, P.
DAMOKLES -- The Database System for the UNIBASE Software Engineering Environment.
IEEE Bulletin on Database Engineering 10(1):37-47, March, 1987.

[Forgy 81] Forgy, C. L.
OPS5 User's Manual.
Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.

[Forgy 82] Forgy, C. L.
Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.
Artificial Intelligence 19, September, 1982.

[Garcia-Molina 83]
Hector Garcia-Molina.
Using Semantic Knowledge for Transaction Processing in a Distributed Database.
ACM Transactions on Database Systems 8(2):186-213, June, 1983.

[Garcia-Molina and Salem 87]
Garcia-Molina, H., and Salem, K.
SAGAS.
In Dayal, U., and Traiger, I. (editor), *Proceedings of the ACM SIGMOD 1987 Annual Conference*, pages 249-259. ACM Press, San Francisco, CA, May, 1987.
Special issue of *SIGMOD Record*, 16(3), December 1987.

[Garza and Kim 88]
Garza, J., and Kim, W.
Transaction Management in an Object-Oriented Database System.
In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 37–45. ACM Press, Chicago, IL, June, 1988.

[Gupta 86] Gupta, A.
Parallelism in Production Systems.
PhD thesis, Carnegie Mellon University, Department of Computer Science, March, 1986.
Technical Report CMU-CS-86-122.

[Honda 88] Honda, M.
Support for Parallel Development in the Sun Network Software Environment.
In *Proceedings of the 2nd International Workshop on Computer-Aided Software Engineering*, pages 5-5 - 5-7. 1988.

[Huff and Lesser 88]
Huff, K. E., and Lesser, V. R.
A Plan-based Intelligent Assistant that Supports the Software Development Process.
In Henderson, P. (editor), *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 97-106. ACM Press, Boston, MA, November, 1988.
Special issue of *SIGPLAN Notices*, 24(2), February 1989 and of *Software Engineering Notes*, 13(5), November 1988.

[Kaiser 90] Kaiser, G. E.
A Flexible Transaction Model for Software Engineering.
In *Proceedings of the 6th International Conference on Data Engineering*. IEEE Computer Society Press, Los Angeles, CA, February, 1990.
To appear.

[Kaiser and Feiler 87]
Kaiser, G. E., and Feiler, P. H.
Intelligent Assistance without Artificial Intelligence.
In *Proceedings of the 32nd IEEE Computer Society International Conference*, pages 236-241. IEEE Computer Society Press, San Francisco, CA, February, 1987.

[Kaiser and Perry 87]
Kaiser, G. E., and Perry, D. E.
Workspaces and Experimental Databases: Automated Support for Software Maintenance and Evolution.
In *Proceedings of the Conference on Software Maintenance*, pages 108-114. Austin, TX, September, 1987.

[Kaiser et al. 88a] Kaiser, G. E., Feiler, P. H., and Popovich, S. S.
Intelligent Assistance for Software Development and Maintenance.
IEEE Software 5(3):40-49, May, 1988.

[Kaiser et al. 88b] Kaiser, G. E., Barghouti, N. S., Feiler, P. H., and Schwanke, R. W.
Database Support for Knowledge-Based Engineering Environments.
IEEE Expert 3(2):18-32, Summer, 1988.

[Katz and Weiss 84]
Katz, R., and Weiss, S.
Design Transaction Management.
In *Proceedings of the ACM IEEE 21st Design Automation Conference*, pages 692-693. IEEE Computer Society Press, Albuquerque NM, June, 1984.

[Kim et al. 88] Kim, W., Ballou, N., Chou, H., and Garza, J.
Integrating an Object-Oriented Programming System with a Database System.
In *Proceedings of the 3rd International Conference on Object Oriented Programming Systems, Languages and Applications*, pages 142-152. San Diego CA, September, 1988.

[Klahold et al. 85] Klahold, P.,Schlageter, G., Unland, R., and Wilkes, W.
A Transaction Model Supporting Complex Applications in Integrated Information Systems.
In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 388–401. ACM Press, Austin, TX, May, 1985.

[Laird 86] Laird, J. E.
Soar User's Manual
Xerox PARC, 1986.
Fourth Edition.

[Leblang and Chase, Jr. 87]
Leblang, D. B., and Chase, R. P., Jr.
Parallel Software Configuration Management in a Network Environment.
IEEE Software 4(6):28-35, November, 1987.

[Lorie and Plouffe 83]
Lorie, R., and Plouffe, W.
Complex Objects and Their Use in Design Transactions.
In *Proceedings of the Annual Meeting of Database Week: Engineering Design Applications*, pages 115-121. IEEE Computer Society Press, San Jose CA, May, 1983.

[Lynch 83] Lynch, N. A.
Multilevel Atomicity — A New Correctness Criterion for Database Concurrency Control.
ACM Transactions on Database Systems 8(4):484-502, December, 1983.

[Martin 87] Martin, B.
Modeling Concurrent Activities with Nested Objects.
In *Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 432–439. IEEE Computer Society Press, West Berlin, September, 1987.

[Minsky and Rozenshtein 88]
Minsky, N. H., and Rozenshtein, D.
A Software Development Environment for Law-Governed Systems.
In Henderson, P. (editor), *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 65-75. ACM Press, Boston MA, November, 1988.
Special issue of *SIGPLAN Notices*, 24(2), February 1989 and of *Software Engineering Notes*, 13(5), November 1988.

[Miranker 86] Miranker, D. P.
TREAT: A New and Efficient Match Algrithm for AI Production Systems.
PhD thesis, Columbia University Department of Computer Science, June, 1986.

[Moss 85] Moss, J. E. B.
Nested Transactions: An Approach to Reliable Distributed Computing.
MIT Press, Cambridge, MA, 1985.

[Nestor 86] Nestor, J. R.
Toward a Persistent Object Base.
In Conradi, R., Didriksen, T. M., and Wanvik, D. H. (editors), *Lecture Notes in Computer Science*. Volume 244: *Advanced Programming Environments*, pages 372-394. Springer-Verlag, Berlin, 1986.

[Perry 89] Perry, D. (editor).
5th International Software Process Workshop.
ACM Press, Kennebunkport, Me, 1989.
To appear.

[Pradel et al. 86] Pradel, U., Schlageter, G., and Unland, R.
Redesign of Optimistic Methods: Improving Performance and Availability.
In *Proceedings of the 2nd International Conference on Data Engineering*, pages 466-473. IEEE Computer Society Press, Los Angeles, February, 1986.

[Pu et al. 88] Pu, C., Kaiser, G., and Hutchinson, N.
Split Transactions for Open-Ended Activities.
In *Proceedings of the 14th International Conference on Very Large Databases*, pages 26-37. Morgan Kaufmann, Los Angeles CA, August, 1988.

[Rowe and Wensel 89] Rowe, L. A. and Wensel, S. (editors).
1989 ACM SIGMOD Workshop on Software CAD Databases.
ACM Press, Napa, CA, 1989.

[Salem et al. 87] Salem, K., Garcia-Molina, H., and Alonso, R.
Altruistic Locking: A Strategy for Coping with Long Lived Transactions.
In *Proceedings of the 2nd International Workshop on High Performance Transaction Systems* , pages 19.1 - 19.24. Pacific Grove CA, September, 1987.

[Skarra and Zdonik 89] Skarra, A. H., and Zdonik, S. B.
Concurrency Control and Object-Oriented Databases.
Object-Oriented Concepts, Databases, and Applications.
ACM Press, New York, NY, 1989, pages 395-421.

[Smith et al. 85] Smith D. R., Kotik, G. B., and Westfold, S. J.
Research on Knowledge-Based Software Environments at Kestrel Institute.
IEEE Transactions on Software Engineering SE-11(11):1278-1295, November, 1985.

[Tichy 85] Tichy, W. F.,
RCS — A System for Version Control.
Software — Practice and Experience 15(7):637-654, July, 1985.

[van Biema ar] van Biema, M.
The Constraint-Based Paradigm.
PhD thesis, Columbia University Department of Computer Science, To appear.

[Weihl 88] Weihl, W.
Commutativity-Based Concurrency Control for Abstract Data Types (Preliminary Report).
In Shriver, B. (editor), *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, pages 205-214. IEEE Computer Society Press, Kona, HI, January, 1988.

[Yeh et al. 87] Yeh, S., Ellis, C., Ege, A., and Korth, H.
Performance Analysis of Two Concurrency Control Schemas for Design Environments.
Technical Report STP-036-87, MCC, Austin, TX, June, 1987 .

[Kaiser et al. 88b] Kaiser, G. E., Barghouti, N. S., Feiler, P. H., and Schwanke, R. W.
Database Support for Knowledge-Based Engineering Environments.
IEEE Expert 3(2):18-32, Summer, 1988.

[Katz and Weiss 84]
Katz, R., and Weiss, S.
Design Transaction Management.
In *Proceedings of the ACM IEEE 21st Design Automation Conference*, pages 692-693. IEEE Computer Society Press, Albuquerque NM, June, 1984.

[Kim et al. 88] Kim, W., Ballou, N., Chou, H., and Garza, J.
Integrating an Object-Oriented Programming System with a Database System.
In *Proceedings of the 3rd International Conference on Object Oriented Programming Systems, Languages and Applications*, pages 142-152. San Diego CA, September, 1988.

[Klahold et al. 85] Klahold, P.,Schlageter, G., Unland, R., and Wilkes, W.
A Transaction Model Supporting Complex Applications in Integrated Information Systems.
In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 388-401. ACM Press, Austin, TX, May, 1985.

[Laird 86] Laird, J. E.
Soar User's Manual
Xerox PARC, 1986.
Fourth Edition.

[Leblang and Chase, Jr. 87]
Leblang, D. B., and Chase, R. P., Jr.
Parallel Software Configuration Management in a Network Environment.
IEEE Software 4(6):28-35, November, 1987.

[Lorie and Plouffe 83]
Lorie, R., and Plouffe, W.
Complex Objects and Their Use in Design Transactions.
In *Proceedings of the Annual Meeting of Database Week: Engineering Design Applications*, pages 115-121. IEEE Computer Society Press, San Jose CA, May, 1983.

[Lynch 83] Lynch, N. A.
Multilevel Atomicity — A New Correctness Criterion for Database Concurrency Control.
ACM Transactions on Database Systems 8(4):484-502, December, 1983.

[Martin 87] Martin, B.
Modeling Concurrent Activities with Nested Objects.
In *Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 432-439. IEEE Computer Society Press, West Berlin, September, 1987.

[Minsky and Rozenshtein 88]
Minsky, N. H., and Rozenshtein, D.
A Software Development Environment for Law-Governed Systems.
In Henderson, P. (editor), *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 65-75. ACM Press, Boston MA, November, 1988.
Special issue of *SIGPLAN Notices*, 24(2), February 1989 and of *Software Engineering Notes*, 13(5), November 1988.

[Miranker 86] Miranker, D. P.
TREAT: A New and Efficient Match Algrithm for AI Production Systems.
PhD thesis, Columbia University Department of Computer Science, June, 1986.

[Moss 85] Moss, J. E. B.
Nested Transactions: An Approach to Reliable Distributed Computing.
MIT Press, Cambridge, MA, 1985.

[Nestor 86] Nestor, J. R.
Toward a Persistent Object Base.
In Conradi, R., Didriksen, T. M., and Wanvik, D. H. (editors), *Lecture Notes in Computer Science*. Volume 244: *Advanced Programming Environments*, pages 372-394. Springer-Verlag, Berlin, 1986.

[Perry 89] Perry, D. (editor).
5th International Software Process Workshop.
ACM Press, Kennebunkport, Me, 1989.
To appear.

[Pradel et al. 86] Pradel, U., Schlageter, G., and Unland, R.
Redesign of Optimistic Methods: Improving Performance and Availability.
In *Proceedings of the 2nd International Conference on Data Engineering*, pages 466-473. IEEE Computer Society Press, Los Angeles, February, 1986.

[Pu et al. 88] Pu, C., Kaiser, G., and Hutchinson, N.
Split Transactions for Open-Ended Activities.
In *Proceedings of the 14th International Conference on Very Large Databases*, pages 26-37. Morgan Kaufmann, Los Angeles CA, August, 1988.

[Rowe and Wensel 89] Rowe, L. A. and Wensel, S. (editors).
1989 ACM SIGMOD Workshop on Software CAD Databases.
ACM Press, Napa, CA, 1989.

[Salem et al. 87] Salem, K., Garcia-Molina, H., and Alonso, R.
Altruistic Locking: A Strategy for Coping with Long Lived Transactions.
In *Proceedings of the 2nd International Workshop on High Performance Transaction Systems*, pages 19.1 - 19.24. Pacific Grove CA, September, 1987.

[Skarra and Zdonik 89] Skarra, A. H., and Zdonik, S. B.
Concurrency Control and Object-Oriented Databases.
Object-Oriented Concepts, Databases, and Applications.
ACM Press, New York, NY, 1989, pages 395-421.

[Smith et al. 85] Smith D. R., Kotik, G. B., and Westfold, S. J.
Research on Knowledge-Based Software Environments at Kestrel Institute.
IEEE Transactions on Software Engineering SE-11(11):1278-1295, November, 1985.

[Tichy 85] Tichy, W. F.,
RCS — A System for Version Control.
Software — Practice and Experience 15(7):637-654, July, 1985.

[van Biema ar] van Biema, M.
The Constraint-Based Paradigm.
PhD thesis, Columbia University Department of Computer Science, To appear.

[Weihl 88] Weihl, W.
Commutativity-Based Concurrency Control for Abstract Data Types (Preliminary Report).
In Shriver, B. (editor), *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, pages 205-214. IEEE Computer Society Press, Kona, HI, January, 1988.

[Yeh et al. 87] Yeh, S., Ellis, C., Ege, A., and Korth, H.
Performance Analysis of Two Concurrency Control Schemas for Design Environments.
Technical Report STP-036-87, MCC, Austin, TX, June, 1987 .