Use of Plugin Architecture and Full Source Licensing in the Deployment and Support of the Conflict Analysis & Simulation Tool (CAST)

John Shue

8 November 2012
Overview

• Plugin Architecture
  – What problems are solved by plugin architecture
  – How it is accomplished
• Embedded Python
  – Key enabler for plugin architecture
  – Other benefits of embedding Python
• Full Source Licensing
  – End user benefits
  – Vendor benefits
The Conflict Analysis & Simulation Tool (CAST) was developed by ManTech to provide a MS&A framework to rapidly simulate and visualize conflict scenarios for:

- mission effectiveness analysis
- requirements analysis
- trade studies
- technology assessment
- tactics/CONOPS development

CAST is commercially available, but has also been employed on support contracts with Government customers.

Modularization through a plugin architecture is required to support a disparate client base and problem domains.
Framework Description

• The CAST framework provides the core components which are used to create, run, and analyze simulations.

• The CAST framework is extended to develop system models and scenarios specific to the client’s problem domain.
  – DHS DNDO: Radiation Detection
  – DoD: Naval Warfare
  – International: Air Warfare
Customization through Plugin Architecture

Core framework extended and customized by adding features via plugin architecture
Problems Solved by Plugin Architecture

• Plugin architecture provides clear separation for
  – Security/classification
  – Export control
  – Source code funding streams
  – Intellectual property

• Specific models and scenarios cannot be shared
  – These models can exist completely within the plugin

• Deployment is simplified
  – End user’s configuration management can focus on
    the plugin-level
Variety of Customizations

• Simulation framework must be flexible to support a variety of customizations to occur at the plugin level
  – Graphical User Interface (GUI)
  – Visualization
  – System models
  – Data inputs
  – Analysis outputs
  – Agent behaviors
Aspects of a Plugin Architecture

• Successful plugin architecture requires:
  – Flexible architecture
    • Object-oriented design
    • Composition versus inheritance
    • Runtime configuration
  – Data driven approach
  – Use of open standards
Object-oriented Design

• Code organized into a hierarchical structure of classes
• Data and associated code are grouped together
• Hierarchy of classes supports generalization where common aspects exist at higher levels and unique aspects are kept at lower levels
• In object-oriented design, inheritance is a rigid compile-time constraint that cannot be changed at runtime
Composition versus Inheritance

- To improve flexibility, object composition is a technique that can be used to specialize objects at runtime.
- This can be thought of as “has a” (composition) versus “is a” (inheritance).
- The ability to compose behavior or functionality lies in creating interfaces.
- An example of kinematic motion for entities
  - Inheritance would dictate a hierarchy of classes for each entity with specialized movement algorithms (Vehicle, Ship, Aircraft, Airplane, Helicopter)
  - Composition provides an interface for movement and encapsulates the movement algorithms into a component attached to the base entity (Vehicle class with movement functors)
Runtime Configuration

• To provide a flexible runtime architecture, framework must provide the ability to be configured at runtime.
• Runtime configurability plays nicely with an embedded scripting language capability (more detail later)
• Example of selecting system models with different fidelity
  – Object-oriented design with composition allows different fidelity models to be attached to entities at runtime, likely driven by data input
Data Driven Approach

• Runtime configuration driven by data inputs
  – Execution changes based on inputs
  – Dynamic selection of runtime components versus hard-coded or static single-choice execution

• Data driven system modeling
  – Algorithm versus data
  – Requires algorithms to be selectable and initialized at runtime
  – This approach doesn’t work for all system models, in which case, design falls back to traditional methods
Use of Open Standards

• Use of Extensible Markup Language (XML) for data
  – Easy to parse
  – Human readable
  – Numerous tools exist for data entry and manipulation
  – Schema can be developed to enforce data structure

• Use of existing best of breed cross-platform Open Source libraries
  – OpenSceneGraph
  – osgEarth
  – wxWidgets
  – Allows users to find support within those open source communities for advanced customizations
Embedded Python

- Embedded scripting language is a key enabler for plugin architecture
  - Python is a popular scripting language, others include Lua, Ruby, and JavaScript
  - Python scripts are text files which can be treated as input data
  - Interpreted nature of Python language allows for flexible initialization of runtime environment as opposed to hard-coded initialization
  - Example: CAST scenarios are implemented as Python scripts to construct and initialize entities
Other Benefits of Embedding Python

- Rapid prototyping without recompile
  - Exposure of C++ code within CAST framework via SWIG (Simplified Wrapper Interface Generator)
  - Allows development of new system model prototypes quickly
  - Once satisfied with prototype, system model can be implemented in C++
- Allows user to execute Python code while simulation is running
  - This feature can be used to inspect the state of the simulation by accessing simulation objects
  - Can also be used to alter the state of the simulation which is helpful for debugging and testing
Full Source Licensing

• End user is empowered to dig into the internals of the simulation framework
  – No “black box” barrier
• Similar benefits to Open Source licensing
  – Ability to see source code to get deeper understanding of framework’s internals
  – Ability to modify source code to create advanced customizations
  – Ability to debug into framework’s source code to identify cause of potential issues
Full Source Licensing

• End users can set the level of software support they require
• Over time end user increases familiarity with codebase source code
  – Initially very hands on: Vendor heavily involved in customization efforts (training, consulting, maintenance)
  – As end user develops proficiency with codebase requires less vendor involvement (maintenance)
Vendor Benefits from Full Source Licensing

• Enforces robust configuration management and software engineering best practices
  – Required in order to support multiple customers with multiple released versions
  – Separation of work-for-hire from core CAST development
• Provides a level of transparency to the customer because source code is not hidden
Questions?