Tactical Behavior Composition

Evan C. Clark
Joel C. Eichelberger
Physical Science Laboratory
New Mexico State University
Las Cruces, NM 88003
575-496-9915
research@evanclark.net, eichel@psl.nmsu.edu

Jeffrey A. Smith
U.S. Army Research Laboratory
Survivability/Lethality Analysis Directorate
White Sands Missile Range, NM 88002-5513
575-678-1332
jeffrey.a.smith@us.army.mil

Keywords:
tactics description language, commander agent, tactical decision making

ABSTRACT: *Behavior composition for computer generated forces is a technique that facilitates the creation and validation of agent behavior. It refers to the practice of creating reusable primitives that can be combined to construct new complex agent behaviors. Research in behavior composition has often focused on the use of procedural primitives. This paper discusses a framework for commander agent behavior composition that includes not only procedural primitives, but also those representing tactical concepts such as spatial relationships, subordinate coordination, terrain analysis, firepower and mobility. These primitives give the domain expert the ability to influence the manner in which tactical decisions are made. These primitives are elements of a tactics description language called Tesla Using the Tesla language, a tactical behavior expert composes tactic templates which can later be used by commander agents in course of action development and to solve tactical problems.*

1. Introduction

Both military modeling and simulation and commercial gaming require software agents that can solve tactical problems. For both industries, realism and immersion are enhanced when commander agents can dynamically adapt to tactical challenges in a reasonable way. However, because the current level of artificial intelligence technology does not permit a software agent to derive its tactical behavior from first principles, some medium is required to facilitate the transferral of tactical expertise from domain experts to software agents.

One technique that has been developed to facilitate this transferral of domain expertise is behavior composition. This technique has been used to allow a domain expert to directly configure the actions an agent will undertake.

This paper describes an approach to agent behavior configuration that extends the number of things a domain expert can specify, giving him or her a greater influence not only on what actions an agent performs but also on how it performs them.

Section 2 motivates this approach by discussing the advantages behavior composition systems already enjoy. Section 3 gives a general overview of the Tesla language and its use in agent configuration. Section 4 provides an example of using this approach. Section 5 describes Tesla's composition primitives. Section 6 discusses the implications of this approach on testing and validation.

2. Background

In the context of commander agent configuration, behavior composition refers to the practice of combining reusable primitives to construct new complex agent behaviors. What constitutes a primitive may vary by echelon and from system to system, but in all cases, a primitive refers to functionality implemented in source code and packaged up so as to be available to an editor application or scripting engine.

Behavior composition is used as an alternative to specifying all agent behavior in code, providing more productive roles for software engineers and domain experts alike. In such an arrangement, software engineers develop behavior primitives rather than *ad*

hoc complex behaviors. It is the nature of these primitives to be modular, encapsulated and reusable (Fu, 2003) (Reece, 2004). Modular and encapsulated code is easier to develop and verify, while code reuse engenders an overall increase in productivity. Engineer productivity is also increased when the time spent soliciting requirements from domain experts is limited to a finite set of primitives rather than a larger set of more complex behaviors.

Domain expert productivity is also benefited by behavior composition, which allows them to use a language directly relevant to their domain. Further, when equipped with an appropriate tool set, the reliance on software developers is dramatically reduced (Summers, 2004). This has the added benefit of increasing the overall productivity of teams that are limited by software engineer availability.

Perhaps the strongest argument in favor of composition systems is that they facilitate model verification and validation. They do this not only because access is extended to those who lack training in software development, but because when behaviors are implemented in code the domain knowledge so represented is mingled with and obscured by code that fulfills other roles.

Behavior composition systems generally fall into one of two broad categories. The first category, knowledge-based systems (also called rule-based systems or embedded expert systems), is characterized by the use of some form of finite state machine (FSM). Examples of this approach can be found in: Obst (2001), Gilgenbach (2006), Fu (2003), Reece (2004), and Kosecka (1997). States in the FSM represent different things in different systems. They can correspond to activities, goals, or behaviors, but in each case, they devolve into actions taken by the unit the agent commands. Typically, only one state may be active at a time. Transitions between states are governed by Boolean expressions whose fluents reflect some bit of the agent's knowledge or some environmental condition. Figure 1 shows an example of FSM-based behavior composition for tactical reasoning.

In order to be used in tactical decision making, there must be a place for tactical concepts in any given knowledge-based system. Some of these concepts, such as time and the ordering of events and actions, are expressed naturally by the arrangement of primitives in an FSM. But other tactical concepts, such as spatial relationships, subunit coordination, cover and concealment, positional analysis and attrition, must be captured in source code in either the actions associated with states or in the fluents' evaluation functions.

Goal-based systems are another broad category into which many behavior composition systems fall. In these systems, a goal condition or optimization function is specified external to the agent. The agent performs a search of some kind to discover a sequence of actions that meets its assigned objective. This search occurs at execution time and gives the agent the ability to dynamically adapt to its particular circumstances. In goal-based systems, domain experts ensure that plan inputs such as atomic actions and their pre- and post-conditions are appropriate to the domain rather than directly specifying action sequences or flow charts. In this sense, the act of composition is shared between the domain expert and an automated planner. Zhang (2001) and Pittman (2008) are examples of this approach.

As with knowledge-based systems, goal-based systems also have the ability to aid in tactical reasoning. But as with knowledge-based systems, apart from temporal relationships and the ordering of events and actions, tactical reasoning must be done in source code.

Both knowledge- and goal-based systems may be termed procedural composition systems, because they focus on agent actions and the manner in which sequences of actions are chosen.

It is the purpose of this paper to assert that non-procedural primitives can also be used in behavior composition and that the gains in accessibility and productivity made possible by procedural composition systems can be extended by increasing the number and kinds of primitives made available to domain experts.

3. Overview

This approach utilizes both procedural and non-procedural composition. To do so, it uses a tactics description language called Tesla to capture tactical concepts and convey them from a human expert to a software agent in a format that is accessible to both.

As depicted in Figure 2, the domain expert uses an editor to create a tactic template. In this template is encoded enough of a tactic's underlying concepts that an agent can later use it to apply the tactic to its particular situation.

Figure 3 shows a simple tactic template displayed in the Tesla editor. In this tactic, the commander agent directs a single subordinate unit to move to a destination while avoiding observation by all known enemies.

The Tesla language is part graphical and part textual. The graphical part is the sketch view which

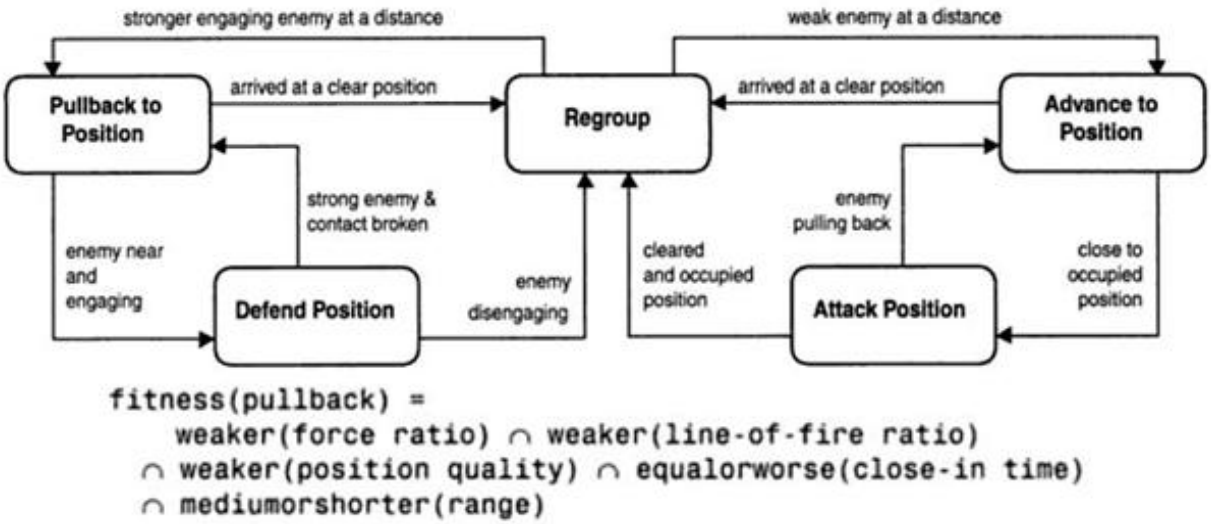

Figure 1: Tactical behavior in a knowledge-based composition system (Gilgenbach, 2006)

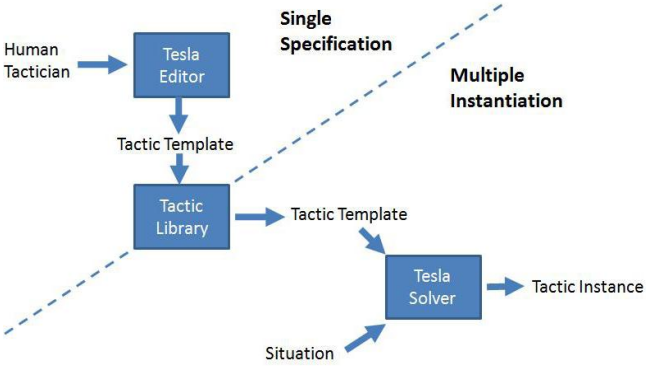

Figure 2: Tesla use case

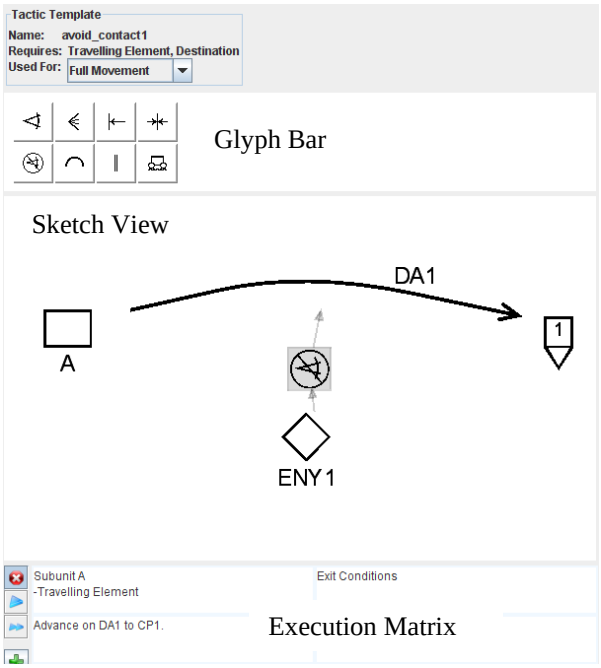

Figure 3: Simple tactic template

corresponds roughly to a course of action sketch. Found in the sketch view are 1) all entities (including relevant control measures) that take part in the tactic and 2) the constraints that define how entities and control measures may be converted from abstract concepts into instances of a particular situation.

The textual part of a template is the execution matrix. As with the sketch view, its semantics and syntax are borrowed from military course of action development (FM 3-90, 2001). Both parts of the language are described in more detail below.

3.1 Nominals

One of the principal elements of the Tesla language is the *nominal*. In grammar, a nominal is a noun phrase. In the Tesla language, a nominal is a unit, location or object on the battlefield.

The example in figure 3 contains four nominals. Starting on the left and proceeding in a clockwise manner, they are: a subunit (A), a generic direction of attack (DA1), a checkpoint (CP1) and an enemy unit (ENY1).

Nominal icons come mainly from US military symbology (FM 1-02, 2004). Note that the subunit and enemy unit symbols do not have echelon designators, because in a template they can refer to any echelon.

3.2 Constraints

In the Tesla language, constraints modify nominals. In this respect, they serve as adjective phrases indicating what kind of object the nominal should be. Above the sketch view in figure 3 is the constraint glyph bar.

Constraints are chosen from this glyph bar, configured and added to the nominals they modify.

The template in figure 3 contains a single constraint. This constraint points from ENY1 to DA1. It is read to mean, "Constrain DA1 such that it is concealed from all enemies identified as belonging to ENY1."

The natural language expression of a constraint can sometimes be ambiguous. To remove this ambiguity, each constraint has one or more associated *location metrics*. A location metric contains the algorithmic interpretation of the constraint that the domain expert wants to use in the tactic. The concealment constraint from figure 3, for example, can be alternately interpreted as meaning the absence of optical line of sight or as referring to an estimated probability of detection being below some threshold. Each interpretation has a corresponding location metric that can be chosen for the constraint. Other interpretations would also be possible.

3.3 Execution matrix

The Tesla execution matrix is conceptually similar to the execution matrices used in military course of action development. It contains the procedural parts of the tactic template. In it, each subunit has a column, and each phase in the course of action has a row. Every cell in the execution matrix contains instructions for that column's subunit. Cells in a row are executed simultaneously. In the Tesla language, instructions are composed of a task word and some number of modifying phrases. These modifying phrases are task word specific and generally relate to one or more nominals from the sketch view.

The execution matrix from figure 3 has a single subunit and a single phase. Its instruction has the task word, *Advance*, with the modifying phrases, *on DA1* and *to CP1*.

3.4 Resolution

Template resolution is the process by which a template is applied to the agent's particular situation. It consists of mapping each nominal to an appropriate counterpart in the agent's environment. In the template from figure 3, for example, subunit A would be mapped to one or more of the agent's subordinates; DA1 would be mapped to a concealed route; CP1 would be mapped to a location; and ENY1 would be mapped to a group of known or suspected hostile units.

In order to ensure that a proper mapping is found, the domain expert assigns and configures a so-called *nominal resolver* to each nominal in the template. Each type of nominal has one or more nominal resolvers to choose from, and each nominal resolver is responsible for making sure that a mapping is found that obeys each of the constraints placed on the nominal.

Once each nominal has been resolved, the instructions in the execution matrix refer to concrete locations and objects rather than abstractions. At this stage, these instructions can be used to generate maneuver and fire orders for subordinates.

4. Example Tactic

To illustrate how a tactic template works, this section examines an implementation of the fix-flank tactic. In this tactic, a force is divided into fixing and flanking elements. The fixing element engages the enemy unit and seeks to pin it in place. The flanking element takes a concealed route to a position of advantage from which it can surprise and flank the enemy. Parts of this template are shown in figures 4 and 5.

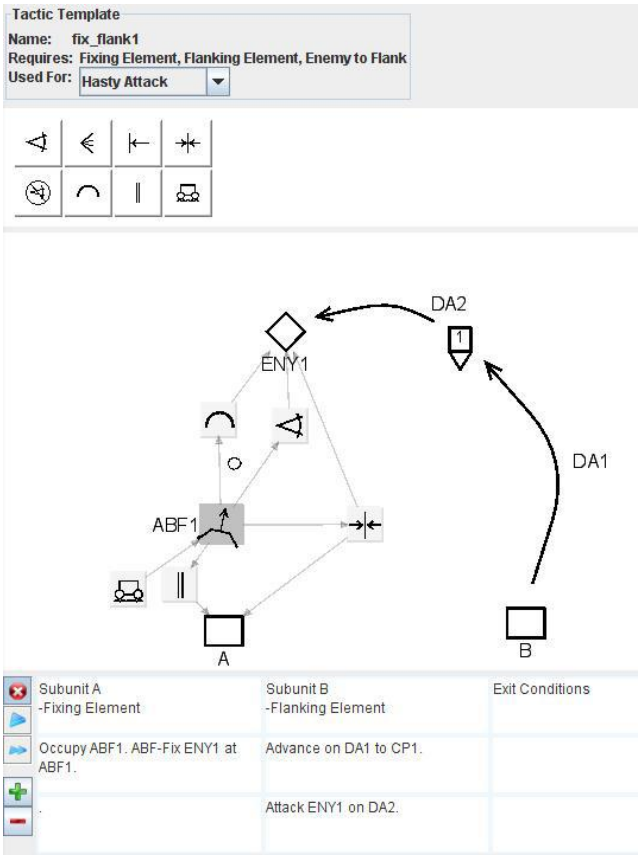

Figure 4: Fix-Flank tactic template

In the fix-flank template, subunit A is the fixing element. It moves to ABF1, an attack by fire position, from which it can engage ENY1. In order for the solver to select a suitable location for ABF1, five constraints are supplied that indicate the properties that

ABF1 must have in order to play its role as a fixing position in this tactic. In the Tesla editor, when a nominal is selected, its constraints become visible. Figure 4 shows the fix-flank template with ABF1 selected. Starting above ABF1 and proceeding in a clockwise direction, its constraints are interpreted as meaning:

- A unit at ABF1 should have cover from ENY1.
- A unit occupying ABF1 should be able to see ENY1.
- ABF1 should be roughly between subunit A's starting position and ENY1.
- ABF1 should be somewhat near subunit A's starting position.
- ABF1 should be on trafficable terrain.

The other nominals from this template also have constraints specified in a similar manner.

Figure 5 shows the user interface for the nominal resolver that was chosen for ABF1. This type of nominal resolver is called a *location scorer resolver* because it uses the constraints' location metrics to score and rank candidate locations. In the location scorer resolver, the domain expert chooses whether to use constraints as a basis for excluding locations as candidates or to use them as contributing to a location's score. As seen in the first two rows of figure 5, only locations with line of sight to all of ENY1 and at least some cover from ENY1 are considered as candidates.

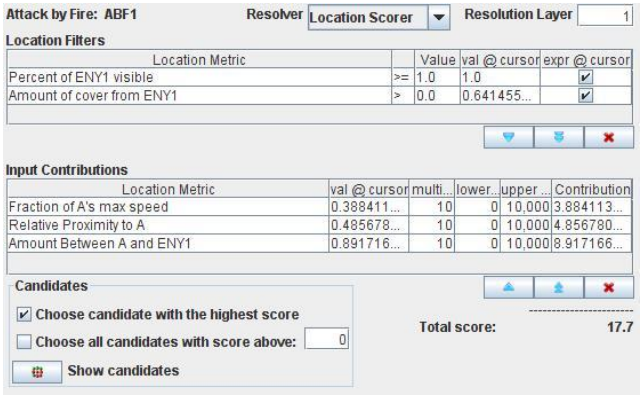

Figure 5: Location scorer resolver configuring ABF1

Location metrics create values that range from zero to one, making them suitable for nominal resolvers that use fuzzy logic. This property also makes it easy to visualize how location metrics operate. Figure 7 shows heat maps for the five location metrics used by the ABF1 nominal resolver.

To apply the template to a situation, the Tesla solver iterates over each nominal and invokes its nominal resolver. The order of resolution matters, since the outcome of one mapping can be used as an input into a subsequent nominal resolver's location metric. In the fix-flank example, A, B and ENY1 are template inputs, meaning that in order to use the template, the agent must supply mappings for these three nominals. The other nominals, ABF1, DA1, CP1 and DA2 are all resolved using constraints, location metrics and nominal resolvers as configured by the template developer.

Figure 6 shows the fix-flank template resolved in two different situations. The top situation is the same as the one from figure 7.

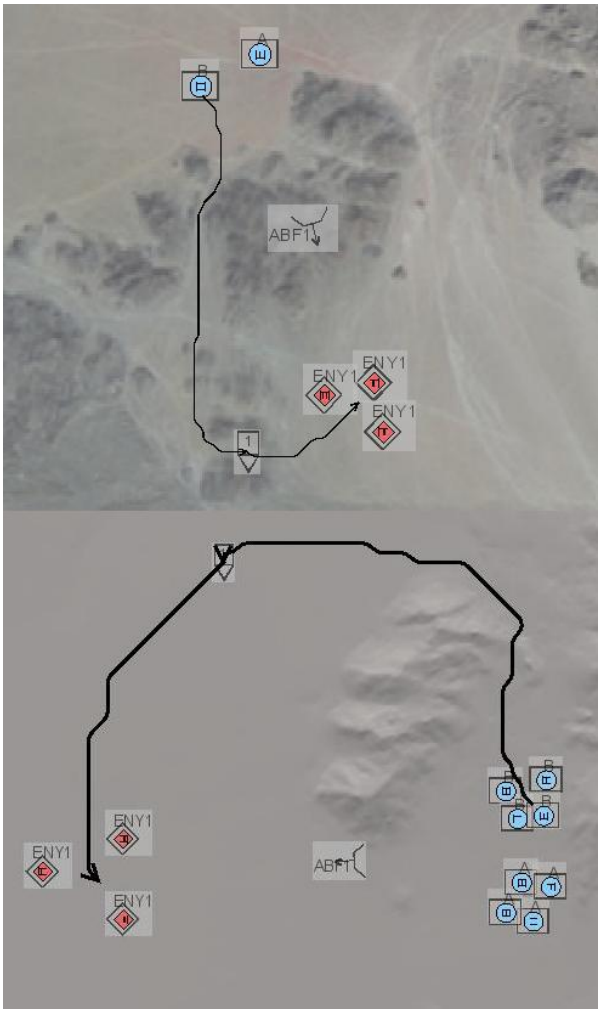

Figure 6: Two resolutions of the fix-flank tactic

5. Tesla Composition Primitives

Each type of behavior primitive in a composition system represents a kind of functionality available to the domain expert for manipulation and validation. The behavior primitive types available indicate the points where the system is easily extensible.

This section discusses some of the composition primitives available to a domain expert in Tesla.

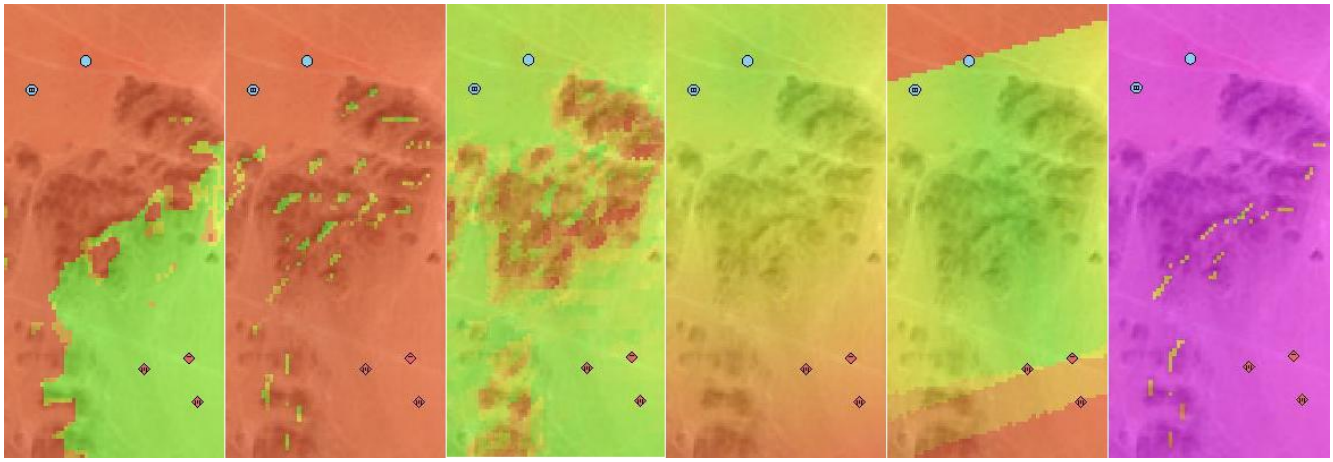

Figure 7: Location metrics used in the fix-flank example. From left to right, they are: Percent visible, Amount cover, Fraction of max speed, Relative proximity and Amount between. The last panel shows the composite scores as calculated by the location scorer resolver. In each panel, green indicates a metric value of one, while red indicates a metric value of zero. In the last panel, magenta indicates a location that has been filtered out and not considered as a candidate.

5.1 Nominals

The number of kinds of battlefield objects that can be represented by the Tesla language is increased by adding more nominals. Nominal types currently supported in the language are:

- subunits i.e. a subordinate of the commander agent
- enemy units
- locations - e.g. point target, support by fire position, point of interest
- line segments - e.g. linear target, lane
- segmented lines - e.g. unit border, phase line
- routes - e.g. avenue of approach, direction of attack
- areas - e.g. objective, free fire zone

5.2 Constraints and location metrics

Constraints and location metrics represent the most basic tactical concepts that can be expressed in the Tesla language. They provide the building blocks for terrain and positional analysis and reasoning over firepower, mobility, communications and sensing. As domain experts develop templates for which existing constraints and locations metrics do not suffice, new ones can be requested of and implemented by a software engineering team.

5.3 Nominal resolvers

The algorithms found in nominal resolvers are themselves behavior primitives. Nominal resolvers currently exist for location selection, enemy classification, route planning and template input handling. More can be built and added to the framework as necessary.

5.4 Verbs and verb modifiers

Similar to other systems, these procedural primitives map to actions that must be individually implemented in source code. But these actions should be much simpler to implement because they are for individual subordinates and not for the unit as a whole. Subunit coordination is done in the template editor rather than by a software engineer.

5.5 Expressivity

The Tesla language allows for the representation of sophisticated tactical concepts. Its primitives can be used to design coordinated attacks, plan ambushes, identify kill sacks and areas of overlapping fire, trace infiltration routes, find overwatch positions, plan defensive positions and so forth.

A reverse slope defense is one that keeps the defender concealed from the attacker until the attacker has approached to close range (such as by defending the reverse side of a hill). This allows the defender to neutralize any weapon range overmatch the attacker might have by forcing the engagement to occur at close range. This concept can be included in a tactic by using and giving proper weights to direct fire constraints. Conversely, an agent can be configured to capitalize on a weapon range overmatch by applying different weights to those same constraints.

Some tactical concepts have fine distinctions that can be difficult for a software agent to make. For example, three different tasks, attack, suppress and fix, all involve seeking advantageous terrain and engaging the enemy. All three are successful if the enemy is destroyed, but the manner in which the tasks are

executed is sometimes different. For attack, the desired effect is the destruction of the enemy. For suppress, the desired effect is to make enemy fires less effective. For fix, the desired effect is to prevent enemy movement. Because fix and suppress tasks have more relaxed goals, troops are permitted a more defensive posture when executing these tasks. These distinctions between the attack, suppress and fix tasks can be realized through judicious use of direct fire and line of sight constraints on ABF and SBF nominals.

The expressivity of the Tesla language gives commander agents the ability to reason over sophisticated tactical concepts. This gives an agent the ability to interpret changes to its tactical situation and dynmically adapt when necessary. This adaptability increases model realism. It also makes scenario devleopment less time consuming, because it decreases the number of eventualities that have to be explicitly scripted for.

6. Iterative Refinement and Behavior Validation

Figure 8 shows the Tesla editor application. It is divided into a template editor and a situation editor. The template editor allows the user to create and view tactic templates. The situation edtor is where the template is tested. It allows the user to create a number of situations against which to test the template.

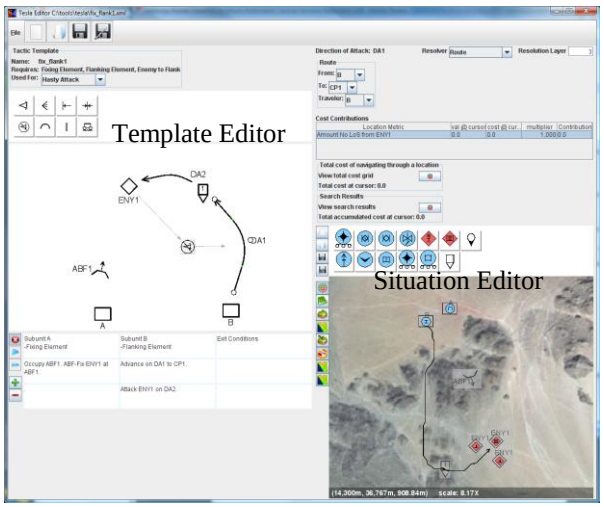

Figure 8: Tesla Editor

The ability to quickly test a template has a number of significant implications. First, it allows template development to be a process of iterative refinement. The domain expert creates a template and a situation and then invokes the solver to see how it interprets the template. If there are unexpected results, debugging is facilitated by overlays showing the contributions of individual parts of the template. These overlays, such as the heat maps from figure 7, are displayed in the situation editor. As problems are worked out, the domain expert creates more situations and tests the template against them as well. The process continues until the user is confident that the template is flexible enough to be applicable in many situations.

This same functionality is useful in behavior validation. Rather than waiting to validate a template until the agent can use it in a fully configured simulation, the validating authority can see how a tactic is used in a number of situations. If applicable, the template can be checked for validity at different echelons as well. These situations are saved with the template library and can be invoked again later, allowing the template library to be separately validated at any time

The easy and full access to this aspect of agent behavior is a significant aid to the validation process.

7. Conclusion

Although the Tesla language shares similarities with other composition systems, it is qualitatively different from many of them. In the military context, the decisions of commanders are more often manifest through communication and the actions of their subordinates than through their own shooting, moving and sensing. For a commander agent to develop a course of action for its subordinates requires it to reason about what it knows about friendly and enemy force positions, composition and capability. As a tool for commander agent configuration, Tesla encodes formulae for the deployment of maneuver forces rather than encoding procedures for equipment operation.

The Tesla language, editor and solver constitute part of a kind of knowledge-based system. It does not compete with automated planners or systems that use FSMs, since they solve different kinds of problems. Procedural composition systems are primarily concerned with determining *what* to do, whereas this approach seeks to identify *how* something should be done. Rather than competing with procedural composition systems, this approach should be viewed as complementary. When equipped with the appropriate metadata, these templates can serve as robust primitives in a higher-level composition system. In particular, they can provide a mechanism for managing subordinate coordination, which can be problematic for a purely procedural system.

The approach described in this paper aids in the specification of commander agent behavior. It is offered as a way to extend the benefits of composition systems to more functionality than is exposed in purely

procedural systems. Doing so facilitates validation and verification by giving domain experts more direct access to agent behavior, enables a more cost effective division of labor between domain experts and software engineers and provides a highly extensible framework for configuring tactical agent behavior.

8. References

Reece, D., McCormack, R., and Zhang, J., (2004). A Case-Based Reasoning Tool for Composing Behaviors for Computer Generated Forces. *In Proceedings of Behavior Representation in Modeling and Simulation*.

Fu, D., Houlette, R., and Jensen, R., (2003). A Visual Environment for Rapid Behavior Definition. *In Proceedings of Behavior Representation in Modeling and Simulation*.

Gilgenbach, M., McIntosh, T., "A Flexible AI System through Behavior Compositing," *AI Game Programming Wisdom 3*, pp 251-264, Charles River Media, 2006.

Obst, O., (2001). Specifying Rational Agents with Statecharts and Utility Functions.

Howden, N., Curmi, J., Heinze, C., Goss, S., and Murphey, G., (2003). Operational Knowledge Representation: Behavior Capture Modelling and Verification. In *Proceedings of the Eighth International Conference on Simulation Technology and Training*.

Lötzsch, M., Bach, J., Burkhard H., Jüngel, M.. (2004). Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL. In *Proceedings 7th International Workshop on RoboCup*, 114-124.

Kosecka, J., Christensen, H., (1997). Experiments in Behavior Composition. *Journal of Robotics and Autonomous Systems, vol. 19,287-298*.

Zhang, Y., Biggers, K., Sheetal, R., Sepulvado, D., Yen, J., Loerger, T., (2001). A distributed intelligent agent architecture for simulating aggregate-level behavior and interactions on the battlefield. In *Proceedings 5th Multi-Conference on Systemics, Cybemetics, and Informatics, 58-63*.

Pittman, D., "Command Hierarchies Using Goal-Oriented Action Planning," *AI Game Programming Wisdom 4*, pp 383-391, Charles River Media, 2008.

Summers, J., McLaren, M., Aha, D., (2004). Towards Applying Case-Based Reasoning to Composable Behavior Modeling. *In Proceedings of Behavior Representation in Modeling and Simulation*.

Department of the Army. (2001). *Field Manual 3-90, Tactics.* Washington, D.C..

Department of the Army. (2004). *Field Manual 1-02, Operational Terms and Graphics*. Washington, D.C..

Author Biographies

EVAN CLARK is a software engineer at the Physical Science Laboratory of New Mexico State University. He works as the simulation subject matter expert for the System of Systems Survivability Simulation (S4). He graduated from Brigham Young University with a B.S. in Electrical Engineering in 1997. He received a M.S. in Computer Science from New Mexico State University in 2006 and a Ph.D. in 2009. His research interests include agent modeling, decision theory, military modeling & simulation, human vision and audition, computational geometry and visual languages.

JEFFREY SMITH is an Electronics Engineer for the Survivability\Lethality Analysis Directorate (SLAD) of the Army Research Laboratory. He is the lead engineer for developing and fielding a Systems of Systems Analysis capability for SLAD and a provider of survivability, lethality and vulnerability analysis expertise to the System of Systems Survivability Simulation (S4) agent based model, a core component of this capability. He graduated from New Mexico State University with a B.S. (1984) degree in Electrical Engineering. He has an M.S. (1998) and a Ph.D. (2004) in Industrial Engineering (Operations Research/Stochastic Systems) and a minor in Mathematics (Statistics/Probability Theory). He entered Federal Service in 1978 as a Co-op engineer, and continued as an Electronics Engineer with the completion of his B.S.E.E. 1984. He has worked his entire career with the U.S. Army in the area of close combat weapons, assessing the effectiveness and hardness of various weapons systems, and the survivability of numerous developmental and fielded combat platforms. He has a lifelong interest in military history and combat simulations.

JOEL EICHELBERGER is the communications subject matter expert and lead communications model developer at the Physical Science Laboratory of New Mexico State University. In 2001, he earned a B.A. in Computer Science and Business from Northwood University, where he was the University's Network Administrator. In 2009, while working at NMSU, he earned his M.B.A.. He has interests in military communications systems and the impact of information on the battlefield, with a focus on the modeling and simulation of said systems.