A Mission Plan Representation for Autonomous Reasoning of Consequences

Brian C. Becker
Avelino Gonzalez
University of Central Florida
Orlando, FL 32816
407-823-5027
Brian@BrianCBecker.com, gonzalez@mail.ucf.edu
Keith Garfield
Institute for Simulation and Training
3280 Progress Drive
Orlando, FL 32826
407-882-1342
kgarfiel@ist.ucf.edu
Keywords:
Autonomous reasoning, replanning, dependences

ABSTRACT *For an autonomous agent, the ability to assess the impact of an event on the agent's current course of action is essential if the agent is to react to events encountered during a mission. One aspect of achieving such a reasoning capability is to develop a representation for the mission plan amenable to assessment. We present the Mission Plan Dependence Graph (MPDG) and discuss its uses with respect to autonomous reasoning of consequences and replanning. This is an initial definition of the MPDG, which is part of larger reasoning model being developed. The MPDG represents a mission plan as a directed, acyclic graph explicitly representing actions, actions sequences, and dependences between actions. The model leverages concepts developed within the Program Dependence Graph program representation by applying them to planning graphs. The MPDG directly supports reasoning of primary and secondary effects of events on a mission plan, mission execution, and replanning operations. The MPDG also support multi-agent and multi-echelon planning environments.*

1. Introduction

Planning is essential to an action oriented, goal-driven agent. In general it is not possible to develop plans for all contingencies *a priori*. Therefore, replanning is an equally essential capability to allow the agent to react to events as the mission unfolds. While successes have been achieved within limited domains, the challenge remains to develop robust systems capable of operating within complex and dynamic environments. One aspect of achieving a generalized planning system is to develop a representation of a mission plan. The goals for such a representation are to represent plans concisely, allow queries regarding the impact of unanticipated events, allow operations to denote mission progress and replanning, and represent plans involving multi-agent cooperation.

This paper presents the *Mission Plan Dependence Graph* (MPDG), which is part of a larger *Autonomous Reasoning of Consequences* (ARC) model. The ARC is composed of a *World Model* (WM), an MPDG, and a *Reasoning Agent* (RA). The WM represents real world agents and the environment they operate in, and defines an agent's internal model of a real world environment. The MPDG represents a mission plan by graphically specifying a series of actions intended to accomplish some result in the WM and specific dependence relationships between them. These dependences are adapted from similar concepts developed for use in formal computational models, such as the Program Dependence Graph (PDG) [Horwitz and Reps, 1992]. The *Reasoning Agent* (RA) performs operations on the MPDG to assess plan status, determine how real time events may affect the plan, and perform replanning when necessary. In this paper we define the MPDG and provide an overview of its uses. The remainder of this paper provides an overview of mission planning approaches, an overview of the ARC model components and features, a formal definition of the MPDG, operations on the MPDG, and conclusions.

2. Background

Automated planning by software agents continues to be the focus of a great deal of study. In a typical planning problem, an agent is presented

with an initial state and a desired goal state and must develop a sequence of actions, or mission plan, to accomplish the goal state. Obtaining a globally optimal sequence of actions is not possible in the general case due to the inherent intractable nature of obtaining optimal sequences (the Traveling Salesman problem), access to incomplete or ambiguous information about the environment, and the potential for conflicting criteria defining mission success. The problem is aggravated by the dynamic nature of environments in which non-trivial planning tasks are carried out. Replanning is critical in environments where novel situations or unexpected events are encountered [Sapena and Onaindia, 2002, Interrante and Rochowiak, 1994]. This implies that an agent must have an ability to assess the consequences of new information or events on an existing mission plan, and be capable of modifying the plan accordingly.

Various paradigms have been attempted to impart mission planning capabilities to software agents. While each have strengths, none have achieved a robust capability for agents operating in complex, dynamic environments. Cognitive solutions attempt to mirror the processes of the human mind [Lehman et al., 1996]. These approaches have the advantage of allowing a human observer to query the agent's rationale for developing a particular plan. These solutions traditionally suffer from limitations on knowledge representation systems, coupled with potentially vast amounts of knowledge required to operate in a complex environment. Computational solutions, such as the Goals, Operators, Methods, and Selection (GOMS) model [Card et al., 1983], treat plan development as a search through (potentially large) solution space. Neural network solutions attempt to mirror the physical structure of the brain. Neural network solutions have had success in identifying patterns and trends indicating which course of action may be profitable. Genetic Algorithm (GA) techniques treat behavior development as a search for a near-optimal solution through a series of progressive candidate solutions that are treated as genetic material [Campbell et al., 2006]. Computational, neural network, and GA solutions tend to require large computational efforts that limit their usefulness in adapting to novel situations.

The MPDG is an outgrowth of the *planning graph* paradigm. Planning graphs grew out of a form of problem solving techniques that were in turn influenced heavily by theorem proving [Blum and Furst, 1997, Fikes and Nilsson, 1971]. A theorem proving problem is one in which a set of true statements or conditions are given as an initial condition, and a conclusion is reached through a sequence of logical operations acting on the known true statements (or other true statements derived from them). The parallel between achieving a goal state from an initial state through a plan of sequential actions is readily made. Blum and Furst's [Blum and Furst, 1997] *GraphPlan* planning system is a popular example of this paradigm that is discussed further in Section 5.

Complex environments may contain multiple cooperating agents attempting to accomplish a single task [Kalofonos and Norman, 2004, Campbell et al., 2006]. This may require that an agent be aware of another agent's capabilities and world view, a process known as perspective taking [Hiatt et al., 2004]. Emergent multi-agent systems investigate the utility of reactive agents producing sophisticated emergent behavior [Campbell et al., 2006].

Planning and replanning activities may be represented within formal systems through symbols and operations on these symbols. Desirable traits for such a system include that it is efficient in representation, amenable to queries by a human or software agent, represents constraints among competing actions, and supports operations allowing assessment and modification.

3. ARC Components

In this section we develop a notation to discuss and develop the MPDG. We begin by defining ARC WM components and their relationship to a mission plan. The ARC model, and more specifically the MPDG, describes real world events that are to occur in a specific sequence to attain some goal. Informally, a model is a mathematical approximation of a real world system representing a finite number of measurable features of interest of the real world system [Walton et al., 2003]. Models specify a set of attributes representing features of interest, and may be coupled with arithmetic/logical (A/L) functions that update the values of these attributes to represent the state and behavior (state changes) of the real world system being investigated. The WM contains individual attributes, collections of attributes (states), actions modifying attribute values, agents performing the actions, and mission plans designating sequences of actions to achieve a goal state.

We define a feature of a model, denoted as f, as a

measurable aspect of a real world system represented in the model (Definition 1). An attribute, denoted a, is a range of values that may be assigned to a feature [Petty and Weisal, 2003, Garfield, 2006]. We use the formal definition given in Garfield [Garfield, 2006] within the ARC model. This definition is in accordance with the requirement that simulations contain explicitly defined boundaries and constraints [Walton et al., 2003]. We refer to an individual value residing in an attribute as an *attribute value*, and denote it as α. We further denote that an attribute vector, A, is a tuple of attributes (Definition 2), and an attribute value vector, A_V, is a tuple of specific values from each attribute contributing to the vector (Definition 3).

Definition 1 (Feature) *A feature, f, is a representation of a measurable aspect of the real world system being studied.*

Definition 2 (Attribute Vector) *An attribute vector, A, is an ordered tuple of attributes, $A = (a_1, a_2, a_3, \ldots, a_k)$*

Definition 3 (Attribute Value Vector) *An attribute value vector, A_v, is an ordered tuple of attribute values, $A_v = (\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k)$, such that $\alpha_i \in a_i$, for $1 \leq i \leq k$.*

We define an *entity* within a scenario as a set of attributes representing a corresponding real world physical object or conditions. The association of a set of attributes with a single physical object implies that if a single attribute in the set is present in the scenario, then the entire set in present. For example, the addition of a UAV to a mission scenario requires that all attributes representing that specific UAV be added to the scenario. Similarly, the loss or removal of the UAV requires the removal of all attributes in the set.

We define a *state* as the values of an arbitrary set of attributes. A state of an entity at any point in a scenario is given by the values of the entity attributes. State attributes are not necessarily associated with a single physical entity, but are grouped according to measures of interest of mission progress and goals. For example, typical mission states of interest are the initial (or start) state, current state, and goal (or final) state. These states are used to define mission status within the WM, but may not all components may belong to the same entity.

Definition 4 (State) *A state, S, is an attribute value vector of interest to observers and participants in a mission plan.*

Actions and events are mechanisms within the ARC model enacting state changes. An action is composed of a designation of the acting agent, a pre-action state, a post-action state, conditions that must be met for the action to be performed, and a time duration. The ARC model utilizes discrete actions as primitives. These actions have clearly identifiable pre-action states, post-action states, and durations. Post-actions states are typically defined through a transformation function acting on a portion of the pre-action state. Continuous actions are approximated through iteration. For example, movement for a UAV may be defined through the primitive action Move, which translates the UAV through a unit of linear space and requiring some fixed unit of time, t. Continuous UAV movement is approximated through the iterated form Move(i), translating the UAV through i units of linear space and requiring $i * t$ units of time.

Definition 5 (Action) *An action, δ is a tuple $\{id, P(S_{COND}), S_o, S_f = F_T(S_o), dt\}$ where:*

1. *id is an identifier for the agent performing the action,*
2. *$P(S_{COND})$ is a Boolean predicate describing a conditional state, S_{COND}, that must evaluate to True before the action may be executed,*
3. *S_o, is an attribute vector describing state of the agent relevant to the action prior to the execution of the action,*
4. *$S_f = F_T(S_o)$, describes the final, or post-action, state of the agent relevant to the state as a transformation on the initial state, and,*
5. *dt represents the estimated time required for the action to complete.*

Agents are defined as a set of attributes and a set of actions available for the agent to perform. Thus, a surveillance UAV may be defined as a set of attributes denoting the limits of location coordinates, altitude, speed, fuel load, etc..., and a set of actions the UAV may perform to alter the state of these attributes within those limits. Related entities are *resources*, which consist of attributes but have no ability to act.

Resources may be produced, transferred, and consumed by agents through their actions. An *environment* is a subjective term, consisting of all entities and attributes external to a set of agents. Thus an environment is relative to the point of view of those operating in it. For example, to a team of UAVs the environment is composed of the terrain and surveillance targets, but to a single UAV the environment also contains the other UAVs.

Definition 6 (Agent) *An agent is defined by its attribute vector, A_V, and the set of actions, denoted Δ it may perform.*

We define a mission as a sequence of actions and decision points selecting between alternate potential courses of action. In a text format, a sequence of actions is created by listing actions in the order they are intended to occur. A decision point, P, is roughly comparable to an IF-THEN-ELSE structure, and is essentially an attribute, a, paired with a selector function, F_S. Upon execution, the selector function operates on an input attribute value vector and resolves to a single attribute value, $\alpha = F_S(A_{V input})$, $\alpha \in a$. The value of α determines which action sequences will be executed. For a single agent, missions are necessarily sequential, and only one action sequence will initiate as a result of the selector function. For multi-agent environments, multiple sequences of action may be initiated.

It is useful to be able to describe certain relationships between the actions and decision points in a mission. We borrow the notion of a *dependence* from the field of computer science [Kuck et al., 1981] to describe specific relationships between elements of a plan. Program dependences specify computational sequences that must be respected in order to maintain program meaning. Within the MPDG, mission dependences specify a partial ordering of action sequences that must be respected to maintain mission results. This has direct implications in reasoning about consequences of events, mission planning, and re-planning.

Flow dependences describe the relationship between producers and consumers of a resource or information. *Control* dependences describe the relationship between decision points and the actions resulting from resolution of these decision points. *Def-order* dependences are a result of an interaction between *flow* and *control* dependences. Mission dependences are formally defined below:

Definition 7 (Control Dependence) *Action δ_B is control dependent on Decision point P_A iff*

1) P_A is a mission decision point containing an expression that will resolve to a single value, $\alpha \in a$.
2) Exactly one value, $\alpha_B \in a$ results in δ_B's execution.

3) There are no intervening statements for which (1) and (2) apply to δ_B.

Definition 8 (Flow Dependence) *Action δ_B is flow dependent on δ_A iff δ_A contains some attribute, a, in its post-condition state, S_F, and δ_B requires that attribute in its pre-condition state, S_o, or predicate, $P(A)$, and there are no intervening actions on some action sequence path from δ_A to δ_B for which this applies.*

Definition 9 (Def-Order Dependence) *δ_B is Def-Order dependent on δ_A iff*

1) Both δ_A and δ_B both contain a value for attribute a in their final state set, S_F.

2) δ_A precedes δ_B in a strict execution sequence.

3) There is some action δ_C that is flow dependent upon both δ_A and δ_B for attribute a.

4. The Mission Plan Dependence Graph

The MPDG is an acyclic, directed graph composed of a node set and four edge sets. The node set is composed of a *Start* node, *action* nodes, *decision point* nodes, and an *End* node. The *Start* node contains an initial state of interest, S_i, and represents a unique point initiating the mission plan. *Action* nodes identify the agent performing the action, the pre-action state, and post-action state. *Decision* nodes represent decision points in the mission plan and allow a single action sequence to branch into two or more possible sequences as a result of a predicate, $P(A_V)$. The edge sets represent *sequence*, *flow* dependence, *control* dependence, and *def-order* dependence relationships within the mission. Temporal dependences are not explicitly represented in the MPDG, but are implied. For example, if action B requires a resource or information supplied by action A, then we imply that action B cannot start until action A is completed. This is implied within the MPDG by the presence of the *flow* edge from A to B. A simple example of a UAV surveillance mission having two possible flight paths and the corresponding MPDG is given in Figure 1.

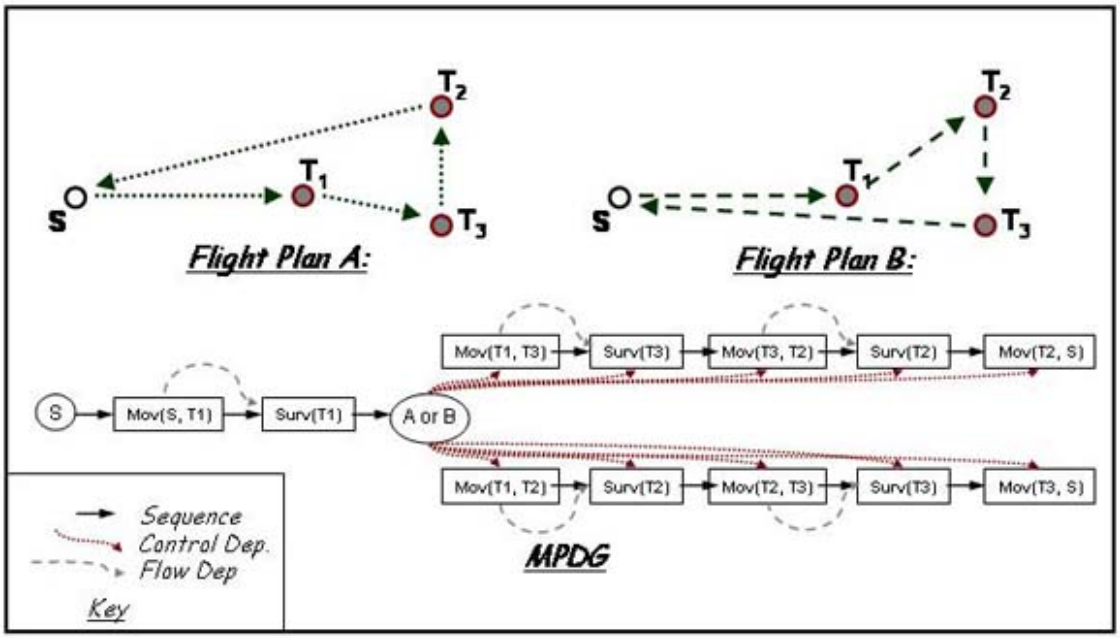

Figure 1: Example: MPDG Containing Two Possible Mission Paths

Definition 10 (Action Sequence) *An action sequence, S, is a tuple of actions indicating that each action occurs one after the other in a discreet fashion.*

5. Related Work

We briefly present work related to the MPDG.

Program Dependence Graphs The Program Dependence Graph (PDG) has achieved wide spread acceptance as a useful tool for software engineering, program analysis, and automated sequencing of program statements [Garfield, 2006, Horwitz and Reps, 1992, Zhao, 1999]. The PDG gains its utility by explicitly representing dependences (corresponding to those in Definitions 7, 8, and 9) between program statements. These dependences impose a partial ordering on sequences of statement execution that maintain program meaning [Kuck et al., 1981]. This provides the basis for "safe" program transformations exploring alternate execution sequences, which corresponds to exploring alternate sequences of actions comprising a plan. The PDG structure also allows rapid determination of program effects through traversal of the PDG edges. Slicing, described in Section 6, is an example of this type of analysis.

Planning Graphs A planning graph is a directed, leveled graph composed of two node sets and three edge sets [Blum and Furst, 1997]. Nodes are divided into *proposition* nodes, encoding a proposition about the system, and *action* nodes operating on the propositions. The two nodes sets reside at alternating levels in the graph, such that initial propositions reside at time one, followed by actions performed at time one, resulting propositions at time two, followed by actions taken at time two etc... The edge sets correspond to *preconditions*, *add-edges* , and *delete-edges*. Precondition edges connect propositions with actions within the same time step, while add- and delete-edges connect actions to their Add- and Delete-effects in the next time step. Propositions are carried forward in time through "no-op" actions.

The MPDG differs from the planning graph in several respects. Planning graph edges depict information flow between actions, while MPDG edges represent dependences between actions. The MPDG represents the passage of time as an attribute of an action, as opposed to a property of graph structure. The MPDG allows actions of past time steps to directly affect actions performed in any future step as opposed to carrying effects forward through "no-op" actions.

6. Operations on the MPDG

In this section we briefly describe operations on the MPDG and their utility in assessing consequences of action and potential replanning activities.

Determining Effects Determining if an existing course of action is affected by an event is critical to identifying conditions requiring replanning of the mission. Within the ARC model, actions affected by an event are identified in a straightforward fashion through set intersection operations. If the intersection of an action's action attribute set and the attribute set modified by an action or event is non-empty, then the action is affected. If the event is in the future, a backwards traversal of sequence edges will identify decision points in the plan where alternate courses of action may yet be pursued. The presence of dependence edges allows the reasoning agent to identify other actions affected in a secondary manner by the event (see *Slicing* discussion below).

The RA must also assess the degree to which an action is affected by an event. The degree of effect may be determined by evaluating durations and resource usages along action sequences. This allows sequential paths of actions to be selected based on specific criteria, such as minimizing time or resource usage to achieve the goal state. Since the MPDG is self-describing to this degree, the reasoning agent has available all information required to make the assessment.

Mission Execution and Real Time Replanning Mission execution may be represented directly within the MPDG through a process known as *graph rewriting* [Garfield, 2006]. Graph rewriting is a node by node process that transforms a graph through a set of rewriting rules. In the case of mission execution, a set of rewriting rules may be developed for the MPDG to update agent and mission states as actions are performed, trace which actions have been completed or are capable of being initiated, and track mission success criteria as the mission is executed. The rewriting rules describing mission execution will leave the graph intact and allow after action review to be performed using the MPDG as a record of events.

In contrast to rewriting, replanning may require that portions of the MPDG be removed and recreated. Replanning is assisted by the presence of the dependence edges, which impose a partial ordering on the actions, thus guiding the replanning process. *Lazy* execution schemes [Garfield, 2006] may be developed that initiate actions only after it has been determined that they contribute to achieving the goal state. Essentially, a demand for actions is propagated through the MPDG backwards from the goal state node, identifying which action and decision nodes contribute to the goal state. Changes in mission goals may be addressed in this manner. When such a scheme is in use, the MPDG may explicitly represent a number of potential actions and alternate courses of action, with the most effective sequences selected in real time as the mission unfolds.

If replanning is required, modifying the sequence of actions may be done in real time by modifying sequence edges. Since the dependence edges enforce a partial ordering on the action nodes, only useful alternate sequence paths are investigated. In cases where exact sequences are not important, sequence edges are omitted and "next action" are selected from a set of candidate actions that have no incoming dependence edges. In other cases, such as the "Traveling Salesman" problem, there may be no benefit in terms of dependence edge restrictions from selecting one action over another, but there may be benefits in terms of resource or time usage. As Traveling Salesman is a known NP-complete problem the MPDG is only as efficient as other representations in resolving optimal action sequences.

Aggregation and Dis-aggregation We borrow the terminology of *aggregation* from the Modeling and Simulation (M&S) community to describe the act of collapsing an action sequence into a single conceptual action. This definition of aggregation within the MPDG has direct correlation to *function composition* in software programming and *chunking* [Lehman et al., 1996] or *abstraction* [Nicolescu and Mataric, 2002] in machine learning systems. The inverse operation, *dis-aggregation*, explicitly represents the primitive and iterated actions comprising an aggregate action. Aggregation and dis-aggregation techniques allow for efficient, scalable representation of missions while maintaining fine grain visibility at selective points of interest in the mission [Tan et al., 2001, Interrante, 1991]. They also allow for mission phases to be represented contextually, such as in the Context Based Reasoning (CxBR) paradigm [Salva, 2003, Gonzalez and Saeki, 2001]. Aggregation and dis-aggregation mechanisms also allow a single behavior representation system to describe multi-echelon behaviors and plans. For example, a mission plan de-

veloped at the brigade level may include companies as agents with primitive and iterative actions scaled appropriately. A company commander developing company level actions required to fulfill the company's portion of the overall mission may plan using platoons and squads as agents, and then aggregate the results upward into the battalion plan. Similarly, a single MPDG may be processed to provide different views to users operating at different echelon levels.

We formally define aggregation for a single agent within the MPDG in Definition 11.

Definition 11 (Aggregation) *Given action sequence* $\delta_1, \delta_2, \ldots, \delta_k$*, we define an aggregate action,* $\delta_{AGG} = \{\ agent_i d,\ P_{\delta_1},\ S_{\delta_1},\ S_{\delta_k},\ t_1 + t_2 + \ldots + t_k\}$.

Mission Slicing The program slice was introduced by Weiser [Weiser, 1984] in the context of software programming and debugging. As devised by Weiser, program slice identifies portions of a program that potentially affects, or may be affected by, a specific point of interest in the program. Although originally introduced for debugging purposes, the software community soon realized their use in a wide range of applications such as parallelization, program differencing, program testing, complexity measurement, and reverse engineering [Tip, 1995, Zhao, 1999]. Numerous publications detail techniques for obtaining a program slice on a graphical program representations [Tip, 1995, Zhao, 1999, Walkinshaw et al., 2003, Liang and Harrold, 1998, Chen and Xu, 2001, Allen and Horwitz, 2003]. A slicing procedure operating on a graph extracts the nodes related, directly or indirectly, to a specific computation in the original program. The problem of creating a program slice is essentially a graph reachability problem [Tip, 1995]. The backwards slice is created by traversing dependence edges backwards through the dependence graph. The forward slice is created by traversing dependence edges forward from the point of interest.

Slicing can be applied directly to the MPDG such that a mission slice identifies mission elements that potentially affect, or may be affected by, a specific point of interest in the plan. Slices may be used to identify plan elements that may require modification based on the expectation of a change to future missio parameters. Similarly, the impact of failure or modification of a current plan element may be assessed in terms of future consequences. Additionally, tasks and action sequences that may be performed in parallel are identified as residing in separate slices.

7. Conclusions

We have presented a new representation for mission planning, the MPDG, explicitly representing mission activities and their dependence to one another. The MPDG borrows from well established and successful principles used in program representations, specifically the Program Dependence Graph. The MPDG contains information regarding specific sequences of action taken within the plan, and constraints in the form of dependence relations that ensure that re-planned sequences achieve the same result as the original plan. We have provided a technique, the mission slice, that allows agents to identify plan components that are directly and indirectly affected by events occurring as the plan is carried out. We have shown that the MPDG is amenable to multi-echelon planning scenarios and aggregate actions.

The MPDG inherently supports parallel operations, and thus multi-agent plans. Each action (node) within the MPDG is self-describing, allowing perspective-taking without requiring direct knowledge of the acting agent's world view or inherent capabilities. There are several opportunities for further research using the MPDG in multi-agent environments. In the immediate future, the representation is to be implemented and tested. Numerous problem domains and planning algorithms exist in the literature [Blum and Furst, 1997, Sapena and Onaindia, 2002], and may serve as benchmarks for comparison. Future scenarios may employ cooperating agents sharing a world view to some degree [Jennings, 1995, Durfee, 1993]. The MPDG may be expanded to include multi-agent primitive actions to accommodate these scenarios.

Smith et. al. [Smith et al., 2003] developed View Centric Reasoning (VCR) to reason about parallel events occurring in an environment containing distributed observers, each with potentially unique imperfect views of event sequences. This paradigm may be used in conjunction with the MPDG to reason about the degree of information transfer required between agents to ensure cohesiveness of action among distributed, cooperating agents. Interesting research questions along these lines include "How much information may be lost before a plan breaks down through lack of cooperation?" and "Can a plan be devised that is less susceptible to communication breakdowns?"

Temporal reasoning may be included as part of behavior selection [Likhachev and Arkin, 2001].

We have not explicitly included temporal reasoning within the MPDG, but imply temporal constraints through dependences. More research may be done to improve the granularity and clarity of temporal dependences between actions by applying a temporal reasoning system (such as the transitive relations developed by [Kovarik, 1994]) to the MPDG.

8. Acknowledgments

This work was sponsored, in part, by the US Army Research Laboratory under Cooperative Agreement W911NF-06-2-0041. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARL or the US Government. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[Allen and Horwitz, 2003] Allen, M. and Horwitz, S. (2003). Slicing java programs that throw and catch exceptions. *Proceedings of the ACM SIGPLAN 2003 Workshop on Partial Evaluation and Semantics Based Program Manipulation*.

[Blum and Furst, 1997] Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281, 316.

[Campbell et al., 2006] Campbell, A., Wu, A. S., Shumaker, R., Garfield, K., Luke, S., and DeJong, K. A. (2006). Empirical study on the effect of synthetic social structures on teams of autonomous vehicles. In *Proceedings of IEEE International Conference on Networking Sensing and Control*.

[Card et al., 1983] Card, S., Moran, T. P., and Newell, A. (1983). *The Psychology of Human Computer Interaction*. Erlbaum, Hillsdale, NJ.

[Chen and Xu, 2001] Chen, Z. and Xu, B. (2001). Slicing object-oriented java programs. In *SIGPLAN Notices*, volume 36, pages 33,40, New York, NY. ACM Press.

[Durfee, 1993] Durfee, E. (1993). Organizations, plans, and schedules: An interdisciplinary perspective on coordinating ai agents. *Journal of Intelligent Systems*, 3:157, 187.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theroem proving to problem solving. *Artifical Intelligence*, 2:189–208.

[Garfield, 2006] Garfield, K. (2006). *A Sparse Program Dependence Graph for Object Oriented Programming Languages*. PhD thesis, University of Central Florida.

[Gonzalez and Saeki, 2001] Gonzalez, A. J. and Saeki, S. (2001). Using contexts competition to model tactical human behavior in a simulation. In *Intelligent Proceedings of the Third International and Interdisciplinary Conference on Modeling and Using Context*.

[Hiatt et al., 2004] Hiatt, L. M., Trafton, J. G., Harrison, A. M., and Schultz, A. C. (2004). A cognitive model for spatial perspective taking. In *Proceedings of the Sixth International Conference on cognitive Modeling*, pages 354, 355.

[Horwitz and Reps, 1992] Horwitz, S. and Reps, T. (1992). The use of program dependence graphs in software engineering. *Proceedings of the 14th International Conference on Software Engineering*.

[Interrante, 1991] Interrante, L. (1991). A model for selective attention in monitoring and control reasoning tasks. In *IEEE International Conference on Decision Aiding for Complex Systems*.

[Interrante and Rochowiak, 1994] Interrante, L. D. and Rochowiak, D. M. (1994). Active rescheduling for automated guided vehicle systems. *Intelligent Systems Engineering*, 3(2):87,100.

[Jennings, 1995] Jennings, N. (1995). Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Journal of Intelligent Artificial intelligence*, (2):195.

[Kalofonos and Norman, 2004] Kalofonos, D. and Norman, T. J. (2004). An investigation into team-based planning. In *Proceedings of Systems, Man, and Cybernetics*.

[Kovarik, 1994] Kovarik, V. J. (1994). *An Efficient Method for Representing and Computing Transitive Closure Over Temporal Relations*. PhD thesis, University of Central Florida.

[Kuck et al., 1981] Kuck, D., Kuhn, R., Padua, D., Leasure, B., and Wolfe, M. (1981). Dependence

graphs and compiler optimizations. *Proceedings of the Eighth ACM Symposium on the Principles of Programming Languages*.

[Lehman et al., 1996] Lehman, J. F., Laird, J. E., and Rosenbloom, P. S. (1996). A gentle introduction to soar, an architecture for hmuan cognition. *Invitation to Cognitive Science*, 4.

[Liang and Harrold, 1998] Liang, D. and Harrold, M. (1998). Slicing object using system dependence graph.

[Likhachev and Arkin, 2001] Likhachev, M. and Arkin, R. C. (2001). Spatio-temporal case-based reasoning for behavior selection. In *Proceedings of the International Conference on Robotgics and Automation*.

[Nicolescu and Mataric, 2002] Nicolescu, M. N. and Mataric, M. J. (2002). A hierarchical architecture for behavior-based robots. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

[Petty and Weisal, 2003] Petty, M. D. and Weisal, E. W. (2003). A formal approach to composibility. In *Proceedings of the 2003 Interservice/Industry Training, Simulation and Education Conference*, pages 1763, 1772.

[Salva, 2003] Salva, A. (2003). Situational awareness through context based situational interpretation metrics. Master's thesis, University of Central Florida, Orlando, FL.

[Sapena and Onaindia, 2002] Sapena, O. and Onaindia, E. (2002). A planning and monitoring system for dynamic environments. *Journal of Intelligent and Fuzzy Systems*, 12:151–162.

[Smith et al., 2003] Smith, M. L., Parsons, R. J., and Hughes, C. E. (2003). View-centric reasoning for linda and tuple space computation. *IEEE Proceedings-Software, Special Issue on Communicating Process Architecture 2002*, 150(2):71,84.

[Tan et al., 2001] Tan, G., Ng, W. N., and Moradu, F. (2001). Aggregation/disaggregation in hla multi-resolution distributed simulation. page 0076.

[Tip, 1995] Tip, F. (1995). A survey of program slicing techniques. *Journal of programming languages*, 3:121–189.

[Walkinshaw et al., 2003] Walkinshaw, N., Roper, M., and Wood, M. (2003). The java system dependence graph. *Third IEEE International Workshop on Source Code Analysis and Manipulation*.

[Walton et al., 2003] Walton, G., Goldiez, B., Hofer, R., and Kaup, D. (2003). Mathematical foundations for modeling and simulation. *Proceedings of the SPIE*, 5091:310, 320.

[Weiser, 1984] Weiser, M. (1984). Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352, 357.

[Zhao, 1999] Zhao, J. (1999). Applying program dependence analysis to java software. *Proceedings of the International Symposium on software Engineering for Parallel and Distributed Systems*.

Author Biographies

BRIAN C. BECKER is earning a B.S. degree in Computer Engineering at the University of Central Florida.

KEITH GARFIELD is an information specialist for the Institute for Simulation and Training at UCF. He earned his Ph.D. from UCF in 2006. He is researching human-agent interactions, behavior representations, and formal models of simulation.

AVELINO GONZALES is a professor in the Department of Electrical and Computer Engineering, and acting director of the Department of Civil Engineering, at UCF. He earned a Ph.D. in electrical Engineering from the University of Pittsburgh and is an IEEE Fellow. His research interests include machine learning, and validation and verification of knowledge-based systems.