LONG DURATION TRANSACTIONS IN SOFTWARE DESIGN PROJECTS*

Henry F. Korth and Gregory D. Speegle

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-89-16 June 1989

* This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 4355.

Long Duration Transactions in Software Design Projects*

Henry F. Korth
Gregory D. Speegle

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

Abstract

Computer-assisted design applications pose many problems for database systems. Standard notions of correctness are insufficient, but some notion of correctness is still required. Formal transaction models have been proposed for such applications. However, the practicality of such models has not been clearly established. In this paper, we consider an example of a software development application and apply the formal model of [KS88] to show how this example could be represented as a set of database transactions. We show that although the standard notion of correctness (serializability) is too strict, the notion of correctness in the model of [KS88] allows sufficient concurrency with acceptable overhead. We extrapolate from this example to draw some conclusions regarding the potential usefulness of a formal approach to the management of long-duration design transactions.

1 Introduction

Database systems were developed originally to manage large volumes of relatively uniform data. Typical applications included banking, payroll, and reservations. Transactions against these databases were generally short, simple transactions. Although these transactions may have been *entered* interactively, people did not interact with *running* transactions. Thus, transactions typically would run for a fraction of a second.

More recently, database concepts have been applied to such areas as computer-aided design, office information systems, knowledge bases and computer-aided software engineering. These new applications require long-duration, interactive transactions, and cooperation among users. The transaction model developed for early database applications falls short of meeting the requirements of design transactions, as has been noted by numerous authors including [BKK85, BBG86, KLMP84].

Current methods for dealing with these new applications fall into two basic categories, ad-hoc systems or traditional concurrency control. Ad-hoc systems such as [KLMP84, LP83] allow cooperation among users based on an intuitive model of interactions. These systems lack a formal definition of correctness, and as such, it is not possible to give a mathematical characterization of the notion of correctness guaranteed. Indeed, such systems ultimately rely on the users of the system to correct any problems which may develop during execution. The justification for such schemes is that only the correctness of the final version matters. The users, being the designers or programmers, know far better than the system whether or not the final product is correct. Thus, correctness is the responsibility of the users, and high performance can be obtained.

The problem with this approach is that people are not adept at managing concurrent activity. Although the users know about their design, they do not necessarily understand the implications of their concurrent activities. It may be the case that their actions prevent a final, correct design to exist since correct changes to the design by different designers could be invalid when performed concurrently. This problem, especially if it was a subtle error, could go unnoticed and yield an incorrect design.

*This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 4355

On the other hand, traditional concurrency control is well-defined [Pap86, BHG87], but inadequate for long-duration transactions. The traditional concurrency control model is based on the concept of serializability. Serializability requires that all concurrent executions be equivalent to a non-concurrent (serial) execution. Systems which enforce serializability do not rely on users to ensure correctness, since the correctness of serializable schedules follows immediately from the correctness of individual transactions. Most protocols that ensure serializability use lock-based techniques. The difficulty with this is that in a design environment, it is possible for a lock to be held for days. Clearly it is unacceptable to force a transaction to wait that length of time for another transaction to release a lock. Other techniques, such as timestamping or validation, use transaction aborts to prevent non-serializable executions. However, it is clearly unacceptable to abort a long-duration transaction except in the most extreme of circumstances. These consequences of requiring serializability are so severe that the traditional transaction model is considered impractical for long-duration design applications.

An alternative is the model presented in [KS88]. This model incorporates three semantic enhancements in order to allow a better representation of transactions in these new applications. Explicit predicates are used in order to facilitate cooperation between users, nested transactions to provide multiple levels of atomicity within long-duration transactions, and multiple versions to improve concurrency. In [KS88] no name is given for this model. For simplicity of reference in this paper, we shall refer to this model as the NT/PV model (for Nested Transactions with Predicates and Versions). The NT/PV model allows a wide degree of flexibility in the use of predicates, nesting, and versions. The traditional transaction model is a special case of the NT/PV model. In [KS88], we identified classes of schedules that correspond to several interesting specializations of the NT/PV model and described concurrency control protocols to illustrate that the model can allow far greater concurrency than that allowed under serializability.

However, the NT/PV model has never been applied to any realistic problem from one of the new application areas. Due to its increased semantics, the NT/PV model leads to greater overhead than both ad-hoc systems and serializability. Therefore, despite the potential benefits of the NT/PV model, the practicality of this model is open to question. To answer this question, we present a realistic example from the field of computer-aided software engineering. This example is drawn from studies of three designers working independently on the same problem [GKC87, GCK88, GC87, GC88]. In our hypothetical example, they are given access to a hypertext tool called gIBIS [CB88b, CB88a], and their interactions are modeled using designer-based ad-hoc concurrency control, serializability, and an instance of the NT/PV model.

We assume a general familiarity with the traditional transaction model. Details of that model can be found in several texts, including [BHG87, KS86]. In the next section, we give an overview of the NT/PV model. Further details of this model can be found in [KS88]. In Section 3, we apply the software-engineering example of [GKC87, GC87, GC88, GCK88] to the NT/PV model. After discussing its intuition, we present the details of the example which shows the advantages of systems based on the NT/PV model. In the final section, we draw conclusions about the applicability and practicality of the model and suggest future directions for enhancing and modifying the model.

2 Informal Overview of the Model

The standard database model defines a transaction to be a series of read steps and write steps of items in a database. A schedule is an interleaving of steps from different transactions, such that the order of all steps within a transaction is preserved. If all steps of every transaction are grouped together, then the schedule is called a serial schedule. If a schedule is equivalent to a serial schedule, then the the schedule is called serializable. Serializability is the correctness criteria used in the standard database model.

Serializability is used in many applications today. Banking systems and airline reservation systems are two of the more common examples. However, serializability is unacceptable for a new class of applications which we call long duration transaction systems. Such a system exists whenever a transaction must interact with a human user in order to execute correctly. This definition includes not only traditional long duration transactions such as computer-aided design (CAD) applications, but also less discussed applications such as computer-aided software engineering (CASE), interactive programs like artificial intelligence knowledge bases and office automation systems.

It is shown in [Yan82] that unless additional information is used for concurrency control, serializability requires two-phase locking [EGLT76]. If two-phased locking is used for long duration transactions, then either long duration waiting is imposed on the transactions waiting for a lock or user work must be aborted. Neither option allows satisfactory system performance, so currently these applications currently require human involvement to ensure correctness of the system as a whole [DG87, KLMP84, HL82, Kat82]. This involvement is required because these applications involve semantics beyond serializability, which cause that model to be inappropriate.

In an effort to capture the essential semantics involved in long duration transaction systems, we have developed a new model. This model combines three enhancements to the standard model so that broader and more appropriate correctness criteria can be established. These enhancements are nested transactions, explicit predicates and multiple versions, which yield the name of the new model, nested transactions with predicates and versions, or NT/PV.

2.1 Nested Transactions

Nested transactions were first defined in [Mos85] as a model for distributed databases. Later, they were used to describe the interactions between designers in computer-aided design applications [BKK85, KKB88]. Nested transactions are an important extension of the standard database model, allowing greater semantics to be captured, and greater concurrency to be achieved.

Nested transactions are based on the concept of the *subtransaction*. A one-level subtransaction is a collection of read and write steps which the rest of the transaction views as a single operation. Thus, either all of the read steps return values, or none of them do, and either all of the write steps update the database, or none of them do. A subtransaction can be extended to contain not only read and write steps, but also other subtransactions, thus changing a transaction from a sequence of steps into a hierarchy of subtransactions.

A good way to think of this hierarchy is as a tree, where each node in the tree is either a subtransaction, or a database access, such as a read or write step. The database accesses are the smallest units in nested transactions, and are the leaves on the tree. The nodes immediately beneath a subtransaction node in the tree are the sub-subtransactions and database accesses which compose the subtransaction. In the NT/PV model, the nodes in a subtransaction are restricted to either subtransactions or database accesses, but not both. This does not reduce the representative power of the model, since a database access can be modeled as a subtransaction containing one operation.

Nested transactions are very useful for modeling long duration transactions. In [BKK85, KKB88], the concept of cooperating transactions is modeled using nested transactions. A cooperating transaction is a subtransaction which performs a specific task for another user, such as designing a pin interface for a computer chip, or a subroutine for a program. By encapsulating all of the cooperation in a separate subtransaction, the user can continue to work secure in the knowledge that he will see only the final result, and will not be bothered by intermediate results or changes to internal data values. However, the other users of the system, will only see the finished product when it is completed, and not subroutines without programs. Thus consistency both within and without a transaction can be maintained.

In the NT/PV model, all transactions are assumed to be subtransactions of an imaginary root transaction. These transactions are called top-level transactions. Every transaction is associated with a set of subtransactions or a set of database access steps. Additionally, each transaction has a partial order defined for the set of subtransactions. In the standard database model, an implicit total ordering exists for every transaction. The relaxation in the NT/PV model allows for greater overall throughput since steps can be performed in any order consistent with the partial order.

2.2 Explicit Predicates

All database concurrency control is based on the notion that it is possible to determine whether or not a database is correct. It may be the case that determining correctness is very difficult, but it is possible to do so. It is also assumed that a transaction which begins executing with a correct database, will finish executing with a correct database, provided it executes by itself. Thus, one way to derive a correct database

is to define an empty database as correct, and then run one transaction at a time on it. The final result, by definition, is a correct database.

Another alternative is to explicitly state, using predicate logic, what it means for a database to be correct. Correctness is then based on the values of the database entities. The values can then be placed into the formulas, and the correctness of the database can be determined. Recently, other methods have been created to check consistency constraints, such as automated theorem proving [MH89].

In [KKB88], this explicit consistency constraint is enforced by requiring every transaction to satisfy an invariant which is implied by the database consistency constraint. In the NT/PV model, we relax this criteria in two important ways. First, instead of an invariant, each subtransaction has a precondition and a postcondition. It must be the case that the postcondition is implied by the database consistency constraint. This preserves the consistency of the database. The precondition of every transaction describes the database state which is required for the transaction to execute correctly.

The second relaxation involves subtransactions other than top-level transactions. A subtransaction is also partially defined by preconditions and postconditions. However, these predicates do not have to be implied by the database consistency constraint. It is assumed that these subtransactions place the database in a temporarily inconsistent state which another subtransaction will correct later. This is similar to what happens in the standard model during a funds transfer application. After one account has been debited, but before the other account has been credited, the transfering funds are missing. This is acceptable since no other transaction can access the database in this state. Likewise, in the NT/PV model, only other subtransactions of the same transaction can access the database in these incorrect states.

It should be noted that the predicates in [KS88] are restricted to predicate logic, and are required to be in conjunctive normal form. This paper relaxes that requirement to any first order logic expression. However, even in our earlier work, the predicates assigned to transactions are used to define the broadest classes of correctness. As with serializability, every transaction is assumed to perform correctly if it is executed by itself. With the NT/PV model, this translates to requiring that if a transaction executes on a database which satisfies its precondition, then it leaves the database in a state which satisfies the postcondition. It is the responsibility of the concurrency control protocols to make sure this happens. An example of such a protocol is in [KS88].

2.3 Multiple Versions

In the standard database model, every data item in the database has exactly one value at any time. However, this value may change as updates are made. If instead of writing over the old value of a data item, a new version of the data item is created containing the new value and the old value is saved, it is possible to allow greater concurrency [Pap86, BHG87]. This happens because a transaction might need to read the old value, and if it has been destroyed, the transaction must be aborted. A system which keeps old data values for the data items uses multiple versions.

The greater concurrency of multiple versions is not free. Additional overhead in terms of both space and computing power is needed to take advantage of this enhancement. However, many long duration transaction systems already must contend with the concept of an *alternative*. An alternative is simply a different way to accomplish the same thing. For example, a different set of logic gates to form a chip, or a different algorithm to solve the same problem. Effectively, these are simply multiple versions of data items. Thus, the overhead required to exploit multiple versions is already present in many long duration systems.

One of the mechanisms required by multiple versions, is some method to determine which of many versions a transaction should read. Typically, a version function tries to assign to transactions versions which satisfy the correctness criteria. There exist many examples of version functions which can be used to ensure serializability [Pap86, BHG87].

In the NT/PV model, different criteria are used for correctness, therefore a different version function is needed. Here the version function must assign values to the data items in the precondition of the transaction such that it is satisfied. However, it must also assign versions in such a way that the partial order of the parent transaction is not invalidated. For example, if the parent transaction requires subtransaction p to precede subtransaction q, then p cannot read versions written by q. The presence of multiple versions makes it easier for a precondition to be satisfied, since every value a data item has had is still in the database, but

multiple versions also makes the problem NP-complete to determine if a database state satisfies a predicate [KS87]. An example of such a version function can be found in [KS88].

2.4 Executions of Transactions

By combining the enhancements of sections 2.1 through 2.3, the NT/PV model of transactions systems can be defined. In the NT/PV model, every database entity has multiple versions, and every transaction contains four parts. A transaction contains a set of subtransactions or database accesses, an ordering on that set, a precondition and a postcondition. It is assumed that if a transaction operates on a database state which satisfies its precondition and it is not interfered with by other transactions, it will leave the database in a state which satisfies its postcondition. However, in order to model the execution of the transaction, additional information is required.

In the NT/PV model, this information is based on a relation and a function for each transaction. The relation is over the set of subtransactions or database accesses associated with the transaction, and usually denotes conflicts within this set. The only formal restriction placed on this relation is that if two items are related in the partial order associated with the transaction, then this relation cannot reverse that relationship. In other words, if a designer defines subtransaction p to occur after subtransaction q, then if p and q conflict, it must be the case that all conflicting steps of q follow the corresponding steps in p.

The function assigns a database state to the transaction. This database state represents the values read by the database, and as such, this database state must satisfy the precondition of the transaction. If the transaction does not read a database entity, and the entity is not in the precondition, then any value can be assigned to the data item.

This function and relation greatly increase the expressive power of the NT/PV model. It is now possible to discuss conflicts between transactions, and even define concepts such as conflict serializability as a restricted form of correcntess within the model [KS87]. Likewise, the reads of a transaction can allow us to define requirements for view serializability, since we now know what values a transaction read [KS87]. Given these final pieces of the model, it is possible to define many correct classes, and model work done by others in database concurrency control.

2.5 Definition of the NT/PV Model

A transaction under the NT/PV model is represented as a four-tuple, $\mathbf{T} = (T, P, I, O)$, where T is the set of subtransactions in $\mathbf{T}$, P is a partial order on T, I and O are predicates on the database, with I being the input constraint, and O being the output constraint.

The traditional model can be represented within the NT/PV model by setting both I and O to be the database consistency constraint. The transactions in T are sets of database operations. Thus, $\mathbf{T}$ is a set of traditional transactions, each of which preserves the database consistency constraint.

If we choose to take advantage of the flexibility of the NT/PV model, we do not place a major burden on the transaction writer. Under the traditional model, a transaction must preserve the database consistency constraint. In the NT/PV model, a transaction must guarantee predicate O if it is run without concurrency on a database that satisfies predicate I. As in the traditional model, the transaction writer is responsible for the correctness of this code. However, in neither the traditional model, nor the NT/PV model must the transaction writer be concerned with concurrency. It is the responsibility of the transaction management component of the database system to ensure that correct transactions are not adversely affected by concurrency.

In this paper, we shall address the user-level application of the concepts of the NT/PV model rather than the system-level implementation aspects of the model. In [KS88], we give a formal definition of the NT/PV model and describe protocols to manage concurrency under the model. In the protocols we defined there, the predicates were required to be predicate calculus statements. However, this is too restrictive for our current purposes, and later work [KS] relaxed this requirement by allowing first-order logic statements.

3 A Software-Engineering Scenario

The example is a hypothetical interaction among three designers working to design an algorithm for coordinating the activities of an unknown number of elevators servicing a building with an unknown number of floors. This problem is called *the N-lift M-floor problem.* The N-lift M-floor problem is relatively simple compared to most tasks which would require long duration transactions in the software engineering field. However, it is complex enough to allow interchanging of ideas between different designers, and still simple enough to be presented in a reasonable format. In [GKC87, GC87, GC88, GCK88], the activities of several software designers were studied as they worked on this problem. For our example, we have chosen to focus on the three designers whose activities were described in detail in those papers. Following the notation of the original source of this example, we refer to these 3 designers as P3, P6 and P8.

In order to capture the design process, the hypertext tool gIBIS, [CB88b, CB88a], is used. gIBIS is a tool for representing discussion between people on a particular topic. The tool presents a graph consisting of issues, positions, arguments, and others, which are the nodes, and various kinds of arcs showing the relationship between the nodes. The allowed arcs are *generalizes, specializes, replaces, questions, is-suggested-by, responds-to, supports, objects-to* and *external.* For example, if an argument supported a particular position, a *supports* arc is drawn from the argument node to the position node.

Given the gIBIS tool, we assume a database consisting of gIBIS nodes and arcs. This database contains various discussions by different people on many different topics. Users may browse the gIBIS database by using gIBIS commands. Updates to this database are expressed by additions of nodes and arcs, which are also gIBIS operations. For simplicity, we will assume that an update to a given node in the gIBIS structure corresponds to a write operation on exactly one database item.

In [GC87, GC88], the designers were assumed to work in isolation. Our example assumes interaction among the designers. In this paper, the designers all begin work as they did in the case study. However, they are influenced by the efforts of the other members of the design team, causing the final design to be different from anything they individually developed. In all cases, an attempt is made to preserve the basic motivations of each designer, as described in [GC87, GC88]. Secondly, the notion of a transaction is completely absent from [GC87, GC88]. We assume in this paper that each designer has a notion of significant units of work, and does not wish to allow other designers to see incomplete work. Therefore, related gIBIS operations are grouped together into transactions.

Representing this problem in a readable format is very difficult. The gIBIS tool uses a sophisticated graphical interface to allow users to easily browse and update the system with "point-and-click" type operations. Nodes are grouped by their arcs and users navigate the graph to retrieve the desired information. However, such clustering cannot easily be represented on paper. Therefore static pictures with numbers representing the nodes are used. This format is not as useful as the gIBIS tool for practical systems, and should not be used to comment negatively on the usefulness of gIBIS.

The actual example presented is based on unconstrained interaction between the designers. Thus, whenever a designer is ready to perform an operation, a corresponding gIBIS step is presented. This is done so that the actual interactions between designers can be modeled as closely as possible, so that the advantages and disadvantages of serializability, no control, and the NT/PV model can be made clear. Whenever an interesting situation develops, we provide a discussion on mechanisms provided by the NT/PV model which would alter the interactions. Likewise, in these situations serializable protocols are also discussed. However, it is important to realize that the actual steps give in this example are based on designers operating with no concurrency control at all.

Finally, it should be noted that there are many different ways for systems using the NT/PV model to be implemented. One method would make each major design effort a top-level subtransaction. Each of these subtransactions could be broken down into smaller design efforts, such as the size of the example presented here. Even further refinement could be accomplished with each designer having his own subtransaction, which have subtransactions which consist of helpful activities performed by other designers. Such an overall strategy is proposed in [BKK85], and should be followed. The example problem presented here assumes that the efforts of P3, P6 and P8 are encapsulated in a subtransaction. Our discussion emphasizes the interaction among the designers.

3.1 The Specifics of the Problem

We can now present our example of a computer-assisted software engineering project. The problem is to design the control software for a general system of elevators within a general building. Thus, the actual number of floors and lifts are unknown. The system must observe the following restrictions:

1. Each lift has a set of buttons, 1 button for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited (i.e. stopped at) by the lift.

2. Each floor has two buttons (except the ground and top), one to request an up-lift and one to request a down-lift. These buttons illuminate when pressed. The buttons are canceled when a lift visits the floor and is either traveling in the desired direction, or visiting the floor with no requests outstanding.

3. When a lift has no requests to service, it should remain at its final destination with its doors closed and await further requests.

4. All requests for lifts from floors must be serviced eventually, with all floors given equal priority.

5. All requests for floors must be serviced eventually with floors being serviced sequentially in the direction of travel.

6. Each lift has an emergency button which, when pressed, causes a warning signal to be sent to the site manager. The lift is then deemed "out of service". Each lift has a mechanism to cancel its "out of service" status.

3.2 Introduction to the Design Scenario

Each designer applied a different technique to solving the problem. P3 started almost at once to apply logic programming techniques to the problem. He attempted to define the system in terms of logical assertions and generate a solution from them. P6 seemed to be familiar with problems similar to this one, made some basic decisions about the problem, and then started refining his solution. P8 took the most time to generate a solution, exploring the environment to a much greater extent than the others. He applied a system development technique called Jackson System Design [Jac83] to build a model of the environment, and then started to develop his solution. This hypothetical example combines the approaches taken by these designers into one effort. The resulting design combines elements from all three designers into something unlike what each had derived on his own.

Although this problem is relatively simple compared to most design problems, it is still a complex problem consisting of over 70 steps. The problem is described in a narrative on groups of gIBIS steps. The format for each gIBIS step is

##.P#: link-type link-node; node-type: description

where ##. is the time order of the gIBIS statement, P# is either P3, P6 or P8, link-type is one of the allowed arcs in gIBIS or none, link-node is the node the new node is linked to, node-type is one of the allowed gIBIS nodes, and description is a short passage describing the node.

A set of read and write operations are presented which correspond to each group of gIBIS statements. These are lower-level operations of which the users are unaware. These operations, or database steps, are of the form $X_i(j)$, where X is either R for a read step or W for a write step, $i \in \{3, 6, 8\}$ and will correspond to the designer performing the operation, and j will correspond to the database item (gIBIS node) on which the operation is performed.

It should also be noted that the data items in a typical design application are not single valued entities. These entities require several fields, some of which may even be repeating fields, in order to be represented adequately. In this example, there is only one type of entity, the gIBIS node. A node in the database contains several fields. These fields are named *arcs*, which consists of two values, one for type of arc and the

other for the node on the other end of the arc, *type*, which is one of the types of nodes allowed by gIBIS, *resolved*, which is the number of the node which resolves an issue, *topic*, which is a short description of the node, and *node*, which contains the number of the node. In this paper, if g is the name of the node, then g.arcs, g.type, g.resolved, g.topic, and g.node would be the names of the fields. For brevity, g.arcs is used to represent the value of the node on the other end of the arc, and g.arcs.type is the type of arc.

Likewise, the semantic information required by the NT/PV model is also associated with some groups of gIBIS steps. This information must be provided by the users, and is therefore strictly overhead. However, the example shows that this overhead is minor compared to the consequences of using either no correctness criteria or serializability.

The example is divided into four phases. In each phase the designers are working on a different part of the problem. In phase 1, the designers state what they consider to be the first problems to be solved. In phase 2, they establish some dialogs to solve some of the basic problems. In phase 3, the designers present progressively more complete solutions, and in phase 4 they finalize the gIBIS structure to reflect their work accurately.

3.3 The Example - Phase I

The first phase of the design effort is a general discussion of the properties and possible solutions of the N-lift M-floor problem. The designers each begin with their own reactions to the problem. P8, who in this example originally stated the problem, begins by applying system development techniques and enumerates the entities involved in the problem. P6, who has seen this type of problem before, starts by tackling the issue of distributed versus centralized control. Finally, P3 begins by using the "generate-and-test" method to design a solution.

gIBIS steps

1. P8: No links; Issue: How do we solve the N-lift M-floor problem?
2. P6: Responds-to 1; Position: Use centralized control
3. P8: Generalizes 1; Issue: What are the entities involved?
4. P3: Responds-to 1; Position: Use global request queue

DB steps

$W_8(1)R_6(1)R_3(1)W_6(2)W_6(1)W_8(3)W_8(1)W_3(4)W_3(1)$

From the database steps, it is clear that P6 and P3 responded to the initial question without seeing P8's idea. Note that each gIBIS update corresponds to two database writes unless there are no links involved. One of the writes inserts the new node, and the other write updates the already existing node to indicate the connection to the new node.

Soon after inserting his remarks, P3 reads the statements made by P8 and comments on them, while P8 and P6 continue with their own thoughts. P3 does not understand the design philosophy that P8 is trying to follow, and therefore thinks P8 is wasting his time.

gIBIS steps

5. P8: Responds-to 3; Position: Floors are entities
6. P3: Questions 3; Issue: Do we need to discuss this?
7. P8: Responds-to 3; Position: Lifts are entities
8. P6: Supports 2; Argument: Easy algorithms to implement
9. P8: Responds-to 3; Position: Requests are entities
10. P3: Responds-to 6; Position: No, it is a waste of time
11. P3: Supports 10; Argument: Isn't helpful for easy solutions
12. P3: Supports 10; Argument: Partial solution can be quickly derived

DB steps

$R_3(2)R_3(3)W_8(5)W_8(3)W_8(7)W_8(3)W_6(8)W_6(2)\ W_8(9)W_8(3)W_3(10)W_3(6)W_3(11)\ W_3(10)W_3(12)W_3(10)$

While P3 was thinking about the problems of trying to use P8's "method", whatever that turned out to be, he was also thinking about his own idea, concerning the use of a global request queue. As he thinks about it, he decides it was really much better to use his global request idea as an argument in favor of the more general question of control, rather than a solution of its own. Unfortunately, P6 has finished his thoughts on central control and reads the database before P3 could make his changes. P6 thought the same thing that P3 did, but since he did not write the node, he argues his position. P3 uses the drastic measure of deleting the node and inserts a new node. The results are quite unexpected for the designers. Note also that P8 reads P3's comments, and defends his approach.

gIBIS steps

13.P6: Questions 4; Issue: Isn't this an argument for central control rather than a position for solving the problem?
14.P3: delete 4
15.P6: Responds-to 13; Position: Yes, it is too specific
16.P3: Specializes 2; Position: Use global request queue
17.P6: Supports 16; Argument: It is only one possible solution using central control.
18.P8: Responds-to 6; Position: We need to understand the basics of the problem before solutions can be attempted

DB steps

$R_6(3)R_6(4)W_6(13)W_6(4)W_3(4)W_6(15)W_6(13)W_3(16)W_3(2)\ W_6(17)W_6(16)R_8(6)R_8(10)R_8(11)R_8(12)W_8(18)W_8(6)$

A graph representing the gIBIS structure after these steps is figure 1.

Already several problems can be noticed with this execution. Obviously, the arcs which are now hanging should be deleted. But if that happens, what should be done with the nodes which would be left disconnected from the rest of the gIBIS structure? Clearly, once this problem arises, the designers must work out what happened and how to solve it. This is purely extra work which in some instances might be very difficult.

These difficulties should be prevented by computer systems designed for these applications. Let us first consider how the traditional model would handle this. Serializability prevents these errors by forcing work to be done as if everything executed in a serial order. The biggest question here involves deciding what is a unit of work, (i.e. what is a transaction). One possibility is to claim that every update is a transaction. However, this is unsatisfactory since the same problem would arise in a serial order. Another possibility is to group all of the steps performed by a given designer together in one transaction. This would prevent this type of error since P6 would not see node 4, because P3 would delete it before he finishes. However, legitimate scenarios are not allowed using this transaction granularity. In fact, the first four gIBIS steps generate non-serializable executions because no serial execution of those database steps is equivalent to them.

On the other hand, the NT/PV model not only allows legitimate designer interaction, but also prevents this incorrect structure. This is accomplished by using additional semantic information which must be provided by the users. They need to inform the system of the predicates their work should maintain, and also which parts of their work should be considered independent from the rest of it.

As stated earlier, we assume that every major design effort is an independent subtransaction. Let **T** be the subtransaction corresponding to our example design effort. Certain operations by the designers will be grouped into subtransactions of **T**. Predicates must be defined for **T** and for all of the subtransactions of **T**. One possibility is the consistency constraint C1 below:

$C1 : \forall g \in G, g' \in g.\text{arcs} \rightarrow g' \in G$

where G represents the set of gIBIS structures.
In English, this statement requires all nodes connected by an arc must be nodes in the structure. This consistency constraint prevents the hanging arc problem. However, that is not sufficient for the input constraint of **T**. A better predicate is C2.

$C2 : \forall g \in G, (g' \in g.\text{arcs} \rightarrow g' \in G) \wedge (g.\text{type=issue} \rightarrow \exists g' \in g.\text{arcs such that } g.\text{resolved}=g')$.

Again, in English this statement implies that every issue has been resolved. This predicate could be the database consistency constraint for the gIBIS structures.

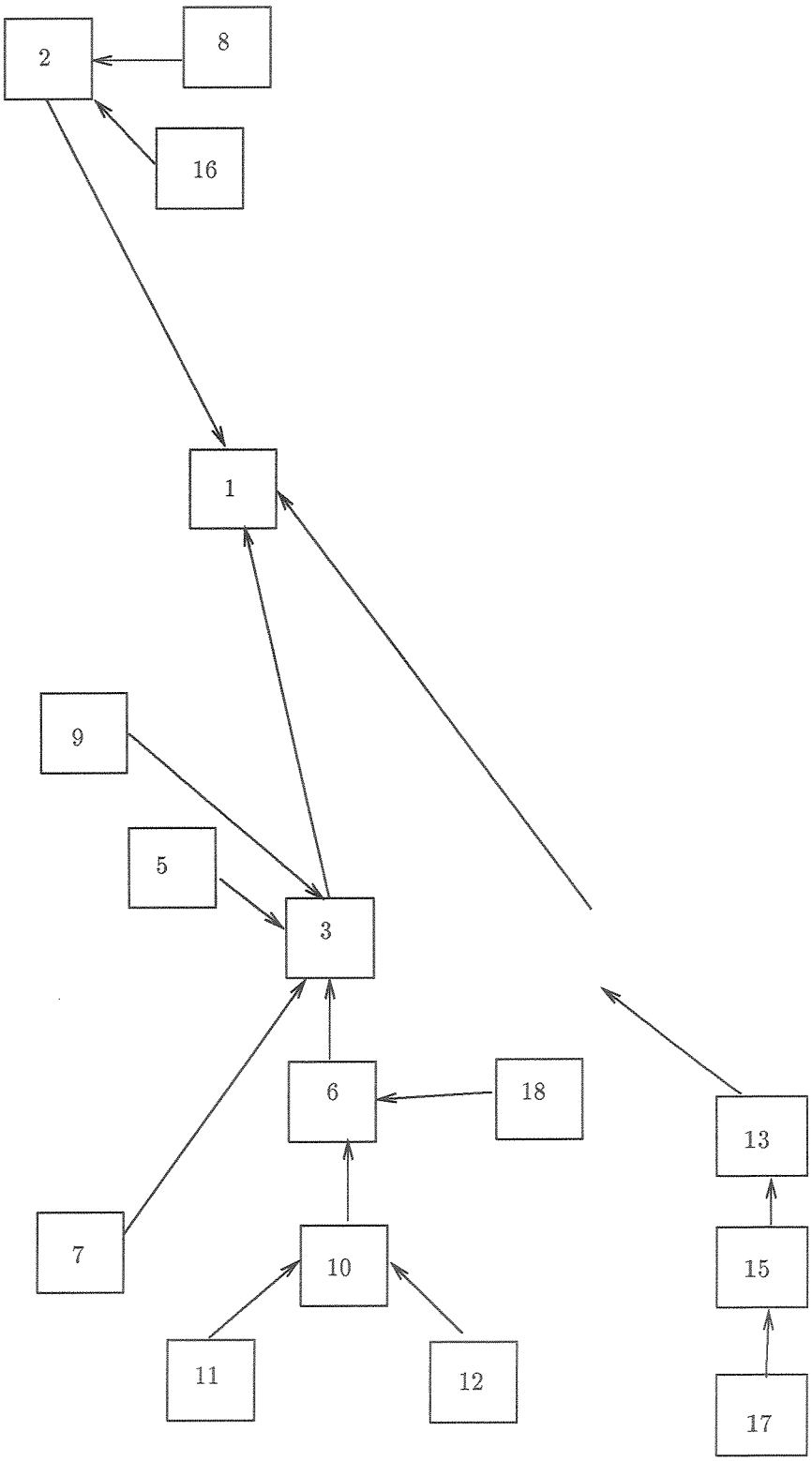

Figure 1: gIBIS structure after phase I

This database consistency constraint must be preserved after the conclusion of T, and should be part of the input predicate of the transaction. However, the transaction is also concerned about the state of a specific gIBIS structure, one dealing with the N-floor, M-lift problem. Therefore, the input constraint of T should be

$C3$: $\forall g \in G, g' \in$g.arcs $\rightarrow g' \in G \wedge g$.type=issue $\rightarrow \exists g' \in g$.arcs such that g.resolved=$g' \wedge \forall g'' \in G$ g''.topic $\neq$ N-lift,M-floor.

The first gIBIS step, performed by P8, is an independent operation, since P8 wants the other designers to be able to work as soon as possible. However, even a single step is a subtransaction, so P8 would have to declare an input constraint, an output constraint and insert the step into the partial order P. Since this step is the first subtransaction, P is not changed. Likewise, the input constraint for T can be used for this step, and the output constraint is simply O1.

O1: $\exists g \in G$ such that g.topic = N-lift, M-floor.

Next, all three designers began to formulate their own designs. In this case, all three would start their own subtransactions, and each would have I1 for an input constraint.

I1: $\exists g \in G$ such that g.topic=N-lift,M-floor.

Note that this statement is much weaker than the database consistency constraint. This is one of the biggest advantages of using input constraints. The output predicates for these transactions require only that there are no hanging arcs, so we use C1, from above.

The sequence of operations generated by these subtransactions would differ from the gIBIS sequence after step 5. Step 6 in the sequence has P3 reacting to some of the early statements made by P8. Clearly, if the all of the steps made by P8 are considered to be a single transaction, then P3 can't react until all of the steps are finished. Also, consider what happens when P6 attempts to argue about the placement of node 4 while P3 is deleting node 4 at the same time. Since the output constraint of P6 claims there should be no hanging arcs, the transaction cannot terminate. The system should inform the designer of the problem, and let him correct it. Since the problem has just happened, it is easier for the designer to determine the appropriate response than if the problem had to be rectified later.

3.4 Phase II

In the second phase of this example, the designers begin to interact in a much more direct manner. Two of the designers, P3 and P6, begin a dialog concerning the advantages and disadvantages of centralized and distributed control. Meanwhile, P8 continues to elaborate on his design methodology. He eventually realizes that his approach is leading towards the same conclusions as the others.

gIBIS steps:

19.P6: Responds-to 1; Position: Use distributed control
20.P8: Is-suggested-by 9; Issue: What is the order of incoming requests?
21.P3: Supports 2; Argument: Can elect new central controller if needed
22.P8: Generalizes 20; Issue: What kind of control is needed?
23.P6: Supports 19; Argument: No single point of failure
24.P6: Objects-to 2; Argument: System halts if central site goes down
25.P3: Supports 2; Argument: Crashes are rare
26.P6: Supports 19; Argument: Handles asynchronous inputs
27.P3: Supports 2; Argument: Easily integrates requests

DB steps:

$W_6(19)W_6(1)R_3(19)$ $W_8(20)W_8(9)W_3(21)W_3(2)R_6(21)W_8(22)W_8(20)W_6(23)W_6(19)$ $R_3(23)W_6(24)W_6(2)$ $R_3(24)$ $W_3(25)W_3(2)R_6(25)W_6(26)W_6(19)$ $R_3(26)W_3(27)W_3(2)R_6(27)$

Notice that P6 and P3 are presenting the standard arguments about centralized versus distributed control. They are using the gIBIS format to discuss an important part of their design, and as such, need to be able to display their views quickly to the other person. This is a common design problem, where two users are cooperating on the same design. Even if both of the users access the same data item, it is not a conflict,

because they are working together. However, their work should be distinct from what P8 is doing, since P8 is taking a different approach. Therefore, P6 and P3 need the concept of a cooperating transaction [BKK85].

gIBIS steps:
28.P8: Add Specializes arc from 22 to 2;
29.P8: Add Specializes arc from 22 to 19;
30.P8: Specializes 9; Position: Floor requests
31.P8: Specializes 9; Position: Lift requests
32.P3: Other 16; External: Partial solution to problem, using global request queue

DB steps:
$W_8(22)W_8(2)W_8(22)W_8(19)W_8(30)W_8(9)W_8(31)W_8(9)\ W_3(32)W_3(16)\ R_8(19)R_8(21)R_8(23)R_8(24)R_8(25)\ R_8(26)\ R_8(27)$

A graph representing the gIBIS structure after these steps is figure 2.

If no concurrency control is used, then P8's actions become a problem, as his work, which accesses some of the same nodes as that of P3 and P6, must be presented to the other designers. Although P3 and P6 would eventually want to see this work, it can only serve as a distraction from their current discussion. Thus, some mechanism for allowing different amounts of interaction is needed. However, serializability is too restrictive, because cooperating transactions are typically non-serializable.

The NT/PV model provides an elegant means for implementing cooperating transactions. Either P6 or P8 would create a subtransaction with an input constraint like I2 and an output constraint like C1.

I2: $\exists g \in G$ such that g.topic=N-lift,M-floor $\wedge$ $\exists g \in G$ such that g.node=10.

Each designer would then perform operations as subtransaction with *true* being the input constraint and outpout condition for every subtransaction. Therefore, the designers could cooperate as they needed, and P8 would be uninvolved. P8 could neither influence nor be influenced by their work, until it was completed. Note also that P3 and P6 could, if they needed, have more complex subtransactions and thus perform larger amounts of work without interaction with the other designer. Thus, cooperating transactions are easily accomplished in the NT/PV model.

3.5 The Example - Phase III

During this stage of the design, [GC87] noted that P3 is generating incorrect solutions. He tested his solutions and discovered this, but in a cooperative environment, it is certainly more likely for the other designers to find faults in his design. Thus, in the stage of the design, P3 would generate a design, one of the others would find a problem with it, and then he would patch the problem. After the control problem is resolved, the other two designers pushed on towards another problem, that of scheduling. Notice here how many of the activities performed by a designer seem to be independent of not only the other designers, but of other activities performed by that designer. However, each activity is much more complex than earlier ones, and would take much longer to perform. This points out a problem with serializability, which we will discuss at the end of this section.

gIBIS steps:
33.P6: Responds-to 32; Position: Incorrectly orders requests
34.P6: Is-Suggested-By 33; Issue: How should scheduling be accomplished?
35.P3: Other 2; External: Partial solution correcting found error
36.P8: update 1 to resolved by 2

DB steps:
$R_6(32)W_6(33)W_6(32)R_3(33)W_6(34)W_6(33)W_3(35)W_3(2)\ W_3(1)$

The designers finally reached a decision on control, either by voting or some other mechanism, and indicated this by marking node 1 as "resolved". In the graphs of the gIBIS nodes, a resolved node has an uppercase "R" in it. Only issues can be resolved, and it is assumed they are resolved by one of the positions attributed to it.

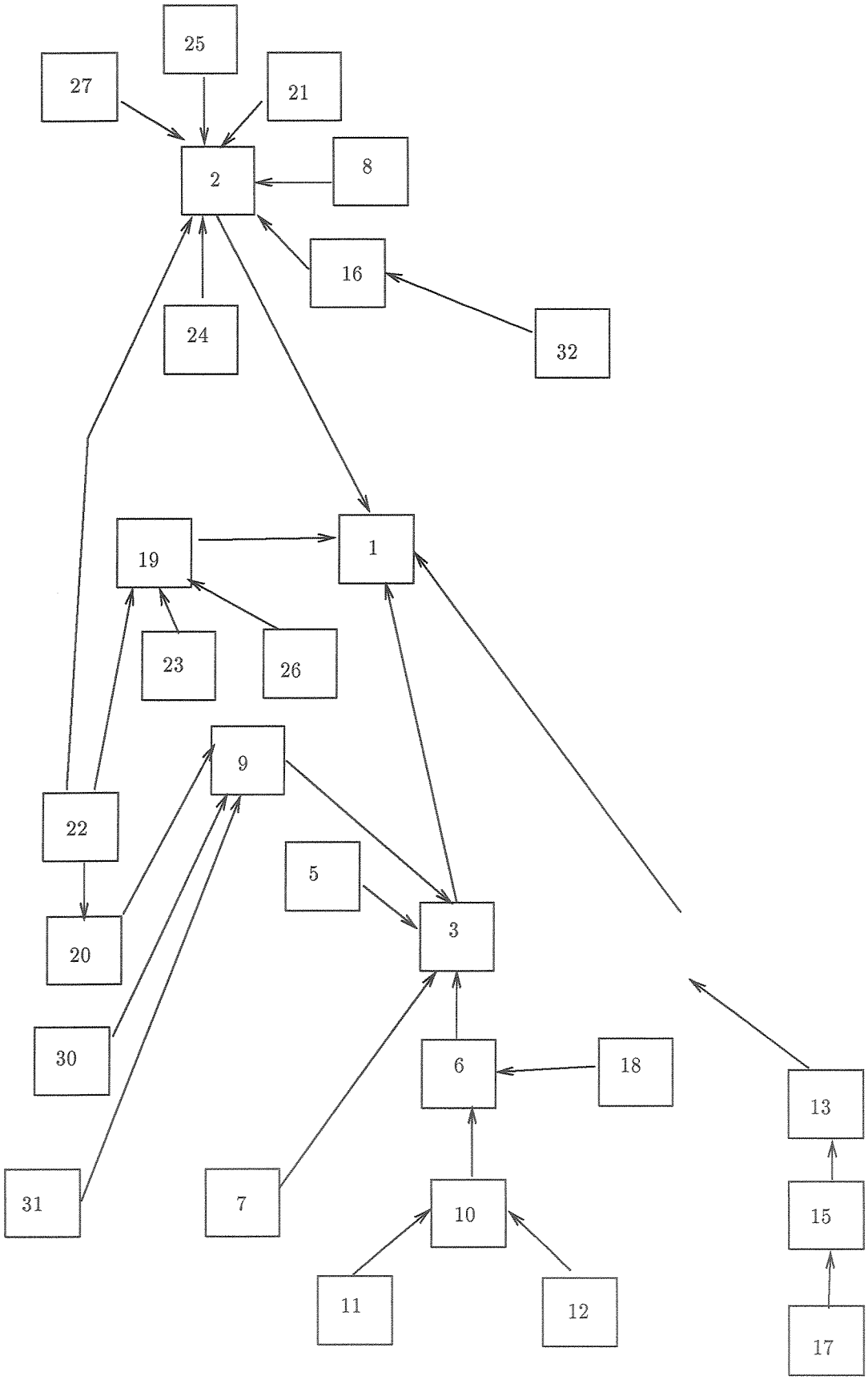

Figure 2: gIBIS structure after Phase II

Also note that since his solutions were being rejected, P3 tries a different approach. He proposes a solution as a communicating finite state machine, or CFSM, model. A CFSM consists of states and transitions. A transition is caused by either sending a message, receiving a message or an internal activity. P8 and P6 also start using them, and P6 even presents some reasons for it in the gIBIS structure.

gIBIS steps:
37.P8: Is-Suggested-By 31: Issue: How can we handle just single requests?
38.P8: Add Is-Suggested-By arc from 30 to 37
39.P3: Other 2; External: CFSM model of single lift
40.P8: Other 37; External: CFSM model of single request handling with central control.
41.P3: Add Other arc from 7 to 39
42.P8: Add Supports arc from 40 to 2
43.P3: Add Other arc from 31 to 39

DB steps:
$W_8(37)W_8(31)W_8(37)W_8(30)$ $W_3(39)W_3(2)W_8(40)W_8(37)W_3(7)W_3(39)W_8(2)W_8(40)$ $W_3(31)W_3(39)$

Notice how P3 and P8 are beginning to reach similar conclusions, although they have approached the problem from totally different perspectives. Likewise, P6 continues to develop the general questions which the others perceive only as specific problems. The designers will expand their models in the next steps, until a final CFSM model exists.

gIBIS steps:
44.P6: Responds-to 34; Position: Model central control as CFSM
45.P6: Supports 44; Argument: Describes conditions well, since control must both communicate and record state of all lifts.
46.P6: Responds-to 35: Position: Uses non-deterministic scheduling which does not satisfy constraints
47.P6: Is-Suggested-by 44; Issue: How should communication be done?
48.P6: Responds-to 47; Position: Model lifts as CFSM devices
49.P6: Supports 48; Argument: Represents communication with central control and state of lift
50.P6: Other 47; External: CFSM model of lifts
51.P3: Add Generalizes arc from 39 to 50
52.P8: Other 34; External: CFSM of scheduling
53.P3: Other 52; External: Better CFSM model for scheduling
54.P6: Other 53; External: Final CFSM model for lifts

DB steps:
$W_6(44)W_6(34)$ $W_6(45)W_6(44)W_6(46)W_6(35)R_3(46)$ $W_6(47)W_6(44)W_6(48)W_6(47)$ $W_6(49)W_6(48)W_6(50)$ $W_6(47)$
$R_3(50)R_8(50)W_3(39)W_3(50)W_8(52)W_8(34)R_3(52)R_6(52)$ $W_3(53)W_3(52)R_6(53)R_8(53)W_6(54)W_6(53)$

The basic problem has been solved. However, one question remains for the programmers. P6 wonders if they should develop the actual protocol used for the communications. P8 responds by saying that is too fine a level of detail for their work, and the designers agree that they are finished. They only thing that remains is to clean up the gIBIS structure, which is phase four.

gIBIS steps:
55.P6: Generalizes 47; Issue: What is the protocol for the CFSM?
56.P6: Responds-to 55; Position: Alternating bit protocol
57.P8: Responds-to 55; Position: No protocol is needed for the design
58.P6: Supports 57; Argument: Easy to implement and sufficient
59.P8: Supports 58; Argument: Systems programmer should use what is available.

DB steps:
$W_6(55)W_6(47)R_8(55)W_6(56)W_6(55)R_8(56)$ $W_8(57)W_8(55)W_6(58)W_6(57)W_8(59)W_8(58)$

A graph representing the gIBIS structure after these steps is figure 3.

In both of the first two phases of this example, we have claimed the database steps related to the gIBIS steps were not serializable. At those times, we merely stated that certain steps were not serializable and did not elaborate on the effect this would have on the design. In this phase we can clearly present the problems associated with requiring serializability.

First, serializability requires the concept of a transaction. There are three basic approaches which can be taken to define transactions for the designer's work. At the extremes, a transaction can be defined to consist of all of the work performed by a designer, or a single update. The third alternative is to allow designers to define transactions as logical units of their work, much as the NT/PV model allows designers to define subtransactions as they need them.

The extreme notions are not workable in design databases. If we assume the designers are using two-phase locking to ensure serializability, a designer must lock a data item before he can update it, and he must hold that lock until he has acquired all the locks he needs. It is certainly possible for a designer to hold a lock for most of the duration of his transaction. Since a transaction consists of all of the operations in his design process, he could hold locks for several *days*, during which time, no other designer could access that data item. In our example, since resolving an issue involves updating the corresponding node, two-phase locking would cause P8 to hold a lock on data item 1 until gIBIS step 36 above. Therefore, P3 and P6 could not even read the node to find out the problem to be solved, until after this step. Clearly, this is not acceptable.

Another alternative is to make every update a transaction. This definition of a transaction has some advantages over the other extreme. Transactions are all of short duration, so long duration waits cannot occur. However, using single updates as a transaction amounts to using no concurrency control at all. As soon as a designer does anything, all of the other designers can see and act upon it. This prevents designers from correcting mistakes, such as when P3 realized his idea for node 4 would be better as node 16 or from grouping logical steps together, such as P3's efforts with gIBIS steps 39, 41 and 43 above.

Therefore, the only practical alternative is to require designers to define their own transactions. This method is an extension to the standard model, but even this is not enough. For example, assume P3 considers steps 39, 41 and 43 to be a transaction, while P8 considers steps 37, 38, 40 and 42 to be a transaction. Let T3 be the transaction started by P3 and T8 be the transaction started by P8. The database steps listed below step 43 are exactly those required to implement these transactions. Notice that T3 accesses data item 2 before T8 does. Thus, the serial order equivalent to this schedule must be T3 T8 if the accesses to data item 2 are to be the same. However, T8 accesses data item 31 before T3 does. Thus the equivalent schedule must be T8 T3. Clearly, no serial schedule can accommodate both orderings. Therefore, even allowing designers to define transactions is not enough to allow serializability to model the designers' interactions.

The NT/PV model does not have this difficulty. Designers are given the power to define subtransactions, thus eliminating both of the problems associated with extreme transaction sizes. Likewise, the power of explicit predicates can allow the designer interaction required. Both transactions would have C1 as their output constraint while T3 would have I3 as its input constraint and T8 would have I3′ as its input constraint.

I3: $\exists g \in G$ such that g.topic=N-lift,M-floor. $\wedge \exists g \in G$ such that g.node=2,7,31.

I3′: $\exists g \in G$ such that g.topic=N-lift,M-floor. $\wedge \exists g \in G$ such that g.node=2,30,31. Since the input constraints are satisfied, the transactions can execute. Likewise, when both transactions terminate, they satisfy their output constraint, and the designers' interaction is allowed.

3.6 The Example - Phase IV

In the final phase of this example, the designers are required to clean up the gIBIS structure so that it adequately represents their thoughts, agreements, disagreements, and so forth. This job consists of adding arcs which the designers believe would make the design more comprehensive, as well as denoting how the issues were resolved. This phase shows the value of output constraints in the NT/PV model. The gIBIS steps are presented first.

gIBIS steps:

60.P8: Responds-to 3: Position: Floors,lifts,and requests are entities

61.P3: delete hanging arc from 13

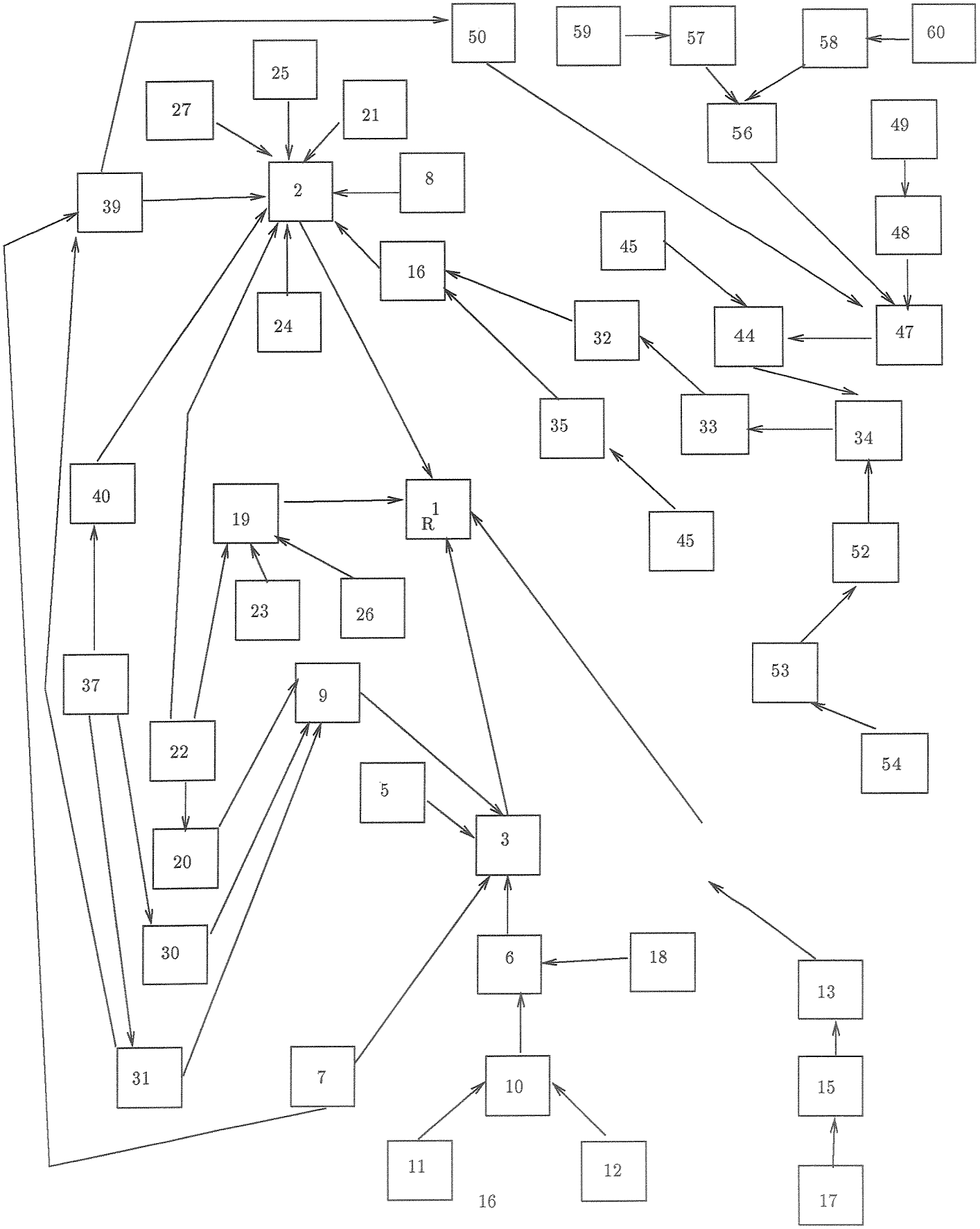

Figure 3: gIBIS structure after Phase III

62.P8: Add Generalizes arc from 9 to 60
63.P3: delete hanging arc to 1
64.P8: Add Generalizes arc from 5 to 60
65.P3: Add Is-Suggested-By arc from 13 to 16
66.P8: Add Generalizes arc from 7 to 60

DB steps:

$W_8(60)W_8(3)W_3(13)W_8(9)W_8(60)W_3(1)W_8(5)W_8(60)W_3(13)W_3(16)\ W_8(7)W_8(60)$

Steps 60 through 66 above represent the "clean-up" work designers in a user-based concurrency control system typically have to perform. The designers had to search the database looking for problems, and then perform the steps needed to correct them. P8 noticed that no single node resolved the issue of node 3, so he created one and linked it in. P3 noticed the hanging arc problem, deleted the hanging arcs and reconnected the figure. Although it is simple in this case to point out the obvious problems, in general this can be a very difficult problem.

The remaining steps are required to show others the resolutions of the issues raised. Since each issue needs to be resolved, this requires the designers to go back through the structure and update each issue node accordingly.

gIBIS steps:

67.P3: update 13 to resolved by 16
68.P3: update 3 to resolved by 61
69.P3: update 55 to resolved by 57
70.P8: update 6 to resolved by 18
71.P8: update 37 to resolved by 40
72.P8: update 20 to resolved by 22
73.P8: update 22 to resolved by 2
74.P3: update 34 to resolved by 44
75.P3: update 47 to resolved by 48
76.P3: commit gIBIS structure

DB steps:

$W_3(13)W_3(3)W_3(55)W_8(6)W_8(37)W_8(20)W_8(22)W_3(34)W_3(47)$

A graph representing the gIBIS structure after these steps is figure 4.

Under the NT/PV model, the two designers could each create a subtransaction, each with input constraint I4. Each subtransaction would have C3 as its output constraint since these are the last subtransactions in this transaction. Whenever the designers tried to quit before the output constraints were satisfied, the system would inform the designers of the problem. Therefore, none of the problems associated with designer-dependent concurrency control occur, as the system automatically prevents it.

I4: $\exists g \in G$ such that g.topic=N-lift,M-floor. $\wedge \exists g' \in G$ such that g'.type=issue $\wedge g'$.resolve is undefined.

Notice that some of these steps, such as 61 and 63, would not be needed if the NT/PV model is used. These steps delete hanging arcs which would have been handled when they happened, rather than at this late stage of the process.

4 Conclusion

We have studied the hypothetical interaction among three designers in an attempt to model a real problem in a software-engineering database application. The designers are based on actual programmers, and they are given an existing, if still experimental software development system to use.

We have used this problem to show many of the difficulties in modeling long duration transaction systems. We have seen that the traditional model is unable to model adequately the concurrency necessary in this example. Treating the work of each designer as a transaction (with no subtransactions) led to the inability of

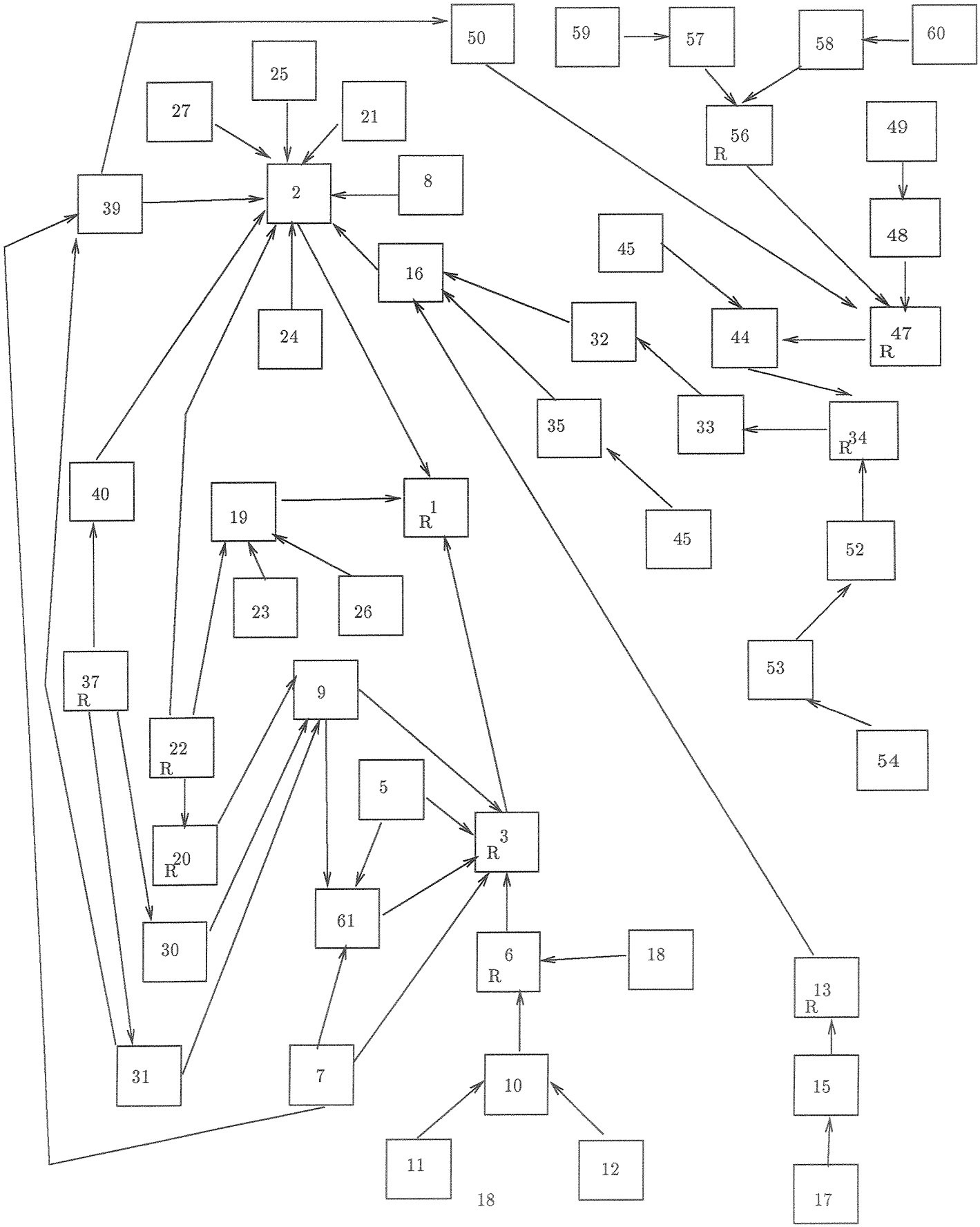

Figure 4: Figure 4

designers to cooperate, yet treating each step as a transaction allowed designers to see incomplete subtasks being performed by other designers.

We have seen also that, using the NT/PV model, we are able to decompose the designers' activities into a collection of nested transactions. At the highest level, the entire example presented here is a transaction. It is split into subtransactions that correspond to logical subtasks in the design process, with the designers interacting both cooperatively within a subtransaction and independently through distinct subtransactions.

We have shown how the designers' transactions would be allowed to interact under the NT/PV model. The restrictions imposed by the model serve to ensure that the designers take necessary actions to maintain database consistency. These restrictions, however, did not become an obstacle to the design process, as would have been the case if a traditional, serializability-based approach were taken.

We have not addressed system-level issues in this paper. To establish fully the practicality of the NT/PV model, we need to build a working transaction manager based on the model and measure its performance on a real-world application. In [KS88] we provide some insight into systems issues related to the model. Further discussion of protocols and implementation will be the subject of future reports.

Acknowledgements

We wish to acknowledge Dr. Laurie Werth of the University of Texas at Austin for her aid and guidance in the synthesis of the example problem.

References

[BBG86] C. Beeri, P.A. Bernstein, and N. Goodman. A model for concurrency in nested transaction systems. Technical Report TR-86-03, Wang Institute of Graduate Studies, 1986.

[BHG87] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[BKK85] F. Bancilhon, W. Kim, and H. Korth. A model of cad transactions. In *11th Conference on Very Large Databases*, 1985.

[CB88a] J. Conklin and M. Begeman. gibis: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, October 1988.

[CB88b] J. Conklin and M. Begeman. gibis: A hypertext tool for team design deliberation. Technical Report STP-016-88, MCC, 1988.

[DG87] H.C. Du and S. Ghanta. A framework for efficient ic/vlsi cad databases. In *13th Conference on Very Large Databases*, 1987.

[EGLT76] K. Eswaren, J. Gray, R. Lorie, and I. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, November 1976.

[GC87] R. Guindon and B. Curtis. Control of cognitive processes during software design: What tools would support software designers? Technical report, MCC, 1987.

[GC88] R. Guindon and B. Curtis. Control of cognitive processes during software design: What tools are needed? In *Chi 88*, 1988.

[GCK88] R. Guindon, B. Curtis, and H. Krasner. A model of cognitive processes in software design: An analysis of breakdowns in early design activities by individuals. Technical report, MCC, 1988.

[GKC87] R. Guindon, H. Krasner, and B. Curtis. Breakdowns and processes during the early activities of software design by professionals. Technical report, MCC, 1987.

[HL82] R. Haskin and R. Lorie. On extending the functions of a relational database system. In *8th ACM SIGMOD Conference*, 1982.

[Jac83] M. Jackson. *System Development.* Prentice Hall, 1983.

[Kat82] R. Katz. A database approach for managing vlsi design data. In *19th Design Automation Conference*, 1982.

[KKB88] H. Korth, W. Kim, and F. Bancihlion. On long-duration cad transactions. *Information Sciences*, October 1988.

[KLMP84] W. Kim, R. Lorie, D. McNabb, and W. Plouffe. A transaction mechanism for engineering design databases. In *10th Conference on Very Large Databases*, 1984.

[KS] H. Korth and G. Speegle. Theoretical aspects of the nt/pv model. work in progress.

[KS86] H. Korth and A. Silberschatz. *Database Systems Concepts.* McGraw-Hill, 1986.

[KS87] H. Korth and G. Speegle. Formal model of correctness without serializability. Technical Report TR-87-XYZZY, University of Texas, 1987.

[KS88] H. Korth and G. Speegle. Formal model of correctness without serializability. In *SIGMOD International Conference on Management of Data*, 1988.

[LP83] R. Lorie and W. Plouffe. Relational databases for engineering data. Research Report RJ 3847 (43914), IBM, 1983.

[MH89] W. McCune and L. Henschen. Maintaining state constraints in relational databases: A proof theoretic basis. *Journal of the Association for Computing Machinery*, 1989.

[Mos85] J. E. B. Moss. *Nested Transactions - An Approach to Reliable Distributed Computing.* The MIT Press, 1985.

[Pap86] C. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, 1986.

[Yan82] M. Yannakakis. Issues of correctness in database concurrency control by locking. *Journal of the ACM*, 29(3):718–740, July 1982.