Simulation Templates in the SUMMIT System

Ernest Friedman-Hill
Todd Plantenga
Heidi Ammerlahn
Sandia National Laboratories
PO Box 969
Livermore, CA 94550
ejfrie@sandia.gov, tplan@sandia.gov, hrammer@sandia.gov

Keywords:
Simulation, Integration, Templates, Assembly

ABSTRACT: Emergency management personnel at all levels can benefit from the use of simulation as a planning or exercise support tool, but knowledge of existing resources and the expertise needed to provision a simulation with data, execute it, and interpret and synthesize results are not uniformly available to all. To meet these needs, the DHS Science & Technology Directorate is funding the development of a Standard Unified Modeling and Mapping Integration Toolkit (SUMMIT). SUMMIT brings distributed simulation codes together with metadata, heuristic domain knowledge, a uniform interface, integration capability and automation. The system emphasizes integration of existing resources around the central notion of simulation templates, which serve as a conduit through which experts can package a category of models together with domain knowledge and best practices for their use. Simulation templates provide an abstraction that presents different interfaces to model users and model owners, while hiding the details of model assembly and execution. This paper will discuss the initial SUMMIT architecture and demonstrate how simulation templates allow for guided discovery, provisioning, combination, assembly and presentation of simulations. SUMMIT is platform and runtime agnostic, but this paper will focus on a cascade-model implementation in Java.

1. Introduction

Emergency response professionals cannot consistently identify and use best-in-class modeling and simulation tools, and their underlying data and domain expertise, across all-hazards planning, training and exercises, and operations. Hundreds of potentially useful models exist across academia, commercial industry, national laboratories, and government; for example, the National Center for the Study of Preparedness and Catastrophic Event Response (PACER), a DHS Center of Excellence, has recently created a list of more than 110 simulation models of relevance to catastrophic event planning and response [1]. Making these models available and useful to the emergency response community has been recognized as a capability gap through the Department of

Homeland Security’s (DHS) Integrated Product Team (IPT) [2] for Incident Management, which includes the Federal Emergency Management Agency (FEMA) and the Office of Emergency Communications (OEC) as key stakeholders.

The Integrated Modeling, Mapping, and Simulation (IMMS) Program is a DHS S&T-funded research program to address this gap. The IMMS program vision is to create a capability for linking together modeling and simulation tools to enable analysts, emergency planners, and incident managers to more effectively, economically, and rapidly prepare, analyze, train, and respond to real or potential catastrophic events.

To realize this vision, a software architecture called SUMMIT – the Standard Unified Modeling and Mapping Integration Toolkit – is being iteratively designed and prototyped. Currently, SUMMIT requirements have been identified, an initial architecture design has been defined, and a prototype reference implementation of key SUMMIT services developed. Subsequent phases of the program will refine the architecture specification and implementation, and incorporate feedback and lessons learned from prototype deployments and pilots.

---

1 Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company for the United States Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000. This work was funded by the Infrastructure & Geophysical Division of the Department of Homeland Security’s Science & Technology Directorate (DHS/S&T/IGD) and managed by the National Aeronautics and Space Administration (NASA).
This paper describes several key aspects of the first SUMMIT architecture design along with the SUMMIT 1.0 reference implementation.

2. Design Goals
The SUMMIT system is intended to help emergency planning and response users discover and utilize a wide range of relevant integrated modeling and simulation tools.

Flexibility is the primary design principle for SUMMIT since the architecture must be able to accommodate many different kinds of modeling and simulation tools, span a broad suite of hazard and catastrophic event domains, and support multiple modes of use including planning, training/exercises, and operations. To this end, the SUMMIT architecture focuses on the development of design concepts and proof-of-concept implementations which leverage existing technologies and standards. Two key design concepts are Resource Discovery and Resource Integration.

2.1. Resource Discovery
One of the fundamental design assumptions for the SUMMIT architecture is that the user may not be aware of all of the resources (e.g., simulation models and data) that are available in a given domain. Therefore, the first operation a SUMMIT user typically performs is discovery. After connecting to the SUMMIT system, a user is first queried about the specific scenario(s) of interest. That query process will elicit scenario attributes (e.g., specific hazards or events of interest, geographic locations, time scales, and other parameters) and information about metrics and desired output results (e.g., number of casualties, damage to critical infrastructure).

Using that information, the SUMMIT system will identify general classes of models that could be executed to meet those requirements. The user then chooses specific models to run, configures and submits the computation, and eventually receives annotated results. The results are archived for future reference.

2.2. Model Integration
Our survey of available modeling and simulation tools in the IMMS domain showed that many of the tools are packaged as complete solutions, not intended for integration with other software. Nevertheless, we have identified many situations in which loose integration between simulations is desirable [3]. Simulation integration is a large and difficult problem. The Department of Defense’s High-Level Architecture (HLA) is a well-known contributor in this space [4]. SUMMIT is not intended as an alternative to HLA; it operates at a higher level by identifying appropriate resources and relying on frameworks like HLA for the actual integration.

There are several levels at which resources must be integrated. These include low-level communication, semantic compatibility, and graphical interface aggregation. Each of these areas will require specific attention and is discussed separately below.

We take a robust, reality-based approach to low-level data exchange which maximally leverages existing integration work. Central to our approach is the notion of a federation execution architecture (FEA). We partition the set of all possible simulations into FEAs which are directly integrable based on intent or technology. For example, all simulations built using a particular HLA Runtime Infrastructure (RTI) would comprise an HLA FEA. Within an FEA, many simulations can be directly connected, while others can be easily adapted. This use of direct connections means that simulations will be efficiently usable by the SUMMIT system without special considerations by their developers.

Integration between FEAs will require bridging services. Bridging services will move data between resources belonging to different FEAs. We currently handle this directly for the most common case of serial execution by defining a set of conventions we call the cascade FEA. The cascade FEA uses a file exchange metaphor to integrate resources, similar to the concept of “pipes” in UNIX. It works well for communication between semantically compatible, distinct resources.

We will address semantic compatibility of models by storing metadata about all resources and using this metadata to generate trivial connectors or translators as needed. The metadata description of a resource will refer to concepts in the IMMS ontologies, including domain concepts like “wind field,” mathematical ideas like units, and lower-level information about data representation. SUMMIT will understand a rich set of standard interfaces and formats which resources may expose; a resource that presents a standard software interface will need only trivial metadata to describe it. Whenever possible, these interfaces and formats will be existing standards rather than new inventions. Our emphasis will be on allowing existing resources to be used with the system with little to no modification.

Graphical interface integration and aggregation has traditionally been a difficult problem to tackle without placing strong requirements on simulation developers. We are exploring several approaches ranging from the use of screen-sharing and virtual-worlds technologies, to specifying a mechanism for simulation developers to directly create graphical plug-ins for the SUMMIT client.
3. Architectural Overview

The SUMMIT architecture is designed to allow for maximum flexibility, placing few restrictions on federated models but still providing necessary capabilities for integration. At the core of the architecture is a metadata repository containing information about models and how they can be accessed, executed, and interconnected. A client-side component allows for interaction with the user, while a set of distributed core services provide the system functionality.

The central concept of the SUMMIT architecture is the simulation template. A simulation template generically represents a class of computations that can be performed on behalf of the user that addresses a particular hazard/incident scenario. A simulation template is a directed graph of slots which itself has its own semantics. A slot is a sort of contract; it specifies both the semantics of an idealized simulation model and a software interface for interacting with that model. The software interfaces and graphs are used by the federation runtime builder to assemble compound models and to execute models in the context of the SUMMIT system. The semantics are also used to locate useful models during the discovery process. Figure 1 is a UML class diagram showing how templates and slots are related.

Figure 1 – UML Class Diagram for Simulation Templates

We therefore see that the discovery process previously described should result in a list of applicable simulation templates. In order to run a full simulation, the user selects a simulation template from those discovered for the desired scenario and analysis objective. The user then chooses specific resources to populate the data and model slots. Differentiating characteristics supplied by the resource providers and feedback from previous executions help guide resource selection. The simulation template itself contains concrete details for connecting the resource slots, including what federation technology should serve as the communication and synchronization backbone.

The populated simulation template is ready for execution after the data and model slots are chosen and template-level inputs are specified. The Federation Runtime Builder (FRB) builds an executable Federation Runtime (FR) from a populated simulation template. The FR manages execution detail -- based on template metadata provided by the ontology, the Federation Runtime establishes the appropriate infrastructure for synchronizing model and data communication, starts the appropriate models and establishes data connections, and mediates exchange of information between the user and simulation federates.

The complete SUMMIT architecture design necessarily includes some additional components, especially for storing simulation results and communicating them to the user; however these components are not the focus of the present paper.

4. Reference Implementation

The SUMMIT 1.0 Reference Implementation is entirely written in the Java programming language [5]. The SUMMIT client program is built on the Eclipse

The server-side component of the SUMMIT reference implementation is organized around four separate core services, which can be installed together or distributed across multiple physical computers. These services are the Discovery Engine, the Storage Manager, the Queue Manager, and the Federation Runtime Builder. The Discovery Service implements the discovery protocol discussed in a previous section. The Storage Manager is a simple HTTP-based file server, and is used to store both executable simulations and computed results. The Queue Manager is capable of fetching executable simulations from the Storage Manager and running them, storing the results and making their location available to the SUMMIT client. Finally, the Federation Runtime Builder generates Java code capable of instantiating, initializing, and executing a configured simulation. Figure 3 shows how the core services interact with the client.

![Sequence diagram showing the interactions of the core services in the SUMMIT reference implementation.](image)

4.1. Networking

Ultimately the SUMMIT system is intended for deployment on a wide-area network such as the Internet. This requirement places some significant restrictions on networking mechanisms. SUMMIT can leverage existing firewalls, security policies, and other infrastructure by operating over an HTTPS-based protocol like Web Services, and ultimately that is our intention. Web Services can be easily tunneled through firewalls, and they can easily make use of PKI-based security. They’re widely used and widely understood, making them ideal for adoption by a large number of model builders.

In our initial reference implementation, however, we have concentrated mainly on the discovery and model assembly process, and have used a simpler networking technology (Java Remote Method Invocation, or RMI) [8] for communicating between the client and the core services. In particular, the core services, the desktop client, and the execution service can all run on separate, independent machines, communicating via RMI. Although not supported by the current reference implementation, our planned architecture includes multiple distributed execution services as well.

Besides RMI, the Storage Manager uses HTTP to serve stored files, which is compatible with the overall system goals.

4.2. Model Integration

In general, it is the philosophy of the SUMMIT framework to place very few constraints on developers. Integrating a model into the framework should be simple and quick, and require little learning. Wherever possible, the framework should perform all setup, communication, and teardown duties, freeing developers from unnecessary responsibilities.

The SUMMIT reference implementation does not include an explicit API for model wrapping. To integrate a model with SUMMIT, a developer needs to write a Java class which implements the slot interface for each slot that the model might occupy. The interfaces are comprised of simple accessor and mutator methods, and the general usage paradigm is that the mutators are used to set properties used as inputs, and the accessors are used to extract results. A single no-argument, void-returning method must be provided to execute the model itself, and the class must have a public, no-argument constructor. Models are allowed to assume that the current working directory is available for storing results files.

In addition to the model wrapper itself, a developer must provide some metadata to the SUMMIT system, including the name of the class, the name of the executor method, the location of a jar file containing the wrapper, and additional information about any supplementary files or libraries needed at runtime.

4.3. Template Representation

In the SUMMIT reference implementation, slots are represented by Java interfaces, and each model must provide a wrapper which implements one or more of these
slot interfaces. Templates describe how model inputs and outputs are interconnected at an abstract level; the JavaBeans naming conventions for properties and methods [9] are used to translate those descriptions into concrete code.

The data types of the model properties must either be primitives, or must be chosen from a collection known as the DataType Library (DTL). The DTL includes general types representing concepts like time, temperature, concentration, and geographic location, as well as more specific types representing concepts relevant to emergency planning. Each entry in the DTL includes a Java class representing the type as well as a graphical component that the client can use to create configuration GUIs for a model.

4.4. The Federation Runtime Builder

Models are assembled into executable programs by the Federation Runtime Builder (FRB). The FRB uses the slot and template descriptions to guide the generation of code which instantiates, configures, connects, and executes all the models in a complete simulation. The assembled simulation is then packaged as a standalone application (albeit one configured to run in a specific environment, possibly on a specific machine) which can be archived for later use or immediately sent to the execution manager to run.

In the SUMMIT system, a Federation Runtime (FR) is an executable simulation, including the network of assembled models and their input data. The Federation Runtime Builder (FRB) is a service that accepts a populated simulation template and generates a FR from it. We did not want simulation templates to mandate a specific architecture or implementation, so that in theory a single template could be interpreted in multiple ways through the establishment of appropriate sets of local conventions. For example, FRs based on direct assembly of XML Web Services components, CORBA servants, or HLA federates are possible. The reference implementation is based on serial execution of models conforming to the JavaBeans component architecture, but nothing about the SUMMIT system design requires this.

A core requirement for SUMMIT is that models located on remote servers could be executed in situ. For purely pragmatic reasons this makes a great deal of sense, as many models have complex and idiosyncratic installation, setup, and maintenance requirements that would make it impractical or impossible to install all SUMMIT models on a single server or server farm. On the other hand, WAN communications between multiple models, although sometimes necessary, should be avoided whenever possible in favor of direct connections on a single host. Therefore the FRB creates, but does not execute, FRs, which can then be sent to a remote server on which a model is installed for execution as appropriate.

The task of the FRB is to generate code which instantiates all of the model wrappers, sets all the free inputs (model inputs which are not connected to the outputs of other models) and then executes each model in turn, moving data as required from the outputs of one model to the inputs of the next, making the final outputs available. There are several ways this could be done. First there is the nature of the “glue” code itself: models could be bound together with generated scripts, or real Java code could be generated instead. We chose to use generated code, mainly to make debugging easier. Secondly, the generated code could use late binding and remain largely ignorant of data types, or it could consist of type safe code which rigorously checks data types at compile time. Again, in support of easier debugging and clearer error messages, we chose to use early binding and compile time checking.

There remain a few complexities that must be addressed. The populated simulation template includes values for the free inputs of all the slots. These values include both scalar data and instances of compound data types drawn from a collection called the Data Type Library (DTL). The DTL includes a set of standard types for describing time, geographical coordinates, and other useful quantities. All of these values must somehow be transmitted to the assembled FR, the execution of which may occur at any future time, on another computer. We chose to accomplish this by using the Java serialization APIs to save all the inputs to a file which is then packaged inside the FR. Code in the FR loads the contents of this file when the model runs and applies the inputs to the models by setting the appropriate JavaBeans properties. This has several beneficial side effects: besides making model execution very simple, it simplifies the process of storing simulation runs along with their results.

Sequencing and synchronization of connected models is in general a substantial topic. In our serial reference implementation, the main issue is that the connected graph of the simulation template does not explicitly specify the order in which the models the slots should be executed. The FRB chooses an order by sorting the slots according to their dependencies on outputs from other slots.

5. Future Directions

Using SUMMIT 1.0, we’ve been able to demonstrate working examples of our approach to model discovery, integration, and execution. Based on these experiences, we have defined some priorities for follow-on work.
5.1. SUMMIT Software Development Kit (SDK)
Defining templates, slots, and models in terms of metadata is flexible and expressive, but the RDFS data format is terse and complex. Entering metadata to represent a new resource can be a complex undertaking. We’ll therefore be developing web-based tools for creating, modifying, and querying resource metadata. These include both developer-facing tools for working with individual resources as well as administrator tools for managing an entire SUMMIT installation.

5.2. Networking and Security
RMI communication is a simple mechanism that’s easy to implement and excellent for experimentation, but it works only with Java, lacks security and doesn’t scale well. It is therefore not suitable for large-scale deployment. SUMMIT 2.0 will explore the use of Web Services, based on XML and HTTPS, for communications. By transitioning our reference implementation to use a Web Services model, we will enable the SUMMIT system to leverage existing firewalls and Internet security infrastructure. Furthermore, the cross-language portability of Web Services will allow a diverse ecosystem of SUMMIT implementations to interoperate.

5.3. Concurrent execution and HLA
The FRB in SUMMIT 1.0 uses a cascade architecture for all communications between models: the models are executed serially, and outputs from each model are injected into subsequent models in the sequence as appropriate. The SUMMIT architecture is more general than this, however, and nothing about the architecture precludes parallel execution of models that communicate using an event-based paradigm like the HLA. Future versions of SUMMIT will continue to refine the cascade federation group, and will also add support for a parallel federation group, most likely using the HLA.

5.4. Advanced visualization and scalable access technologies
Finally, APIs for interfacing commercial and next-generation visualization technologies to the SUMMIT architecture will also be prototyped. These technologies will support the collaborative use of SUMMIT-brokered resources by multiple remote users (e.g., through virtual worlds) as well as provide new ways of fusing and exploring the results generated by federated SUMMIT models.

6. References

Author Biographies
ERNEST FRIEDMAN-HILL is a principal member of the technical staff at Sandia National Laboratories. He is the creator of the Jess rule engine and author of Jess in Action from Manning Publications. His research interests include machine intelligence, distributed computing, and simulation.

TODD PLANTENGA is a principal member of the technical staff at Sandia National Laboratories. He has contributed to and led a number of software projects in both research and commercial arenas. He holds a Ph.D. with focus on large-scale scientific computing algorithms.

HEIDI AMMERLAHN is the manager of the Informatics & Decision Sciences department at Sandia National Laboratories. Her research interests include the development and application of simulation tools and advanced information systems to enable better catastrophic event preparedness and response.