Exploiting Domain-Specific Knowledge To Refine Simulation Specifications*

David Pautler
Steven Woods
Alex Quilici

Department of Electrical Engineering
University of Hawaii at Manoa
Honolulu, HI 96822

Abstract

This paper discusses our approach to the problem of refining high-level simulation specifications. Our domain is simulated combat training for tank platoon members. Our input is a high-level specification for a training scenario and our output is an executable specification for the behavior of a network-based combat simulator. Our approach combines a detailed model of the tank-training domain with non-linear planning and constraint satisfaction techniques. Our initial implementation is successful in large part because of our use of domain-knowledge to limit the branching factor of the planner and the constraint satisfaction engine.

1 Introduction

There has been considerable work in turning specifications (either formal or informal) into executable programs. Some systems use algebraic techniques and design tactics to synthesize and refine data structures, representations, and algorithms that provide solutions to scheduling and optimization problems [6, 8]. Other systems use deductive program synthesis [11] to combine components to construct physical or numerical simulations [10, 4].

We have been working on a different but related problem. Our focus is in refining high-level specifications into more detailed specifications. Specifically, we concentrate on forming executable simulation scripts from high-level, partial simulation specifications. This paper presents our mechanism for doing so, which relies heavily on domain modeling, non-linear planning, and constraint satisfaction techniques.

*This project has been funded by the DARPA CAETI program, under contract number N66001-96-C-8502.

2 The Problem

The Army currently does much of its tank training using ModSAF (Modular Semi-Automated Forces), a distributed, interactive, combat simulator [3]. In ModSAF, different battlefield actions are programmed in as parameterized tasks. For example, one task might be a particular type of helicopter attack, and its parameters include how many helicopters are involved in the attack, their initial location, the actual location of the attack, and so on. ModSAF provides an architecture for controlling the execution of these tasks and their interaction with human-generated actions (such as firing a weapon within a physical tank simulator).

To set up a particular simulation, it is necessary to specify exactly what simulated forces are present, which tasks they should execute, when these tasks should be executed, and so on.[1] The problem is that these detailed, formal specifications must be formed from less-detailed, relatively informal training specifications.

Figure 1 shows a typical specification for a tank platoon training exercise. It loosely specifies where the scenario should take place (a general region), what formations and battle actions are required (some actions that should definitely be trained for), and how long the scenario should last (a time range). These training requirements essentially specify a minimal set of actions to execute and a set of constraints on those actions (such as the locations where they occur).

The focus of our work is automatically turning these high-level specifications into a concrete script for a training scenario. This scenario contains detailed specifications of the actions of the simulated forces, as well as orders to be given to the human participants in the scenario.

Figure 2 shows an example of a refined specification (in

[1] Automatically generating the implementation of these tasks from specification of behavior and doctrine is an open research question currently being addressed by others [1].

A. Use the region in map "NK" bounded by (180, 370) in the lower left corner and (280, 500) in the upper right corner.

B. Start the WHITE platoon in the HERRINGBONE formation with no enemy unit firing on WHITE.

C. Include at least the following movement techniques and battle actions:
   COLUMN-TRAVELING-FORMATION
   CONTACT-WITH-SMALL-ARMS-FIRE
   DEFEND-AGAINST-AIR-ATTACK

D. Terminate with WHITE in a COIL formation.

E. The exercise should last between 60 and 180 minutes.

Figure 1: An informal training specification.

A. Make start at (200, 380) and end at (275, 455).
   Forces platoon through narrow gap, necessitating COLUMN formation.

B. Place no enemy units near start position.
   Avoids starting simulation with trainees being fired upon

C. Order platoon to assemble in HERRINGBONE formation.

D. Place SMALL-ARMS-FIRE enemy unit at (245, 400), aiming North.
   Forces platoon to pass this enemy unit triggering CONTACT-WITH-SMALL-ARMS-FIRE drill.

E. Place HELICOPTER enemy unit at (260, 450), aiming SouthEast.
   Forces platoon to pass this enemy unit triggering DEFEND-AGAINST-AIR-ATTACK drill. Also puts mountain between enemy and platoon, for enemy helicopter cover, and a forest near by, for platoon cover.

F. Order platoon to move out of its stopped formation when helicopters have cleared.

G. Order platoon to move toward end position.

H. Order platoon to stop in COIL formation.

Figure 2: An overview of the resulting, refined specifications.

English, for readability). The refined specification includes enabling assembly and movement actions, specifies the order and location of actions, and places of enemy forces to allow those actions.

While our particular problem deals with combat simulation, we are addressing the more general question of refining specifications for systems to be constructed from components. The refinement process determines additional components, what their parameter values should be, and how they should be stitched together.

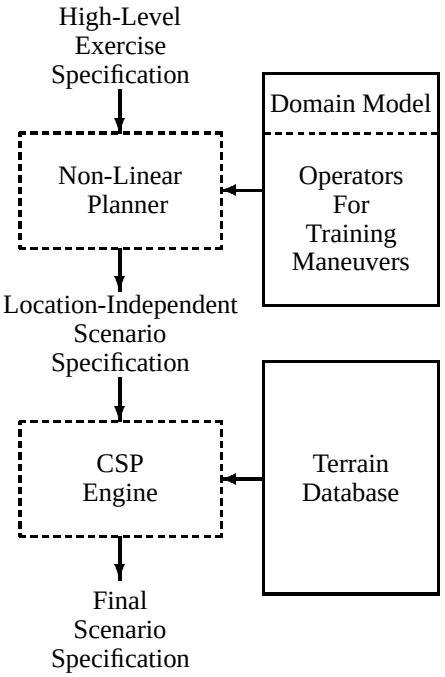

Figure 3: An overview of our architecture.

3 Overview Of Our Solution

We have constructed an environment that provides a forms-based tool for specifying a high-level training exercise specification (desired actions, starting and ending regions, and so on) and produces a detailed exercise specification meeting these requirements. If, for some reason, the resulting specification isn't suitable, the user can modify the original exercise specification and repeat the process to generate a new, more appropriate, simulation specification. For example, the initial specification may fail to exclude actions that are actually undesirable for this exercise (e.g., do not use a WEDGE-TRAVELING-FORMATION), which may result in their being used to flesh out the original specification. The user can then mark those actions as undesirable and repeat the process to generate new specifications.

Figure 3 shows the architecture of our solution. We divide the task of refining specifications into two parts: determining which actions should take place, and then determining where these actions should take place. To determine which actions should take place, we use a domain model representing actions as planning operators and apply standard non-linear planning techniques. To specify where these actions should take place, we use a terrain database describing properties of different terrain locations and apply constraint-satisfaction techniques.

The motivation for this architecture is that there are two different types of enabling conditions on actions. One type can be fulfilled by applying other operators. An example is the enabling condition that one must be moving before initiating an attack. We achieve this type of enablement by using planning to form a set of partially-ordered and terrain-independent actions. The other type cannot be satisfied by applying other operators. It is exemplified by enabling conditions involving terrain constraints, such as that tanks cannot travel over mountains. We ensure that these enabling conditions are satisfied by using constraint-satisfaction techniques to fit planned acts to scenario terrain.

4 The Domain Model

Our domain model is a collection of operators describing each of the actions in the tank-training domain. Each action corresponds to a currently simulated ModSAF task, and a high-level user specification may indicate that one or more of these tasks must be executed. Figure 4 lists these actions, which can be loosely divided into movement techniques (different ways of getting from one place to another) and battle actions (different ways of engaging an enemy).

Following standard planning practice, each operator is represented in terms of its enablements and effects [2]. Figure 5 contains an example, the operator NORMAL-TRAVEL-METHOD (shown in English, rather than our frame-based representational language).

The NORMAL-TRAVEL-METHOD operator illustrates the different enabling conditions mentioned earlier. One of its enabling conditions is that the platoon has moved out from its starting position, an enablement that is filled by any Move-Out action. Another enabling condition of this operator is that the path must be clear from start to finish, an enablement that can be filled only by placing the execution of the operator in a particular subset of terrain locations. Terrain-related enabling conditions on other operators specify things such as the needed width of a path, the type of nearby terrain, whether the path is clear, whether an enemy is visible or invisible, and so on.

5 Refining Specifications

Figure 6 sketches our algorithm for refining the initial specifications into a simulation script. Essentially, we divide the process into two tasks: determining which actions are necessary to fulfill the original specification and anchoring each action to particular place in the terrain.

Move-Out(platoon, stopped-pos)
Move-out-from-stationary-formation
Move-out-from-halted-traveling-formation
Air attack requires platoon to halt ASAP, usually before it has time to form a stationary formation (stationary formations are more secure than traveling formations)

Assume-Travel-Formation(platoon, start, end)
Travel-in-column-formation
Columns provide the least security but allow movement through narrow passageways.
Travel-in-wedge-formation
Most secure and most favored formation.
Travel-in-staggered-column-formation
Less secure than wedge, but narrower
Travel-in-echelon-formation
Maximum security to either flank
Travel-in-vee-formation
Good security for flanks, and narrower than echelon
Travel-in-line-formation
Less secure than wedge, but preferred for assaults

Employ-Travel-Method(platoon, start, end, formation)
Normal-travel-method
Travel-with-overwatch
Slower but more secure, so used when attack expected.
Travel-with-bounding-overwatch
Even slower and even more secure.

Engage-Enemy(platoon, start, end, formation, travel-method)
Contact-with-small-arms-fire-drill
Platoon continues moving while firing on source of small arms fire. Small arms units tend to hide in cover.
Defend-against-air-attack-drill
Platoon fires on helicopter, and then veers from path to find cover ASAP. Helicopters fire while popping up from behind small mountains.
Change-of-direction-drill
Change-of-direction-to-assault-enemy-drill
Change-of-direction-to-avoid-obstacle-drill
Change-of-formation-drill
React-to-indirect-fire-while-stopped-drill
When stopped platoon receives shower of fire, it either returns fire or moves out from its position.

Complete-Movement(platoon, end, formation, method, drill)
Move-To-Leg-End-Following-Bounding-Overwatch
Move-To-Leg-End-Following-Non-Bounding-Overwatch

Terminate(platoon, end)
Stop-in-coil-formation
Most secure stopped formation, but longer to set up than herringbone or a halting during a traveling formation.
Stop-in-herringbone-formation
Halt-in-travel-formation

Figure 4: The operators in our domain model.

Operator-Normal-Travel-Method

Arguments:
- Platoon
- Current-Travel-Formation
- Start-Pos
- End-Pos

Enablements:
1. Platoon has moved from Start-Pos.
2. Path between Start-Pos and End-Pos is clear.
3. Forward-Fire-Capability as determined by Current-Threat-Formation is good to excellent.

Effects:
1. Move from Start-Pos to End-Pos with excellent speed and moderate security.
2. Previous-Formation is now Traveling-Formation.
3. A Battle-Action can happen between Start-Pos and End-Pos.

Figure 5: A detailed look at one operator in our domain model.

Determine Actions

A. Assign the operator corresponding to each initially required action to a leg (resulting in the partial instantiation of one or more legs).

B. For each leg, invoke a planner to instantiate the unspecified operators required for each leg so that the enabling conditions for required actions are fulfilled.

C. Concatenate the individual legs together to form a complete set of actions (resulting in a complete, ordered specification of all actions to be executed).

Map Actions To Terrain

D. Form a constraint satisfaction problem by running through each of these instantiated actions, adding unfulfilled enablements as constraints, and adding unspecified parameters as variables.

E. Solve the resulting constraint satisfaction problem (resulting in the specification of locations for each action).

Figure 6: Our approach to refining simulation specification.

5.1 Forming A Complete Action Specification

Our approach to refining scenario specifications starts by forming a revised specification that apportions each required action to a *leg*. An instantiated leg corresponds to a sequence of operators, where each operator plays a particular role, called a *stage*. Figure 7 is a detailed description of the six stages in a leg, which we have already used implicitly to group the operators in Figure 4 (i.e., Move-Out, Assume-Travel-Formation, Employ-Travel-Method, Engage-Enemy, Complete-Movement, and Terminate). Each of the operators in our domain model is classified as belonging to one of these stages. In an instantiated leg, a stage can be filled by one and only one operator.

Move-Out: Do an action that involves moving from a stopped position into a standard stationary formation, such as a COIL or HERRINGBONE.

Assume-Travel-Formation: Do an action that involves moving into a particular type of traveling formation, such as a WEDGE or COLUMN.

Employ-Travel-Method: Possibly augment the travel method with variants of a traveling method for greater security or speed, such as switching to an TRAVELING-OVERWATCH.

Engage-Enemy: Participating in a standard battle action, such as firing and looking for cover, in response to encountering enemy units.

Complete-Movement: Move to the end point of the leg.

Terminate: Optionally, stop in a standard stationary formation, although this may only be a temporary stop before starting on a new leg.

Figure 7: The stages in a leg.

As an example, the three specified actions in Figure 1 (MOVE-COLUMN, SMALL-ARMS-FIRE, AIR-ATTACK) can be apportioned to two legs (MOVE-COLUMN as the Employ-Travel-Method and SMALL-ARMS-FIRE as the Engage-Enemy in one leg, AIR-ATTACK as the Engage-Enemy in another leg). We can't place all these actions in a single leg, as there are two Engage-Enemy operators.[2]

Apportioning each required action to a leg segments the task of determining the necessary actions into largely independent parts that can be planned for independently, greatly simplifying the planning problem. That is, after placing each action in a leg, we can generate a plan by fleshing out the remaining actions in that leg. We then recombine the legs into a sequence of actions to determine the complete set of actions for the users desired scenario.

5.2 Specifying Action Locations

At this stage, we have refined the initial specification to a set of actions. To complete the process of refining the specification, we need to anchor each action to the terrain. We do this by treating the assignment of actions to locations as a constraint satisfaction problem [12].

A constraint satisfaction problem (CSP) typically consists of three major components: a set of variables, a finite set of domain values for each variable, and a set of constraints among the variables that restrict domain value assignments. A solution of a CSP is a set of domain-value-to-variable assignments such that all inter-variable constraints

[2] In general, if an exercise involves two traveling formations or two battle actions, they must occur on different legs, which accurately represents how real-world training acts occur.

are satisfied. There are a variety of well-understood methods for solving these problems [13].

The variables in our CSP represent the start and end points of exercise legs and the positions of enemy units and terrain features. As a result, these variables take terrain locations as values. For example, a Defend-Against-Air-Attack requires a location where the attack should take place, as well as a location for where the attacking aircraft can be based. The location constraints in our CSP correspond to a subset of the enablements of the operators and include things such as a point being a particular type of terrain (e.g., a forest). Figure 8 lists the most important of the currently required spatial constraints.

The CSP itself is formed by running through each plan operator and gathering up the enabling conditions that are relevant to spatial locations.

Once the CSP is set up and solved, all actions have been bound to specific locations. The result is a refined specification that adds the necessary but unmentioned actions and that binds all actions to specific terrain locations.

5.3 The Justification For Our Approach

Any plan-based technique must face a potential combinatorial explosion of possible interactions between operators, effects, and enablements. Our work is making use of two well-known techniques to control the complexity of planning problems: hierarchical decomposition and hierarchical approximation [14, 9].

Hierarchical decomposition involves breaking a planning problem into actions at different levels of abstraction. This structure allows the planner to initially form plans using high-level operators and to then use the constraints that arise from forming these high-level plans to more efficiently form low-level plans. We are applying this technique in our use of legs and stages. A leg is an abstract, partially-specified, domain-specific plan, where each stage is a set of possible operators that share similar characteristics. This reduces our planner's job to binding each stage to an operator in a consistent fashion, as opposed to trying to work from scratch to find a suitable sequence of operators.

Hierarchical approximation involves planning with simplified versions of the available operators and progressively making them more detailed. We use hierarchical approximation in terms of ordering enablements by their criticality. The idea is first to plan for the enablements that are the most difficult (or critical) to achieve: we focus first on satisfying movement and action preconditions, and then we focus on preconditions related to spatial constraints. This ordering arises from our assumption that for our problem domain the spatial constraints are the least critical.

Node Constraints (constraints on a single point).

Terrain(type): The location must be a particular type of terrain (plain, forest, mountain).

Radial-Space(n): There must be radial space of size N at the given location.

In-Region(r): The location must be located in a particular square region.

Empty: The location must be empty.

Enemy-Unit(type): There must be an enemy-unit of the specified type at the given location.

Arc Constraints (constraints between two points).

Leg-Clear: No obstructions between two points.

Leg-Lateral-Space(w): Path between points has a minimum width.

Min-Separation(d), Max-Separation(d): There must be at least (at most) a specified separation between points.

Same-Terrain: Both points share the same terrain.

Direction(d): The first point must be in the specified direction (East, West, North, South) from the second.

Figure 8: Some of our current spatial constraints.

6 Implementation

We have completely implemented the approach discussed within this paper using by pre-existing Lisp-based implementations of a planner [16] and a constraint satisfaction engine [15] we had developed for earlier projects. The system produces both a textual specification and a visual depiction of the scenario, as shown in Figure 9.

We have generated a variety of detailed scenario specifications using the 24 operators described here and using a terrain database detailing the armor school training ground at Fort Knox. The operators have between 3 and 10 enablements and 4 to 7 effects each. It takes on the order of 10 seconds per leg to determine which actions should be included in the plan corresponding to that leg, and it takes on the order of a minute to locate these actions by solving the resulting CSP (using a standard Forward-Checking with Dynamic-Rearrangement CSP solver). This suggests that our breakdown of specification refinement into two distinct tasks has led to an efficient overall refinement process.

Forming and representing the current domain model has taken approximately six man-months of effort. Designing and implementing the system on top of the existing tools has taken another six man-months of effort.

7 Applications To Software Synthesis

We have applied our approach only to the problem of refining simulation specifications. However, it may well be

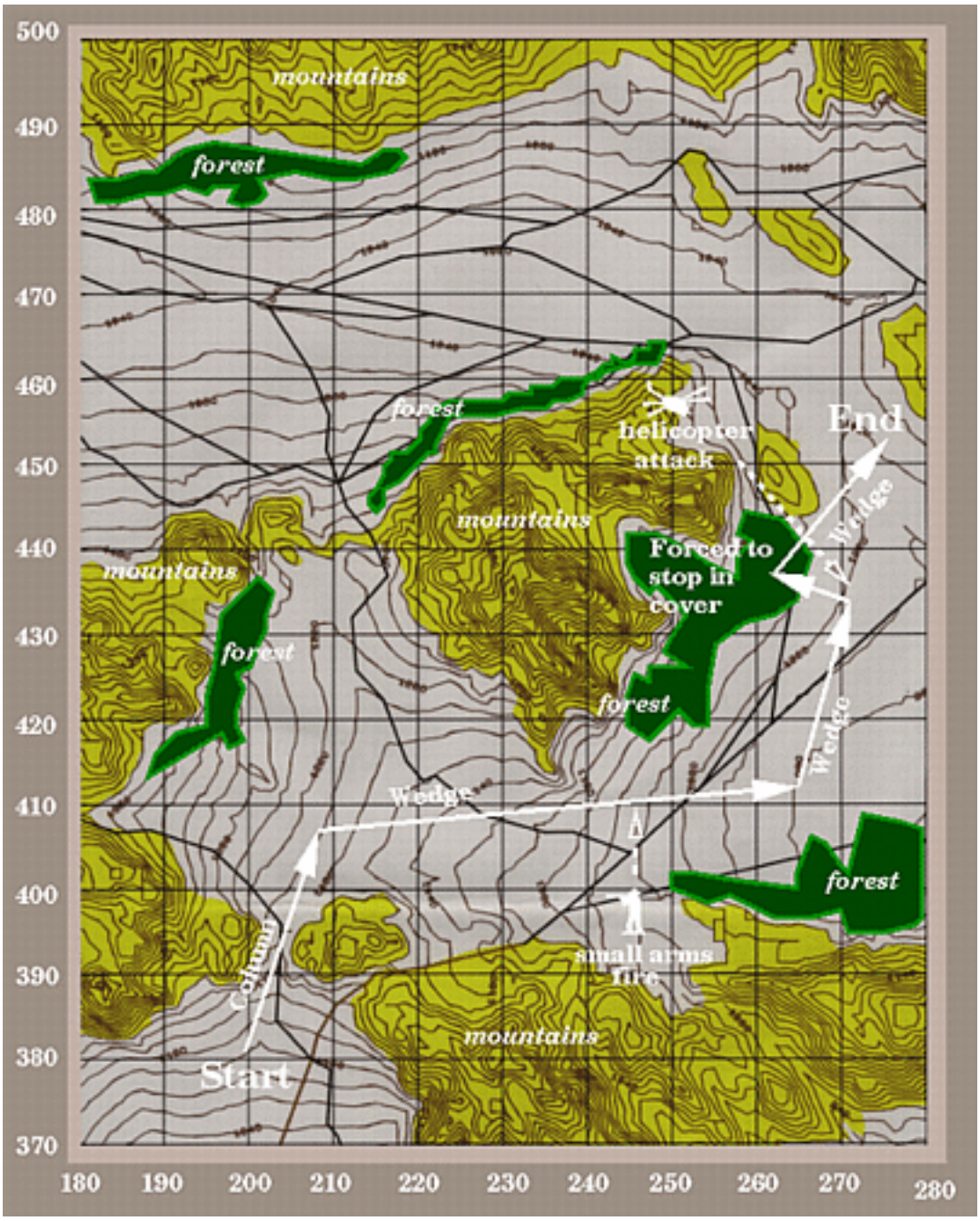

Figure 9: A visual depiction of a refined scenario specification.

applicable more generally to synthesizing applications from software components. In particular, our "legs" correspond to high-level, abstract scripts of how different types of components might fit together to perform high-level tasks, and our initial specification consists of requests to use some particular components. Our "terrain database" corresponds to a list of resources available to these components as they are executing (such as machine types, network connections, buffers, memory, and so on).

In this view, our planner is taking a set of requested components (e.g., particular actions on a data stream) and then determining what other components are needed, what their parameters should be, and how they all fit together. It does so by representing each component as an operator, with a set of preconditions that must be established by executing other components (e.g., the requirement that certain attributes must have been computed) and an additional set of conditions that must be true about the world in which the component executes and that cannot be achieved by other components (e.g., the requirement that certain resources, such as a particular network topology or message passing scheme be available). Our constraint satisfaction engine would make sure that these components can be assigned suitable resources.

8 Future Work

We are currently working on addressing the limitations of our current system and on applying our approach to additional domains.

8.1 Addressing Current Limitations

Despite our current system's successful implementation, it has several limitations that need to be addressed.

One minor problem is that we have not yet completely captured the domain of tank-related combat. In particular, there are six other battle actions that are used in current training that we need to incorporate into the domain model. We estimate that extending our domain model to include operators for these actions will represent approximately two additional man-months of effort.

Another minor problem is that we currently restrict what is allowable in terms of the initial, high-level specification. We are in the process of loosening these restrictions to allow the user to provide additional ordering constraints on actions, to request that an action be executed a particular number of times or in a particular region, and to provide partial parameters for selected actions.

A more major problem with our current approach is that it assumes that any specified set of actions can be fixed to terrain locations. While this assumption seems to hold for the specific high-level scenario specifications we have tried, with a more restrictive terrain database it's possible that our algorithm would be unable to find locations for a planned set of actions, while another set of actions that met the same initial specifications would have been suitable. One approach to extending our architecture to handle this situation is to iteratively try to generate new action specifications when an attempt to fit a plan to the terrain fails. This can be done by trying alternative ways of binding the initially specified actions to legs, as well as trying to find multiple solutions to the bindings with a given leg.

One final problem is that our current approach assumes we are specifying the behavior for the training exercise for a single platoon. A harder problem is to specify interacting training exercises for multiple platoons on the same terrain database. As a result, we are now looking at extending our work to generate simulation specifications that involve multiple interacting scenarios. Our approach to tackling this problem is to add an intermediate "plan merging" [5, 17] step between specifying actions and fitting the actions to the terrain.

8.2 Applying To Additional Domains

We are now working on applying our approach to two new domains.

The first target is to perform specification refinement for a visual data-flow programming system we had developed previously for constructing telemetry processing systems [7]. In this system, the user uses an annotated data-flow graph to visually specify components that are required, their relationships to other components, and type constraints on the information flowing between components. The system helps the user to refine this specification by suggesting candidates for unspecified components that meet the user's original specification. Our plan is to apply our approach to completely automate refining the specification.

The other target domain is generating detailed scene specifications for video games. In many video games, the user is supposed to move from one point to another within a particular artificial world, interacting with or avoiding a variety of different hazards. The elements of this world are often programmed as a set of objects, and the game internally has an architecture for controlling the invocation of operations of these objects and the interaction of these objects with human-generated actions (e.g., moving a joystick or pushing a button). As a consequence, there are many similarities between our simulation specifications and video game scene specifications. Our plan is to take a high-level specification of desired character actions and relationships and to generate a detailed, executable specification of the actions that should happen in the scene. The user can then

observe the scene behavior and use those observations to determine how to change the original specifications.

9 Conclusion

This paper has discussed the architecture of a system to refine high-level specifications for combat training simulation into detailed, executable simulation scripts. The system's overall architecture treats refining simulation specifications as a planning problem and makes use of a detailed, operator-based model of the tank training domain. The model can efficiently refine high-level training specifications into detailed specifications for the scenarios currently used in the Fort Knox armor training school.

We rely on domain-specific implementations of general AI planning heuristics to control the large search space typically encountered while refining specifications. In particular, we divide up enablements into those that can be planned for directly and those that can be represented as constraints on the world, and we use the domain-specific abstraction of "legs" to further reduce the planning problem. This effort provides additional evidence of the utility of applying domain-specific knowledge in moving from simulations to executable code.

References

[1] Balzer, B. Generating Semi-Automated Synthetic Forces. 1996 Computer Generated Forces Working Group, Orlando, Fl, 1996.

[2] Chapman, D. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333-377, 1987.

[3] Ceranowicz, A., "Modular Semi-Automated Forces", Technical Report. The Loral Advanced Distributed Simulation Group, Cambridge, MA, 1995.

[4] Ellman, T. and Murata, T. Deductive Synthesis of Numerical Simulation Programs from Networks of Algebraic and Ordinary Differential Equations. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*, Syracuse, NY, Sept. 1996.

[5] Foulser, D. and Li, M. and Yang, Q., A Quantitative Theory of Plan Merging. In *Proceedings of the 1991 AAAI Conference*, Anaheim, CA, pp. 673–679, 1991.

[6] Gomes, C., Westfold, S., and Smith, D. Synthesis of Schedulers for Planned Shutdowns of Power Plants. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*, Syracuse, NY, Sept. 1996.

[7] Gorlick, M. and Quilici, A. Visual Programming-in-the-Large versus Visual Programming-in-the-Small. In *Proceedings of the 1994 Visual Languages Symposium*, St. Louis, MO, October, pp. 137-144, 1994.

[8] Graham, R., and Bailor, P. Synthesis of Local Search Algorithms by Algebraic Means. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*, Syracuse, NY, Sept. 1996.

[9] Knoblock, C. Search reduction in hierarchical problem solving. In *Proceedings of the 1991 AAAI Conference*, pp. 686-691, 1991.

[10] Lowry, M., Philpot, A., Pressburger, T., Underwood, I. A Formal Approach to Domain-Oriented Software Design Environments. In *Proceedings of the Ninth Knowledge-Based Software Engineering Conference*, Monterey, CA, Sept 1994, pp. 48-57.

[11] Manna, Z., and Waldinger, R. Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, 18(8):674-705, 1992.

[12] Tsang, E. *Foundations Of Constraint Satisfaction*. Academic Press Limited, 24-28 Oval Road, London England, NW1 7DX, 1993.

[13] VanBeek P. and Kondrak, G. A theoretical evaluation of selected backtracking algorithms. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pp. 541–547, 1995.

[14] Wilkins, D. *Practical Planning: Extending the Classical AI Planning Paradigm* Morgan Kaufman, Palo Alto, CA, 1988.

[15] Woods, S. 1996. *A method of program understanding using constraint satisfaction for software reverse engineering.* Ph.D. Thesis, Department of Computer Science, University of Waterloo, Waterloo, Canada.

[16] Woods, S. *An implementation and evaluation of a hierarchical non-linear planner.* Master's Thesis. Technical report CS-91-17. University of Waterloo, Ontario, Canada, 1991.

[17] Yang, Q. *Intelligent Planning - Algorithms and Analyses for Plan Reasoning*, Springer Verlag, New York, New York, 1997.

[18] Yang, Q., Tennenberg, J. and Woods, S. On the implementation and evaluation of Abtweak. *Computational Intelligence*, vol 12., 1996.