SRI International

A Representation of Parallel Activity Based on Events, Structure, and Causality

Technical Note 401

December 8, 1986

By: Amy L. Lansky
Artificial Intelligence Center
Computer Science and Technology Division

**APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED**

This research has been made possible by the Office of Naval Research, under Contract N00014-85-C-0251, and by the National Science Foundation, under Grant IST-8511167. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of the Office of Naval Research, NSF, or the United States government.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

Most AI domain representations have been based on state-oriented world models. In this paper we present an event-based model that focuses on domain events (both atomic and nonatomic) and on the causal and temporal relationships among them. Emphasis is also placed on representing *locations* of activity and using them to structure the domain representation. Our model is based on first-order temporal logic, which has a well-understood semantics and has been employed extensively in concurrency theory. We show how temporal-logic constraints on *event histories* (records of past activity) can facilitate the description of many of the complex synchronization properties of parallel, multiagent domains.

1 Introduction

The duality between events and states is a well-known phenomenon. In a state-based representation, the world is viewed as a series of states or "snapshots" that are altered by events. Events are modeled solely in terms of their state-changing function. Alternatively, the dual, event-based approach represents the world in terms of a set of interrelated events. In this context, the "state" of the world at any particular point in time is represented in terms of the set of events that have occurred up to that moment (see Figure 1).

Most AI domain representations have relied on state-based models. In this paper we explore the dual view and examine its impact on the representation of multiagent domains — domains in which parallel activity is inherent and vital. As with most dualities, the choice of one representation over another may not affect any essential capability for expression; after all, one dual representation can usually be converted into the other. However, the mere form of a representation may make certain kinds of properties more natural to express and reason about. We believe that an event-based approach holds this advantage with respect to many of the complicated properties of multiagent worlds.

The representation described in this paper is based on the GEM concurrency model [17,18,19,20]. As advocated by philosophers such as Davidson [6] and as manifested in several AI representations such as Allen's and Georgeff's [1,10], GEM reifies events and explicitly represents their causal and temporal interrelationships. However, unlike previous AI representations, events are the *primary* elements of our world model and state is defined strictly in terms of past event activity. Thus, the work described in this paper explores the use of events and event relationships in way that is more general than previous work on knowledge representation and reasoning.

Another important aspect of the GEM representation is an explicit emphasis on *location* of activity. Specific mechanisms are provided for structuring events into logical locations of occurrence as well as for grouping those locations together in various ways. These event structures help organize the way a domain is described – for instance, particular domain constraints can be localized within a given context or subset of events. Structural contexts can also be used to actually represent properties of the domain. For example, particular event sets can be used to represent locations of forced sequential activity, scopes of potential causal effect, or the boundaries of localized forms of knowledge. In this way, domain structure helps to attack aspects of the frame problem. Domain structure can also be utilized as a heuristic in guiding computation; for example, it can serve as a guideline for the decomposition of planning tasks.

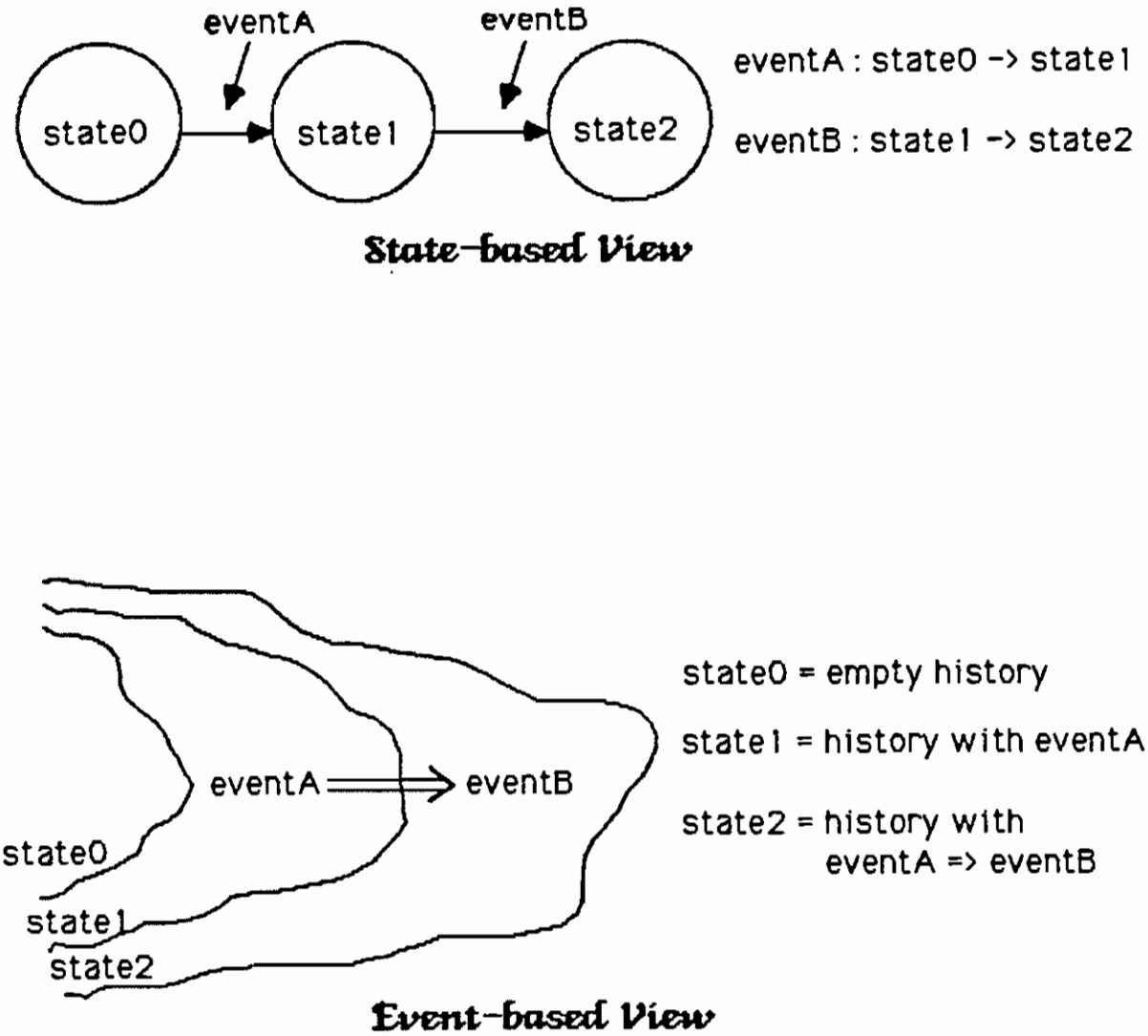


Figure 1: State/Event Duality

Within an event-based model, such as the one we are proposing, a notion of "state" is most naturally defined in terms of past activity: the state of the world at any point in time is merely a record of the events that have occurred and their interrelationships (once again, see Figure 1). *State descriptions* can be used to characterize sets of states, just as in a state-based model. These descriptions are typically formulas that describe patterns of past activity – for example, "the state in which a red robot has registered a request for tool X followed by a request for tool Y, but has not yet received either tool" or "the state in which Jack and Jill have just simultaneously begun running down the hill." These "behavioral" descriptions of state are formulas that explicitly describe temporal and causal relationships *between events.* In GEM, behavioral state descriptions (as well as all domain constraints) are stated as first-order temporal-logic formulas. Because these formulas are cast directly in terms of events and event relationships, a wide range of behavior-related properties can be described succinctly. Indeed, descriptions of states in which simultaneous activity has occurred are impossible to formulate in many state-based representations. As we will show, more conventional state descriptions based on state predicates can also be utilized in our event-based framework (see Section 6).

The primary aim of this paper is to convey the expressive power behind a world model based on structured, interrelated events. We begin in Section 2 by motivating our underlying approach and relating it to other work. Section 3 provides a more formal description of the GEM world model and presents a semantics for our domain descriptions. The power behind this representation is then more fully illustrated in Section 4 through construction of a complex domain description. Finally, Sections 5 and 6 focus on the modeling of nonatomic events, on building state descriptions, and on the frame problem. In Section 7 we conclude with a brief discussion of our current work on building a planner based on this event-oriented framework.

2 Motivation and Background

2.1 A Scenario

One of the primary goals behind any representational mechanism is to capture the properties of a domain in a useful and coherent way. Consider the following scenario. Three sets of friends decide to meet for dinner at an elegant restaurant. Each person must find his or her own mode of transport to the restaurant (personal car or taxi) and the first person from each party to arrive must book a reservation for his or her group. The maitre d' at the restaurant, Felix, happens to be a somewhat mercenary fellow who gives preference to parties that bribe him. In fact, everyone knows that a $50 bribe will guarantee eventual seating. However, he does have some scruples; he will seat a party only if a table of the correct size is available and

if the entire party has arrived (members of a party must be seated simultaneously). All other things being equal, he will then seat parties on a first-come-first-served basis. After being seated, guests may then choose, order, and eat their meals.

This scenario, though somewhat complex, is typical of situations we confront in our day-to-day lives. To describe it requires representation of several interlocking synchronization constraints (all of Felix's seating rules) as well as multiple ways of achieving goals (traveling to the restaurant by car or taxi). It also manifests some naturally emerging forms of structure: individuals are grouped into parties; certain locations of activity may be viewed as resources at which only limited forms of activity may occur (the tables at the restaurant, the attention of Felix); knowledge is partitioned (some of Felix's actions and habits are known by all, whereas others may not be). These kinds of properties are found in many domains, including factory management and scheduling, robotics, and organizational coordination.

When planning for a domain such as this, it is clear that some activities are loosely coupled and can be planned separately (each person's plan for getting to the restaurant), while others must be tightly coordinated (the seating of the three parties). In addition, the expansion of some nonatomic actions will preserve a sense of locality and maintain constraints that may have been solved at a higher level of abstraction (for example, each person's menu selection plan). In other cases, however, nonatomic-event expansion will result in activity that spans across locations accessible to others, and may thus require rechecking after expansion (possible contention for the use of a limited number of taxis).

In Section 4 we shall illustrate our GEM-based representational approach by building a specification of this scenario. In Section 7 we outline a planner currently under construction that is based on GEM and that explicitly uses domain structure to guide the planning process. First, however, we take a brief look at other common forms of domain specification and the ways in which they might tackle the restaurant problem.

2.2 Other Approaches

Most traditional AI domain representations model the world as a sequence of states, and actions or events as relations between sets of states [8,23,27].[1] States descriptions are typically constructed in terms of a set of state predicates, and actions are defined in terms of preconditions and postconditions on state. This is the basic descriptive framework underlying classical planning systems such as STRIPS [8], NOAH [28], and other planners [32,33,34]. Some of these representations require that all events be totally ordered in time [8,23]. In others, events are ostensibly partially ordered, but it is still assumed that potentially interacting events occur in some total order that conforms to the partial order [28]. For instance, this premise

[1]Although they are often viewed as distinct, we shall use the terms "action" and "event" interchangeably.

underlies use of the STRIPS assumption [8]. While the STRIPS assumption can be used to determine the effect of individual events on world state, it cannot be used to determine the combined effect of simultaneous events. Since parallel, multiagent domains may sometimes even *require* that events occur at the same time (for example, two robots picking a heavy block up together), this is a definite limitation.[2]

Another disadvantage of traditional state-based frameworks is that they spread the representation of some kinds of properties across several action descriptions. For example, to encode the restaurant scenario's seating rules, several state predicates would have to be maintained to encode the "synchronization state" of the reservation desk: who is waiting, in what order they arrived, who gave the biggest bribe, how large the parties are, what tables are available, etc. The preconditions and postconditions of reservation and seating-related actions would involve complex manipulation of and interactions among these predicates. Each of the seating constraints is essentially spread out among the descriptions of the actions that are involved. Changes in the constraints may therefore entail fairly complex and nonobvious changes in action descriptions. Clearly, it would be simpler if each of Felix's rules were encoded as a separate, succinct constraint on the relationship among domain actions.

More recent approaches to domain representation have been based on *state intervals*. Events (actions) and/or predicates are modeled as state sequences occurring over an interval of time, and domain properties are described in terms of interval-ordering constraints [1,7,13,24,29]. This form of representation has the benefit of allowing the state of the world *during* an event to be modeled explicitly and reasoned about. Event intervals can be temporally related in all possible ways (for example, they can be interleaved or simultaneous). Most interval-based models also explicitly utilize relationships between events (although sometimes only relationships between types or classes of events are allowed). However, events themselves are not considered the primary objects of the world model.

Unfortunately, many of these interval-based approaches do not capture the semantics of concurrent activity in a clear manner. For example, merely equating an event type with a set of state intervals (typically, the intervals which represent successful event occurrences) gives no clue as to how events are *achieved* or *composed* – e.g., what can or cannot be done by an agent.[3] This hampers reasoning about interactions and relationships between events, as well as about how events may fail. Georgeff's recent paper elaborates this point [10].

---

[2]Recent work by Georgeff [10] has made progress in extending the situation-calculus framework to accommodate simultaneous events. A model-based approach is used, along with a semantic (rather than syntactic) frame rule. However, properties that involve particular relationships between events (such as simultaneity) must still be described in Georgeff's framework by using what is essentially event-based description. Thus, although his model is state based, explicit relationships between events are used.

[3]However, Allen and Koomen [2] do utilize a simple form of event decomposition similar to NOAH operators.

Some interval-based models also do not adequately capture existing or potential relationships between event instances. It is useful to be able to reason about specific causal event-pairs, or the particular events composing a nonatomic event, not just causal and composite relationships between classes of events. For example, in the restaurant scenario, it is important to know precisely which reservation events correspond to which seating events. Otherwise, Felix's seating rules could not be enforced. In addition, most interval-based representations employ a notion of causality that implies eventuality – i.e., if a class of events A has a causal relationship with a class of events B, then, if an event of type A occurs, an event of type B must too. It is therefore difficult to talk about situations in which an event has occurred but has not yet caused (and perhaps never will cause) its corresponding effect. For example, one might view a party's entering a restaurant and making a reservation as causing the party to be subsequently seated. However, such a seating need not necessarily materialize.[4]

Finally, none of these domain representations utilize event location (i.e. structural relationships between events) in any complex way.[5] These kinds of relationships are important if we want to capture those aspects of a domain that are truly affected by locality – for example, forced sequentiality within certain regions, or boundaries of causal effect.

As we shall illustrate, an ontology based on structured, interrelated events has a distinctly different flavor from the interval-based approaches, although it shares with them many of the advantages over more traditional representations. Information about event relationships is captured explicitly. We can easily talk about particular event instances and their various temporal, causal, and simultaneity interrelationships, as well as how particular events constitute a specific nonatomic event. Event intervals and interval relationships can also be utilized. Complex structural relationships among events (i.e. various kind of event locations and groupings of locations) are also represented. The temporal logic underlying our model has a well-understood semantics and has been used extensively in concurrency theory [26,20]. Our use of temporal-logic constraints over *history sequences* (sequences of accumulating records of past behavior) is distinct from most previous approaches (although Stuart uses a similar idea [31]). Because histories include all information about previous events and their interrelationhips, they facilitate use of complex information about the past.

[4]While it is nonstandard, we have found it advantageous to view causality as a phenomenon more akin to *enablement*. To say that class A causes class B means that any event of type B must have been "enabled by" an event of type A. However, an occurrence of an event of type A does not guarantee that it will cause an event of type B. If, however, such a relationship *does* exist, it is perfectly reasonable to say that it is causal. If an eventuality requirement is also desired, it must be stated explicitly.

[5]Of course, many models do associate events with their performing agent. Nonetheless, this is a very limited form of event structure.

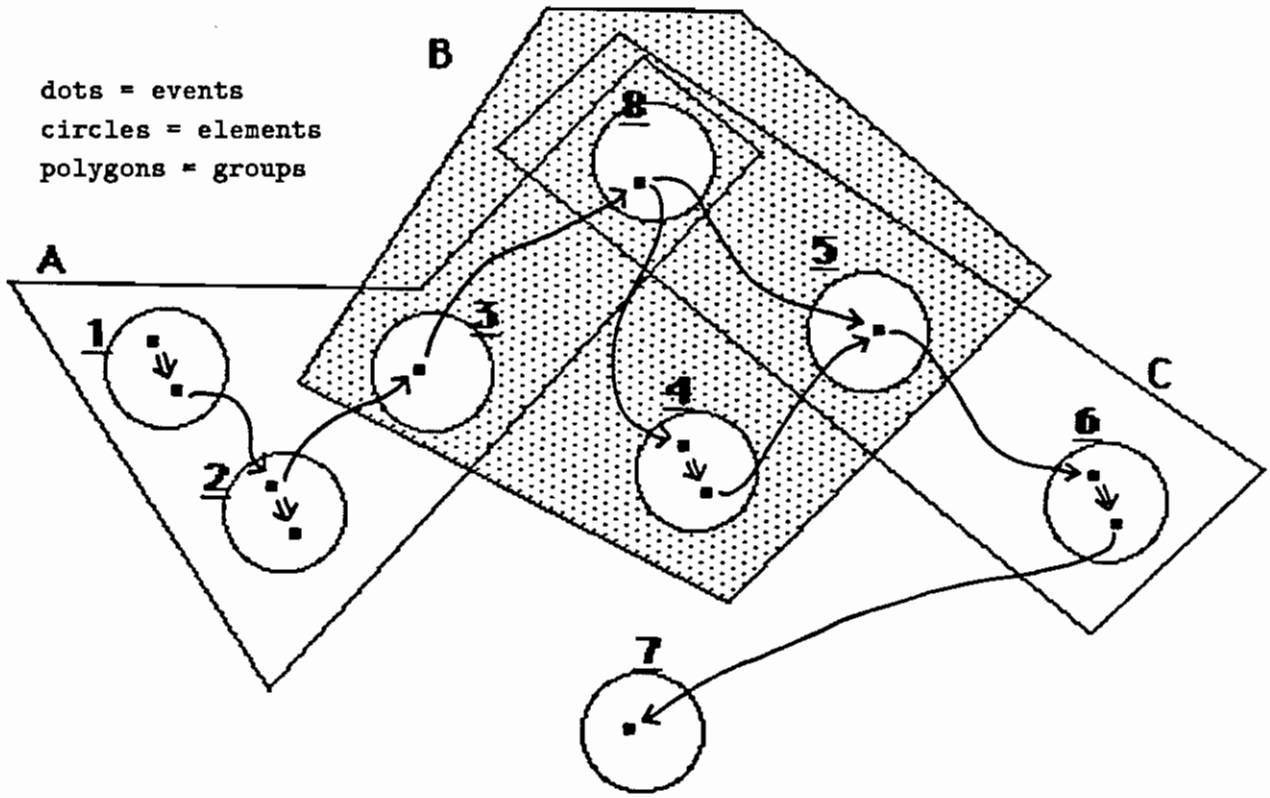


Figure 2: Events, Elements, and Groups

2.3 GEM: An Informal View

Because our approach is somewhat unconventional, it is useful to begin with an informal description of the GEM world model. This discussion will be formalized later.

Our event-oriented representation is based on the view that the world can be modeled as a myriad of interrelated events occurring at locations (see Figure 2). The most basic events are *atomic*; they are not observably decomposable. Nonatomic events are then composed of these atomic events. Three partial relations may hold between events: a causal relation $\rightsquigarrow$, a temporal order $\Longrightarrow$, and a relation that embodies required simultaneity $\rightleftharpoons$. Structural relationships are also used to describe event locations as well as nonatomic event composition.

A useful way of understanding our world model is as a two-tiered structure. The upper tier is based on partially ordered sets of events related by $\rightsquigarrow$, $\Longrightarrow$, and $\rightleftharpoons$ as well as by structural relationships. We call a particular set of interrelated events a *world plan*. (Figure 2 might be viewed as a pictorial representation of a particular world plan.) The lower tier of our world model consists of the set of executions *admitted by* a world plan. It is this lower tier that is usually identified as the "world model" in most state-based representations. Because each

world plan may allow many possibly executions, branching state models are conventionally used to represent this execution-level view of the world. However, we have found it easier to reason about the world primarily in terms of world plans (the upper tier). These structures are definitely a more compact representation than branching state models – in fact, they correspond directly to the usual notion of a "plan."

A world plan also more clearly represents what is actually "knowable" about a domain — i.e., it models the *observable* and *necessary* qualities of a domain. For example, if $\Longrightarrow (e1, e2)$ is true of a world plan, $e1$ must occur before $e2$ in every domain execution. However, if two events are observably unrelated temporally (they are unrelated by $\Longrightarrow$ in the world plan), in some world executions they may occur in a particular order, while in others they may be simultaneous. (In fact, any two events that are unrelated by $\Longrightarrow$ are *potentially* simultaneous.) In contrast, if $\rightleftharpoons (e1, e2)$ is true of a world plan, $e1$ and $e2$ must occur simultaneously in every world execution. The distinction between known relations and the executions admitted by them is especially useful when dealing with parallel domains. For example, people can usually perceive the known temporal order of specific world processes (and thereby can reason easily about them), but find it difficult to know exactly how these processes are interleaved (the actual world executions). World plans are thus a much more intuitive way of viewing the world than are world executions.

Because GEM's causal relation is nonstandard in some ways, it merits some additional clarification. As mentioned earlier, our causal relation $\rightsquigarrow$ is weaker than in other representations (for example, McDermott's [24]) in that it decouples causality from eventuality. A class of events (say *Reservations*) may hold a causal relationship to another class of events (*Seating* events), but the mere occurrence of an event of the first class does not necessarily entail that it *will inevitably* cause an event of second type. Once an event *does* cause another event, however, this relationship is represented explicitly. Of course, if an eventuality requirement is also desired, it can be specified by using a temporal logic constraint (i.e., we could say, "Every reservation must eventually cause a seating").

Our causal relation also implies an ordering in time (if e1 causes e2, it must also occur before e2). Therefore, we distinguish between causality and *identification* of events (for example, the identification of a light-switch-flipping event with the event of turning on the light). However, our use of the simultaneity relation $\rightleftharpoons$ enables modeling of event identification and other forms of required simultaneous activity. As in most representations using causality, GEM's causal relation is irreducible and must be induced from outside the logic.

As mentioned earlier, events in a world plan are also clustered into locations. The most basic type of location is a locus of forced sequential activity; that is, all events belonging to such a location must be totally ordered within the temporal order $\Longrightarrow$. We call these sequential locations *elements* and they are depicted in Figure 2 as circles.

Elements (and the events composing them) may also be grouped into larger regions of activity. We call these locations *groups*, depicted in Figure 2 as polygonal areas. When forms of activity logically occur at the same "location," but are not necessarily sequential, it is natural to model the location as a group consisting of many elements. For example, one might model a robot arm as a group consisting of many sequential elements. As illustrated in Figure 2, groups can be composed hierarchically or overlap.

Group structure is used by GEM to impose constraints on the causal relationships among events. In essence, the group boundaries may be considered just that – boundaries of causal access. Thus, groups A and C in Figure 2 might each be used to represent locations of activity within a robot (perhaps different segments of an arm) that can be causally related only through activity in group B (a joint). In contrast, the activity at element 7 is accessible to all locations (can be caused by activity at all locations – i.e., it is *global*). The activity at element 8 is not only accessible to activity within groups A, B, and C, but, because it is part of groups A, B, and C, it can affect or cause activity at all locations (it could perhaps represent the robot's brain). A group may also be associated with *ports* or access holes that serve as causal interfaces between the group and the events outside it.

The "state" of the world at any particular moment is modeled in GEM as the sum of past activity that has occurred. States embody not only what is true at particular moments, but also what has occurred in the past; we therefore call them *histories* or *pasts*. This view of state as an event history actually conforms quite naturally to what humans know about the world. If we allow events that model *observations*, then what can possibly be known at any particular moment (i.e. the known state of the world) will be derivable from the events that have occurred (i.e., past observations and actions). If we further categorize past activity into those sets of events occurring at, or observable from, particular sets of locations – say, those associated with the particular group modeling a robot – we can then model the "beliefs" of that robot as its localized view of past activity.

GEM's entire representational capability is built upon the basic framework we have just described. A GEM *specification* describes a particular domain by delineating the kinds of events and event structures found in the domain and then imposing constraints on the set of possible world plans (and therefore on the set of possible world executions). In the GEM specification language, constraints on actions and their interrelationships are formulated in first-order temporal logic over sequences of histories (world executions). These constraints may be scoped or applied within locations of activity. All of these capabilities are formalized and illustrated in the next two sections.

3 Domain Model

As stated in the preceding section, the GEM model may be viewed as two-tiered: the upper tier models the world in terms of *world plans* (sets of interrelated events, elements, and groups); the lower tier models the actual physical executions allowed by world plans. Each world plan is composed of a set of unique objects, called *events*, that are related by a temporal ordering $\Longrightarrow$, a causal relation $\rightsquigarrow$, and a simultaneity relation $\rightleftharpoons$. Events are also grouped into *elements* which may further belong to *groups* (groups may also belong to surrounding groups).

$W = < E, EL, G, \Longrightarrow, \rightsquigarrow, \rightleftharpoons, \varepsilon >$

- $E =$ A set of event objects
- $EL =$ A set of element objects
- $G =$ A set of group objects
- $\Longrightarrow$: $(E \times E)$ The temporal ordering
- $\rightsquigarrow$: $(E \times E)$ The causal relation
- $\rightleftharpoons$: $(E \times E)$ The simultaneity relation
- $\varepsilon : (E \times (EL \cup G)) \cup ((EL \cup G) \times G)$ A subset relation between events and elements or groups in which they are contained, as well as between elements and groups and the surrounding groups in which they are contained.

For now we assume that all events are atomic. Thus, each event in a world plan models an atomic event that has occurred in the world domain, each relation or ordering relationship models an actual relationship between domain events, and each element or group models a logical location of activity. In Section 5 we shall extend this basic model to accommodate nonatomic events. Note that our assumption of event atomicity does not imply that events are totally ordered; they *may* happen simultaneously. From an intuitive standpoint, it might be useful for the reader to view each atomic event as the endpoint of some logical world action.

Every event in a world plan must be distinct; it may be viewed as a unique token. Events may be parameterized and may also be organized into types, each of which represents some class of world events. For example, *Paint(Object, Color)* could represent the class of world events, each of which paints an object a certain color. A specific instance of this type would be *paint(ladder,red)*. Lowercase tokens are used to denote specific event instances, while uppercase is used for event classes or types. A similar convention is used for parameter values and types, as well as for group and element instances and types.

As described earlier, events are related by three kinds of partial relationships, $\Longrightarrow$, $\rightsquigarrow$, and $\rightleftharpoons$. The temporal order $\Longrightarrow$ is an irreflexive, antisymmetric, transitive relationship that models event ordering in time.[6] The causal relation $\rightsquigarrow$ is irreflexive and antisymmetric, but *not* transitive – it represents "direct" causality between events. Every domain is, by default, associated with a constraint that requires causally related events to also be temporally related ($\rightsquigarrow$(e1,e2) $\supset \Longrightarrow$(e1,e2)), but the reverse is not true; just because two events may be forced to occur in some sequence does not mean that they are causally related. Finally, the simultaneity relation $\rightleftharpoons$ is reflexive, symmetric, and transitive, and models a *necessary* simultaneous occurrence of events.

In addition to being ordered, events are also clustered into *elements*. These elements (as well as other groups) are further clustered into *groups*. The events considered part of a group are precisely those belonging to the group's constituent elements and groups. The structure of a domain is conveyed by the set membership relation ε.

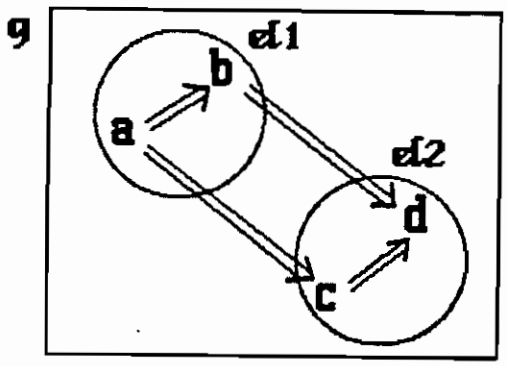
Particular domains are represented in GEM by domain specifications. Each specification will allow a set of world plans and the world executions conforming to those world plans. A specification is built by delineating locations of activity (elements and groups), stating the types of events that may occur at these locations, and, finally, imposing constraints on the possible relationships among those events. Some of these constraints are domain-specific (for example, the constraints that describe the seating rules of the restaurant domain); others apply to all domains (e.g., the total ordering constraint on events belonging to the same element). The basic form of a constraint is a first-order temporal-logic formula that is applied to the possible executions of a given world plan. The next section describes a semantics for such constraints.

3.1 World Executions: Histories and History Sequences

A world plan, as we have described it, is actually a structure that captures or represents many potential executions in the domain being modeled. For example, consider the world plan in Figure 3. It can actually be executed in three ways:

Execution 1:	1st a	2nd b	3rd c	4th d
Execution 2:	1st a	2nd c	3rd b	4th d
Execution 3:	1st a	2nd b,c	3rd d	

[6]Note that we make no use of explicit time values. In multiagent domains, it is actually disadvantageous to rely on such values for purposes of synchronizaton; the use of a partial temporal ordering is a much safer avenue for assessing the relative occurrences of events. However, actual time can be incorporated into GEM by associating each event with a time parameter and requiring that the temporal ordering conform to these event time stamps: $(\forall\, e1(t1),\, e2(t2))\, [\, t1 < t2 \supset \Longrightarrow (e1(t1), e2(t2))\,]$.



$\Longrightarrow(a,b)$ $\Longrightarrow(a,c)$ $\Longrightarrow(b,d)$ $\Longrightarrow(c,d)$
$\varepsilon(a,el1)$ $\varepsilon(b,el1)$ $\varepsilon(c,el2)$ $\varepsilon(d,el2)$
$\varepsilon(el1,g)$ $\varepsilon(el2,g)$

Figure 3: A World Plan

Note that, in the third execution, b and c occur simultaneously. Although we know that *one* of these world executions may occur, we cannot assume any one of them actually does.

The possible executions of a world plan may be viewed as linear sequences of " states," where each state is a record of past activity. We call such a state a *history* or *past*.[7] Each *history* α of a world plan W is simply a set of partially ordered events that is a *prefix* of that world plan; it therefore may be described in the same way as a world plan, i.e., for history α, we could use $< E_\alpha, EL_\alpha, G_\alpha, \Longrightarrow_\alpha, \rightsquigarrow_\alpha, \rightleftharpoons_\alpha, \varepsilon_\alpha >$, where $E_\alpha \subset E$, $EL_\alpha \subset EL$, $G_\alpha \subset G$, $\Longrightarrow_\alpha$ is a subrelation of $\Longrightarrow$, etc. Each history represents a possible point in an execution of the world plan, plus everything that has happened until then. Essentially, it is a record of past activity up to some moment in time.

For the world plan in Figure 3, there are six possible histories or pasts, consisting of the following sets of events (as well as their interrelationships):
α_0 :{} α_i :{a} α_j :{a,b} α_k :{a,c} α_m :{a,b,c} α_n :{a,b,c,d}
For instance, state α_j describes the state in which events a and b have occurred, and in which, moreover, the relations $\Longrightarrow_{\alpha_j}(a,b)$, $\varepsilon_{\alpha_j}(a,el1)$, $\varepsilon_{\alpha_j}(b,el1)$, and $\varepsilon_{\alpha_j}(el1,g)$ all hold.

Given this notion of history ("state"), the possible world executions permitted by a world plan may be described as sequences of histories – which we shall call *valid history sequences* (VHS). For each VHS, every history in the sequence (except the first) must be a superset of

[7]The reader should be warned that the term *history* has been used by others in different ways – for example, for a particular sequence of states. Here the term refers to a snapshot of the past – i.e. a record of the events that have occurred and their interrelationships.

its predecessor. Moreover, two events may enter a given VHS in the same history only if it is possible for them to have occurred simultaneously, i.e., if there is no explicit temporal ordering relationship between them. For example, if $\Longrightarrow$(e1,e2), then e1 and e2 would have to enter a VHS in distinct histories. By the same token, if two events *must* take place simultaneously (e.g., $\rightleftharpoons (e1, e2)$), they must always enter a given VHS in the same history. A history sequence is said to be *complete* if it starts with the empty history.

For the world plan in Figure 3, there are three possible complete VHSs — S1, S2, and S3 — corresponding to the three possible executions of the world plan given earlier:

S1: α_0 α_i α_j α_m α_n $\equiv$ 1st a 2nd b 3rd c 4th d
S2: α_0 α_i α_k α_m α_n $\equiv$ 1st a 2nd c 3rd b 4th d
S3: α_0 α_i α_m α_n $\equiv$ 1st a 2nd b,c 3rd d

Note that one way of representing the possible history sequences of a world plan might be as a branching tree. For example, we would have

$$\begin{array}{ccccccccc} & & & \nearrow & \alpha_j & \rightarrow & \alpha_m & \rightarrow & \alpha_n \\ \alpha_0 & \rightarrow & \alpha_i & \rightarrow & \alpha_k & \rightarrow & \alpha_m & \rightarrow & \alpha_n \\ & & & \searrow & \alpha_m & \rightarrow & \alpha_n & & \end{array}$$

This corresponds to the branching tree of states used by McDermott; a chronicle corresponds to a VHS [24]. However, by representing this tree as a *world plan* (i.e., the form depicted in Figure 3), information about possible world executions and observable relationships between events is conveyed in a more compact form.

3.2 Constraint Semantics

Now that we have valid history sequences, we have a framework for defining the semantics of formulas in first-order linear temporal logic. First, we consider simple nontemporal first-order formulas Q. Each such formula is composed in the standard fashion, with the usual quantifiers and connectives ($\wedge, \vee, \neg, \supset, \Longleftrightarrow, \forall, \exists, \exists!$).[8] Event, element, and group instances, as well as event-parameter values, are used as constants, over which range event, element, group, and event-parameter variables. The predicates that may be used in these formulas are *occurred*, the infix predicates $\rightsquigarrow$, $\Longrightarrow$, $\rightleftharpoons$, ε, and equality, as well as arithmetic comparison of parameter values. The interpretation of formulas is standard. Given a history α described by $< E_\alpha, EL_\alpha, G_\alpha, \Longrightarrow_\alpha, \rightsquigarrow_\alpha, \rightleftharpoons_\alpha, \varepsilon_\alpha >$, we have

$$\begin{array}{lcl} \alpha \models occurred(e) & \equiv & e \epsilon E_\alpha \\ \alpha \models e1 \Longrightarrow e2 & \equiv & \Longrightarrow_\alpha (e1, e2) \\ \alpha \models e1 \rightsquigarrow e2 & \equiv & \rightsquigarrow_\alpha (e1, e2) \\ \alpha \models e1 \rightleftharpoons e2 & \equiv & \rightleftharpoons_\alpha (e1, e2) \\ \alpha \models x \; \varepsilon \; y & \equiv & \varepsilon_\alpha (x, y) \end{array}$$

[8] The quantifier $\exists!$ denotes existence of a unique object.

A typical nontemporal first-order formula is the following: (∀ e:E)[occurred(e) ⊃ (∃ f:F)[f⇝e]]. This might be read as follows: "Every event of type E that has occurred must have been caused by an event of type F." Free variables in a formula are considered, by default, to be universally quantified.

Linear temporal operators are modal operators that apply formulas to sequences.[9] The most common temporal operators used are □ (*henceforth*), ◇ (*eventually*), ○ (*next*), and ∪ (*weak until*). In most temporal logics they apply formulas to sequences of states [26]. GEM follows traditional formulations of linear temporal logic, but applies the modal operators to sequences of histories (i.e., to VHSs). Given a valid history sequence of the form $S = \alpha_0, \alpha_1, ...$, we use the notation $S[i]$ to denote the i^{th} tail sequence of S – i.e., $\alpha_i, \alpha_{i+1}, ...$ Note that $S = S[0]$. Also notice that every tail sequence of a VHS is also a VHS.

We then define the semantics of temporal formulas as follows:

$$
\begin{array}{lll}
\textit{Henceforth } P: & S[i] \models \Box P & \equiv (\forall j \geq i)\, S[j] \models P \\
\textit{Eventually } P: & S[i] \models \Diamond P & \equiv (\exists j \geq i)\, S[j] \models P \\
\textit{Next } P: & S[i] \models \bigcirc P & \equiv S[i+1] \models P \\
P \textit{ Until } Q: & S[i] \models P \cup Q & \equiv (\forall j \geq i)\, S[j] \models P \vee \\
 & & \quad (\exists j \geq i)[\, S[j] \models Q \wedge (\forall k, i \leq k < j)\, S[k] \models P \,]
\end{array}
$$

A nontemporal formula Q is true of $S[i]$ if it is true of α_i: $S[i] \models Q \equiv \alpha_i \models Q$.

To enhance the specification of properties dealing with past activity, we also introduce the backwards temporal operators △ (*before*), $\overleftarrow{\Box}$ (*until now*), $P \overleftarrow{Q}$ (*Q back to P*), and *INIT* (*initially*). Although somewhat nonstandard, they have been used elsewhere [3].

$$
\begin{array}{lll}
S[i] \models \triangle P & \equiv & S[i-1] \models P \\
S[i] \models \overleftarrow{\Box}\, P & \equiv & (\forall k, 0 \leq k \leq i)\, S[k] \models P \\
S[i] \models P \overleftarrow{Q} & \equiv & S[i] \models \overleftarrow{\Box}\, Q \vee \\
 & & \quad (\exists j, 0 \leq j \leq i)[\, S[j] \models P \wedge (\forall k, j < k \leq i)\, S[k] \models Q \,]
\end{array}
$$

We define *INIT P* to be $\overleftarrow{\Box}$ $[(\neg(\exists e) occurred(e)) \supset P]$. In other words, P is true of the empty history. At the beginning of a VHS, $S[0] \models \triangle P$ is *false* for all P.

Because we shall be creating structured specifications in which constraints are imposed on limited contexts, it is also useful to define *localized* or scoped versions of the temporal operators. For example, what happens *next* in a particular context may be different from what happens next in the domain as a whole.

[9] This is in contrast to branching-time temporal logics, which regard time as a branching tree. In these logics, the modalities can vary according to how they are applied to the various paths through that tree. We have found linear temporal logic to be adequate and simpler to use.

Suppose we have a VHS of form $\alpha_0....\alpha_n$. Let us assume that *context* is a set of events. We then define $\alpha_0....\alpha_n|_{context}$ to be the history sequence remaining after histories that satisfy the following formula have been removed:

$$occurred(e) \wedge \triangle\neg occurred(e) \supset \neg e \in context.$$

In other words, we eliminate from $\alpha_0....\alpha_n$ all histories that are formed solely through the addition of events outside *context*.

Given any valid history sequence S, we then define the scoped temporal operators as follows:

$$\mathsf{S} \models \Box_{context}\ \mathsf{P} \equiv \mathsf{S}|_{context} \models \Box\ \mathsf{P}$$
$$\mathsf{S} \models \Diamond_{context}\ \mathsf{P} \equiv \mathsf{S}|_{context} \models \Diamond\ \mathsf{P}$$
$$\mathsf{S} \models \bigcirc_{context}\ \mathsf{P} \equiv \mathsf{S}|_{context} \models \bigcirc\ \mathsf{P}$$

and similarly for other temporal operators.[10]

Finally, first-order temporal logic formulas may be applied to a GEM *world plan* by viewing a world plan W as the set of all its complete valid history sequences. A world plan satisfies a constraint if and only if all tail sequences of its complete valid history sequences satisfy that constraint

$$W \models P \equiv (\forall\ \text{complete VHS}\ \ S\ \ \text{of}\ \ W)(\forall i \geq 0)S[i] \models P\ .$$

For example, the world plan in Figure 3 satisfies $\Box(occurred(d) \supset b \Longrightarrow d \wedge c \Longrightarrow d)$, but not $\Box(occurred(c) \supset occurred(b))$ (VHS S2 does not satisfy it).[11]

[10] The use of contexts may also be convenient for describing scoped or localized forms of state. For example, a state of the world α relative to a particular context would be a state α', where all noncontextual events have been removed. If a particular context corresponds to the events that are part of, or visible to, a particular agent, then the scoped state with respect to that agent would correspond to the agent's perspective upon, or beliefs about, the past.

[11] The temporal operator $\Box$ can actually be removed from both of these constraints, because they apply to *all* tails of complete VHSs of a world plan.

4 Domain Specifications

In the next three sections, we demonstrate how GEM domain specifications are built by actually constructing a description of the restaurant scenario presented earlier. We begin with an overview of the general structure of the GEM specification language and describe how typical kinds of constraints are formed. In Sections 5 and 6 we address such issues as the specification of nonatomic actions, state description, and the frame problem. A complete specification of the restaurant scenario is given in the appendix.

4.1 Specification Structure

The GEM specification language is a set of notational conventions for writing constraints on world plans. Viewed semantically, a specification σ is equivalent to (can be expanded into) a set of first-order temporal-logic formulas over valid history sequences. Each specification defines a class of world plans by stating explicitly: (1) what types of events may occur; (2) how those events must be clustered into elements and how elements and groups are clustered into groups; and (3) what constraints exist on relationships between events and their parameter values.

Just as elements and groups model the structural aspects of a domain, they also serve as the structural components of our specification language. Each specification σ consists of a set of element and group declarations, along with a set of explicit constraints on the events that belong to those elements and groups. Each element is associated with a set of event types, and each group is composed of a set of elements and other subgroups. The events belonging to an element may be only of the designated types associated with it. The events belonging to a group are taken to be those belonging to the group's elements and subgroups. Constraints are "scoped" within the context in which they are declared; i.e., they are imposed only on those events that belong to the element or group with which they are associated. However, the temporal operators are scoped with respect to a context only if scoped temporal operators are used. Thus, if we write $\bigcirc P$, then P must be true of the next state in the *entire world execution*, not just the next state in which an event in the particular element or group occurs.

GEM also includes a mechanism for describing element and group *types*. These may be parameterized and are definable as refinements of other previously defined types. Each instance of a defined type is a unique element or group with a structure identical to that of its type description. From a semantic standpoint, the use of types and instances may be viewed as a simple text substitution facility; each type instance is shorthand for a separate but identical element or group declaration.

For example, we might describe the class of restaurant tables as follows:[12]

```
RestaurantTable (size:INTEGER) = ELEMENT TYPE
EVENTS
  Occupy(p:Party)
  Vacate(p:Party)
CONSTRAINTS
  :
END RestaurantTable
```

A declaration of the form

```
table[1..5] = RestaurantTable(10) ELEMENT
```

would declare table[1]...table[5] to be tables of size 10. The notation table[1].size yields table[1]'s size value (in this case, 10). table[1].Occupy and table[1].Vacate refer to the class of Occupy and Vacate events belonging to table[1], respectively. The notation table[1].occupy(p) denotes a particular Occupy event instance.[13]

The structure laid out by a set of group and element declarations creates a framework associated with implicit (default) constraints. Domain-specific constraints are then added on top of this framework. Default constraints include the following (we give only an informal description of these constraints here; for a more formal description, see [19]):

- The only events, elements, and groups allowed within a valid world plan are those delineated by the specification. We are essentially minimizing the potential structure of world plans with respect to the domain specification. Events must be clustered into elements and element/group structures must be formed as described in the specification.
- All events belonging to the same element must be totally ordered temporally:
  $(\forall$ e1,e2,elem) [e1εelem $\wedge$ e2εelem $\wedge$ e1$\neq$e2 $\supset$ e1 $\Longrightarrow$ e2 $\vee$ e2 $\Longrightarrow$ e1].
  For instance, we might represent the restaurant lobby as follows:

[12]This description models only two types of events that can take place at a table. Of course, if we wish a table to be associated with broader forms of activity, it could be modeled as an element with more event types or, alternatively, as a group consisting of many elements. For example, to represent the simultaneous lifting of both sides of a table, each side of a table could be modeled as an element associated with "lifting" events. We could also model these events as being performed by the agents that do the lifting. We could even do both (have lifting events at the table *and* the lifting agents) and identify the two. This form of event identification is illustrated in Section 4.4.

[13]A more typical event notation might be occupy(table[1],p). However, we have found dot notation to be very useful for denoting events that occur at particular elements or groups.

```
lobby = ELEMENT
EVENTS
  Enter(f:Friend)
END lobby
```

While lobby is not associated with any explicit constraint, its events must still be totally ordered – i.e., people may enter the lobby only one at a time.

- As stated earlier, we use groups as a way of representing limitations of causal effect. Essentially, the "walls" of a group form a boundary through which causal effect may probe outward, but not inward.[14] The one exception to this rule is the use of *ports*: "holes" in the group boundary. If an event is a port for a group g, that event can be affected by other events outside g. Let us assume that the atomic formula $port(e, g)$ is true for every event e that is a port of group g. The formal constraints on the causal relation imposed by group structure may be described as follows.

  Suppose that $e1 \epsilon el1$ and $e2 \epsilon el2$. Then $e1$ may cause $e2$ ($e1 \rightsquigarrow e2$) only if:

  $$access(el1, el2) \vee [port(e2, g) \wedge access(el1, g)].$$

  We define $access(x, y)$ to be true if either (1) x and y belong to the same group or (2) there is some surrounding group g' such that y belongs to g' and x is contained within g' – i.e., y is "global" to x. We say that an element el or group g *belongs* to a group g' if it is explicitly declared as one of the components of group g'. We say that el (or g) is *contained* within g' if there is some hierarchical scoping of groups $g1...gn$ such that el belongs to $g1$, $g1$ belongs to $g2$, ... and gn belongs to g'. (By convention, we assume that all elements and groups modeling the world are contained within a single surrounding group.)

  For example, the specification structure for the restaurant domain is shown in Figure 4 (only one party with one friend, one taxi, and one restaurant table are depicted). Notice how the friend has access to the taxi, the reservation desk, the restaurant lobby, and also to the Vacate actions at the table (an asterisk marks port event types). However, a friend cannot directly affect Felix's personal observations, nor can she directly occupy a table (guests must be seated by Felix).

We now define some of the domain-specific constraints for the restaurant domain. Because many constraint forms arise repeatedly, it is useful to have abbreviations for them. We shall take some liberties in devising these abbreviations, appealing to the reader's intuition. However, all of these constraints are more rigorously defined elsewhere [19].

[14] This is much like the notion of scope in programming languages, except that groups may overlap as well as form hierarchies.

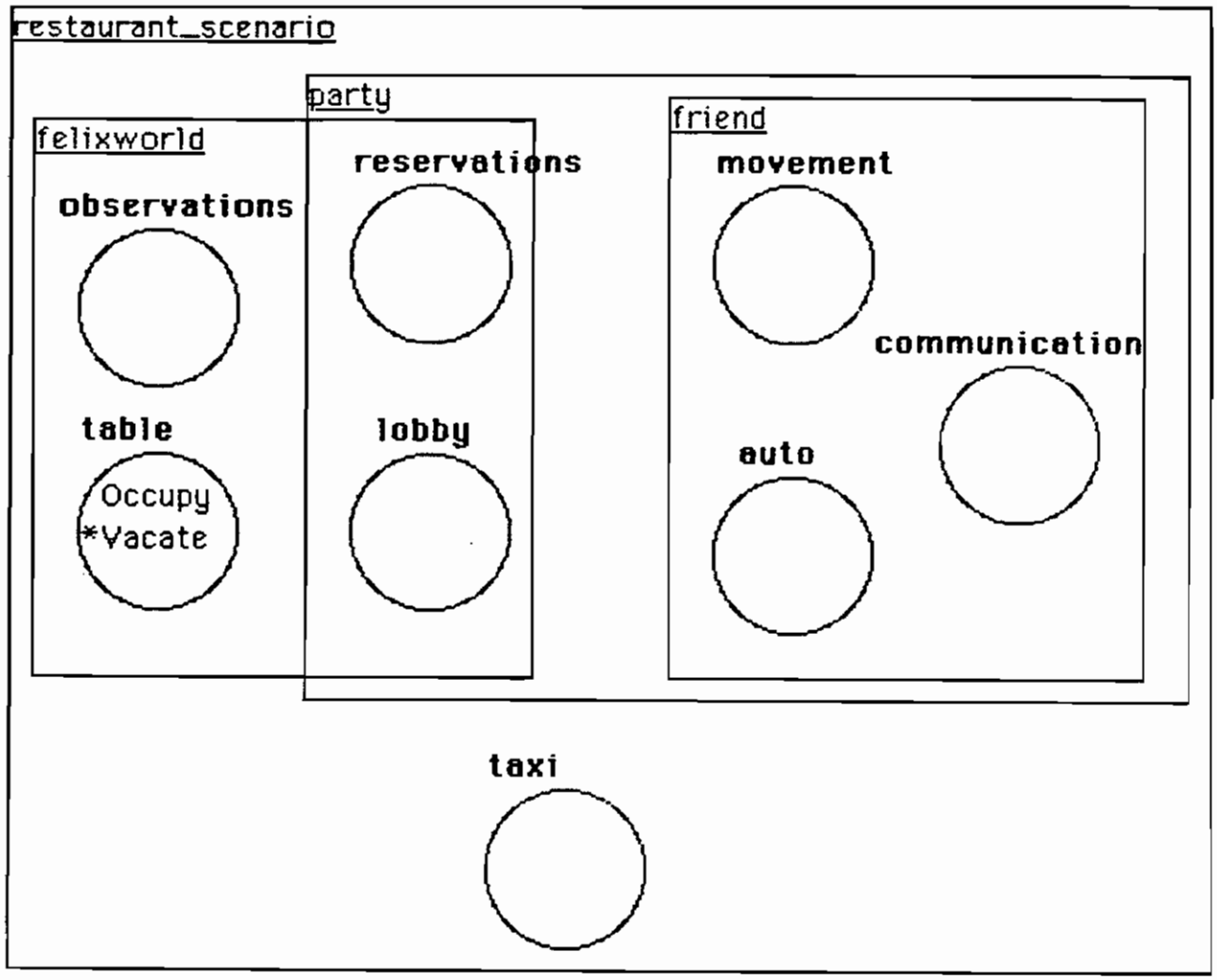


Figure 4: Restaurant Domain Structure

4.2 Prerequisite Constraints

Probably the most common kind of constraint is the *strong prerequisite*, denoted $E1 \longrightarrow E2$. This constraint requires that each event of type $E2$ be caused by exactly one event of type $E1$, and that each event of type $E1$ can cause at most one event of type $E2$. In essence, this is a one-to-one causal requirement. The definition of this constraint is as follows:

$$
\begin{aligned}
&E1 \longrightarrow E2 \equiv \\
&(\forall e2 : E2)(\exists!\, e1 : E1)[e1 \rightsquigarrow e2]] \wedge \\
&(\forall e1 : E1)(\exists \textit{ at most one } e2 : E2)[e1 \rightsquigarrow e2]
\end{aligned}
$$

In Section 4.3 we use a strong-prerequisite constraint to describe the one-to-one causal relationship between reservation and seating events: Reserve $\longrightarrow$ Seat. We can also use it to define the constraints of the RestaurantTable element type. Constraint 1 uses a regular-expression notation as shorthand for a more complicated pattern of strong-prerequisite constraints.[15] Each Vacate(p) event must have a one-to-one causal relationship with a preceding Occupy(p) event, and each Occupy event (except the first) must have a one-to-one causal relationship with a preceding Vacate event. This restricts the events at a table to be of the form occupy(p1)$\rightsquigarrow$vacate(p1)$\rightsquigarrow$occupy(p2)$\rightsquigarrow$vacate(p2)$\rightsquigarrow$.... Namely, a table is a resource that can be used by only one party at a time.[16]

```
RestaurantTable (size:INTEGER) = ELEMENT TYPE
EVENTS
  Occupy(p:Party)
  Vacate(p:Party)
CONSTRAINTS
1) ( Occupy(p) ⟶ Vacate(p) )*⟶
END RestaurantTable
```

Taxis and cars are other resources that are described in a similar fashion. In the following we use the element type hierarchy to first describe a Vehicle element type, and then further define taxis and personal automobiles as Vehicles.

---

[15] $(x)* \longrightarrow$ denotes zero or more repetitions of x separated by $\longrightarrow$. A formal semantics for prerequisite expressions is given in my dissertation [19].

[16] Note that, although a party may "occupy" a table only once before it vacates, this does not preclude members of a party from leaving the table in the interim. As we shall see later, while some of the seating actions of party members are identified with "occupying" the table, not all are. Likewise not all departures from the table are identified with the party's actually vacating the table.

Vehicle = ELEMENT TYPE
EVENTS
  Occupy
  Drive(loc1,loc2:Location)
  Vacate
CONSTRAINTS
1) (Occupy $\longrightarrow$ Drive(loc1,loc2) $\longrightarrow$ Vacate)$^{*\longrightarrow}$
END Vehicle

Taxi = Vehicle ELEMENT TYPE
Auto = Vehicle ELEMENT TYPE

Another useful form of prerequisite constraint combines strong prerequisite constraints to model activity that forks or splits from an event, or joins into an event:

$$\begin{array}{lrcl} EventFork: & E \longrightarrow \{E1,...En\} & \equiv & (\forall i, 1 \leq i \leq n)\ E \longrightarrow Ei \\ EventJoin: & \{E1,...En\} \longrightarrow E & \equiv & (\forall i, 1 \leq i \leq n)\ Ei \longrightarrow E \end{array}$$

Another common form of prerequisite relationship between events is the nondeterministic prerequisite $\{E1...En\} \longrightarrow +E$, defined as follows:

$$\begin{array}{l} \{E1,...,En\} \longrightarrow +E \equiv \\ (\forall e : E)(\exists! ei : \{E1,...,En\})[ei \rightsquigarrow e] \wedge \\ (\forall ei : \{E1,...,En\})(\exists \textit{ at most one } e : E)[ei \rightsquigarrow e] \end{array}$$

In other words, each event of type E must be caused by exactly one event of type $E1$ or $E2$ or... En, and an event of type $E1$ or... En can cause at most one event of type E. This is useful for specifying situations in which exactly one (undetermined) member of a set of possible event types can cause another event.

Another form of nondeterministic behavior is the nondeterministic fork:

$$\begin{array}{l} E \longrightarrow +\{E1,...,En\} \equiv \\ (\forall ei : \{E1,...,En\})(\exists!\ e : E)[e \rightsquigarrow ei] \wedge \\ (\forall e : E)(\exists \textit{ at most one } ei : \{E1,...En\})[e \rightsquigarrow ei] \end{array}$$

In other words, all events of type $E1$ or ... En must be caused by an event of type E, but each event of type E can cause only one such event.[17]

[17]It is interesting to note that that both the nondeterministic prerequisite and the nondeterministc fork can be described using the strong prerequisite if we combine an event class set into a single event class. For example, if we let $F = E1 \cup ... \cup En$, then $\{E1,...En\} \longrightarrow +E \equiv F \longrightarrow E$ and $E \longrightarrow +\{E1,...En\} \equiv E \longrightarrow F$.

4.3 Priority, Mutual Exclusion, Eventuality

To specify Felix's seating rules, we need a way of describing synchronization properties among events. This is easily accomplished with temporal formulas. First we define the abbreviation e1 cbefore E2 (e1 is causally before the class of events E2) as follows:

$$e1\ cbefore\ E2 \equiv occurred(e1) \wedge \neg(\exists e2 : E2)[e1 \rightsquigarrow e2]$$

This may be read, "$e1$ has occurred but has not yet caused an event of type $E2$."

We can now express priority and mutual-exclusion properties by using the following kinds of constraints:

- *Priority of causal transitions from e1 to events of type E2 over those from e3 to events of type E4*:

  $$(e1\ cbefore\ E2 \wedge e3\ cbefore\ E4) \supset \Box\,[(\exists e4 : E4)e3 \rightsquigarrow e4 \supset (\exists e2 : E2)e1 \rightsquigarrow e2]$$

  In other words, if $e1$ is pending at $E2$ *and* $e3$ is pending at $E4$ at the same time, then, from that moment on, if $e3$ actually does cause an event $e4$, $e1$ must have already caused an event $e2$ ($e1$ must cause its corresponding event before $e3$ does).

- *Mutual exclusion between intervals in which e1 and e3 are causally pending*:

  $$\neg(e1\ cbefore\ E2 \wedge e3\ cbefore\ E4)$$

If we define *tbefore* as follows:

$$e1\ tbefore\ E2 \equiv occurred(e1) \wedge \neg(\exists e2 : E2)[e1 \Longrightarrow e2]$$

then temporal forms of mutual exclusion and priority can be expressed as well – e.g., constraints of the form

$$\neg(e1\ tbefore\ E2 \wedge e3\ tbefore\ E4).$$

Going back to the restaurant scenario, we have the following description of Felix's reservation desk:

reservations = ELEMENT
EVENTS
  Reserve(p:Party, b:Bribe)
  Seat(p:Party, t:RestaurantTable)
CONSTRAINTS
1) To be seated, a party must have a reservation. Moreover, each reservation is good for only one seating.
  Reserve(p,b) $\longrightarrow$ Seat(p,t)

2) Parties can be seated only at tables of the right size.
  occur(seat) $\supset$ seat.p.size = seat.t.size

3) A bigger bribe will get you seated faster.
  reserve1 cbefore Seat $\wedge$ reserve2 cbefore Seat $\wedge$ reserve1.b > reserve2.b $\supset$
  $\Box$ [($\exists$ seat2:Seat) reserve2$\rightsquigarrow$seat2 $\supset$ ($\exists$ seat1:Seat) reserve1$\rightsquigarrow$seat1]

4) All other things being equal, seating is first-come-first-served.
  reserve1(p1,b1)$\Longrightarrow$reserve2(p2,b2) $\wedge$ b1=b2 $\wedge$
  present(p1) $\wedge$ present(p2) $\supset$
  $\Box$ [($\exists$ seat2:Seat) reserve2$\rightsquigarrow$seat2 $\supset$ ($\exists$ seat1:Seat) reserve1$\rightsquigarrow$seat1]

5) A $50 bribe will definitely get you seated.
  reserve.b $\geq$ $50 $\supset$ $\Diamond$ ($\exists$ seat:Seat) reserve$\rightsquigarrow$seat
END reservations

Note how the temporal operator $\Diamond$ (eventually) is used to describe the rule for eventual seating, given a $50 bribe. The state description present(p) represents states in which all members of party p are present in the lobby. It is defined to be true precisely at the time all the friends in party p have arrived in the restaurant lobby but have not yet been seated. The definition given below thus illustrates the use of event-based formulas to define state predicates (see Section 6).

present(p) $\equiv$ ($\forall$ f:Friend, fεp) ($\exists$ enter(f):Lobby.Enter)
  enter(f) cbefore Reservations.Seat

4.4 Simultaneity

One way of establishing relationships between the private events of restaurant guests and those of the restaurant's logical components is to form identifications between them. For example, we can identify a reservation event performed by one of the friends in a party with the reservation event at the restaurant desk. Similarly, we require that Felix's seating of a party coincides with a sitting action by each member of the party. These identifications will

force all restaurant guests to comply with Felix's seating rules. Event identification (as well as other forms of required simultaneity) is accomplished by using the simultaneity relation $\rightleftharpoons$. In particular, we use the following kinds of constraints:

$$E1 \sim E2 \equiv (\forall e1 : E1)(\exists e2 : E2)[e1 \rightleftharpoons e2]$$
$$E1 \approx E2 \equiv (\forall e1 : E1)(\exists e2 : E2)[e1 \rightleftharpoons e2] \wedge (\forall e2 : E2)(\exists e1 : E1)[e1 \rightleftharpoons e2]$$

Note that $E1 \approx E2$ is equivalent to $E1 \sim E2 \wedge E2 \sim E1$. The constraint $E1 \sim E2$ identifies all events of type $E1$ with events of type $E2$, but not vice versa. For example, all of Felix's Seat events must be identified with Sit actions by members of a party, but not all Sit actions by a particular person need be identified with seating by Felix.

We begin with a preliminary description of a friend. Each friend is made up of movement events, communication events, and use of a personal automobile. Note that events at each of the elements composing a friend must be sequential, but events occurring at different elements may be simultaneous (as long as they conform to the domain constraints). Thus, people can potentially communicate and move at the same time.

```
Friend = GROUP TYPE (m:Movement, c:Communication, a:Auto)
Movement = ELEMENT TYPE
EVENTS
  Ride(loc1,loc2:Location)
  Walk(loc1,loc2:Location)
  Sit(tableloc:Location)
  Eat
END Movement
Communication = ELEMENT TYPE
EVENTS
  Reserve(p:Party, b:Bribe)
  OrderFood(food:Food)
END Communication
```

A party consists of a set of friends, the reservation desk, and the lobby: [18]

[18] The notation f.c.Reserve denotes events of type Reserve occurring at the Communication element c of Friend f. SELF is used to denote the group constant associated with each particular group instance. Thus, when the Party type definition is instantiated, SELF will be replaced by each Party instance's group constant. The function setsize yields the cardinality of a set.

Party(size:INTEGER) = GROUP TYPE ({f}:SET OF Friend,reservations,lobby)
CONSTRAINTS
1) size must be the size of the set of friends.
size = setsize({f})
2) A reservation by a friend is identified with a reservation at the desk.
f.c.Reserve(p,b) $\approx$ reservations.Reserve(p,b)
3) All members of a party must be seated simultaneously.
($\forall$ f' ϵ {f}) reservations.Seat(SELF,t) $\sim$ f'.m.Sit(t)
4) In order to be seated, all members of the party must be present.
($\forall$ f' ϵ {f}) lobby.Enter(f') $\longrightarrow$reservations.Seat(SELF.Table)
5) The first friend to enter the lobby must make a reservation.
($\forall$ f1,f2 ϵ {f}) occurred(lobby.enter1(f1)) $\wedge$
$\neg$($\exists$ lobby.enter2(f2)) [lobby.enter2(f2)$\Longrightarrow$lobby.enter1(f1)] $\supset$
$\Diamond$ ($\exists$ reserve1:f1.c.Reserve(SELF,b)) occurred(reserve1)
END Party

We can also now specify Felix's world, consisting of the reservation desk, the lobby, a set of tables table[1]...table[10] (these are assumed to have been instantiated), as well as Felix's personal observations or thoughts. One sort of observation that Felix may make is whether a table is empty. Later on we shall add an extra constraint that allows Felix to make such observations only if the table is indeed empty. For now, however, we assume that Felix always makes accurate observations. He will not seat a party unless *he* thinks a table is unoccupied. Note that this is different from saying that a seating may take place when the table is free.[19]

observations = ELEMENT
EVENTS
EmptyTable(t)
END observations

felixworld = GROUP (reservations, table[1..10], lobby, observations)
PORTS(table[i].Vacate)
CONSTRAINTS
1) Felix must observe that a table is empty before he can seat a party there.
observations.EmptyTable(table[i]) $\longrightarrow$ reservations.Seat(Party.table[i])
2) Seating a party at a table is the same as occupying the table.
reservations.Seat(p,table[i]) $\approx$ table[i].Occupy(p)
END felixworld

[19] We could have chosen to do this as well; we just wanted to illustrate the use of "observation" events.

5 Nonatomic Events and Hierarchical Description

We now digress from our development of the restaurant domain specification to discuss the use of nonatomic events within the GEM framework. The inclusion of such events is relatively straightforward once it is realized that a nonatomic event can be described by using two or more internal events. In particular, we associate each nonatomic event type E with two atomic event types E' and E'', representing the initiation and termination of E. We also add an additional constraint: $E' \longrightarrow E''$ (there must be a one-to-one causal relationship between the initial event and terminal event for each nonatomic event). A nonatomic event e is *in progress* if the formula e' cbefore E'' is true. Using this notation, we can describe the various possible ordering relationships between two nonatomic events a and b as follows (these are the same interval relationships used by Allen [1]):[20]

$$
\begin{array}{lcl}
\text{a before b} & \equiv & \text{a}'' \Longrightarrow \text{b}' \\
\text{a equal b} & \equiv & \text{a}' \rightleftharpoons \text{b}' \wedge \text{a}'' \rightleftharpoons \text{b}'' \\
\text{a meets b} & \equiv & \text{occursnext(a}'') \supset \bigcirc \text{ occursnext(b}') \\
\text{a overlaps b} & \equiv & \text{a}' \Longrightarrow \text{b}' \wedge \text{a}'' \Longrightarrow \text{b}'' \\
\text{a during b} & \equiv & \text{b}' \Longrightarrow \text{a}' \wedge \text{a}'' \Longrightarrow \text{b}'' \\
\text{a starts b} & \equiv & \text{a}' \rightleftharpoons \text{b}' \wedge \text{a}'' \Longrightarrow \text{b}'' \\
\text{a finishes b} & \equiv & \text{b}' \Longrightarrow \text{a}' \wedge \text{b}'' \rightleftharpoons \text{a}''
\end{array}
$$

The use of initial and terminal events will be the basis for our addition of nonatomic events to the GEM domain model. We extend world plans to include nonatomic events, and also add a relation κ, which models the composition of a nonatomic event – i.e., if $\kappa(e, f)$, then e is a part of nonatomic event f. Thus, we now have world plans of the form

$$W =< E, EL, G, \Longrightarrow, \rightsquigarrow, \rightleftharpoons, \varepsilon, F, \kappa > .$$

where F is a set of nonatomic events, and $\kappa : ((E \cup F) \times F)$ is the part-of relation between atomic (or nonatomic) events and nonatomic events. We require that, in all world plans, there should exist for each nonatomic event f two atomic events f' and f'' (its initial and terminal events) such that $\kappa(f', f)$ and $\kappa(f'', f)$. We also have the following additional constraints:

$$
\begin{array}{l}
occurred(f) \supset f' \rightsquigarrow f'' \\
e \rightsquigarrow f \supset e \rightsquigarrow f' \\
e \Longrightarrow f \supset e \Longrightarrow f' \\
f \rightsquigarrow e \supset f'' \rightsquigarrow e
\end{array}
$$

[20] The abbreviation occursnext(e) is defined by ¬ occurred(e) ∧ ○ occurred(e).

$$f \Longrightarrow e \supset f'' \Longrightarrow e$$
$$e \rightleftharpoons f \supset e \epsilon F \wedge f' \rightleftharpoons e' \wedge f'' \rightleftharpoons e''$$
$$f \varepsilon el \supset f' \varepsilon el \wedge f'' \varepsilon el$$
$$f \varepsilon g \supset f' \varepsilon g \wedge f'' \varepsilon g$$

Note that a nonatomic event can be simultaneous only with another nonatomic event. We consider them to be simultaneous if their endpoints are simultaneous. This is equivalent to Allen's relation equal (see [1]).

To model a nonatomic domain action, we can now simply use two atomic events – its initial and terminal events. Usually, however, it is preferable to associate nonatomic actions with a particular form of behavior. For example, we might want to associate a nonatomic event type with particular intervals over which some formula holds.

Suppose we have a formula P that is true of particular histories.[21] By using the following constraint, we can identify a nonatomic event type F with every convex interval in which P is true:

$$\mathsf{P} \wedge \triangle\neg\mathsf{P} \supset$$
$$(\exists\ \mathsf{f'}{:}\mathsf{F'})\ \mathsf{justoccurred(f')} \wedge \mathsf{P}\ \mathsf{U}\ (\neg\ \mathsf{P} \wedge (\exists\ \mathsf{f''}{:}\mathsf{F''})\ [\mathsf{lastoccurred(f'')} \wedge \mathsf{f'} \rightsquigarrow \mathsf{f''}])$$

where

$$\mathsf{justoccurred(f')} \equiv \mathsf{occurred(f')} \wedge \triangle\ \neg\ \mathsf{occurred(f')}$$
$$\mathsf{lastoccurred(f'')} \equiv \triangle\ \mathsf{justoccurred(f'')}$$

Namely, in any history in which P becomes true, there occurs some event f' and, when P is about to become false again, f'' occurs.

Alternatively, we can choose to model nonatomic actions as particular patterns of behavior. This is actually much more appealing in an event-based framework. To describe how a nonatomic event is achieved, we use an abbreviation of the following form:

$$\mathsf{F} \doteq \mathsf{E1} \longrightarrow \ldots\ \mathsf{event\ pattern}\ \ldots \longrightarrow \mathsf{En}$$

This states that the nonatomic event type F is composed of a pattern of other event types, beginning with an atomic *initial*-event type (here E1) and ending with an atomic *terminal*-event type (En). These initial and terminal event types are then identified (by using $\sim$) with F' and F'' respectively. If more than one way of achieving F is supplied, events f' and f'' for each nonatomic event f may be identified with any of the possible initial-event/terminal-event

[21] In other words, P is a *state description*.

pairs of event patterns that could compose f. Finally, we require that for all events *ei* of type E1 ... En that compose an event f, $\kappa(ei, f)$ must hold. Any arbitrary event pattern or set of constraints may be used to describe the set of events composing a nonatomic event. Such nonatomic events are therefore very similar to the notion of process used by Georgeff and me [11].

We now return to the restaurant scenario. We can use a nonatomic-event description to define the different methods the friends have for traveling to the restaurant. Earlier we associated each friend with a Movement element containing an event type Ride(loc1,loc2). We can now view Ride as a nonatomic event that can be expanded in two ways:

Ride(loc1,loc2) $\doteq$ auto.Occupy $\longrightarrow$ auto.Drive(loc1,loc2) $\longrightarrow$ auto.Vacate

Ride(loc1,loc2) $\doteq$ taxi.Occupy $\longrightarrow$ taxi.Drive(loc1,loc2) $\longrightarrow$ taxi.Vacate

The element variables auto and taxi must be bound to the friend's personal automobile or to any taxi, respectively.

Note that, once Ride events have been expanded, they may cause interference with other events in the domain that were not observable beforehand. For example, if there is only one taxi in town and no friend wishes to use his or her personal auto, the Ride events for all friends will, after expansion, be forced into a total ordering.

Unfortunately, it seems inevitable that the expansion of *arbitrarily* defined nonatomic events will result in added complications (or even a violation of domain constraints) at the resulting lower level of description. It is precisely for these reasons that hierarchical planners such as NOAH must *recheck* domain constraints after each event expansion. This is clearly an undesirable state of affairs. It can lead to a combinatorial explosion in the cost of reasoning about and planning in such domains.

One way of getting around this problem is to limit the forms of behavior that can constitute a nonatomic event. This limited form of behavior must lack interaction with other events in the domain; it must somehow be encapsulated. Group structure, with its associated causal limitations, is a candidate for achieving this needed encapsulation.

Consider a nonatomic event type F occurring at element *elem*. Rather than assume that initial- and terminal-event types F' and F'' also belong to *elem* (as is normally done), we create a group g with port event types F' and F'' (see Figure 5). Port events of the form f' and f'' serve as an interface between the rest of the domain and the protected forms of activity within group g (which compose events of type F). We use the abbreviation

F $\doteq g$ E1 $\longrightarrow$ ... $\longrightarrow$En

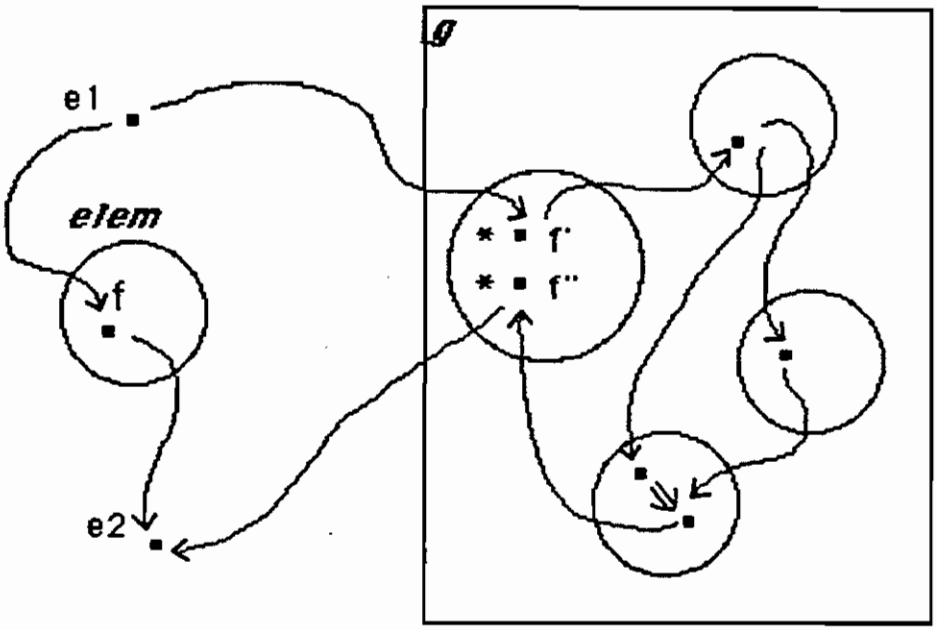


Figure 5: Protected Nonatomic-Event Expansion

for this kind of protected or limited event expansion. All events composing an event of type F must belong to group g.

We can use protected event expansion to describe the nonatomic event type OrderFood(food) in the Friend specification. To each Friend group we add a new subgroup of type FoodOrdering.

FoodOrdering = GROUP TYPE (r:Sight, t:Thought, s:Speech)

Each Sight element is assumed to have an event type Read(menu), Thought has an event type Choose(food), while Speech has an event type Utter(food). We then add to the Friend specification the following constraint (fo denotes the FoodOrdering group belonging to the particular Friend):

OrderFood(food) $\doteq fo$
fo.Sight.Read(menu) $\longrightarrow$fo.Thought.Choose(food) $\longrightarrow$ fo.Speech.Utter(food)

Each FoodOrdering group has no ports other than those of type OrderFood' and OrderFood''. Thus, if no constraints refer to events in the FoodOrdering group other than those constraints explicitly associated with FoodOrdering, and if these constraints are also localized (i.e. they utilize scoped modal operators), then interactions between a friend's food ordering activity and other domain events cannot take place.

6 State Description

As we have been trying to illustrate throughout this paper, many domain properties can easily be described without the use of state predicates and, in some cases, more naturally. However, such predicates are often useful for encoding or abbreviating aspects of past behavior. Thus, we might want to write a constraint of the form "P is a precondition of event e," without also stating in that constraint how P was achieved. For example, given some sort of definition for P, we could use the Precondition constraint defined below:[22]

Precondition(e,P) $\equiv$ occursnext(e) $\supset$ P .

where
occursnext(e) $\equiv \neg$ occurred(e) $\wedge \bigcirc$ occurred(e)

The approach we will take to atomic state formulas once again emphasizes the duality between state-based and event-based representations; just as many state-based descriptions represent events as relations between states, so shall we represent state predicates dually as formulas pertaining to events. The truth or falsity of an atomic state formula will thus depend on the definition of its predicate.

In some cases, these defining formulas will form a complete description of the state predicate. For example, we might define a predicate TableEmpty as follows:

TableEmpty(table) $\equiv$
$\neg$ ($\exists$ occupy:table.Occupy) occurred(occupy) $\vee$
($\exists$ vacate:table.Vacate) vacate cbefore table.Occupy

If, for some history and table, there has never been a table.Occupy event *or* there is a table.Vacate event that has not yet been followed by a corresponding Occupy event (i.e, the table has been vacated but not yet reoccupied), then TableEmpty(table) is true in that history. If this formula evaluates to false, TableEmpty(table) is also false.

We shall call such formulas *complete predicate definitions.* Given the TableEmpty definition, we have the following constraint on Felix's observations:

($\forall$ empty(t):observations.EmptyTable) Precondition(empty(t),TableEmpty(t)).

Notice that no frame problem arises when complete predicate definitions are used; the atomic formula TableEmpty(table) is defined to be true for a particular table if and only if its

[22]To say that P is a precondition of *all* events of type E, we would write: ($\forall$ e:E) Precondition(e,P).

corresponding formula is true. Consequently, there can be no question about the effects of unrelated events on its truth value; once true (or false), TableEmpty(table) remains true (or false) for a particular value of table until its defining formula becomes false (or true).

Sometimes, however, we shall want to use weaker, *incomplete* predicate definitions – e.g., assertions of the form formula1 $\supset$ P and formula2 $\supset \neg$ P. In this case, we know that, if formula1 is true, so is P. However, if formula1 is false, we *cannot* conclude $\neg$ P. This would be the case, however, if formula2 were true.[23] Other types of incomplete predicate definitions might include use of the temporal operators. These describe ways of attaining P for particular histories. For example, we might have formula1 $\supset \bigcirc$ P or INIT P.

We may also want to build predicate definitions not only with behavioral (i.e., event-based) formulas, but also with formulas that utilize other state predicates. For example, we might have

$$\neg \text{ P} \vee \text{Q} \vee \text{formula1} \supset \text{R}$$
$$\text{formula2} \equiv \text{P}$$
$$\text{R} \vee \text{formula3} \supset \text{Q} \ .$$

where formula1, formula2, and formula3 are purely event-based. To find valuations for P, Q, and R in a particular history, we could use an iterative-evaluation mechanism. Behavioral portions of formulas would be evaluated first and then the formulas would be iteratively simplified wherever possible. Of course, sometimes this will not yield a truth value for all atomic formulas. For example, if formula2 and formula3 are true and formula1 is false, then we have R $\wedge$ P $\wedge$ Q. However, if formula1 and formula3 are false and formula2 is true, we can conclude P but nothing more about R and Q (except that $R \Longleftrightarrow Q$).

In order to apply incompletely-defined predicates to *all* histories in an execution, it is necessary to have some sort of frame rule to assert the persistence of P (or $\neg P$) despite the occurrence of other events. Such a rule is defined in Section 6.2, which essentially minimizes the effect of events on state formulas. In the future we hope to explore the use of circumscription in the GEM framework [22].

Note that, unlike formalisms based on STRIPS-like action descriptions, the use of predicate definitions (both complete and incomplete) suffers from none of the problems created by the possibility of simultaneous action. Since effects upon a world state are not prescribed in the context of individual event descriptions (as is normally done in many state-based frameworks), there is no confusion regarding the effect of simultaneous activity or any other form of behavior. If the formula defining the truth-value of an atomic formula P is true of a history, P itself is true of the history.

[23] Of course, for these definitions of P and $\neg P$ to be consistent, $\neg(formula1 \wedge formula2)$ must hold.

For example, suppose we have a domain with two events $e1$ and $e2$, plus the requirement that, if $e1$ and $e1$ occur simultaneously, Q will be true. Otherwise P will be true. In GEM we would simply state that $\mathsf{e1} \rightleftharpoons \mathsf{e2} \supset \mathsf{Q}$ and $\neg\ \mathsf{e1} \rightleftharpoons \mathsf{e2} \supset \mathsf{P}$. These effects on P and Q cannot be expressed in the classical STRIPS framework nor in in any framework that does not accommodate event simultaneity and explicit relationships between events.

6.1 Add/Delete Axioms

The add/delete lists used in STRIPS-like frameworks to define the effects of events on state can easily be cast in terms of predicate definitions. Let $\tilde{v}$, $\tilde{w}$, $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ denote tuples of free variables and/or constants. Event type $E(\tilde{x})$ denotes the class of events of type E whose parameters match $\tilde{x}$; and $P(\tilde{z})$ matches those atomic formulas formed from predicate symbol P and a tuple of variables or constants matching $\tilde{z}$.

Given this notation, for every event type $E(\tilde{x})$ that adds $R(\tilde{y})$ under precondition $P(\tilde{z})$, we use a declaration of the form $Adder(E(\tilde{x}), P(\tilde{z}), R(\tilde{y}))$ and, for every event type $F(\tilde{w})$ that deletes $R(\tilde{y})$ under precondition $Q(\tilde{v})$, we have $Deleter(F(\tilde{w}), Q(\tilde{v}), R(\tilde{y}))$. If these *Adder* and *Deleter* declarations characterize the effects on $R(\tilde{y})$ completely, we can use the following predicate definition for $R(\tilde{y})$:[24]

$$
\begin{aligned}
&\mathsf{R}(\tilde{y}) \equiv \\
&(\exists\ \mathsf{e}{:}\mathsf{E}(\tilde{x}))\ [\ Adder(E(\tilde{x}), P(\tilde{z}), R(\tilde{y})) \wedge \mathsf{occurred(e)} \wedge \overleftarrow{\Box}\ \mathsf{precondition(e,P}(\tilde{z})) \wedge \\
&\quad \neg\ (\exists\ \mathsf{f}{:}\mathsf{F}(\tilde{w}))\ [\ Deleter(F(\tilde{w}), Q(\tilde{v}), R(\tilde{y})) \wedge \overleftarrow{\Box}\ \mathsf{precondition(f,Q}(\tilde{v})) \wedge \mathsf{e} \Longrightarrow \mathsf{f}]\ ]
\end{aligned}
$$

This states that $R(\tilde{y})$ is true if and only if it has been made true and has not been subsequently deleted. Of course, we might want to create other rules using *Adder* and *Deleter* (for example, stating that something is true ***unless*** deleted), or use *Adder* and *Deleter* to build incomplete rather than complete definitions. For example, if we stated that the above formula only ***implies*** $\mathsf{R}(\tilde{y})$, it would be equivalent to the STRIPS rule (i.e. the value of $\mathsf{R}(\tilde{y})$ would become ***undefined*** after a *Deleter* event occurs). Notice that we have also assumed that all relevant *Deleter* and *Adder* events have been explicitly stated. Thus, we have invoked a form of closed-world assumption. Another alternative might have been to use some form of circumscription over *Adder* and *Deleter* specifications.[25] While we tend to assume some form

[24] Note that this formula can be expanded into a first-order formula by taking a disjunction over all possible combinations of *Adders* and *Deleters*.

[25] Our use of the closed-world assumption here has been made only with respect to the use of *Adders* and *Deleters* for defining state predicates. It need not necessarily apply to the definition of state predicates in general. For instance, the frame rule given in the next section assumes that some way of determining which events affect which predicates is given, but does not state exactly how. Nor does the rule determine the

of minimization of the effects of events in this paper, we do not wish to take a particular stand on how it is done. We intend to explore this further in future work.

6.2 The Frame Problem

In many ways, several of the difficulties often associated with the frame problem find relief in our structured event-based model. GEM's ability to structure events into elements and groups automatically imposes constraints that limit their effects upon each other. For example, events occurring within nonintersecting groups cannot causally affect one another except through explicitly-defined ports. Likewise, events occurring within the same element cannot occur simultaneously, thereby limiting the amount of reasoning necessary to determine the effects of simultaneity. Another important aspect of group/element structure is that it provides a well-defined *framework* in which nonmonotonic reasoning can take place. For example, while we may discover new qualifications on previously known preconditions for starting a car (e.g., that there be no potato in the tailpipe), groups provide a structure in which to add and use these qualifications. For example, if we model the car as a group, we could add new ports to that group which allow for the newly discovered kinds of effects.

While we may effectively use group/element structure to alleviate aspects of the frame problem, there is still a need for frame rules in GEM. In particular, we still need a way to complete incompletely-defined predicates. As we have so far described, the truth or falsity of a state-based formula with respect to a particular history must be derived from predicate definitions. Given that we want to build a computational system based on our model, it will be inefficient to reevaluate these definitions for every history. This problem is somewhat alleviated by the fact that event-based domain descriptions do not often employ state predicates. Still, some sort of frame rule for carrying over atomic-formula valuations from one history to the next seems to be necessary. The same rule can also be used for extending incomplete predicate definitions to form complete predicate definitions.

We now describe a semantic frame rule of the form used by Georgeff [10]. Such semantic rules avoid the difficulties usually associated with syntactic approaches to the frame problem. For each n-ary predicate symbol P and event e in a world plan, we add a formula $\delta_P(e, \widetilde{x})$ to the world plan, where $\widetilde{x}$ is an n-tuple of free variables. This states that, for every $\widetilde{x}$, if $\delta_P(e, \widetilde{x})$ holds, then $P(\widetilde{x})$ may be affected by e. (Note the similarity between δ and the *Adder/Deleter* classifications used in the previous section.[26]) We then have the following frame rule:

$$\triangle P(\widetilde{x}) \wedge \neg(\exists e)[justoccurred(e) \wedge \delta_P(e, \widetilde{x})] \supset P(\widetilde{x}).$$

value of an atomic formula if it is affected by an event in some ***unknown*** way.

[26] And, as stated in the previous section, while we may wish to minimize δ, we take no stand in this paper on exactly how this is to be done.

In other words, if $P(\widetilde{x})$ is true in a given history and no event occurs that can affect $P(\widetilde{x})$, then $P(\widetilde{x})$ remains true.

While the formula $\delta_P(e, \widetilde{x})$ may certainly be different for every P, e, and $\widetilde{x}$, a useful way of defining δ formulas in general is to assume that the predicate definitions in a particular specification are complete. In other words, we could assume that only events of types explicitly designated as affecting $P(\widetilde{x})$ can affect $P(\widetilde{x})$. Suppose we define $eventset(P(\widetilde{x}))$ to be precisely the set of events so designated. We then assert: $\delta_P(e, \widetilde{x}) \Longleftrightarrow e \in eventset(P(\widetilde{x}))$. The frame rule then reduces to: $\triangle P(\widetilde{x}) \wedge \neg(\exists\, e \in eventset(P(\widetilde{x})))justoccurred(e) \supset P(\widetilde{x})$.

7 Conclusion

This paper has presented a structured, event-based framework for representing the properties of domains with parallel activity. We have attempted to demonstrate its utility in describing the complex properties of such domains in a coherent and semantically sound fashion. Our model, designed explicitly for describing parallel domains, has a well understood semantics (the semantics of first-order temporal logic) that has been used elsewhere in concurrency theory. By the use of first-order temporal-logic constraints on *history sequences*, complex synchronization properties based on causality, temporal ordering, and simultaneity can be expressed easily and naturally.

An important aspect of our model is its explicit representation of event location. This is used to embody the structural aspects of a domain, to localize domain constraints, and to impose constraints on the temporal ordering as well as on causal access. The model also includes the notion of nonatomic events. State-based specifications can be incorporated by describing state in terms of past behavior. We have presented a semantic frame rule for such uses of state. However, we have also stressed the fact that many properties can be expressed without resorting to state-based description.

We are currently constructing a planning system (GEMPLAN) based on the formalism described in this paper. It is being written in Prolog on a Sun 3/50. The present system is capable of generating multiagent solutions to blocks world problems [16]. Given an event-based specification for a domain, the planner builds an initial world plan that reflects a particular problem's initial and goal states. This event network is then filled in through a process of incremental constraint satisfaction.

The planning search space in GEMPLAN may be viewed as a tree with a partial plan at each node. When a node is reached, the system checks to see whether a particular constraint has been satisfied. If it has not, the search space branches for each of the possible plan "fixes" that will satisfy that constraint. Since the kinds of constraints that must be satisfied are much

broader than the pre- and postconditions used by other planners, the derivation of constraint fixes is a nontrivial problem.

Highlights of the current system include a table-driven search mechanism that can be adapted to specific domains, an efficient representation of world plans, facilities for plan explanation, and nonatomic-event expansion. Initial work has also begun on delayed binding of event parameters, on accumulating constraints on unbound variables, and on dependency-directed search and backtracking.

Especially promising is current work on structuring and guiding the search in a way that makes use of the domain structure itself. The planning space is partitioned to reflect the element/group structure of the domain and search through this partitioned space is guided by the aforementioned table-driven mechanism, which is tailorable to the domain. The result is a planning architecture that can generate plans in a manner that can vary according to the specific application, ranging from loosely coordinated, distributed forms of planning (for less tightly synchronized applications) to more tightly coordinated, centralized planning (for those applications in which synchronization constraints are strong and complex).

Another useful result is an algorithm that transforms an n-robot solution into an m-robot solution where $n > m$. By using this algorithm, maximally parallel solutions are generated and then made less parallel as resource constraints warrant. We expect this development to be applicable to many other resource planning problems besides the blocks world and we shall be reporting on it at a later date.

To satisfy violated constraints, we are currently using predefined fixes for common constraint forms. Because of the intractability of solving arbitrary first-order temporal-logic constraints, we considered this to be a good initial approach to this problem. It is similar to Chapman's idea of *cognitive cliches* – i.e., utilizing a set of specialized theories that are common to many domains, rather than trying to solve for the most general theory [5]. A similar idea is incorporated in the ISIS system [9]. However, we are also working on techniqes for deriving some fixes automatically from the logical form of a constraint, at least for a subset of the logic. The synchronization techniques for solving propositional temporal-logic constraints as conceived by Manna and Wolper [21] (and implemented by Stuart [31]) may eventually be applied.

Acknowledgments

I would like to thank Michael Georgeff for his critical reading of several drafts of this paper as well as many enlightening discussions on multiagent-domain representation and planning. Thanks also to David Fogelsong for his help in constructing GEMPLAN and for his constructive comments in reviewing this paper.

References

[1] Allen, J.F. "Towards a General Theory of Action and Time," *Artificial Intelligence*, Vol. 23, No. 2, pp. 123-154 (1984).

[2] Allen, J.F. and J.A.Koomen, "Planning Using a Temporal World Model," *IJCAI-83, Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp. 741-747 (August 1983).

[3] Barringer, H. and R. Kuiper, "Hierarchical Development of Concurrent Systems in a Temporal Logic Framework," *Proceedings of the NSF/SERC Seminar on Concurrency*, Carnegie Mellon University, Pittsburgh, Pennsylvania (July 1984).

[4] Chapman,D. "Cognitive Cliches," AI Working Paper 286, MIT Laboratory for Artificial Intelligence, Cambridge, Massachusetts (April 1986).

[5] Chapman, D. "Planning for Conjunctive Goals," Masters Thesis, Technical Report MIT-AI-TR-802, MIT Laboratory for Artificial Intelligence, Cambridge, Massachusetts (1985).

[6] Davidson, D. Essays on Actions and Events, Clarendon Press, Oxford, England (1980).

[7] Cheeseman, P. "A Representation of Time for Automatic Planning," *Proceedings of the IEEE International Conference on Robotics*, Atlanta, Georgia (March 1984).

[8] Fikes, R.E, P.E.Hart, and N.J.Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, 3 (4), pp. 251-288 (1972).

[9] Fox, M.S. and Smith, S.F. "ISIS – A Knowledge-Based System for Factory Scheduling," *Expert Systems, the International Journal of Knowledge Engineering*, Volume 1, Number 1, pp. 25-49 (July 1984).

[10] Georgeff, M. "The Representation of Events in Multiagent Domains," *AAAI-86, Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania (August 1986).

[11] Georgeff, M. and A. Lansky, "Procedural Knowledge," *Proceedings of the IEEE, Special Issue on Knowledge Representation*, pp.1383-1398 (October 1986).

[12] Hewitt, C. and H.Baker Jr. "Laws for Communicating Parallel Processes," *IFIP 77*, B.Gilchrist,ed., pp. 987-992, North-Holland, Amsterdam, Holland (1977).

[13] Ladkin, P. "Comments on the Representation of Time," *in Proceedings of 1985 Workshop on Distributed Artificial Intelligence*, Sea Ranch, California, pp. 137-158 (1985).

[14] Lamport, L. "Times, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM* Vol. 21, No. 7, pp. 558-565 (July 1978).

[15] Lansky, A.L. "A Representation of Parallel Activity Based on Events, Structure, and Causality," Technical Note 401, Artificial Intelligence Center, SRI International, Menlo Park, California (1986).

[16] Lansky, A.L. "GEMPLAN: Event-based Planning Through Temporal Logic Constraint Satisfaction," Forthcoming Working Paper, Artificial Intelligence Center, SRI International, Menlo Park, California (1987).

[17] Lansky, A.L. "Behavioral Specification and Planning for Multiagent Domains," Technical Note 360, Artificial Intelligence Center, SRI International, Menlo Park, California (1985).

[18] Lansky, A.L. "A 'Behavioral' Approach to Multiagent Domains," *in Proceedings of 1985 Workshop on Distributed Artificial Intelligence*, Sea Ranch, California, pp. 159-183 (1985).

[19] Lansky, A.L. "Specification and Analysis of Concurrency," Ph.D. Thesis, Technical Report STAN-CS-83-993, Department of Computer Science, Stanford University, Stanford, California (December 1983).

[20] Lansky, A.L. and S.S.Owicki, "GEM: A Tool for Concurrency Specification and Verification," *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pp.198-212 (August 1983).

[21] Manna, Z. and P.Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications," *ACM Transactions on Programming Languages and Systems*, 6 (1), pp.68-93 (January 1984).

[22] McCarthy, J. "Circumscription – A Form of Non-Monotonic Reasoning," *Artificial Intelligence*, Vol. 13, No. 1-2, pp.27-39 (1980).

[23] McCarthy, J. and P.Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," *Machine Intelligence*, Vol.4, B.Meltzer and D.Michie, eds., pp. 463-502, American Elsevier, New York, New York (1969).

[24] McDermott, D. "A Temporal Logic for Reasoning About Processes and Plans," *Cognitive Science* 6, pp.101-155 (1982).

[25] Nilsson, N. Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, California (1980).

[26] Owicki, S. and L.Lamport, "Proving Liveness Properties of Concurrent Programs," *ACM TOPLAS 4*, 3, pp.455-492 (July 1982).

[27] Pednault, E.P.D. "Toward a Mathematical Theory of Plan Synthesis," Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, California (forthcoming).

[28] Sacerdoti, E.D. A Structure for Plans and Behavior, Elsevier North-Holland, Inc., New York, New York (1977).

[29] Shoham, Y. "A Logic of Events," Working Paper, Department of Computer Science, Yale University, New Haven, Connecticut (1985).

[30] Shoham, Y. and T.Dean, "Temporal Notation and Causal Terminology," Working Paper, Department of Computer Science, Yale University, New Haven, Connecticut (1985).

[31] Stuart, C. "An Implementation of a Multi-Agent Plan Synchronizer Using a Temporal Logic Theorem Prover," *IJCAI-85, Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Los Angeles, California (August 1985).

[32] Tate, A. "Generating Project Networks," *IJCAI-77, Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, pp. 888-893 (August 1977).

[33] Vere, S.A. "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No.3, pp. 246-267 (May 1983).

[34] Wilkins, D. "Domain-independent Planning: Representation and Plan Generation," *Artificial Intelligence*, Vol. 22, No. 3, pp. 269-301 (April 1984).

APPENDIX: Restaurant Domain Specification

Restaurant Tables (of sizes 1 to 10)

RestaurantTable (size:INTEGER) = ELEMENT TYPE
EVENTS
  Occupy(p:Party)
  Vacate(p:Party)
CONSTRAINTS
1) Tables can be occupied by only one party at a time.
  (Occupy(p) $\longrightarrow$ Vacate(p))$^{*}\longrightarrow$
END RestaurantTable

table[i=1..10] = RestaurantTable(i) ELEMENT

Vehicles

Vehicle = ELEMENT TYPE
EVENTS
  Occupy
  Drive(loc1,loc2:Location)
  Vacate
CONSTRAINTS
1) Vehicles can be occupied by only one passenger at a time.
  (Occupy $\longrightarrow$ Drive(loc1,loc2) $\longrightarrow$ Vacate)$^{*}\longrightarrow$
END Vehicle

Taxi = Vehicle ELEMENT TYPE
taxi[1..5] = Taxi ELEMENT

Auto = Vehicle ELEMENT TYPE
auto[1..15] = Auto ELEMENT

Restaurant Lobby and Reservations

lobby = ELEMENT
EVENTS
  Enter(f:Friend)
END lobby

reservations = ELEMENT
EVENTS
  Reserve(p:Party, b:Bribe)
  Seat(p:Party, t:RestaurantTable)
CONSTRAINTS
1) To be seated, there must be a reservation. Moreover, each reservation is good for only one seating.
  Reserve(p,b) $\longrightarrow$ Seat(p,t)

2) Parties can only be seated at tables of the right size.
  occur(seat) $\supset$ seat.p.size = seat.t.size

3) A bigger bribe will get you seated faster.
  reserve1 cbefore Seat $\wedge$ reserve2 cbefore Seat $\wedge$ reserve1.b > reserve2.b $\supset$
  $\Box$ [($\exists$ seat2:Seat) reserve2$\rightsquigarrow$seat2 $\supset$ ($\exists$ seat1:Seat) reserve1$\rightsquigarrow$seat1]

4) All other things being equal, seating is first-come-first-served.
  reserve1(p1,b1)$\Longrightarrow$reserve2(p2,b2) $\wedge$ b1=b2 $\wedge$
  present(p1) $\wedge$ present(p2) $\supset$
  $\Box$ [($\exists$ seat2:Seat) reserve2$\rightsquigarrow$seat2 $\supset$ ($\exists$ seat1:Seat) reserve1$\rightsquigarrow$seat1]

5) A $50 bribe will definitely get you seated.
  reserve.b $\geq$ $50 $\supset$ $\Diamond$ ($\exists$ seat:Seat) reserve$\rightsquigarrow$seat
END reservations

where

present(p) $\equiv$ ($\forall$ f:Friend, fϵp) ($\exists$ enter(f):Lobby.Enter)
  enter(f) cbefore Reservations.Seat

Friend=GROUP TYPE (m:Movement,c:Communication,f:FoodOrdering,a:Auto)
CONSTRAINTS
1) Each friend must sit and order before they can eat.
m.Sit $\longrightarrow$ c.OrderFood $\longrightarrow$ c.Eat

2) If a friend is eating, they must still be sitting at a table.
justoccurred(c.eat) $\wedge$ m.sit$\rightsquigarrow$c.orderfood$\rightsquigarrow$c.eat $\supset$
m.sit cbefore m.LeaveTable

3) To ride somewhere, a friend may use their own car.
m.Ride(loc1,loc2) $\doteq$ a.Occupy $\longrightarrow$ a.Drive(loc1,loc2) $\longrightarrow$ a.Vacate

4) To order food, a friend must first read the menu, choose a meal, and then tell the waiter.
c.OrderFood $\doteq f$ f.s.Read(menu) $\longrightarrow$f.t.Choose(food) $\longrightarrow$ f.s.Utter(food)

END Friend

Movement = ELEMENT TYPE
EVENTS
Ride(loc1,loc2:Location)
Walk(loc1,loc2:Location)
Sit(tableloc:Location)
LeaveTable(tableloc,loc:Location)
CONSTRAINTS
1) A friend must walk to a table before sitting there, and must be sitting there before leaving.
Walk(loc1,tableloc) $\longrightarrow$ Sit(tableloc) $\longrightarrow$ LeaveTable(tableloc,loc2)

2) To depart from a location x, a friend must first be there.
($\forall$ move(x,y):{Ride,Walk,LeaveTable}) Precondition(move(x,y), at(x))

at(x) $\equiv$
(INIT at(x) $\wedge$ $\neg$ ($\exists$ move:{Ride,Walk,LeaveTable}) occurred(move)) $\vee$
(($\exists$ move(z,x):{Ride,Walk,LeaveTable}) occurred(move(z,x)) $\wedge$
$\neg$ ($\exists$ move':{Ride,Walk,LeaveTable}) move(z,x)$\Longrightarrow$move')
END Movement

```
Communication = ELEMENT TYPE
EVENTS
  Reserve(p:Party, b:Bribe)
  OrderFood(food:Foodstuff)
  Eat
END Communication


FoodOrdering = GROUP TYPE (t:Thought, s:Speech, r:Sight, f:Order)
                PORTS(f.OrderFood', f.OrderFood")


Thought = ELEMENT TYPE
EVENTS
  Choose(food:Foodstuff)
END Thought


Speech = ELEMENT TYPE
EVENTS
  Utter(food:Foodstuff)
END Speech


Sight = ELEMENT TYPE
EVENTS
  Read(menu:ReadingMaterial)
END Sight


Order = ELEMENT TYPE
EVENTS
OrderFood'(food:Foodstuff)
OrderFood" (food:Foodstuff)
END Order
```

comm[1..15] = Communication ELEMENT
move[1..15] = Movement ELEMENT
thought[1..15] = Thought ELEMENT
speech[1..15] = Speech ELEMENT
sight[1..15] = Sight ELEMENT
order[1..15] = Order ELEMENT
foodorder[i=1..15] = FoodOrder GROUP (thought[i].speech[i].sight[i].order[i])
auto[1..15] = Auto ELEMENT
friend[i=1..15] = Friend GROUP (comm[i].move[i].foodorder[i].auto[i])

The Parties

Party(size:INTEGER) =
  GROUP TYPE ({f}:SET OF Friend. reservations.lobby)
CONSTRAINTS
1) (size must be the size of the set of friends)
  size = setsize({f})

2) A reservation by a friend is identified with a reservation at the desk.
  f.c.Reserve(p,b) $\approx$ reservations.Reserve(p,b)

3) All members of a party must be seated simultaneously.
  ($\forall$ f' ϵ {f}) reservations.Seat(SELF,t) $\sim$ f'.m.Sit(t)

4) In order to be seated, all members of the party must be present.
  ($\forall$ f' ϵ {f}) lobby.Enter(f') $\longrightarrow$reservations.Seat(SELF,Table)

5) The first friend to enter the lobby must make a reservation.
  ($\forall$ f1,f2 ϵ {f}) occurred(lobby.enter1(f1)) $\wedge$
    $\neg$($\exists$ lobby.enter2(f2)) [lobby.enter2(f2)$\Longrightarrow$lobby.enter1(f1)] $\supset$
  $\Diamond$ ($\exists$ reserve1:f1.c.Reserve(SELF,b)) occurred(reserve1)
END Party

party[1] = GROUP ({friend[1..3]},reservations,lobby)
party[2] = GROUP ({friend[4..8]},reservations,lobby)
party[3] = GROUP ({friend[9..15]},reservations,lobby)

Felix

observations = ELEMENT
EVENTS
  EmptyTable(t)
END observations

felixworld = GROUP (reservations, table[1..10], lobby, observations)
             PORTS(table[i].Vacate)
CONSTRAINTS
1) Felix must observe that a table is empty before he can seat a party there.
   observations.EmptyTable(table[i]) $\longrightarrow$ reservations.Seat(Party,table[i])

2) Seating a party at a table is the same as occupying the table.
   reservations.Seat(p,table[i]) $\approx$ table[i].Occupy(p)

3) Felix must make correct observations about empty tables.
   ($\forall$ empty(t):observations.EmptyTable) Precondition(empty(t),TableEmpty(t))

   TableEmpty(table[i]) $\equiv$
   $\neg$ ($\exists$ occupy:table[i].Occupy) occurred(occupy) $\vee$
   ($\exists$ vacate:table[i].Vacate) vacate cbefore table[i].Occupy

END felixworld

The Entire Restaurant Scenario

restaurantscenario = GROUP(felixworld, party[1..3], taxi[1..5])
CONSTRAINTS
1) ($\forall$ taxi[i],party[j])
   party[j].f.m.Ride(loc1,loc2) $\doteq$
   taxi[i].Occupy $\longrightarrow$ taxi[i].Drive(loc1,loc2) $\longrightarrow$ taxi[i].Vacate
END restaurantscenario