2009

Rapid Scenario Generation for Multiple Simulations: An Application of the Military Scenario Definition Language (MSDL)

Dodds, R.; Pearman, J.; Baez, F.; Blais, Curtis

Paper 09S-SIW-003, Proceedings of the Spring Simulation Interoperability Workshop, Simulation Interoperability Standards Organization, San Diego
https://hdl.handle.net/10945/31206

Rapid Scenario Generation for Multiple Simulations: An Application of the Military Scenario Definition Language (MSDL)

Curtis Blais
MOVES Institute
Naval Postgraduate School
700 Dyer Road
Monterey, CA 93943-5001
831-656-3215 (DSN 756-3215)
clblais@nps.edu

Russell Dodds
Yumetech, Inc.
2915 East Madison Street
Suite 304
Seattle, WA 98112
russell@yumetech.com

Jerry Pearman
Augustine Consulting, Inc.
1277 Castro Road
Monterey, CA 93943
831-656-1034
gmpearma@nps.edu

Francisco Baez
TRADOC Research and Analysis Center
PO Box 8692
Naval Postgraduate School
Monterey, CA 93943
francisco.r.baez@us.army.mil

Keywords:
Simulation, combat modeling, scenario definition, agent-based simulation, MSDL, Pythagoras, IWARS

ABSTRACT: *The Military Scenario Definition Language (MSDL) is a common language for describing components of military scenarios that can be shared across a variety of modeling and simulation systems. Version 1.0 of the MSDL specification was approved as a Simulation Interoperability Standards Organization (SISO) balloted standard in October 2008. Numerous initiatives are in progress to employ the new standard and to realize the benefits of exchanging scenarios files across diverse systems. The US Army Training and Doctrine Command (TRADOC) Research and Analysis Center, Monterey (TRAC-Monterey) is developing a Rapid Scenario Generation tool to assist a user in constructing courses of action that can be exercised in various simulation systems, such as Pythagoras and the Infantry Warrior Simulation (IWARS). This paper describes the Rapid Scenario Generation tool and its use of MSDL as an archival format for storing and exchanging scenario information.*

1. Introduction

The Department of Defense uses a variety of modeling and simulation (M&S) systems for analysis, training, experimentation, acquisition, and mission planning and rehearsal. While some of the systems are used for multiple purposes, many serve a single purpose. Often a requirement exists to represent the same operational situation across multiple systems to serve these various purposes or to examine a problem from different perspectives. For example, a scenario used in a training exercise may be needed to conduct analyses of future force structures, or vice versa. Conversely, a scenario used for conducting analysis may be employed in an operational experiment evaluating Command and Control (C2) systems or new tactics, techniques, and procedures (TTPs). Many events now use a federation of M&S systems to represent battlespace entities and dynamics. Because of differences in design of individual federates, common aspects of the scenario have to be expressed in different ways to be understandable to the individual software. The individual M&S system (or federate) representations are not easily interchangeable, even though they often represent very similar aspects of the situation, such as force structures, initial plans and orders, weather conditions, or terrain. Currently no single scenario description exists to initialize common aspects of the battlespace across all federates.

In October 2008, the Simulation Interoperability Standards Organization (SISO) approved Version 1.0 of the Military Scenario Definition Language (MSDL) [1] to become an international standard for describing military scenarios. MSDL takes steps toward defining common aspects of a scenario that can be utilized across federates or across multiple non-federated systems. The language specifies force structures, environment, and other information for initialization of simulation systems, decision support systems, planning systems, and others. The standard specifies an Extensible Markup Language (XML) schema to provide a common mechanism for validating and loading military scenarios, to promote sharing of scenario files across systems, and to improve scenario consistency among federated and non-federated systems. Now that the standard has been approved, the

M&S community has further need to become aware of the nature and scope of MSDL and to explore application of the standard across a variety of operational contexts. While MSDL may benefit an individual system by providing a well-defined, well-organized expression of scenario information, its greatest benefit to the community will be through broad adoption permitting multiple systems to share scenario descriptions.

This paper describes an application of MSDL for storing and exchanging scenario information across multiple simulations. The US Army Training and Doctrine Command (TRADOC) Research and Analysis Center, Monterey (TRAC-Monterey) is developing a Rapid Scenario Generation (RSG) tool to assist users with constructing course of action (COA) sketches that can be exercised on multiple simulations. The RSG tool currently supports exchange of scenario information between the agent-based simulation Pythagoras and the Infantry Warrior Simulation (IWARS). This paper describes the RSG tool and its use of MSDL as an archival format for storing and exchanging scenario information. The paper first provides a high-level comparative overview of the MSDL, Pythagoras, and IWARS data models as described by their respective XML data formats. The paper then describes the RSG tool and discusses its design approach for incorporating MSDL as an archival and interchange format for storing and exchanging scenario data across the models. The paper concludes with a call to the simulation interoperability community to participate in analysis and development of MSDL usage to broaden community knowledge and to provide inputs to future enhancement of the standard.

2. Overview of MSDL, Pythagoras, and IWARS Scenario Constructs

The following subparagraphs provide brief overviews of the MSDL, Pythagoras, and IWARS XML data models, followed by a conceptual comparison of the information elements in the models to describe commonalities that present achievable interchange opportunities.

2.1 Primary Scenario Constructs in MSDL

The top-level structure of the MSDL XML data model (SISO version 1.0) is shown in Figure 1.[1] MSDL describes locale, forces, intelligence, situation, and course of action for re-use across multiple C2 and M&S systems. The MSDL Specification [1] defines a military scenario as "a specific description of the situation and course of action at a moment in time for each element in the scenario." The scenario description largely reflects common Mission, Enemy, Terrain and weather, Troops and support available, Time available and Civil (METT-TC) elements of a military situation. The purpose is to provide the M&S community with:

- A common mechanism for validating and loading military scenarios
- The ability to create a military scenario that can be shared between simulations and C4I[2] devices
- A way to improve scenario consistency between federated simulations
- The ability to reuse military scenarios as scenario descriptions are standardized throughout the Army, Joint, and international communities and across simulation domains, such as training and analysis

Scenario elements can be individual items of equipment, such as a tank or aircraft, or aggregates of troops and equipments, such as an infantry company. The reality of the situation reflects known or established content in the scenario, such as a certain force structure being employed to conduct an operation or terrain and weather conditions. These descriptions are exact and not the result of interpretation by scenario elements. Intelligence information reflects knowledge of the battlespace that an entity or force may possess at the outset of the execution, such as knowledge of enemy force positions and activities. This information may be incorrect and incomplete, but represents what is known when the execution begins (and on which simulated entities may begin making decisions and taking action). Some simulations do not start with such information, but establish battlespace awareness through simulated detections as the entities and forces begin to interact in the simulation.

The MSDL description of the scenario is expressed as an XML file conforming to an XML schema described and provided in the SISO specification. The MSDL XML schema defines one global element, the *MilitaryScenario* root element. All other constructs in the language are defined as global types, either complex or simple types, to maximize reuse of the definitions in creation of other XML languages. MSDL also has extensibility provisions through the use of the XML Schema *any* construct. This permits an MSDL XML document to contain XML structures defined elsewhere. This flexibility proved particularly beneficial to the current use, as will be described later in this paper.

[1] The *msdl:* prefix in element and type names in the MSDL XML schema refers to the MSDL namespace "urn:sisostds:scenario:military:data:draft:msdl:1". Solid boxes in the figure denote required elements; dashed boxes indicate optional elements.

[2] C4I: Command, Control, Communications, Computers and Intelligence

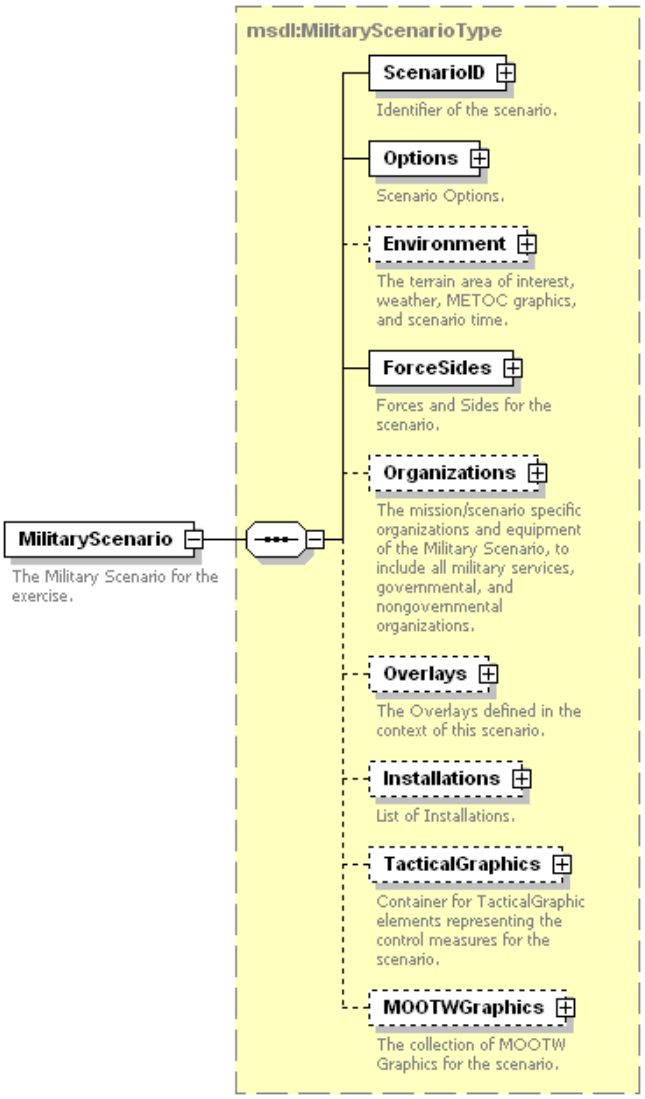

Figure 1. Top-Level Schema Structure of MSDL Scenario Files

We can examine the content of an MSDL scenario description by examining the structure of the language defined in the XML schema. The root element of the XML file is called *MilitaryScenario* and contains the following child elements (these descriptions are illustrative, not exhaustive):

- *ScenarioID* (mandatory) – provides identification of the scenario and its purpose.
- *Options* (mandatory) – provides global parameters about the scenario and its content.
- *Environment* (optional) – describes the simulated physical environment in which the execution is to occur (e.g., area of interest, weather, time).
- *ForceSides* (mandatory) – describes the structure of the forces and sides involved in the execution.
- *Organizations* (optional) – describes the structure of the units and equipment involved in the execution.
- *Overlays* (optional) – describes the logical overlays used to group the intelligence elements/instances in the scenario. Ownership of a specific overlay is determined through the intelligence elements/instances contained in that overlay.
- *Installations* (optional) – describes the detected installations as determined by the intelligence gathering process of each force, side, or unit individually.
- *TacticalGraphics* (optional) – describes the tactical information as known by a particular force, side, or unit individually.

- *MOOTWGraphics* (optional) – describes the detected MOOTWGraphics[3] instances as determined by the intelligence gathering process by each force, side, or unit individually.

The *ScenarioID* element contains metadata about the scenario, including the following information: (1) name assigned to the scenario; (2) type of object model; (3) version of the scenario file; (4) date of last modification; (5) classification level; (6) release restrictions; (7) purpose of the scenario; (8) type or class of application to which the scenario applies; (9) description; (10) any limitations on use of the scenario; (11) history of use; (12) keyword (and identification of taxonomy) characterizing the scenario; (13) identification of the organization or person who has a particular role with respect to the scenario; (14) type and identity of any reference; (15) identification of a glyph for visually representing the scenario; and (16) other data deemed relevant by the scenario author. The *ScenarioID* element, defined through the ModelID schema, includes the *any* compositor, which allows any XML structure from other languages to be inserted and retain validity against the MSDL schema.

The XML design of MSDL employs certain vocabulary from other XML schemas; namely: (1) *ScenarioID* metadata defined in the ModelID_v2006.xsd schema from the Base Object Model Specification (SISO-STD-003-2006) [3]; and (2) meteorological and battlespace domain values defined in the Joint Command, Control, and Consultation Information Exchange Data Model (JC3IEDM[4]) schema JC3IEDM-3.1-Codes-20061208.xsd. The MSDL XML schema declares namespaces assigned to these external schemas and imports these schemas in support of the definition of MSDL-specific elements and attributes.

The use of namespaces is important in dealing with XML vocabularies. The namespace enables a particular term to be uniquely identified within an XML document while permitting multiple vocabularies to be combined to create more complex languages, as in the case of MSDL's use of the ModelID and JC3IEDM vocabularies. This will be important later as we consider practical ways to use MSDL for information storage and exchange across the other simulation systems.

[3] MOOTW: Military Operations Other Than War. More current terminology is Stability, Security, Transition, and Reconstruction (SSTR) Operations (see [2]).

[4] JC3IEDM is a well-established data model maintained by the Multilateral Interoperability Programme (MIP). See http://mip-site.org.

A complete description of MSDL is beyond the scope of this paper. The reader is referred to the current MSDL specification and published XML schemas for a full description of the language.

2.2 Primary Scenario Constructs in Pythagoras

Pythagoras is an agent-based simulation employed by a number of organizations to perform various analyses, often exploring non-traditional or irregular warfare scenarios. Pythagoras represents agents individually, although a number of agents can be initialized with the same set of behaviors and operational characteristics. The latter would be similar to defining a "unit" in MSDL consisting of a homogeneous set of entities. However, individual agents can also be considered as abstractions that can represent any sensing or decision-making object in the scenario (a sensor, a robot, a person, an infantry company, etc.).

The top-level structure of the Pythagoras XML data model (Pythagoras version 1.10.5) is shown in Figure 2. A complete description of the Pythagoras data model is beyond the scope of this paper. The reader is referred to the Pythagoras Manual [4] for a full description of Pythagoras capabilities and data entry.

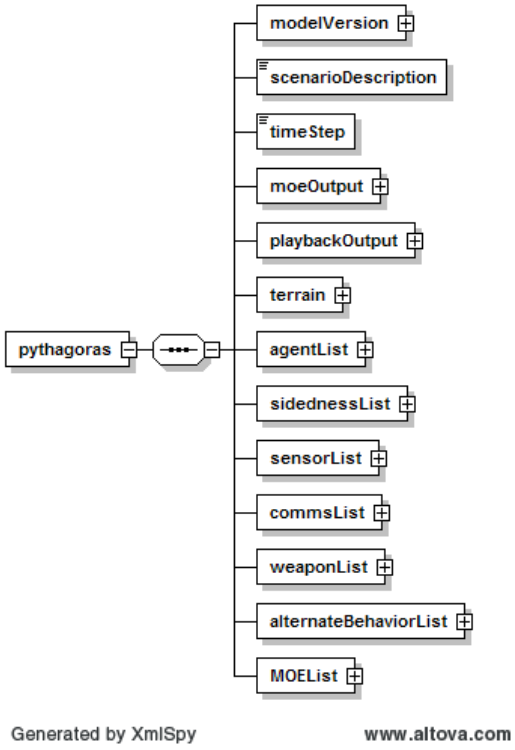

Figure 2. Top-Level Schema Structure of Pythagoras Scenario Files

As shown, the root element in the Pythagoras XML scenario file is *pythagoras* and contains the following child elements (note that all child elements are mandatory in the Pythagoras scenario file; the following descriptions are illustrative, not exhaustive):

- *modelVersion* (mandatory) – provides major, minor, and bugfix version identification for the version of Pythagoras that was used to create the scenario file.
- *scenarioDescription* (mandatory) – textual description of the scenario.
- *timeStep* (mandatory) – number of time steps to execute before ending the model run. Time steps have no real-world analog, other than that considered by the scenario developer for scaling movements and other temporal actions in the model.
- *moeOutput* (mandatory) – identifies end of run measures of effectiveness (MOE) that can be selected for output.
- *playbackOutput* (mandatory) – describes the file that can be used to record certain agent information during scenario execution.
- *terrain* (mandatory) – describes the terrain play box and influences on concealment and movement. The play box size is specified in screen pixels rather than any real-world geographic coordinate system. No scaling is explicitly defined, but considered by the user when setting speeds, weapon ranges, etc. relative to the size of the terrain. Features can be overlaid on the base terrain to set specialized concealment and movement factors for representation of such things as roads, buildings, and forested areas.
- *agentList* (mandatory) – provides a list of agents represented in the scenario. Agents are decision-making objects in the scenario, possibly possessing weapons, sensors, and communications capabilities. Agents have an initial location, an assigned side, and various characteristics relating to such aspects as leadership, obedience, vulnerability, detectability, and movement speed. Agents can possess several resources, including fuel. A major part of the specification of agents is their tendencies, factors indicating how the agent will react to a variety of situations. The data also specify what conditions will trigger changes in agent state (alternate behaviors).
- *sidednessList* (mandatory) – defines the sides represented in the scenario. Sides are defined in terms of the magnitude of red, green, and blue attributes, as well as respective tolerances. Agents may consider other agents to be part of a unit, friendly, or hostile based on defined ranges of values around a particular sidedness red, green, blue setting.
- *sensorList* (mandatory) – defines sensors for useby agents in the scenario. Sensor characteristics include signature type, maximum range, matrix of detection probabilities, error characteristics, broadcast range, and field of view properties.
- *commsList* (mandatory) – defines the communications devices that can be assigned to agents in the scenario. Communications characteristics include indication of dependence on line of sight, on/off status, maximum range, and channel characteristics.
- *weaponList* (mandatory) – defines the weapons that can be assigned to agents in the scenario. Weapon characteristics include indication that the weapon is used in direct or indirect fire, blast and lethality characteristics, influence capabilities (e.g., nonlethal weapons or even information weapons), affiliations on which the weapon is employed, firing rate, maximum range, kill probability matrix, and weapon effects.
- *alternateBehaviorList* (mandatory) – alternate behaviors that an agent can be assigned. An alternate behavior can be assigned to an agent when a particular condition occurs and essentially redefines all attributes of the agent.
- *MOEList* (mandatory) – identifies user-selectable MOEs that will be computed during execution of the scenario.

In contrast to the MSDL schema design, the Pythagoras schema defines all elements globally without declaration of named types. Other schemas can only reference the defined elements rather than declaring new elements of the same type (structure, content) for use, restriction, or extension. This approach constrains reuse of Pythagoras constructs to some extent, but is not a major issue for our purposes. The Pythagoras schema also does not declare a target namespace for its globally declared terms.

2.3 Primary Scenario Constructs in IWARS

IWARS is a constructive, force-on-force, combat simulation used to model individual soldier, team, and small-unit combat operations in complex environments, including urban environments. The simulation is used to support analysis of warrior systems. An IWARS XML scenario file consists of the *Scenario* root element, a *Simulation* child element and an *OutputInfo* child element.[5] For our purposes, we examine here the structure of the *Simulation* child element, shown in Figure 3.

[5] Although the *Scenario* element appears as the element, no top-level schema showing the Scenario element at the root of the scenario files could be found in the IWARS software distribution.

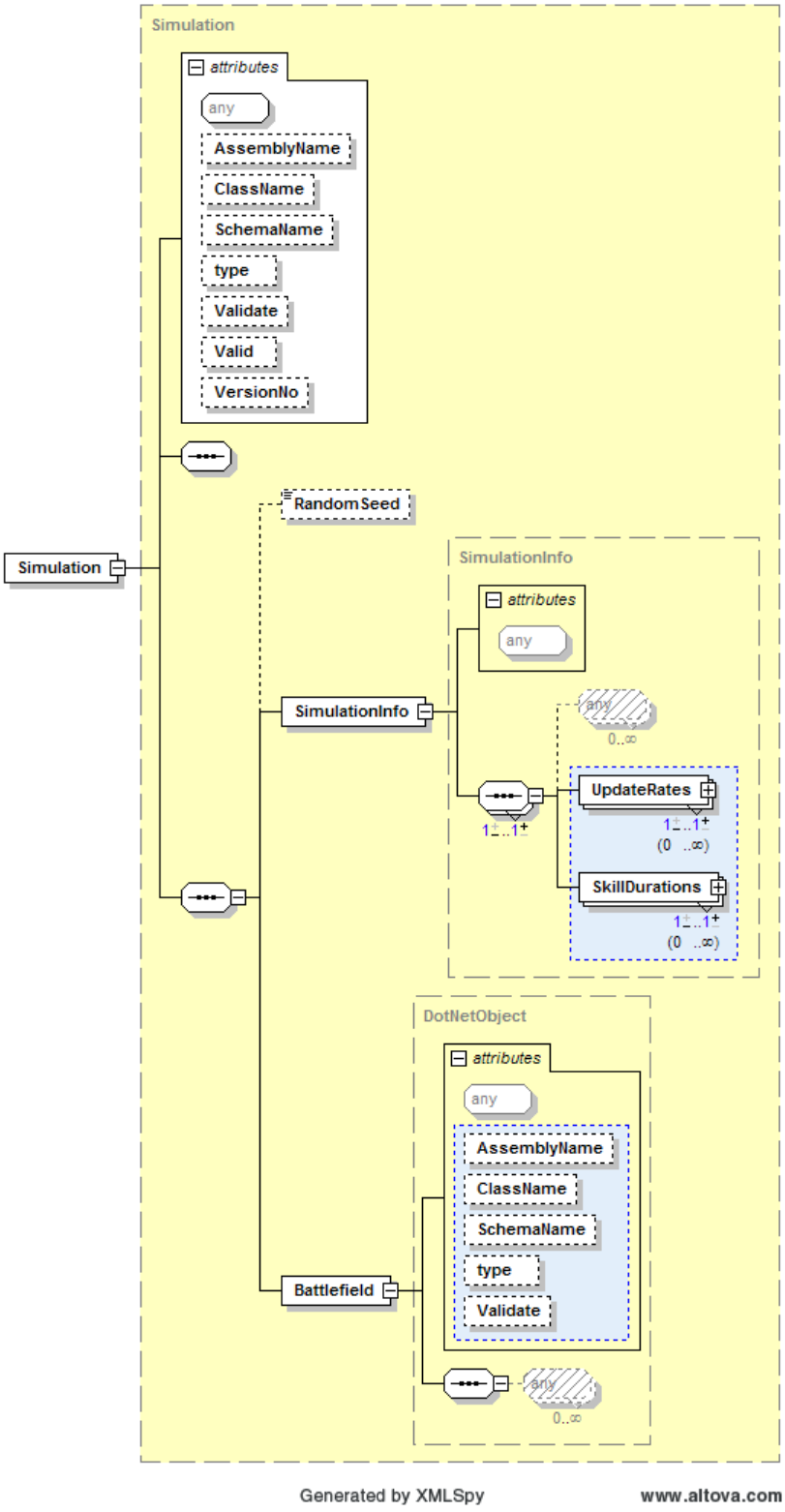

Figure 3. Top-Level Schema Structure of IWARS Scenario Files (Simulation Element)

A complete description of IWARS data is beyond the scope of this paper. The reader is referred to the IWARS User Guide [5] for a full description of the model capabilities and data entry.

Perhaps the most interesting aspect of the schema design shown above is the openness of the top-level structure. The use of the *any* construct, as discussed earlier, allows any elements to be inserted into the structure. This approach allows a great deal of flexibility, although it makes validation (in the XML sense[6]) of scenario files difficult—just about any content would be considered valid, even though the data would not necessarily be loadable by the IWARS application. The IWARS application overcomes this problem by identifying component schemas in the *SchemaName* attribute seen in the *Simulation* and *Battlefield* elements in the figure. However, this is not a useful approach for other applications wishing to read IWARS scenario files through standard XML mechanisms.

IWARS has a separate XML schema describing structure and content of the Battlefield element in more detail. Its overall structure is shown in

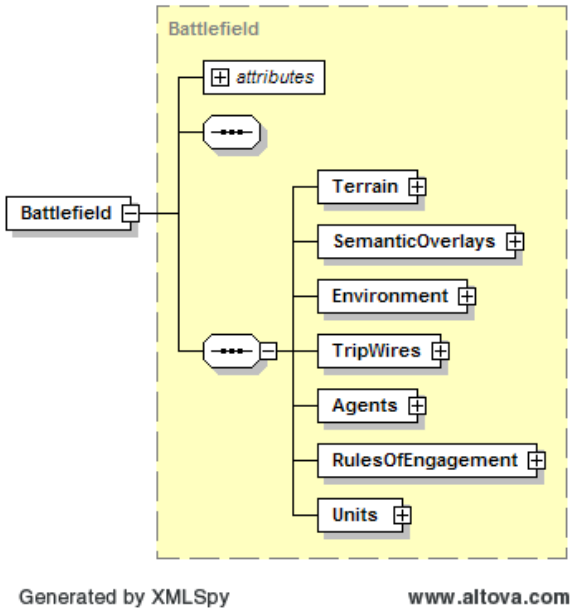

Figure 4. Structure of the IWARS Battlefield Element

A brief description of each of the child elements of the Battlefield element is provided below (these descriptions are illustrative, not exhaustive):

- *Terrain* (mandatory) – identifies the terrain file to be used in the scenario. IWARS terrain files are generated from Open Flight terrain databases and used in the IWARS Environment Engine to give physical meaning to the polygons and textures described in the terrain database.
- *SemanticOverlays* (mandatory) – describes user-created features represented in the environment, such as waypoints and area targets. These are expressed as point features (e.g., waypoints), linear features (e.g., for tripwires, paths, trenches, and tunnels), area features (e.g., area targets), arc-node networks (e.g., used by agents for path selection), and stochastic shields (e.g., terrain features or buildings that an agent may use as cover and/or concealment but that are not explicitly represented in the terrain database).
- *Environment* (mandatory) – provides characteristics of the physical environment, such as *AmbientTemperature*, *RadiantTemperature*, *RelativeHumidity*, *WindSpeed*, *WindDirection*, *Season*, *TimeOfDay*, *Background*, *CloudCover*, *Clutter*, *Location*, and *MeteorologicalVisibilityRange*.
- *TripWires* (mandatory) – describe tripwires, providing *Name*, *LinearFeatureId*, and *BattleFieldEvent* (i.e., trip event and event location).
- *Agents* (mandatory) – a list of one or more *Agent* elements. Each Agent represents a Human in the scenario described by *Name*, *Posture*, *Location* (X, Y, Z coordinates), *FacingDirection*, *Size*, *Weight*, *FatPercentage*, *Sex*, *Age*, *FieldOfRegard*, *EquipmentList*, *SelectedVisualSensor*, *SelectedWeapon*, *SelectedMount*, *SelectedProtection*, *Knowledge*, *BehaviorEngine*, *UnitId*, and *MemberType*.
- *RulesOfEngagement* (mandatory) – defines for each force the circumstances under which they are to fire at members of other forces.
- *Units* (mandatory) – defines one or more Units, where each Unit is composed of agents assigned to specific roles (e.g., Fireteam Leader, SAW Gunner, Grenadier, Rifleman) in the Unit. Each *Unit* element has a *Name*, *Force*, *UnitDescription*, *Formation*, optional set of *UnitMembers*, and optional *UnitMemberType*.

[6] XML validation is performed by checking an XML instance document (containing data) against the governing XML Schema document(s). The XML instance document is *valid* if its content meets the structure and content specifications of the governing XML Schema document(s). Many applications, including Web browsers, automatically check instance document validity when a governing schema is identified.

3. Comparison of Scenario Constructs

A detailed comparison of scenario constructs between MSDL and Pythagoras was provided previously in [6, 7]. The goal was to map information from a Pythagoras scenario file to an MSDL file as the starting point for capabilities of the RSG tool being developed by TRAC-Monterey. This work examined the MSDL XML document structure to determine what information can be mapped to it, directly or through some computational translation, from the Pythagoras XML scenario structure. Since that time, additional work was performed to examine IWARS XML data structures to see what can be mapped from that data model to/from MSDL. The work was equally interested in seeing where mappings are not possible due to incompatible conceptual models or structures which would require an augmented structure to preserve application-specific information while maximizing interchange opportunities through use of the MSDL standard.

Detailed comparison of the data models is beyond the scope and size limitations of this paper. In short, examination of the three data models and the two applications revealed few areas of strict commonality where information from one data model can be directly mapped (transferred) to a data structure in another model. The greatest immediate utility is found in identification of sides and forces (units, agents) with initial positioning and preliminary behaviors (e.g., initial movement actions). This is illustrated in more detail in Section 5 below.

4. Hybrid MSDL Structures for Embedding Pythagoras and IWARS-Specific Content

As mentioned in the previous section, numerous portions of Pythagoras and IWARS initialization data are not directly transferable to MSDL XML structures. For example, Pythagoras performance data, such as movement speeds and sensor and weapon ranges, as well as behavior triggers and other details have no analog in the MSDL structure. Even so, some of the weapon and sensor parameters in Pythagoras input files can be meaningfully transferred to scenarios constructed for other systems, such as IWARS or the Map Aware Non-uniform Automata (MANA) agent-based simulation. Nonetheless, there are several use cases of interest to the current work that motivate the use of MSDL as an archival and interchange format:

- A user develops a scenario in a "simulation-agnostic" application. The scenario information is saved to MSDL for archival purposes and for use by other MSDL-capable systems.
- A user develops a scenario in Pythagoras that is saved in Pythagoras XML format. The user desires to transform the Pythagoras scenario data to MSDL format for archival purposes and for use by other MSDL-capable systems. The transformation needs to (1) extract as much information as possible from the Pythagoras file to "fill" information constructs in the MSDL file, (2) prompt the user or otherwise fill in mandatory MSDL information not available from the Pythagoras file, and (3) save the rest of the Pythagoras file for later regeneration of the original Pythagoras scenario file as needed.
- Same as above, but substituting "IWARS" for "Pythagoras."
- The user develops a scenario in a "simulation-agnostic" application. The application provides the option to save out the scenario data in a simulation-specific format (in our case, Pythagoras and IWARS). Information not available in the application to fill out mandatory information in the target simulation is obtained from "dummy" scenario files so that the exported file meets all minimum XML validity requirements in the target format.

One strategy for producing MSDL from application-specific initialization files is to write out (transform) only the elements from the application-specific file that can be mapped to MSDL elements. This may produce a partial MSDL file that would need supplemental data entry, possibly through an extension to a "File-Export" menu selection to prompt the user for that information. Another approach is to create (on export or save) a hybrid MSDL/application-specific document governed by an XML schema that uses definitions from the respective schema documents to describe a composite document. This approach would preserve all application-specific content in the composite document while also providing as much of the information in MSDL XML structure as possible. Through simple XSLT files, applications or users can then re-create MSDL-specific or application-specific XML files that will validate against their respective XML schemas. In either case, the only concern is obtaining sufficient information from source application-specific scenario files to fully populate all mandatory elements in a valid MSDL XML document, and vice versa. This can be solved by having dummy initialization files available in each format and having the tool user interface or software application prompt the user for any additional information needed to generate (or be used to generate) a valid XML document in the desired format.

In the XML schema structure for MSDL, there is one area where literally anything can be added to the language and still obtain a valid XML document conforming to the MSDL schema. As introduced earlier, the first child element, *ScenarioID*, of the *MilitaryScenario* root element has complex structure defined from the BOM specification. Of particular interest is the final child element in the *ScenarioID* structure, defined in the XML schema as follows:

```
<xs:any namespace="##other" minOccurs="0"
maxOccurs="unbounded" processContents="lax"/>
```

The <xs:any> declaration is called an *element wildcard*. This construct allows the entry of one or more elements from any namespace into this part of the structure of an XML document conforming to the MSDL schema. The "##other" value for the namespace attribute allows elements from namespaces other than the defined target namespace (in this case, the BOM namespace http://www.sisostds.org/schemas/modelID) to be included as part of the wildcard. The processContents="lax" attribute instructs the processor to attempt to validate the wildcard elements if it has access to a global XML Schema definition for them (more on this later).

Consider the following notional (and minimal) MSDL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2008
(http://www.altova.com)-->
<msdl:MilitaryScenario
xsi:schemaLocation="urn:sisostds:scenario:military:data:draft:msdl:1
MilitaryScenario_1.0.0.xsd"
xmlns:modelID="http://www.sisostds.org/schemas/modelID"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:msdl="urn:sisostds:scenario:military:data:draft:msdl:1">
   <msdl:ScenarioID idtag="ID_1" notes="ID_1">
      <modelID:name idtag="ID_2"
notes="ID_1">NCName</modelID:name>
      <modelID:type idtag="ID_3"
notes="ID_1">FOM</modelID:type>
      <modelID:version idtag="ID_4"
notes="ID_1">a</modelID:version>
      <modelID:modificationDate idtag="ID_5" notes="ID_1">1967-
08-13</modelID:modificationDate>
      <modelID:securityClassification idtag="ID_6"
notes="ID_1">Unclassified</modelID:securityClassification>
      <modelID:description idtag="ID_7"
notes="ID_1">a</modelID:description>
      <modelID:poc idtag="ID_8" notes="ID_1">
         <modelID:pocType idtag="ID_9" notes="ID_1">Primary
author</modelID:pocType>
         <modelID:pocEmail idtag="ID_10"
notes="ID_1">String</modelID:pocEmail>
      </modelID:poc>
   </msdl:ScenarioID>
   <msdl:Options>
      <msdl:MSDLVersion/>
   </msdl:Options>
   <msdl:ForceSides>
      <msdl:ForceSide>
         <msdl:ObjectHandle>00000000-0000-0000-0000-
000000000000</msdl:ObjectHandle>
         <msdl:ForceSideName/>
      </msdl:ForceSide>
   </msdl:ForceSides>
</msdl:MilitaryScenario>
```

This file validates against the MSDL schema. If we add the following content after the <modelID:poc> field, the file is still valid against the MSDL schemas:

```
<p:pythagoras xmlns:p="some/notional/pythagoras/namespace">
            <modelVersion>
                  <major>1</major>
                  <minor>10</minor>
                  <bugFix>2</bugFix>
            </modelVersion>
            <scenarioDescription>Terrain: City Core Terrain
Scenario: Movement to Contact - PLATOON engagement Enemy: 4
* 4-man teams Civilain: 15 Hostile Civilian:
15</scenarioDescription>
            <timeStep>1250</timeStep>
</p:pythagoras>
```

However, the processContents attribute is actually absent in the ModelID_v2006_FINAL.xsd schema used by MSDL. This means that a processor will attempt to validate the wildcard elements and will raise a validity error if a global XML Schema definition for the wildcard elements cannot be found (same behavior as when processContents="strict"). We added processContents="lax" to the MSDL schema to enable the example to validate. A better way, but more complex, is to create a composite schema that could import the namespaces and schemas for the various components we want to include in the structure. In our case, the composite schema would import the MSDL MilitaryScenario schema (which, in turn, imports the other MSDL schemas) and the Pythagoras schema, and would then have a single element declared as being of type msdl:MilitaryScenarioType. The following illustrates this idea with a notional schema and XML content in place of Pythagoras data (or any other data).

The approach starts with a simple XML schema that has a single element of type msdl:MilitaryScenarioType, but with additional namespace declarations and schema imports for other content that will be stored in the MSDL MilitaryScenario element structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdl="urn:sisostds:scenario:military:data:draft:msdl:1"
xmlns="http://some/other/namespace"
targetNamespace="http://some/other/namespace"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:import
namespace="urn:sisostds:scenario:military:data:draft:msdl:1"
schemaLocation="MsdlComplexTypes_1.0.0.xsd"/>
   <xs:import namespace="http://myNamespace"
schemaLocation="SomeData.xsd"/>
```

```
   <xs:element name="RSGScenario"
type="msdl:MilitaryScenarioType">
      <xs:annotation>
         <xs:documentation>Root element of an RSG scenario file.
Uses MSDL type MilitaryScenarioType to hold MSDL data with
inserted content from other scenario language namespace(s) (e.g.,
Pythagoras, IWARS).</xs:documentation>
      </xs:annotation>
   </xs:element>
</xs:schema>
```

Let the following XML Schema represent some data from other simulation models (e.g., Pythagoras, IWARS, etc.):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://myNamespace"
targetNamespace="http://myNamespace"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="SomeData" type="xs:string">
      <xs:annotation>
         <xs:documentation>Comment describing your root
element</xs:documentation>
      </xs:annotation>
   </xs:element>
</xs:schema>
```

The point is, this can be any schema for any XML language.

In the MSDL data file, we change the root element from msdl:MilitaryScenario to our new r:RSGScenario element (where r: is the prefix for the namespace of our composite scenario schema), and insert the structure of our arbitrary language (in this case, just the element SomeData), with its own namespace in place of the Pythagoras element we showed earlier. The new file, shown in entirety below, now validates against the original MSDL schema files, but via the reference from our new RSGScenario schema. In addition, the SomeData section is also validated against its schema since that was imported into the RSGScenario schema file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2008
(http://www.altova.com)-->
<rsg:RSGScenario
xsi:schemaLocation="http://some/other/namespace
RSGScenario.xsd" xmlns:rsg="http://some/other/namespace"
xmlns:modelID="http://www.sisostds.org/schemas/modelID"
xmlns:oth="http://myNamespace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:msdl="urn:sisostds:scenario:military:data:draft:msdl:1">
   <msdl:ScenarioID idtag="ID_1" notes="ID_1">
      <modelID:name idtag="ID_2"
notes="ID_1">NCName</modelID:name>
      <modelID:type idtag="ID_3"
notes="ID_1">FOM</modelID:type>
      <modelID:version idtag="ID_4"
notes="ID_1">a</modelID:version>
      <modelID:modificationDate idtag="ID_5" notes="ID_1">1967-
08-13</modelID:modificationDate>
      <modelID:securityClassification idtag="ID_6"
notes="ID_1">Unclassified</modelID:securityClassification>
      <modelID:description idtag="ID_7"
notes="ID_1">a</modelID:description>
      <modelID:poc idtag="ID_8" notes="ID_1">
         <modelID:pocType idtag="ID_9" notes="ID_1">Primary
author</modelID:pocType>
         <modelID:pocEmail idtag="ID_10"
notes="ID_1">String</modelID:pocEmail>
      </modelID:poc>
      <oth:SomeData>Another namespace and data</oth:SomeData>
   </msdl:ScenarioID>
   <msdl:Options>
      <msdl:MSDLVersion/>
   </msdl:Options>
   <msdl:ForceSides>
      <msdl:ForceSide>
         <msdl:ObjectHandle>00000000-0000-0000-0000-
000000000000</msdl:ObjectHandle>
         <msdl:ForceSideName/>
      </msdl:ForceSide>
   </msdl:ForceSides>
</rsg:RSGScenario>
```

Bottom line, by just adding our new RSGScenario schema, using it to import schemas for any other languages we wish to include in the MSDL structure, and substituting our RSGScenario root element for the original msdl:MilitaryScenario root element, we are able to store *and validate* any content we want within the msdl:MilitaryScenarioType structure.

Another simple alternative would be to define the RSGScenario to be a container for the full msdl:MilitaryScenario content, as in:

```
<xs:element name=”RSGScenario>
   <xs:complexType>
      <xs:sequence>
         <xs:element ref=”msdl:MilitaryScenario” minOccurs=”1”
maxOccurs=”1”/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Either approach appears to be fairly easy for software to accomplish.

5. Practical Application: The Rapid Scenario Generation Tool

Scenario generation tends to be a time-intensive process, and in many cases involves repeating steps and duplicating previous modeling activities. TRAC-Monterey prototyped the Rapid Scenario Generation (RSG) tool to reduce time and learning requirements to generate an executable scenario file through the reuse of previously developed and documented scenario components. The RSG tool attempts to simplify development of scenarios through an emphasis on standard terms and symbols and composition of scenarios from libraries of saved scenario components. A goal of the RSG tool is to provide inputs to multiple systems. This can be achieved for individual systems

by specialized export formats, such as is currently implemented for creation of Pythagoras and IWARS input files, or more broadly through use of MSDL as the common interchange format. The RSG tool employs MSDL XML structures to store the representation of an operational situation in a common format. This format can be read directly by any MSDL-capable system. The RSG tool also provides the capability to generate application-specific data models for Pythagoras and IWARS from the common MSDL format. The approach can be extended to enable data exchange across other systems having unique data models.

The RSG tool provides a graphical user interface (GUI) for course of action (COA) sketch development that incorporates common operational terms and symbols. The COA sketch consists of entities of multiple sides, routes, objectives, and terrain. The RSG tool is not intended to develop a complete scenario for any particular simulation, but requires the user to manipulate aspects of the scenario construction using the target simulation (i.e., in our case, Pythagoras or IWARS). This focus on commonality of scenario data versus specialization of scenario data aligns well with the MSDL design philosophy.

Users may develop COA sketches from scratch using the RSG tool, or import scenarios from Pythagoras or IWARS as a starting point. After completing the COA sketch, users may export the executable file scenario to either Pythagoras or IWARS. For instance, the user may import a Pythagoras scenario to the RSG tool, edit the scenario with COA sketch components, and export the executable scenario file to IWARS data format. When a scenario is saved from RSG for storage in a file system, it is saved as MSDL. The tool is able to read the MSDL file into internal software objects for manipulation within the tool, and subsequent exporting to other simulation-specific XML formats at the user's discretion. The following series of figures illustrates development of COA sketches from scratch, and importing and exporting scenarios between Pythagoras and IWARS.

Figure 5 shows a COA sketch developed from scratch using the RSG tool. The analyst selected the desired terrain (background image), defined red and blue sides, identified starting locations for two blue entities and one red entity, and selected an axis of advance from the blue entities to approach the red entity. The left panel highlights the common terms and symbols used to develop the COA sketch. Symbols are easily dragged and dropped onto the play box, edited as necessary, then assigned to an agent. Note that entity and axis of advance symbols conform to the MIL-STD-2525 symbology standard. The terrain shown in Figure 5 is a section of Fallujah, Iraq that the analyst selected using the X3D (Extensible 3D Graphics international standard for 3D graphics on the Web; see http://www.web3d.org) tool implemented within the RSG tool.

RSG gives the analyst the option to export the COA sketch depicted in Figure 5 to Pythagoras or IWARS, capturing sides, entities, starting locations, routes, and a limited number of behaviors. The Fallujah terrain can also be exported to Pythagoras, but not IWARS, due to differences in terrain representations. IWARS currently only supports four terrain files using a custom binary format. None of these match this Fallujah locale. Even so, the initial conditions can be saved into IWARS for further manipulation in that tool for a location of interest.

Figure 6 shows an IWARS scenario imported to the RSG tool. The terrain for the IWARS scenario is the McKenna urban terrain site from Fort Benning, GA. The RSG tool enables editing of the IWARS scenario, to include entity starting locations, routes, and limited behaviors.

Figure 7 depicts the IWARS scenario edited within the RSG tool. In this example, the user modified blue force starting locations and developed an axis of advance for one blue entity.

Finally, Figure 8 shows the edited IWARS scenario exported to Pythagoras utilizing MSDL data exchange formats. Pythagoras does not support MIL-STD-2525 symbology, depicting entities as circles and the axis of advance as a series of way points. The scenario shown in Figure 8 is fully executable in Pythagoras.

The Rapid Scenario Generation codebase utilizes Open Source projects as core components of its implementation; principally, (1) ChefX3D, a scene authoring toolkit; and (2) Xj3D, a 3D runtime viewer. Using these toolkits allowed the RSG developers to rapidly create a full-featured application without having to design and develop many of the standard GUI tasks, such as undo/redo of commands, a catalog of tools, a top-down 2D editor framework, a 3D view of the earth with satellite imagery, and much more. Reuse of these features reduced the development time dramatically, while giving the application many rich features out of the box.

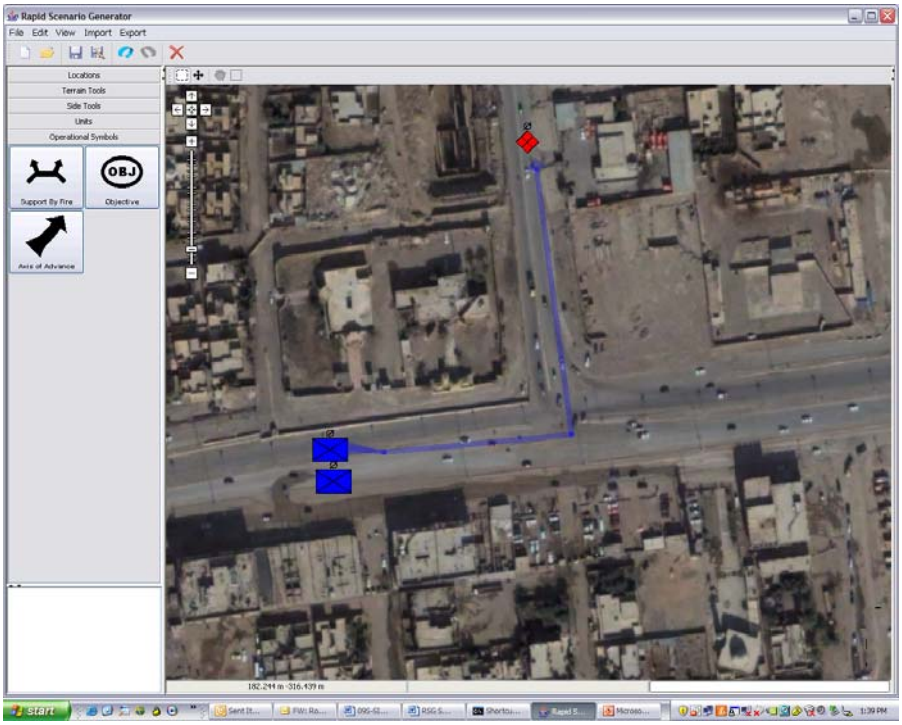

Figure 5. COA sketch developed from scratch using the RSG tool

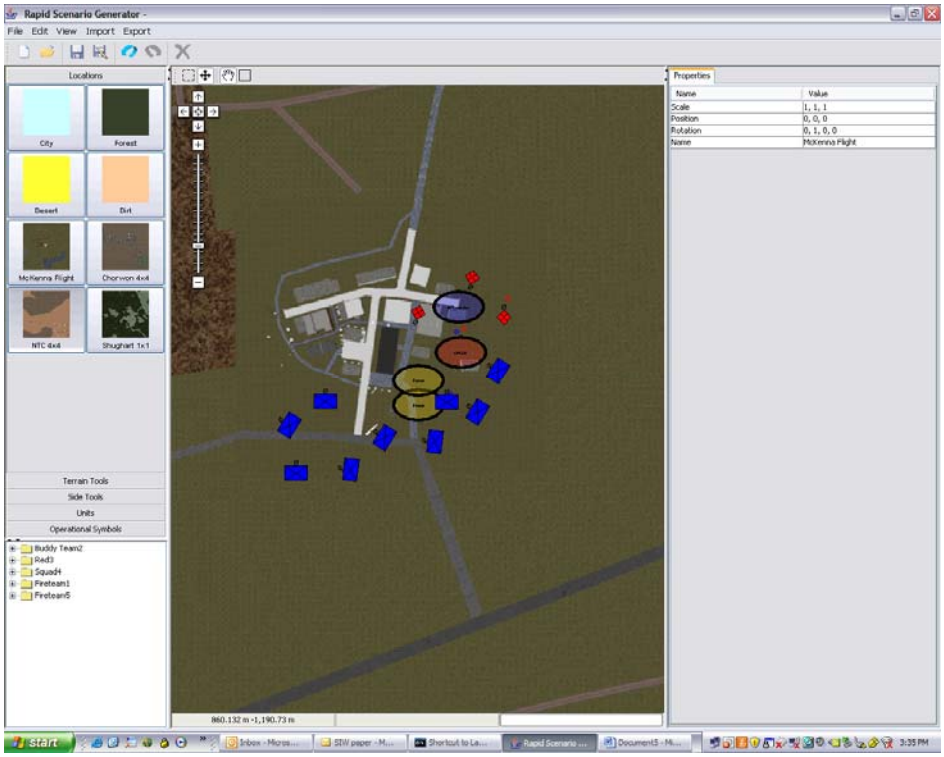

Figure 6. IWARS scenario imported to the RSG tool

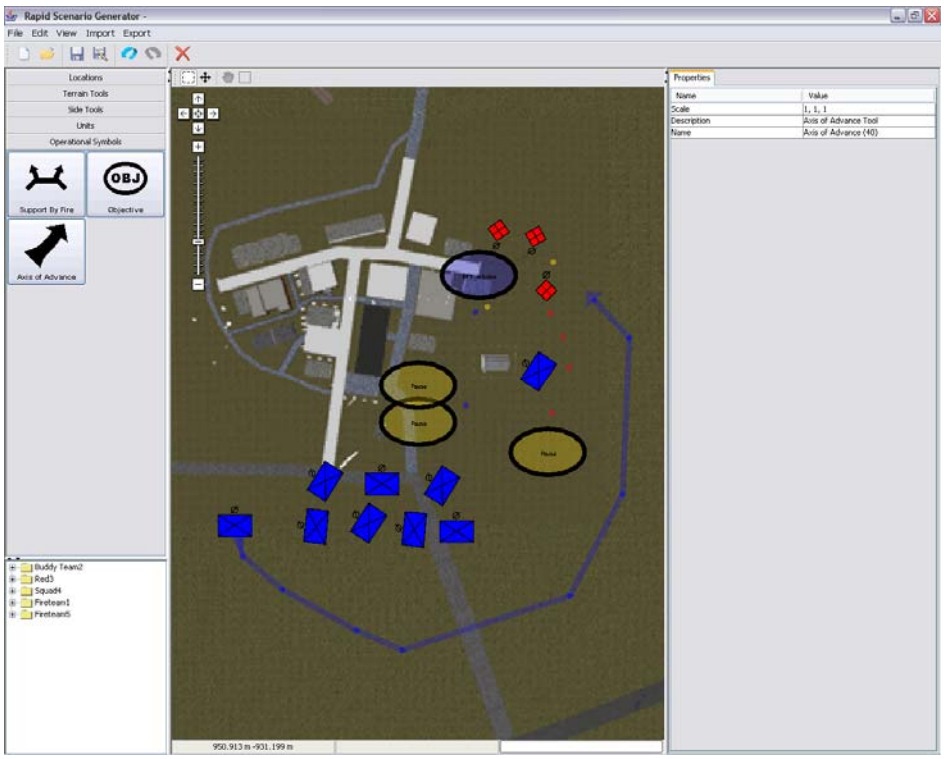

Figure 7. IWARS scenario edited in the RSG tool

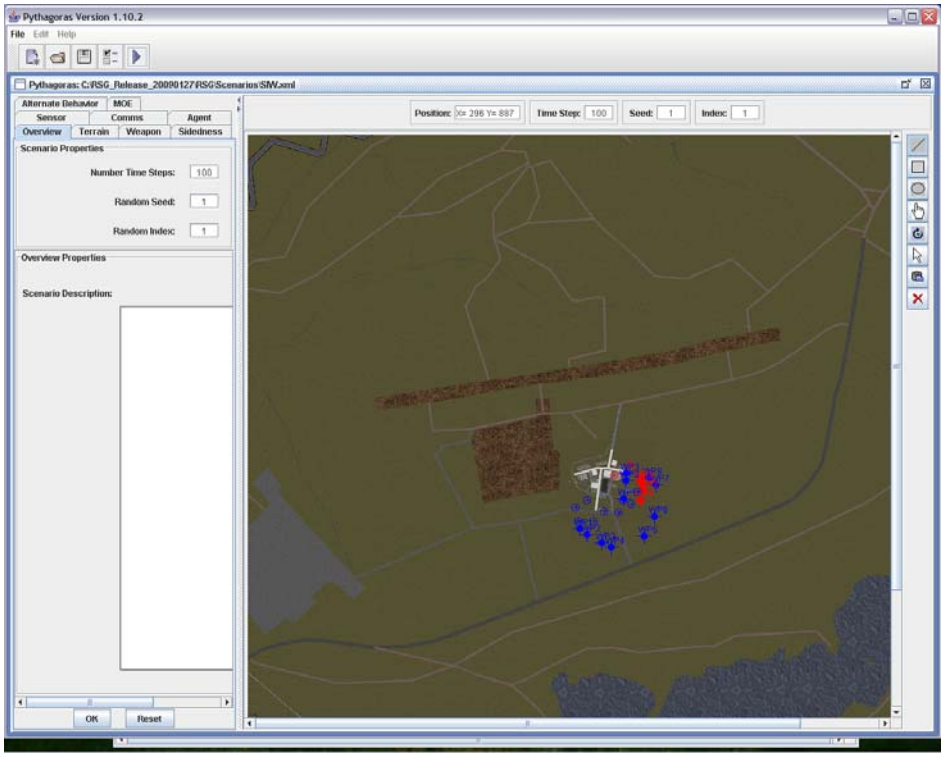

Figure 8. Edited IWARS scenario exported to Pythagoras

ChefX3D has a generic property framework which was used to store the incoming XML from the target simulation. Properties that were determined to be editable are mapped to the ChefX3D property space, while the remaining XML is stored so no data is lost. When exporting, the edited and saved data are merged back into the target XML structures. The ChefX3D property space provides a generic, flexible repository that is used in RSG as a translation medium between systems.

6. Toward Greater Scenario Data Interchange

In constructing MSDL scenario documents for interchange with other systems, we have seen that application-specific information that is not mappable to MSDL constructs can simply be embedded in the MSDL file as Pythagoras- or IWARS-specific content to become part of a scenario repository. There will always be conceptual mismatches between the data contained in an application-specific scenario file and the data contained in the general scenario definition language provided by the MSDL standard. Such mismatches do not negate the value of the effort to pull as much as possible from the application-specific scenario structures to/from MSDL structures. Whatever portion of the MSDL file can be considered populated from an application-specific format, this is a portion that will not require re-creation from scratch in some other simulation system that is able to read and process MSDL. This translates to reduced effort and increased scenario commonality, both of which benefit the community. The conceptual mismatch also serves an important purpose in identifying a potential direction for future enhancement of the MSDL specification. If the community determines that there are characteristic information structures in certain simulations, then it may be worthwhile to promote and design an extension to the MSDL specification to accommodate such structures explicitly. Investigations of this type may also lead us to deeper understanding of the fundamental information elements and semantics of scenarios for a wider range of simulations, in turn leading toward enhancements to MSDL or research into other technical approaches for aligning the conceptual models of a broad set of systems.

7. Conclusion

Development of the RSG tool demonstrates the value of MSDL for scenario description and interchange across numerous systems. As various organizations move forward on development of scenario generation tools using MSDL as the common interchange format to produce initialization data for multiple simulation programs (for example, see [8,9]), more insights into the capabilities and limitations of the standard will become known. MSDL has a well-defined scope of coverage that maintains coherence and ease of use to help achieve the greatest level of acceptance. Properties of XML permit namespaces to be used to extend the language for systems requiring more detailed information or a different set of concepts than those included in the MSDL specification. Others in the simulation interoperability community are encouraged to conduct similar analyses and development to further inform the community on the opportunities and techniques for employing MSDL for storing and sharing scenario information. Through adoption and use by the M&S community, best practices will emerge for creating effective interchange through software and XML-based means.

8. References

[1] Simulation Interoperability Standards Organization: Standard for Military Scenario Definition Language (MSDL), SISO-STD-007-2008, 14 October 2008.

[2] Department of Defense: Military Support for Stability, Security, Transition, and Reconstruction (SSTR) Operations, Department of Defense Directive Number 3000.05, 28 November 2005.

[3] Simulation Interoperability Standards Organization: Base Object Model (BOM) Template Standard, SISO-STD-003-2006, May 2006.

[4] Northrop Grumman: Pythagoras Manual, Version 1.10.2 (available in software distribution).

[5] AMSAA: IWARS User Guide, Version 1.0 Release 5, Army Materiel Systems Analysis Activity, May 2008.

[6] Alt, J., T. Lucas, S. Sanchez, P. Sanchez, C. Blais, and J. Pearman : Chemical, Biological, Radiological and Nuclear Tactical Situation Awareness, TRAC-M-TR-07-016, TRADOC Analysis Center, Monterey, 15 September 2007.

[7] Blais, C.: "Military Scenario Description Language: How Broadly Can It Be Applied?", Paper 08F-SIW-002, *Proceedings of the Spring Simulation Interoperability Workshop*, Simulation Interoperability Standards Organization, Orlando FL, September 2008.

[8] le Roux, W. H.: "Investigation into the Usability of MSDL in South African C2 Tactical Simulations," *Proceedings of the European Simulation Interoperability Workshop*, Simulation Interoperability Standards Organization, June, 2007.

[9] Ullner, F., A. Lundgren, J. Blomberg, N. Andersson, and P. M. Gustavsson: "The Lessons Learned from Implementing a MSDL Scenario Editor", Paper 08F-SIW-001, *Proceedings of the Spring Simulation Interoperability Workshop*, Simulation Interoperability Standards Organization, Orlando FL, September 2008.

Disclaimer

The opinions expressed in this paper are those of the authors and do not necessarily represent those of the Naval Postgraduate School, TRAC-Monterey, or any sponsoring or partner organizations.

Author Biography

CURTIS BLAIS is a Research Assistant in the Modeling, Virtual Environments, and Simulation (MOVES) Institute at the Naval Postgraduate School, Monterey, California. Mr. Blais conducts research in a variety of areas, including design of agent-based simulation of non-traditional warfare and application of Semantic Web technologies to address effective interchange of information across M&S and C2 systems. Mr. Blais is also a Ph.D. candidate in MOVES investigating ontological frameworks for obtaining valued information at the right time in semantically-rich network-centric architectures.

RUSSELL DODDS is a software engineer with over 10 years of experience in a wide range of fields including, telecommunications, internet hosting solutions, education, and simulation/training. Primarily his focus has been on client and middle tier applications for both rich client and web-based applications. For the past 2 years he has been a senior developer for Yumetech, software consulting company specializing in meeting business and government 3D Visualization and Simulation software needs.

JERRY PEARMAN is a retired Army officer with experience as an operations research analyst for the US Army TRADOC Analysis Center (TRAC) - Monterey, and combat aviation experience during Operation Desert Shield/Desert Storm and Operation Iraqi Freedom. Mr Pearman is currently a contractor for Augustine Consulting, Inc. supporting TRAC-Monterey.

FRANCISCO R. BAEZ is a Major in the U.S. Army. He currently serves as an Operations Research Analyst in the U.S. Army Training and Doctrine Command Analysis Center in Monterey, California. He received his B.S. in Logistics from the University of Puerto Rico, and his M.S. in Operations Research from the U.S. Naval Postgraduate School.