A Case-Based Behavior Design Aid for OneSAF

Douglas Reece
Jenifer McCormack
Jackie Zhang

Science Applications International Corporation
Outline

• Problem — expense of CGF behavior building
• Solution approach — software reuse
• OneSAF — behavior design complexity
• A behavior design process and design aid
• Conclusion
Problem

Building CGF Behavior is Expensive

• Operators need new behaviors to support different units, equipment, and tactics
  – Current CGF systems like OneSAF need 100’s of behaviors
• Has required Software Engineer in addition to operator or SME
  – Significant cost in Knowledge Engineering step

Observations about several CGF systems:
• There is often a lot of existing behavior
• Programmers try to use code they know about as an example when writing new behavior
  – But knowledge of existing code may be limited
  – Knowledge of behavior code isn’t common across many programmers
A Solution Approach

Reuse Behavior Software

• Develop behavior component representation that supports reuse
  – Library of primitives with metadata descriptions
  – Grammar for composing complex behaviors

• Save compositions in a repository
  – Reuse compositions as well as primitives

• Use a graphical, interactive tool to make composition easier

Expected benefits:

– Avoidance of unnecessary behavior development
  • Less redundancy in repository

– Faster behavior development

– Non-programmers can compose behaviors
  • (If the grammar is easier than programming language)

– Behaviors have common design
  • Easier for operators to find behaviors
Behavior Software Reuse Challenges

• Finding behaviors relevant to target behavior
  – Primitives and composites
  – Characterizing target and existing behavior
  – Searching repository

• Understanding how behaviors work
  – Exactly what it does
  – Interfaces

• Adapting (more easily than rewriting)
  – Adapting target task to existing primitives
  – Adapting existing composites to target task
  – Extending existing primitives
Remember Composite Behavior Solutions as “Cases”

Reuse of compositions is a Case-Based Design (CBD) problem.

Case Based Design:

- Repository of design solutions
- Solutions partly graphical, spatial, geometric—may not easily accommodate text-based matching or computed indices
- CBD systems provide some matching, plus browsing for solutions for user evaluation

Our objective: Develop a tool to provide user with examples of relevant, working behavior solutions and help in understanding how to modify them
OneSAF

• A composable, next generation CGF
• Represent a full range of operations, systems, and control processes (TTP)
• Entity up to brigade level
• Variable level of fidelity
• Supports multiple Army M&S domain (ACR, RDA, TEMO) applications.

Automated
Composable
Extensible
Interoperable

Software only

Field to:
RDECs / Battle Labs
National Guard Armories
Reserve Training Centers
All Active Duty Brigades and Battalions

Capable of replacing legacy entity-based simulations: BBS, OTB/ModSAF, CCTT/AVCATT SAF, Janus, JCATS MOUT

Platform Independent
"Becoming a *de facto* Joint Model"

- USMC (PM TRASYS) invested FY04 and FY05 dollars to build Marine entities, units, and behaviors in OOS.
- OOS drives USMC CACCTUS training system.
- Coordination ongoing with Air Force SIMAF office to use OOS as the ground maneuver driver in EAAGLES.
- USAF planning to build behaviors for OOS as well.
- JFCOM J-9 plans to use OOS in JUO Experiments in FY05 and 06
- Part of 2004 J-7 Joint interoperability demo at I/ITSEC
- OOS will be under the hood in a variety of virtual simulators, including AVCATT, CCTT, the Common Gunnery Architecture, and the Common Driver Trainer
- Embedded training on all FCS platforms

- Signed Project Agreement with the UK to build their behaviors for OOS
- Several countries are preparing to stand up their own IDE
OneSAF Behavior Vision

- Millions of dollars, hundreds of behaviors (and units, physical models, entity types, etc.)
- Composition descriptions available in repository
- Behaviors are composable with graphical tool

A promising setup for reuse!
OOS Spiral Development:
Evolving Capabilities with User Feedback


Increasing functionality

You are here.

FOC Fielding

Block A ➔ Block B ➔ Block C ➔ Block D

P3I

14th BRIMS Conference
16-19 May 2005
OneSAF Behavior Architecture

• Behavior is procedural

• Grammar: Behavior element =
  – Primitive behavior
  – Conditional branch on predicate function value
  – Composite behavior
  – “Order sender”
  – Communication sender
  – Conditional repeat of element (do-loop)
  – Sequence of elements
  – Parallel set of elements

• GUI to compose behaviors

XML
Java

14th BRIMS Conference
16-19 May 2005
OneSAF Behavior Design Steps

1. Compose timeline (procedure)
   - Select new elements from repository
     • Or, create new primitives or composites
   - Place elements in sequential and parallel blocks
   - Connect conditional branch paths
   - Create behavior inputs and constants

2. Connect data paths
   - Connect element inputs and outputs to behavior and element inputs and outputs

3. Set semantics of primitives
   - Extend I/O of primitive behaviors
   - Modify Java code of primitives or add new primitive
   - Modify or add predicate functions

4. Other
   - Rule sets
   - Facts
   - Data tables
   - Behavior agent capabilities

Fairly complex — a lot to get right
Searching for Behaviors in OneSAF*

*Approximately the Block C version of OneSAF
A Structured Design Process for Reuse

• Look for behaviors that seem similar to the target behavior
• Figure out what the behaviors do
  – Whether and how they match target
• Select one behavior and modify it as needed
  – Or combine parts of multiple behaviors
  – Modification of components makes process recursive
• If reusing sub-behaviors, maybe modify top behavior to accommodate them

A top-down approach with bottom-up feedback
Design Process

1. Task description
2. Search for existing behavior(s)
3. Analyze behavior(s)
   - Review descriptions
   - Analyze I/O
   - Analyze control flow
   - Analyze subs
4. Select and adapt a behavior
   - Change control flow
   - Combine behaviors
   - Add, delete, substitute, or invent sub-behaviors
   - Modify subs

Repeat process as necessary.
Search for Behaviors

Similarity may be in

- Unit type, task type, environment, or other domain categories
  - Define indices with continuous match values
  - Score behaviors by weighted combination of indices

- Structure of control flow
  - Parallel blocks
  - Repeated elements
  - Behavior used for unit and individual entity
  - Etc.

[See Reece, McCormack and Zhang, *BRIMS 2004*]
Searching/Browsing by Domain Info

Domain indices specified

Composites listed according to their matching scores

Select promising composite to examine

Copyright 1998 Institute for Simulation & Training
Searching/Browsing by Structure

Behavior pattern specified (Conditional Loop)

Selected behavior with loop structure
Analyzing What Behaviors Do

- Examine input variables
- Examine input constants
  - Especially, selector values
- Examine data connections between primitives
- Determine branch and loop conditions
- Determine what behavior does
  - Primitive—examine Java code
  - Composite—open in window
- Explore how a primitive is used in other behaviors
Examining Behavior Input Connections

Sub-behaviors in behavior

Behavior input constants and variables

Connection to sub-behavior input
Primitive Behavior Analysis

GUI highlights key primitives for this composite—have selector input.

GUI highlights other composites that use the selected primitive.
Primitive Input/Output Connections

Table shows enumeration inputs (also constant and variable inputs, outputs)
OneSAF Primitive Design—Common Themes

Common theme goal for primitive behaviors:

• Avoid having many similar primitives
  – Combine actions using common core algorithms
  – Combine actions related by a “theme”
  – Use “selector” inputs to select specific primitive function

Ramification:

• New primitive function often involves adding to an existing primitive
• User must examine selector input to see what all a primitive can do
Select and Open Sub-Behavior
Select and Open Sub-Behavior
Modify Behavior

Copy behavior to composer, then:

• Change control flow
• Add or delete sub behaviors
• Modify sub behaviors
  – Composite: open in composer
  – Primitive (!): add Java (no tool support yet)
• Invent new sub-behaviors
  – Recursively apply design process
OneSAF Primitive Design—Level of Detail

Level-of-detail goal for primitive behaviors:
- Avoid requiring user to program data types, structures, simulation data accesses, complex behavior synchronization, etc.

Ramification:
- User can’t program data types, structures, simulation data accesses, complex behavior synchronization, etc.
- Most new entity behaviors require new primitive code
Copy Behavior to Composer to Edit
Experiments

Early experiments with test subjects

- Task: design new behavior from “informal” task description
- Subjects: programmers with some behavior programming experience (OneSAF, or other SAF)
- Very positive feedback about search and browse feature
- Examination of inputs very common step; I/O tables speed this step
- Structural behavior matching is less common
  - but found example of use even with experienced subject
Future Work

• More one-click sub-behavior info displays
• Auto presentation of Java code for Primitive
• Auto generation of some Java code
  – Primitive declarations
  – Enumerated type extensions
• User testing and evaluation of effectiveness
Conclusions

• CGF behavior is complex—hard to get around difficulty in defining (programming) it for simulation

• Reuse potential is there
  – Avoiding unnecessary behavior building
  – Providing examples of working complex behavior

• OneSAF provides a good framework for reuse
  – Behavior language
  – Repository
  – Primitives
  – GUI for composition

• A Case-Based Design tool has the potential for overcoming challenges of behavior reuse