From: AAAI-87 Proceedings. Copyright ©1987, AAAI (www.aaai.org). All rights reserved.

Localized Representation and Planning Methods for Parallel Domains

Amy L. Lansky*
David S. Fogelsong**

Abstract

This paper presents a general method for structuring domains that is based on the notion of *locality*. We consider a localized domain description to be one that is partitioned into *regions of activity*, each of which has some independent significance. The use of locality can be very beneficial for domain representation and reasoning, especially for parallel, multiagent domains. We show how localized domain descriptions can alleviate aspects of the frame problem and serve as the foundation of a planning technique based on localized planning spaces. Because domain constraints and properties are localized, potential interactions among these search spaces are fewer and more easily identified.

1 Introduction

The use of hierarchy is a well-recognized representational technique, not only in AI but in computer science as a whole. Such representations facilitate our understanding of domain descriptions and make it possible to use "divide-and-conquer" problem-solving techniques. However, hierarchical descriptions are only one of the many possible ways of subdividing a domain. Depending on how one plans to use or view a domain representation, an appropriate decomposition might include regions that overlap, form disjoint sets, or take on any other structural configuration – they need not necessarily form hierarchies.

In this paper we discuss a more comprehensive manner of structuring domains, one that utilizes the notion of *locality*. By "locality" we mean a very general notion of structure or decomposition. A localized domain description is considered to be one that is partitioned into *regions of activity*, each of which has some form of independent significance. Regions may be composed of related subregions of activity to form hierarchies or any other kind of structural configuration.

Since we are particularly interested in representing parallel domains, it is also important to account for the potential interactions among regions. To help deal with this problem, we introduce the use of *ports* – well-defined region-boundary locations in which interactions can take place. The notion of a port has been used extensively in the design of distributed systems as well as in some CAD systems [15].

*Artificial Intelligence Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California, and the Center for the Study of Language and Information, Stanford University.

**Computer Science Department, Stanford University.

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 author and should not be interpreted as necessarily representative of the official policies, either expressed or implied, of the Office of Naval Research, NSF, or the United States government.

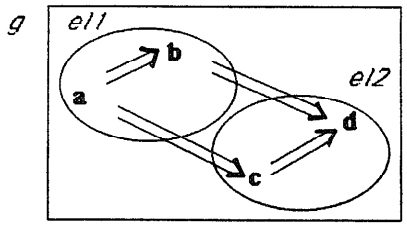
The primary goal of this paper is to examine the role that locality plays in domain representation and reasoning. In particular, we focus on three uses of this structuring concept. First, we show how a localized specification methodology can be used to define domain properties and impose constraints only within relevant regions of activity. Second, by viewing certain types of locations as regions of causal effect, locality can be used as a way of addressing the frame problem. Such use of locality has already been recognized in the AI literature [6,7], but has not been extensively explored. Finally, the localization structure of a domain can provide heuristics for problem-solving in that domain. We present a localized planning technique that partitions both the plan representation and the planning search space according to the structure of the domain. Because constraints are localized, there are far fewer interactions between regional search spaces and, when they do exist, they are more readily identified. While the containment of such interactions is a goal of many existing planning systems [16,18,19,20], most do not localize domain descriptions sufficiently. As a result, the task of determining and coping with interactions is extremely expensive.

The basis of the work described in this paper is GEM (Group Element Model) [8,9,10,11], a model that explicitly represents regions of activity in the manner we have described – i.e., it allows them to be defined and grouped together in arbitrary ways and associated with ports of interaction. Besides its use of domain structure, GEM is unusual in being an event-based (rather than state-based) framework. Domains are described strictly in terms of the events that occur within regions of activity and the causal and temporal relationships between those events. Domain properties are described by first-order temporal logic constraints that limit a domain's potential behaviors. Several domain representations other than our own have been proposed that make use of events and event relationships [1,4,13,14]. However, GEM differs from most of these in having a *purely* event-based domain model (where state descriptions are derived from past event behaviors), as well as in its emphasis on event localization.

The rest of this paper is organized as follows. In Section 2 we present a brief overview of the GEM model. In Section 3 we discuss the influence of locality on the frame problem. Finally, in Section 4, we describe *GEMPLAN*, a localized planner based on the GEM representation.

2 Model and Specification Language

The GEM specification framework was designed for the description of domains with intrinsic parallelism. It has been used not only for AI applications, but also for concurrent program specification and verification. In this section we shall try to suggest the general flavor of the domain model and specification language;



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

Figure 1: A World Plan

much more rigorous and complete definitions can be found elsewhere [8,9,10,11]. While these previous papers have emphasized GEM's use of event-based temporal-logic constraints, this paper focuses on GEM's structuring capabilities.

GEM's underlying domain model is constructed in terms of events[1] that are localized into regions of activity. Event instances may be interrelated by three kinds of relations: the temporal order $\Longrightarrow$, the causal relation $\rightsquigarrow$ (modeling a direct causal relationship between event instances), and a simultaneity relation $\rightleftharpoons$ (modelling the necessary simultaneity of event instances). GEM utilizes two kinds of regions: *elements* and *groups*. Elements are the most basic type of region. Every event must belong to some element, and all events belonging to the same element must be temporally totally ordered – i.e., elements represent regions of sequential activity. Elements (and their constituent events) may then be clustered into groups. Each group represents a region of causally encapsulated activity. The causal laws associated with groups are described in detail in Section 3.

GEM's domain model can be viewed as a two-tiered structure. The upper level consists of *world plans* (see Figure 1). Every world plan consists of a set of events, elements, and groups, and their interrelationships. Each event in a world plan models a unique event or action occurring in the world domain, each relation or ordering relationship models a relationship between domain events, and each element or group models a logical region of activity.[2]

World plans are meant to convey *known* information about a domain. This information may be incomplete in the sense that some potential relationships are left undetermined. For example, if $\Longrightarrow$(e1,e2) is true in a world plan, e1 must always occur before e2 in every behavior of the domain. However, if no relationship exists between the two events, they might occur in either order or even simultaneously. The lower tier of the GEM world model contains the set of potential behaviors or executions permitted by a world plan. These executions must conform to all the relationships established in the world plan – in essence, they represent its possible "completions."

For example, the world plan in Figure 1 can be executed in three possible 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

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 that any one of them actually does. All three executions are thus part of the lower level world model for this world plan. In GEM, these execution sequences of a world plan are modeled as linear sequences of *histories* – i.e., as a set of *history sequences*. Each history α may be viewed as a "state" that encompasses not only the state of the world at some given moment, but everything that has occurred up to that moment – i.e., it is a snapshot of past behavior.[3] The GEM history sequences for the world plan in Figure 1 are as follows:

Execution 1: α_0 α_i α_j α_m α_n
Execution 2: α_0 α_i α_k α_m α_n
Execution 3: α_0 α_i α_m α_n

where α_0 is the empty history, α_i is the history with just event a, α_j is a history in which a has been followed by b (but not c), α_k contains a followed by c (but not b), α_m contains events a, b, and c, and α_n includes all four events.[4]

Given this underlying world model, domains are described by GEM *specifications*. Each specification consists of a set of constraints that limit the allowed executions or behaviors of that domain. A given world plan W is considered to satisfy a set of domain constraints if every one of its history sequences (i.e., executions) satisfies every constraint in the set. The task performed by the GEMPLAN planner is to construct world plans that attain some stated goal and satisfy all of a domain's constraints.

Just as elements and groups model the structural aspects of a domain, they also serve as the structural components of the GEM specification language. Each specification is composed of a set of element and group declarations. Each element is associated with a set of event types (the types of events that may occur at the element). Each element and group may also be associated with a set of first-order linear-temporal-logic constraints. These constraints are localized, applying only to those events occurring within the element or group in which they are defined. Every specification also includes a set of default constraints imposed by the element/group structure of the domain itself. These default "locality" constraints and their effect on the frame problem are discussed in Section 3.

Figure 2 illustrates a sample specification and a possible world plan for a cooking-class domain. The cooking class is described as a group consisting of a set of kitchen subgroups, all of which share a teacher element. Each kitchen also contains a set of student elements and an oven element. Typical constraints that might be used in this domain include rules regarding individual student behavior, limitations on the use of the oven in each kitchen, requirements for student cooperation on certain tasks,

[1] While they are often considered distinct, we use the terms *event*, *event instance*, and *action* interchangeably.

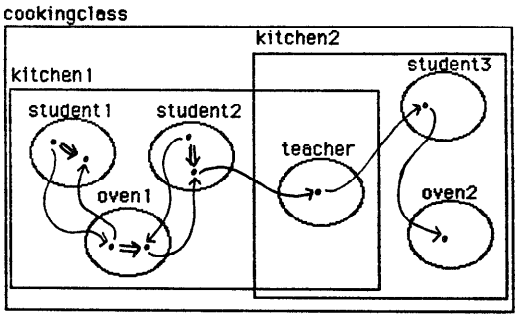
[2] We assume here that all events are atomic. The model is expanded in include nonatomic events elsewhere [8].

[3] The reader should be warned that the term *history* has been used by others in different ways – for example, to denote a particular sequence of states.

[4] One way of representing the possible history sequences of a world plan is as a branching tree. In this example, we would have

$$\alpha_0 \rightarrow \alpha_i \rightarrow \begin{cases} \alpha_j \rightarrow \alpha_m \rightarrow \alpha_n \\ \alpha_k \rightarrow \alpha_m \rightarrow \alpha_n \\ \alpha_m \rightarrow \alpha_n \end{cases}$$

This corresponds to the branching tree of states used by McDermott; a chronicle corresponds to a history sequence [13].




```
Student = ELEMENT TYPE              student1 = Student ELEMENT
EVENTS                              student2 = Student ELEMENT
   Prepare(cake)                    student3 = Student ELEMENT
   OvenRequest
CONSTRAINTS
   :
END Student

Oven = ELEMENT TYPE                 oven1 = Oven ELEMENT
EVENTS                              oven2 = Oven ELEMENT
   Bake(cake)
CONSTRAINTS
   :
END Oven

Kitchen = GROUP TYPE                teacher = ELEMENT
          (teacher,                 EVENTS
           {s}:SET OF Student,         Instruct
           o:Oven)                  CONSTRAINTS
CONSTRAINTS                            :
   :                                END teacher
END Kitchen

kitchen1 = Kitchen GROUP (teacher,{student1,student2},oven1)
kitchen2 = Kitchen GROUP (teacher,{student3},oven2)
cookingclass = GROUP (kitchen1,kitchen2)
CONSTRAINTS
   :
END cookingclass
```

Figure 2: A Cooking Class Domain

or descriptions of appropriate reactions to the teacher's instructions. Figure 2 also illustrates the use of GEM's region type definition and instantiation mechanism.[5] In the full GEM model, region-type inheritance and refinement may also be used [8].

The constraints associated with elements and groups are written as first-order temporal-logic formulas which are then applied to history sequences. Temporal logic has a well-defined semantics and has been used extensively in concurrency theory. While we shall not discuss the details of the logic here, we shall illustrate briefly how quite complex properties can be described.

GEM's temporal operators may be applied to sequences that go forward in time (using the operators $\Box$ (*henceforth*), $\Diamond$ (*eventually*), $\bigcirc$ (*next*), and $P\ U\ Q$ (P *until* Q)) as well as backwards in time ($\triangle$ (*before*), $\overleftarrow{\Box}$ (*until now*), $P \overset{Q}{\leftarrow}$ (Q *back to* P)). For instance, when past behavior dictates the course of future behavior, we would typically use a constraint of the form: $P \supset \Box Q$. This may be read: "if P holds for the events in some history, then, for every history which follows in the history sequence, Q must hold." This constraint form is commonly used for priority requirements as well as for many other naturally occurring domain properties. A simple first-come-first-served requirement for use of an oven might be:

$$(\forall\ \text{ovenreq1,ovenreq2:OvenRequest, k:Kitchen})$$
$$\text{ovenreq1}\epsilon\text{k} \wedge \text{ovenreq2}\epsilon\text{k} \wedge \text{ovenreq1} \Longrightarrow \text{ovenreq2} \supset$$
$$\Box\ [\ \text{serviced(ovenreq2)} \supset \text{serviced(ovenreq1)}\ ]$$

In other words, if two oven requests in the same kitchen occur in some order, they must be serviced in that order. The notation serviced(e) is an abbreviation for a specific event formula that is true of histories in which an oven request has been fulfilled.[6] An example of an *eventuality constraint* is the following: occurred(ovenreq) $\supset \Diamond$ serviced(ovenreq). That is, if an oven request occurs, it must eventually be serviced. Backwards temporal operators may be used to describe event preconditions. For example, justoccurred(e) $\supset \triangle$ precondition(e) may read "if e has just occurred then precondition(e) must hold in the preceding history."

3 Locality and the Frame Problem

Probably the most significant and best-understood aspect of the frame problem is what Georgeff [6] has called the *combinatorial problem*, i.e., how to state which properties remain unaffected by actions. In a recent paper, he shows how *independence axioms* can be used to solve this and other related problems. Assuming that one is able to state which events are independent of which properties, Georgeff offers a general-purpose law of persistence that guarantees that properties will remain unaffected by independent events. One of the undeveloped aspects of Georgeff's theory is exactly how to specify independence axioms. He suggests domain-structuring techniques as a possible solution. Hayes [7] has also suggested that domain structure can be used as a way of delineating frame axioms. We now show how GEM's use of locality can achieve precisely this objective.

GEM specifications are associated with the following implicit constraints imposed by the structure of a domain:

- All events belonging to the same element must be totally ordered temporally. For instance, in our cooking class scenario, each oven can have only one item baking in it at a time. Elements are often used to model limited resources which, by their very nature, are constrained to support only one action at a time.
- Groups are used to represent regions whose boundaries limit inward causal effect.[7] For example, in the cooking domain,

[5] Event, group, and element instances are denoted in lowercase; types are capitalized.

[6] As discussed elsewhere [8], GEM state descriptions are built strictly in terms of formulas on events. This way of defining state descriptions does not result in any loss of expressiveness and maintains the purity and usefulness of event-based descriptions. Indeed, priority properties such as these are much more awkward to describe in formalisms that are based strictly on state.

[7] Thus, the effects of events are assumed to range freely unless they are explicitly blocked. While we could have made a group wall limit outward access as well as inward, we have found this one-way "wall" to be more useful. The effect of a two-way wall can be simulated by enclosing more regions within groups.

the actions of students belonging to different kitchens cannot be related causally. However, within each kitchen, the actions of the teacher, the students, and the oven may be causally interrelated. In addition, because the teacher belongs to all kitchens, causal interactions between kitchens may propogate through the teacher.

One exception to the group 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. For example, we might declare certain student actions as kitchen ports. By doing so, these student actions may be affected by everyone in the cooking class. Let us assume that the atomic formula $port(e, g)$ is true for every event e that has been declared a port of group g. Moreover, suppose that $e1$ belongs to element $el1$, $e2$ to element $el2$. Then the formal constraint on the causal relation imposed by group structure may be described as follows:

$$maycause(e1, e2) \equiv access(el1, el2) \vee [port(e2, g) \wedge access(el1, g)]$$

We define $access(x, y)$ to be true if any of the following holds: (1) x and y belong directly to the same group or (2) y is not contained within any group *or* (3) y is "global" to x. We say that y *belongs directly* to a group g if it is explicitly declared as one of the components of group g. We consider y to be global to x if there is some surrounding group g' such that y belongs directly to g' and x in indirectly contained within g'. For instance, if we added a door element directly to the cooking class group, it would be global and thus accessible by every student.

Group structure is a natural way of defining event independence in a general manner.[8] Suppose a given property P is associated with activity in group R. If event e has no causal access to activity in R (i.e., $\neg(\exists f, f \varepsilon R)[maycause(e, f)]$) or, alternatively, e has no causal access to any event in R that can influence P, then we can assert that e is independent of P. For example, since the entire cooking class lacks ports, no activity outside the class can influence properties of the class. By helping to define independence in a succinct and well-defined fashion, group structure helps solve the combinatorial problem. Elements also help address the combinatorial problem because they limit potential forms of parallelism within a domain. Restriction of the oven to sequential use, for example, ensures the persistence of certain oven properties.

Strictly speaking, of course, events also influence one another by virtue of the explicit constraints associated with domain regions. Depending on the nature and scope of constraints, actions within certain regions may or may not violate the constraints of other regions. This is precisely the advantage of GEM's localization of constraints; if a given region's constraint is known to be satisfied, the introduction of a new event at some other disjoint region can do nothing to violate that constraint. The GEMPLAN planner takes direct advantage of this guaranteed noninterference property.

Of course, depending on the structure of elements and groups, noninterference cannot always be guaranteed: activity occurring within a particular region R might violate a more *global* constraint associated with a larger region containing R. For example, if the cooking class group as a whole were associated with constraints and properties that pertain to, or might be affected by, events of *any* student, then all students could affect one another by interfering with these "global" requirements. However, if a strict and more structured specification-writing methodology is adhered to, localization can be made tighter. For example, if the constraints associated with the cooking class as a whole are restricted to apply only to actions that each kitchen or student makes explicitly accessible (e.g., port events), then event interactions can be well delineated and kept under control. This kind of constraint localization also helps to ensure that subplans generated by localized planning procedures will not interfere with each other, even at a more global level.

[8] Dynamic restructuring of groups and elements is also utilized in an expanded version of the GEM model [10]. However, we do not use it in this paper or in the current version of GEMPLAN.

4 Localized Planning Method

In this section we describe the **GEMPLAN** multiagent planning system. Written in Prolog on a Sun 3/50, it has already been used to generate multiagent solutions to blocks-world problems. Some of GEMPLAN's important characteristics are the construction of synchronized plans through the satisfaction of first-order temporal-logic constraints; the use of localized plan representations and localized planning search spaces; an adaptable, table-driven mechanism for guiding the planning search that can make explicit use of noninterference among localized constraints.[9]

GEMPLAN's task is to construct a world plan (i.e., a set of partially ordered, localized events) *all* of whose executions satisfy a given set of domain constraints and achieve some stated goal.[10] Given an initial world plan (possibly empty), the planner repeatedly chooses a domain constraint, checks to see whether the constraint is satisfied and, if it is not, either backtracks to an earlier decision point in the planning process or goes ahead and modifies the world plan so that the constraint will be satisfied. From a conceptual standpoint, the planning process may be viewed as a search through a tree (see Figure 3). At each node of the tree is stored a representation of the currently constructed world plan. When a node is reached during the planning search, a constraint is checked. To satisfy it, the search space branches for each of the possible ways of repairing or fixing the world plan. These "fixes" may involve the addition of new events, elements, groups, or event interrelationhips.

In this paper, we shall concentrate on describing GEMPLAN primarily from an architectural point of view, stressing its localization of the planning search process. However, since the development of constraint satisfaction algorithms is one of our key research objectives, it merits some brief discussion here. Because of the intractability of solving arbitrary first-order temporal-logic constraints, we decided that a good initial approach to the constraint satisfaction problem would be to use predefined fixes for common constraint forms. This approach 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 [2]. The current GEM-

[9] The current planner, however, does not make use of ports; it only takes advantage of the localization of constraints and the limitations imposed by group/element structure.

[10] The stated goal of the world plan is viewed as one of the constraints to be satisfied.

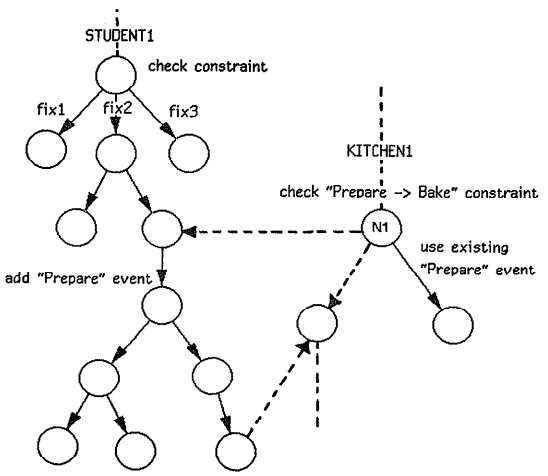


Figure 3: Localized Search Trees for the Cooking Class Domain

PLAN system can satisfy the constraints used in the blocks-world domain: event prerequisites, constraints based on regular-expression patterns of events, the maintenance of state-based preconditions,[11] and nonatomic-event expansion (into patterns of events). The planner also includes a facility for accumulating constraints on the values of unbound event parameters. We intend to add several constraint forms in the future, including various kinds of priorities, mutual exclusion, and simultaneity. This will enable GEMPLAN to handle more sophisticated forms of synchronization than other existing planners. To solve propositional constraints, we hope to utilize the algorithms conceived by Manna and Wolper [12] and implemented by Stuart [17].

The most important feature of GEMPLAN's system architecture is its partitioning of the planning search space and plan representation in a way that reflects the group/element structure of a domain. For each element and group there exists a local search tree and plan representation. The overall planning system may be viewed as a set of mini-planning systems, one for each region. In accordance with the structure of a domain, more global regions have access to their subregions' plans and search spaces. It is these "parent-child" connections that form the glue with which the entire planning system is tied together.

The plan descriptions associated with GEMPLAN tree nodes are built by using *inheritance*: only events and relations that represent *changes* from the parent node plan are stored. The entire plan at each node may thus be derived by following the plan inheritance chain, accumulating plan modifications along the way. This inheritance representation is not only compact, but is also well suited to localized plan representation; the same inheritance scheme can be used for consolidating local plan information to form more global plan descriptions. Each group plan is described as the union of a set of plans for each of its composite regions (which will include all local event occurrences and relationships resulting from the satisfaction of local constraints), along with any relations that are added by virtue of the global constraints.

As an illustration, consider the search trees depicted in Figure 3 – one for the `kitchen1` group of the cooking class, the other for its student subelement, `student1`. Let us assume that we have reached the node labeled `N1` for `kitchen1`, and that a set of `Bake` events has already been inserted into the world plan. At this point, we imagine that the following global `kitchen1` constraint is checked:

$$(\forall\ \text{bake(cake):Bake})(\exists\ \text{prepare(cake):Prepare})$$
$$\text{prepare(cake)} \rightsquigarrow \text{bake(cake)}\ .$$

In other words, each baking event must have been enabled by a student event that prepares a cake. Moreover, we also assume that another constraint allows each cake preparation to enable or cause only one baking event. If the first constraint has already been satisfied (i.e., all `Bake` events already have an corresponding `Prepare` event), the planner will move on to some other `kitchen1` constraint. If this is not the case, however, there are two ways to proceed. First, there may be an existing `Prepare` event that could be used – i.e., a cake has been prepared by some student, but has no corresponding `Bake` event. In this case, a causal relationship would be added between the existing `Prepare` event and the lone `Bake` event. The other fix is to generate a new `Prepare` event involving one of the students. This choice is illustrated in Figure 3 as a branch to the `student1` search space. At this point, the search space for `student1` is resumed where it had left off (in a state where all its internal constraints had been satisfied), the new `Prepare` event is added, and the student's local constraints are rechecked. After `student1`'s constraints have been satisfied, control returns to the `kitchen1` search space. Note that no rechecking of the local constraints for any other student, the teacher, or the oven, is necessary, since these could not possibly be affected by a `student1` event. However some global `kitchen1` constraints may have to be rechecked as a result of this change.

The actual order in which constraints and fixes are applied is determined by a *plan search table* for each local region. This table can be set up by a user to define quite flexible kinds of search. The table provides three types of information: (1) the order in which to apply constraints, (2) the order in which to try constraint fixes, and (3) when and where to backtrack. The particular constraint, fix, or backtracking scheme chosen at any point in time is context-sensitive – it can be determined by the particular situation at hand. Whenever backtracking occurs, the node left behind is still retained for possible later exploration. The search can thus use a mixture of depth- and breadth-oriented exploration, depending on the strategy determined by the table.

If a user does not supply domain-specific search information, a default depth-first search strategy is used. The constraints and fixes within each region are chosen in a given order. Chronological backtracking is used when a plan cannot be fixed. Planning halts when either no new options can be explored or all constraints have been checked successfully.

The use of the GEMPLAN search table has proved to be a quite powerful and flexible means of guiding the planning process. It can easily be constructed to take advantage of a domain's locality properties. When fixes modify the structure of a plan, rechecking for consequent interference with other constraints can be limited to those regions and constraints that could be affected. In contrast, most existing planning systems (which may also use "divide-and-conquer" methodologies) do not localize the domain description sufficiently and therefore cannot exploit the resulting properties of noninterference.

[11] The algorithm used is equivalent to an implementation of Chapman's *truth criterion* [3].

In addition, most planning systems search the plan space in a fairly rigid way – typically, local expansion to a uniform level of description followed by interaction analysis. In contrast, the GEMPLAN planning search can be flexibly tuned. Depending on the structure of a domain, the nature of its constraints, and the content of the search table, the search can be quite distributed and loosely coupled for regions with weak interdependencies, but tightly coupled when regions interact strongly. The table can also be set up to *focus* the search in prescribed ways. Researchers developing the ISIS scheduling system [5] have found resource- and agent-focused search to be useful for job-shop scheduling. In GEMPLAN, the search can be focused on *any* region, as long as the user has specified domain structure, regional constraints, and the search table appropriately.

5 Conclusions

This paper has presented an event-based formalism, GEM, for representing parallel, multiagent domains. GEM can explicitly describe arbitrary forms of domain structure as well as complex constraints on events and their interrelationships. We have shown how the behavioral limitations associated with GEM's structuring mechanisms (elements and groups) can be used to delineate event or property independence. We demonstrated how these implicit structural constraints, along with the localization of explicit constraints and the use of ports, can help solve aspects of the frame problem.

We have also presented the GEMPLAN planning architecture, which directly partitions plan representation and the planning search space according to the group/element structure of a domain. By employing a table-driven search mechanism, the planning process can be guided to take advantage of the locality and interactional properties of a domain. We have used this system to construct parallel solutions to several blocks-world problems; it is our intention to extend its application to more complicated scheduling domains in the near future.

Acknowledgments

We would like to thank those readers and critics who have helped to improve the quality of this paper: Michael Georgeff, Martha Pollack, David Wilkins, Mark Drummond, Steven Rubin, Marc Vilain, and Savel Kliachko.

References

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

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

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

[4] Drummond, M.E. "A Representation of Action and Belief for Automatic Planning Systems," in *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, M.P. Georgeff and A.L. Lansky (editors), Morgan Kaufman Publishers, Los Altos, California, pp. 189-211 (1987).

[5] 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).

[6] Georgeff, M. P. "Many Agents are Better Than One," in *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop*, F. Brown (editor), Morgan Kaufman Publishers, Los Altos, California (1987).

[7] Hayes, P.J. "The Frame Problem and Related Problems in Artificial Intelligence," from *Artificial Intelligence and Human Thinking*, pp. 45-59, A. Elithorn and D. Jones (editors), Jossey-Bass, Inc. and Elsevier Scientific Publishing Company (1973).

[8] 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), also appearing in *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, M.P. Georgeff and A.L. Lansky (editors), Morgan Kaufman Publishers, Los Altos, California, pp. 123-160 (1987).

[9] 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).

[10] 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).

[11] 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).

[12] 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).

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

[14] Pelavin, R. and J.F. Allen, "A Formal Logic of Plans in Temporally Rich Domains," *Proceedings of the IEEE, Special Issue on Knowledge Representation*, Volume 74, No. 10, pp. 1364-1382 (October 1986).

[15] Rubin, S.M., *Computer Aids for VLSI Design*, Addison-Wesley, Reading, Massachusetts (1987).

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

[17] 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).

[18] Tate, A. "Goal Structure, Holding Periods, and 'Clouds'," in *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, M.P. Georgeff and A.L. Lansky (editors), Morgan Kaufman Publishers, Los Altos, California, pp. 267-277 (1987).

[19] 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).

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