

Multisimulations: Towards Next Generation Integrated Simulation Environments

Leila Jalali, Sharad Mehrotra, Nalini Venkatasubramanian,

University of California at Irvine, USA
{jalali, sharad, nalini}@ics.uci.edu

Abstract. In this paper, we consider the challenge of designing a reflective middleware to integrate multiple autonomous simulation models into an integrated simulation environment (multiasimulation) wherein we can model and execute complex scenarios involving multiple simulators. One of the limitations of the simulators is that they are developed by domain experts who have an in-depth understanding of the phenomena being modeled and typically designed to be executed and evaluated independently. Therefore, the grand challenge is to facilitate the process of pulling all of independently created models together into an interoperating multisimulation model where decision makers can explore different alternatives and conduct low cost experiments. We aim to build such integrated simulation environments by creating a loosely coupled federation of pre-existing simulators. We evaluate our proposed methodology via a detailed case study from the emergency response domain by integrating three disparate pre-existing simulators – a fire simulator (CFAST), an evacuation simulator (Drillsim) and a communication simulator (LTESim).

Keywords: Reflective Middleware, Simulation Integration, Metamodels, Methodology.

1 Motivation

In this paper, we consider the challenge of designing a reflective middleware to integrate multiple autonomous simulation models into an integrated environment wherein we can model and execute complex scenarios involving multiple simulators. Modelling and simulation is an important methodology to address a variety of real-world problems; it offers numerous advantages instead of experimenting with the real system itself. Simulation is cheaper, quicker, and enables what-if analyses for better system design [7].

This is particularly true in domains such as emergency response where response plans and methods are validated by simulating disasters and their impact on people and infrastructure. A variety of simulators; e.g., loss estimation tools (HAZUS [13], INLET [14], CAPARS [15], CFD [12]), fire spread simulators (CFAST [10], CCFM [19]), evacuation simulators (DrillSim [9], SDSS [11]), transportation simulators (VISIM [17], PARAMICS [18]) etc., that model different aspects of disasters, their

impacts, and response/mitigation processes have been developed. While these simulators are individually important in understanding disasters, their integrated and concurrent execution can significantly enhance the understanding of the phenomena and interdependencies between multiple aspects of the complex processes. Consider, for instance, a fire simulator, CFAST, that simulates the impact of fire and smoke in a specific region and calculates the evolving distribution of smoke. Since fire and smoke can affect health conditions of individuals in the region of fire, one may wish to further study its impact on the evacuation process as captured within an evacuation simulator, e.g. DrillSim. Similarly, the progress of fire (captured by CFAST) may create infeasible paths/exits for evacuation (as captured by DrillSim). Such what-if analyses can significantly improve the understanding of adverse impacts such as increase in evacuation times or increased exposure to undesired particulates enabling intelligent decision making to improve the response. Not only do we need to understand how will fire and smoke distribute in a specific region, we also need to plan what traffic routes will people use to evacuate the affected regions, what demands will be placed on the hospital services in the region, etc. The individual simulation models such as those for studying the impact of fire and smoke need to be integrated with those analyzing the traffic movement through the highways and arteries of the affected area, and with those analyzing the resource constraints of hospital systems among others.

The need for integrated execution of simulators is well recognized and is the main driver of U. S. Department of Defense (DoD) High Level Architecture (HLA) initiative [4] which has become the de-facto standard technical architecture for military simulations. HLA aims to promote interoperability and reusability among simulators. While HLA is suited to developing new simulators that can be easily integrated, its broader applicability to combining pre-existing simulators is questionable [21]. HLA forces developers to provide a particular functionality or to conform to specific standards in order to participate in the integration process; the rigid assumptions and limitations on participants makes it difficult to integrate pre-existing simulators without significant modification (especially in non-military domains).

In this paper, we consider the problem of integration of pre-existing simulators. We refer to such an integrated simulation environment as a *multisimulation*. We aim to build multisimulations by creating a loosely coupled federation of pre-existing simulators. We explore a reflective middleware approach to address challenges of integrated simulation environments in which interoperability of different simulators can be ultimately achieved in a flexible and efficient manner. Unlike the significant code rewrite required in the HLA case, our multisimulation framework permits individual simulators to maintain their autonomy (i.e. retain their internal representations of time/state etc.), thereby avoiding the need for rigid common interfaces across simulators.

This paper is organized as follows. In Section 2, we discuss our multisimulation architecture and the main challenges. In Section 3 we discuss our methodology for simulation integration that supports the interoperability of multiple existing simulation models. In Section 4 we discuss the implementation of our system prototype. We evaluate our proposed approach via a detailed case study that integrates multiple real world simulators. Finally we draw conclusions.

2 Multisimulation Architecture

We propose a reflective middleware architecture (Figure 1) to support the development of integrated simulation platforms. Our initial efforts [24] focused on using structural reflection [1] to reify (abstract out) the structure of objects and components of the underlying simulators. The base-level consists of the various (pre-existing) simulators that must be integrated. In the proposed architecture, integration of different simulators can be ultimately achieved by using the meta-level for specifying/modeling the properties of the different simulators and reasoning about the interactions among the different simulators. The meta-level is built on base-level simulators; reification of base-level entities yield data structures at the meta-level, modified features of these structures that implement the integration are then reflected to the base-level. A closer look at the base-level simulators themselves reveals that the structural aspects of the simulation application are not merely in the simulator code, backend databases and models stored in domain-specific formats contain aspects of the simulators that may need to be explored as well. In general, there can be many kinds of meta-level entities to cover various integration aspects.

Given the potential black-box nature of simulators developed by experts in diverse domains, we believe that achieving a completely automated plug-and-play integration of simulators is a very difficult, if not infeasible challenge. Our goals are more modest— we intend to develop enabling tools that will simplify the task of simulation integration with a wide range of simulators that vary in the degree to which they expose their interfaces and implementations. Our solution does not require simulator developers to adhere to a strict programming interface or conform to particular design styles - the ability to flexibly interoperate with multiple simulators is our goal.

By using the metamodeling capability the model elements that need to be integrated can be extracted. In other words, in our approach, we formulate the metamodel that captures concepts of interest using a publish- subscribe mechanism for data exchange – here, subscribers (the simulation integration tasks) express interest in aspects that they want to observe (implemented by base-level simulators) –

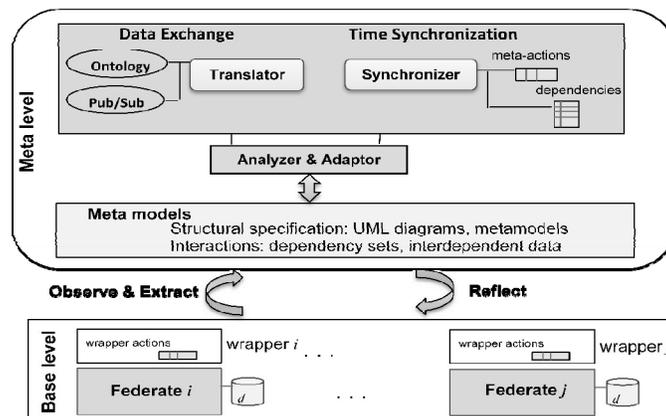


Fig. 1. Multisimulation Architecture.

when changes in these monitored aspects occur at the base simulators, the meta-level entities receive information or updates of interest via publishers. A pre-existing set of ontology models assist in the matching process for the pub-sub implementation of the simulation integration task; these include domain ontologies that are representations of knowledge in a well-circumscribed domain. Interoperability of different simulators can be achieved by sharing and understanding the metamodels. Implementing the semantic constraints for simulation integration is a human in the loop process which results in the annotations that are invisible to base-level computation and are provided to the meta-level.

In contrast to prior work on simulation integration (e.g. HLA) [4], [2], [3], [5], [27] in our architecture we do not need to integrate simulations tightly into a common framework, but we make it feasible to semi-automatically compose simulation models via a looser coupling approach that avoids the need to adhere to a rigid common interface, which can hinder leveraging prior work. We explore a reflective architecture to address challenges of integrated simulation environments in which interoperability of different simulators can be ultimately achieved in a flexible and efficient manner while preserving the autonomy of the individual simulators.

3 Integration Methodology

In this section, we describe the general structure of our methodology and its relevant issues. The complex process of integration is decomposed in several phases, and for every phase several tasks are specified, with the strategies to be followed. We describe step by step, different phases, making use of an example to make it easy to understand.

Figure 2 demonstrates the basic steps of methodology for simulation integration. The first step is to extract metadata from basic simulators and to describe it using metamodels. Next is to analyze metamodels to discover inter-dependencies between simulators. The first two steps are pre-processing steps that are human-in-loop process. When federation runs, meta-level modules ensure the correctness until the end of simulation. In the following we describe each step in details.

3.1 Preprocessing Steps: Extract Simulators' Metadata and Dependencies

The first step is to extract simulator-related meta-data and describe it at meta-level using metamodels. Metamodels are abstracts of lower-level details of integration and interoperability which make the underlying simulator more understandable. Figure 3 shows our meta-model. There are several key classes in the metamodel: model type, actions, model elements (data items) which could be local data or shared data, input or output parameters, actions, and constraints. We construct our metamodel using UML (Unified Modeling Language).

Model type includes information about the type of simulation model. In general simulators can be categorized into Discrete-event, Agent-based, System dynamics. They also can be categorized based on the time management mechanism that they employ as time stepped simulators or event based simulators [1, 7]. In time stepped

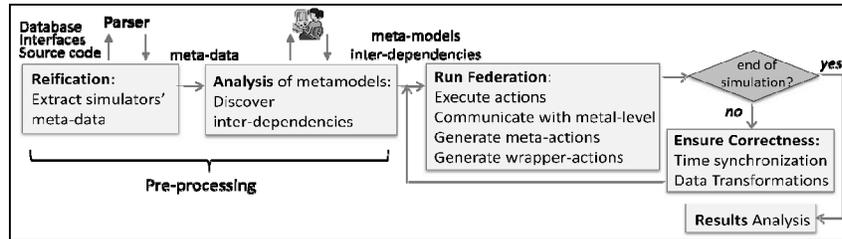


Fig. 2. The Basic Steps of Methodology.

simulators, for each execution of the main control loop the simulation time is incremented by one quantum of time Δt . In the case of event based simulators, execution is driven by an event list, each event has a time stamp (usually causality preserving) and the simulation time jumps from one event time stamp to the next without representing all the values in between. For example for Drillsim [9] we have agent-based and time-step as model type.

Model elements are the main elements of a simulation and can be captured from the interfaces, the source code, or databases. We develop a set of tools to extract the simulator information as metamodels from the base-level simulators. Model elements consist of simulation model features. Since we are interested in structural reflection, currently we only use structural features which include classes and attributes. We may also take behavioral features into account to represent operations and associations in future. We implemented a parser using a tool for large scale code repositories search to extract the entities and attributes from a complex and large simulators using the simulator's source code, interfaces, and databases. Then we group extracted information into features to capture the structure of the simulator. The features are put into the same class if they are considered equivalent.

Since our metamodel needs to take several domain expert simulators into account, the metamodel should be comprehensive, yet extensible. In our metamodel, we also consider input and output parameters. The careful examination of the features in

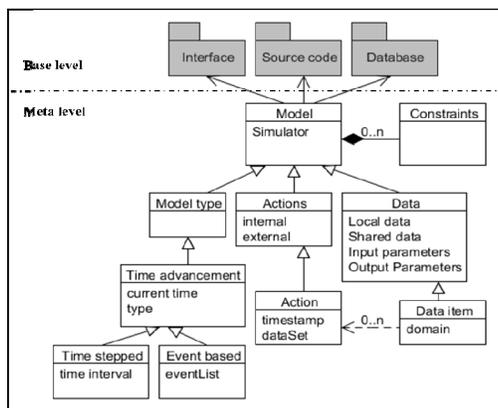


Fig. 3. Basic Metamodel.

various simulators of the different domains has allowed us to identify and categorize common features using key classes. Finally, constraints are the number of limits for the simulation parameters in the simulation model. We will discuss complete examples of metamodels in our case study later.

In the second step, we analyze the metamodels to discover the interdependencies between simulators. We use dependency descriptors to specify the dependency between a data item d in simulator S_i and a data item d' in simulator S_j when updates on d' need to be reflected into d : $S_i \leftarrow S_j = \langle d \in D_i, d' \in D_j, f \rangle$. Note that dependency notion is directional. S_i is the supplier simulator, S_i is the consumer simulator. Here, d and d' are *interdependent data items*. In general, there can be more than one dependency between two simulators describing multiple aspects of their relationships. A dependency function, f , defines the relationship between two data items values. At each iteration, the new value of d is determined by the dependency function $f: Dom(d') \rightarrow Dom(d)$, that is $v = f(v')$. For each dependency between simulators such a dependency function is defined at meta-level.

3.2 Run Federation

We consider each simulator's execution as a sequence of actions (time steps in time stepped simulators or events in event based simulators). Typically simulators execute their set of actions independently from the beginning to the end of simulation in an uncoordinated way. To participate in a federation, each of these simulators needs to be modified, i.e. the introduction of synchronization points at which the simulations needs to stop processing its actions and communicate with the meta-level in order to be synchronized with other simulators.

Simulators are interfaced with met-level by using wrappers. The wrapper determines the external actions for which the simulator needs to communicate with meta-level and sends them to meta-level. External actions are those actions that access at least one data item which is an interdependent data item (there exists a dependency between this data item and another data item in another simulator). Upon receiving such actions from a simulator, the meta-level generates meta-actions to notify its

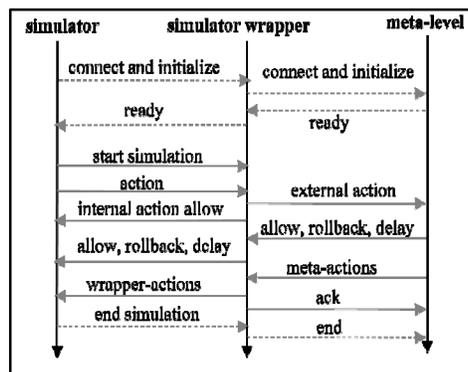


Fig. 4. Base-level federates, wrapper, and meta level interactions (step 3).

dependent simulators.

Figure 4 shows the details of Step 3 in our methodology. First, the wrapper sends a request for connection to the meta-level. The meta-level confirms the connection and sends information about interdependent data items and dependencies to the wrapper. Once simulation starts, the wrapper determines the external actions and sends corresponding requests to the meta-level. Upon receiving a request, the meta-level modules evaluate the dependencies and respond to the wrapper with a decision (allow, rollback, or delay) based on the scheduling approach used (which will be discussed in the next section). It also sends to the wrapper meta-actions that contain the updates by other simulators. The wrapper reflects the received meta-actions on the execution of the underlying simulator. This loop is continued until the end of simulation.

4 Challenges

There are several challenges in integrating multiple autonomous simulation models. The first challenge arises from modeling complex scenarios using multiple simulation models and the analysis of cause-effect relationships between those models. Given the potential black-box nature of simulators developed by experts in diverse domains we believe that achieving a completely automated plug-and-play integration of simulators is a very difficult, if not infeasible challenge. Our goals are more modest – we intend to develop enabling tools that will simplify the task of simulation integration with a wide range of simulators that vary in the degree to which they expose their interfaces and implementations. We do not want to require simulator developers to adhere to a strict programming interface or conform to particular design styles - the ability to flexibly interoperate with multiple simulators is our goal. Another challenge arises from the fact that each simulator uses its own models and entities; these must now be integrated in the context of a single simulation. The simulators need to exchange the data and have a correct interpretation of the data they send and receive. Time synchronization is yet another challenge. When integrating simulators, there is a need for synchronization of time between the different models. In the following we discuss the details:

(a) Managing Complexity of Interoperating Systems. Integration of independently created models can lead to a complex interoperating system of systems that need to be managed efficiently. Understanding the interoperability issues that arise in this context is the main aim of multisimulations. In our proposed approach, we use meta models to describe simulator-related meta-data; a way to infer data transformations, and a means of specifying and automatically executing orchestration. Metamodels can make the underlying simulator more understandable. Additionally, metamodels are abstracts of lower-level details of integration and interoperability. Other challenges arise from heterogeneity of the data that simulation models need to exchange. Since we are working with existing simulation models, it is necessary to analyze the data types used internally by the simulators. It might be possible to adapt prior work on data transformation and integration [26]. We plan to investigate whether such techniques can be extended to account for or detect potential data exchange issues that will arise.

(b) Correctness. Another challenge in integrated simulation environments is to ensure the correctness of multisimulations (e.g. preserving causality among different simulation models). In particular, we focus on time synchronization and data consistency as critical problems that must be addressed to ensure the correct interoperability of the concurrently executing simulators. The simulation clock that controls simulation time during execution of a simulation resides within each simulator itself. Time synchronization mechanisms are needed to ensure causal correctness for models that use different time advancement mechanisms. Most of available synchronization methods need the participants to agree on a common interpretation of time and a common time advancement method. Our goal is to leverage existing simulators, as is, while enabling data interchange between them and to accommodate multiple time management and advancement mechanisms implemented internally in participating simulators, preserving the autonomy of the individual simulators. We need to describe the semantics of the internal time advancement in different simulation models (e.g., whether it is a continuous-time model with observations made at regular time intervals, a discrete-time model with observations only at “ticks,” or a discrete-event model with observations only at irregularly spaced event-occurrence times). The spatial coordinate system must also be specified so that different models can be spatially aligned. We came across both issues in our exercise and resolved them with appropriate interpolations of data and transformations to overcome mismatches.

(c) Scalability. Accurate modeling and analysis of large scale and complex scenarios presents a scalability challenge. Modeling such complex scenarios places considerable stress on the system resources. Such scenarios involve a lot of entities, e.g. agents with complex behavior operating on dynamic environments. In this paper we focus on interoperability and correctness issues. Scalability is currently another ongoing topic of research on multisimulations [25]

We focus on time synchronization and data consistency as critical problems that must be addressed to ensure the correct interoperability of the concurrently executing simulation models. In the following we discuss the details of our approach for federation time synchronization and data consistency.

4.1 Time Synchronization

Time synchronization services is a research area with a very long history. In general the time synchronization mechanisms can fall into two different categories: 1) conservative, and 2) optimistic [16]. A conservative strategy ensures the legality of simulator actions by delaying the actions such that the dependencies are preserved in the concurrent execution of actions of different simulators. This approach prevents action roll-backs. A simulator can proceed if the synchronizer can guarantee that by executing its external action, no dependencies will be violated. In the optimistic strategy, we accept the fact that violations occur, but instead of trying to prevent them by delaying the actions, we simply choose to detect them after the action has executed and then resolve the violation when it does occur; by aborting the actions that caused the violation.

We categorize simulators based on the time management mechanism that they employ as being time stepped or event based (Table 1)[7]. In time stepped simulators,

Table 1. Time-stepped and Event-based Simulators.

Time-stepped simulator	Event-based simulator
<pre> while (simulation in progress) do for each tick do read data; modify data; <i>time = time + Δt</i>; end for end while </pre>	<pre> while (simulation in progress)do Event <i>e</i>= nextEvent; while(<i>e</i>!=null)do process(<i>e</i>); <i>time</i>= timestamp(<i>e</i>); <i>e</i>= nextEvent; end while end while </pre>

for each execution of the main control loop the simulation time is incremented by one quantum of time Δt . In the case of event based simulators, execution is driven by an event list, $E = \{e_m | m = 1, 2, \dots\}$, each event has a time stamp (usually causality preserving) and the simulation time jumps from one event time stamp to the next without representing all the values in between. For every two events e_a and e_b we have the following property: $timestamp(e_a) \leq timestamp(e_b)$ when $a \leq b$. We allow different simulators to have different levels of granularity in their events or timestamps.

Just as in any concurrency controller, the synchronizer can follow a conservative or optimistic strategy for scheduling actions.

Conservative approach. A conservative strategy ensures the legality of schedules by delaying the actions such that the dependencies are preserved in the concurrent execution of actions of simulators. The delay will cause the simulator to freeze until the synchronizer allows it to proceed. This approach prevents action roll-backs. A simulator can proceed if the synchronizer can guarantee that by executing its action, no dependencies will be violated; otherwise, the action will be delayed.

Optimistic approach. In some applications it is quite common to be in a situation where although simulators are working simultaneously on interdependent data, violations are infrequent and dependencies continue to be preserved. When this is the case, an optimistic strategy becomes efficient. In the optimistic approach, we accept the fact that violations occur, but instead of trying to prevent them by delaying the actions, we choose to detect them after the action has executed and resolve the violation when it does occur; by aborting the actions that caused the violation.

Above strategies may become more (or less) effective as a multisimulation progresses. The efficacy of a specific strategy at a point in time is a factor of the underlying dependencies and actions taken by the concurrently executing simulators. Initially, the cost of abort is small, so the optimistic strategy will be preferred. However, as the simulator proceeds, aborts costs become increasingly high. Therefore, the conservative strategy becomes more effective. We plan to propose a hybrid approach that combines the benefits of both the optimistic and conservative strategies by considering the underlying dependencies and the costs of delays and aborts to make an informed decision for an action.

4.2 Data Transformation

In general, the data management module provides data transfer that preserves the meaning and relationships of the data exchanged between two simulators. Since we

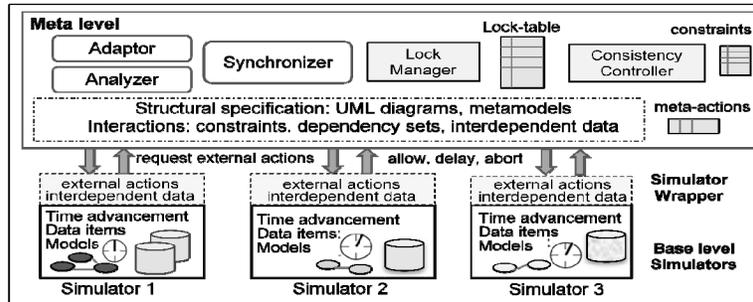


Fig. 5. Multisimulation Framework.

are working with existing simulators, we cannot use the methods based on the common representation of data. Each simulator may have its own data representation which can not be easily modified. We used data translators that work based on the dependencies between federates. If the data translators are implemented correctly, they can provide immediate conduits to publish or subscribe to information.

To integrate a simulator to the multisimulation, adaptors components need to built. The purpose of developing the adaptors is to provide a descriptor of a simulator to implement a standard interface that makes the run-time multisimulation capable of controlling the data exchange between multiple simulators. Adaptors are responsible for transforming data values of entities of one simulator to the corresponding values of entities of another simulator. For instance, a traffic simulator provides updates for other simulators on traffic congestion values for the links in the network geography. If other simulators use a different geography, a conversion must take place to map a value on the first geography to one or more values on other geographies.

We plan to adapt prior work on data exchange [26], which infers a default transformation mapping from a source schema to a target schema. Our goal is to investigate whether such mapping-generation algorithms can be extended to account for or detect potential time-management, geometry-management, and unit-conversion issues that may arise. In any case, there are standard transformations that will be required for the majority of model mash-ups. These include unit conversions, time and space interpolations and aggregations, and database join-operations on files. Future work includes identifying the set of such functions and establishing a standard library. Such transformation functions will need to be highly scalable. Other non-standard transformations can be quite challenging (e.g., aligning or combining different social networks in multiple agent-based simulation models, or allocating household caloric intake among household members).

5 Prototype System Implementation

In this section we discuss the general structure of our methodology, its relevant issues, and the implementation of a prototype multisimulation system. Figure 5 shows different modules in the prototype system. Consistent with our

metaarchitecture design philosophy, the design aims to separate the base level aspects of each simulator (this includes the simulator code, the backend databases and models stored in domain-specific formats) from the meta-level synchronization and adaptation mechanisms. Base-meta interactions occur through simulator wrappers that handle the processing of external actions in each simulator by forwarding requests to meta-level. There are 3 key modules at meta-level: (a) a *Synchronizer* which uses the proposed approaches to monitor and control concurrent execution in the multisimulation. This module also makes use of a lock manager to coordinate concurrent access to simulators data items. (b) an *Analyzer* which analyzes the interactions between simulators using meta-models to capture the dependencies which stored in a separate table and indexed by its corresponding interdependent data items. (c) an *Adaptor* which manages the data exchange and adapts information that is passed between simulators through the design of wrapper modules for each simulator.

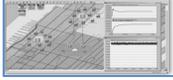
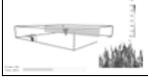
In the initialization steps the wrapper sends a request for connection to the synchronizer. The synchronizer confirms the connection and sends information about interdependent data items and dependencies to the wrapper. Once simulation starts, the wrapper determines which actions are external actions and sends corresponding requests to the synchronizer. Upon receiving a request, the meta-level modules evaluate the dependencies and respond to the wrapper with a decision (allow, rollback, or delay) based on the scheduling approach used. It also sends to the wrapper meta-actions that contain the updates by other simulators. The wrapper reflects the received meta-actions on the execution of the underlying simulator. This loop is continued until the end of simulation.

The implementation of allow and delay in the wrapper is straightforward; it will proceed or freeze the simulation respectively. In the case of rollback, associated with the rollback notification from synchronizer is a time, t , which indicates the time in the past to which simulation needs to be rolled back. One option is to start the simulation from time t , initialize all the interdependent and local data items values to the values that they had in time t , and run the simulation – obviously this involves a high overhead for storing/checkpointing the data item values at each instance of time, especially when working with pre-existing simulators. In the case of simulators when it is not possible to start a simulation from a random time in the past, we will be required to rerun the simulation from its start time until it reaches time t .

5.1 Integrating Real-World Simulators

To ground our work in reality, we develop a case study for simulation integration using three pre-existing real world simulators from the emergency response domain – the primary goal is to validate our approach and synchronization solutions and understand issues in its realization. The specific simulators are (1) CFAST, a fire simulator that simulates the effects of fire and smoke inside a building and (2) Drillsim, an activity simulator that model a response activity evacuation and (3) LTESim, a communication simulator for the next generation wireless network infrastructure. Table 1 summarizes the three simulators and their properties. In our case study, we focus primarily on integrating simulation and models aimed at

Table 2. Three Real-World Simulators.

Evacuation Simulator	Communication Simulator	Fire Simulator
 <ul style="list-style-type: none"> ◆ DrillSim [9], Time stepped ◆ Open source (in Java) ◆ Parameters: health profile, visual distance, speed of walking, num. of ongoing call, etc. Output: num. of evacuees, injuries, etc. 	 <ul style="list-style-type: none"> ◆ LTESim [20], Event based ◆ Open source (in Matlab) ◆ Parameters: num. of transmit and receive antennas, uplink delay, network layout, channel model, bandwidth, frequency, etc. Output: pathloss, throughput, etc. 	 <ul style="list-style-type: none"> ◆ CFAST [10], Time stepped ◆ Black-box (no access to source) ◆ Parameters: bldg geometry, materials of construction, fire properties, etc. Output: temperatures, pressure, gas concentrations: CO2, etc.

informing emergency response policy decision making, but we expect our framework and methods will be applicable to other complex problem domains.

1) Fire simulator: CFAST, the Consolidated Model of Fire and Smoke Transport, is a simulator that simulates the impact of fires and smoke in a specific building environment and calculates the evolving distribution of smoke, fire gases, and temperature [10]. CFAST has several interfaces to input the parameters that contain information about the building geometry, fire properties, and etc. The simulator produces outputs that contain information about temperatures, ignition times, gas concentrations, and etc.

2) Activity simulator: Drillsim is a multi-agent that plays out the activities of a crisis response process, e.g. building evacuation in response to an evolving fire hazard. Drillsim simulates human behavior in a crisis at fine granularities [9] - agents represent an evacuee, a building captain, etc. Every agent has a set of properties associated with it, such as physical perceptual profile (e.g., range of sight, speed of walking) and the current health status of the agent (e.g. injured, unconscious).

3) Communication simulator: LTESim, the communication simulator in our case study, is a LTE System Level simulator [20] which abstracts the physical layer and performs network level simulations of 3GPP Long Term Evolution with lower complexity. We chose LTESim because the LTE standard has several improvements in capacity, speed, and latency and will be the technology of choice for most existing 3GPP mobile operators [8]. LTESim considers several parameters to model the communication infrastructure (such as number of transmit and receive antennas, network layout, bandwidth, pathloss, and etc.).

In our integration scenario, the fire simulator, CFAST, is used to simulate the impact of fire and smoke in a specific region and calculates the evolving distribution of smoke; fire and smoke can affect evacuation process, e.g. people's health condition, in the evacuation simulator, Drillsim, which has impacts on communication patterns in communication simulator, LTESim. Such integration is useful to conduct better what-if analyses and understand various factors that can adversely delay evacuation times or increase exposure and consequently used to make decisions that can improve safety and emergency response times. The first step is to specify meta-models for the three base level simulators and dependencies across them (see Appendix). The following are the examples of information interchanged among simulators:

- A harmful condition in CFAST can affect an individual's health in Drillsim.

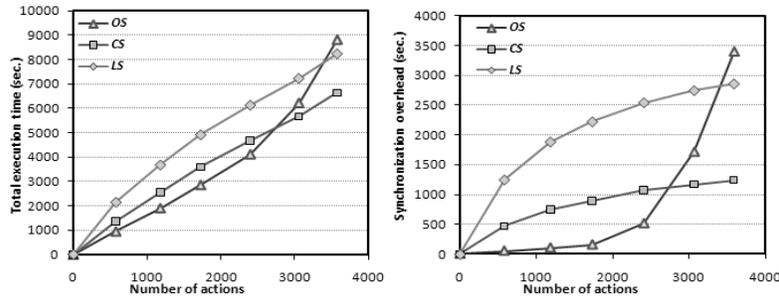


Fig. 6. (a) Average synchronization overhead (b) Total execution time vs. the number of actions.

- Smoke in CFAST can decrease an agent’s visual distance in Drillsim.
- The number of ongoing communications in Drillsim can affect network pathloss and throughput in LTESim.
- Pathloss in LTESim can be used to determine connectivity/coverage in Drillsim.

In our current implementation, several such dependencies specified (the actual number of dependencies required was in the range of 10-50 for most situations).

5.2 Initial Results

Our experiments are based on the case study described above where we integrate 3 real world simulators. In our experiments, we implemented techniques for synchronization across the three simulators: We implemented three different solutions for time synchronization across the three simulators: Lock-step approach (*LS*), conservative approach (*CS*), and optimistic approach (*OS*). Lock-step approach is the most conservative approach for the purpose of evaluating the other two approaches, by having a lock step schedule. All simulators advance step by step, and at any step they synchronize by locking at data item level until the next step. The lock table is maintained at meta-level. By locking shared data in the beginning of action and releasing the locks at the end, we can prevent deadlocks. In *OS*, we consider wait time before each rollback to be 0.5 s.

We studied the synchronization overhead and the total execution time using

Table 3. Comparison between HLA and Multisimulation Architecture.

Criterion	HLA	Multisimulation Architecture
Objective	– Interoperability – Reusability	– Semantic Interoperability, Reusability – Flexibility
Domain	– Defense	– Flexible via use of domain ontologies
Complexity	– Low level knowledge needed – Lack of semantic interoperability	– No need to conform the internal properties – Semantic constraints implemented at the metalevel
Time Management	– Optimistic and conservative methods	– Optimistic and conservative methods
Separation of Concerns	– Merges domain-specific and integrated simulation aspects	– Separate concerns related to simulation domain to those related to integration mechanisms

different techniques. We measure synchronization overhead by adding the synchronization overhead in all simulators. In *CS*, we considered the total delay time, i.e. the duration a simulator is blocked and the locking overhead, i.e. the time needed for acquiring or releasing locks, to calculate the synchronization overhead. In *OS*, we considered the total rollback time. Figure 6-a and 6-b illustrate the average synchronization overhead per time step and average execution time for different numbers of actions. In our base case, the number of dependencies between simulators is 30 (a reasonably large number for our case study). The synchronization overhead in *CS* is much lower as compared to *OS* during later phases of execution. This is due to the high rollback time in *OS*.

Table 3 presents a brief comparison of the reflective architecture to HLA. Using HLA outside the defense domain such as our case study is very complex, if not impossible. In HLA low level knowledge needed from participants. Each simulator must use the common data format that leads to simulations that are very closely coupled to an underlying database. Since the HLA environment is a fully distributed simulation environment, the simulators must fully conform to the designated features of the HLA standard. Note that transforming existing simulators to conform to the standard may not always be feasible. In our reflective architecture each simulator can have its own data representation, internal time management, and data management. Therefore, we do not force the simulators to change their internal properties. Another advantage of our reflective architecture is separation of concerns, that is, separate the concerns related to the simulation domain from those related to the integration mechanisms. Additionally it provides a design that is more adaptable, flexible and easier to extend.

6 Related Work and Conclusions

To best of our knowledge, simulation integration has been studied in two domains – (a) military command-and-control [4], [2], [5], [27], and (b) games and virtual environments [3]. The U.S. Department of Defense (DoD) has promoted the development of standards to provide a common framework in which simulators can be integrated. These include standards such as SIMulator NETworking (SIMNET) [27], Distributed Interactive Simulation (DIS) [5], Aggregate Level Simulation Protocol (ALSP) [2], High Level Architecture (HLA) [4]. These standards provide specific services for interoperability in niche applications, for example DIS for human-in-the-loop simulators or ALSP for war games. The recent HLA effort has become the defacto standard technical architecture for military simulations – it aims to promote interoperability and reusability between simulators.

While HLA is suited to developing new simulators that can be easily integrated, its broader applicability to combine pre-existing simulators is questionable [6], [21]. It is a complex standard designed specifically for the military domain and is not transparent enough – too much low level knowledge is needed from the practitioner. HLA forces developers to provide a particular functionality or to conform to specific standards in order to participate in the integration process; the rigid assumptions and limitations on participants makes it difficult to integrate pre-existing simulators without significant modification (especially in non-military domains). As in the case

with the HLA architecture, solutions in the game community are also prescriptive - they force the developers to provide a particular functionality to participate in the integration process and have different assumptions/limitations on how participants interact. Such methods are unsuited to the integration of pre-existing simulators.

In this paper we proposed a reflective middleware architecture for simulation integration that implements structural reflection to alleviate the flexibility issues in current simulation integration techniques. In this architecture, the meta-level is structured as a series of metamodels representing the various simulators. We have implemented a detailed case study from the emergency response domain by integrating 3 disparate simulators: a fire simulator (CFAST), an evacuation simulator (Drillsim) and a communication simulator (LTESim). Future research will focus on addressing challenges in the complexity associated with generalizing the meta-models for simulators, integrating simulators in other domains including earthquake and transportation simulators, and addressing the challenges of data transformation in multisimulations.

References

- [1] Kon, F., Costa, F., Blair, G., Campbell, R.H.: The Case for Reflective Middleware, *Communications of the ACM*, 45(6), 33–38, (2002)
- [2] Weatherly, R., Seidel, D., Weissman, J.: Aggregate Level Simulation Protocol, Summer Computer Simulation Conference, 953-958 (1991)
- [3] Jain, S., McLean, C.R.: Integrated simulation and gaming architecture for incident management training, *Simulation, Proc. of the Winter Simulation*, 904-913 (2005)
- [4] Kuhl, F., Weatherly, R., Dahmann, J.: *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, New Jersey, Prentice Hall (1999)
- [5] Davis, P.K: Distributed Interactive Simulation (DIS) in the Evolution of DoD Warfare Modeling and Simulation, *Proceedings of the IEEE* 83(8), 1138-1155, (1995)
- [6] Boer, C., Bruin, A., Vebraeck, A.: Distributed simulation in industry- a survey: part 2 -- experts on distributed simulation. *Winter Simulation Conference*, 1061-1068 (2006)
- [7] Fujimoto, M.R.: *Parallel and Distributed Simulation Systems*, John Wiley Inc. (2000)
- [8] McQueen, D.: 3GPP LTE: the momentum behind LTE adoption, *IEEE communication*, Vol. 47, 44-45 (2009)
- [9] Balasubramanian, V., Massaguer, D., Mehrotra, S., Venkatasubramanian, N.: DrillSim: A Simulation Framework for Emergency Response Drills, *ISI*, 237-248, (2006)
- [10] Peacock, R., Jones, W., Reneke, P., Forney, G: CFAST– Consolidated Model of Fire Growth and Smoke Transport (Version 6) User’s Guide, NIST Special Publication (2005)
- [11] De Silva, F.N., Eglese, R.W.: Integrating Simulation Modeling and GIS: Spatial Decision Support Systems for Evacuation Planning, *JORS* 51(4), 423–430 (2000)
- [12] Abanades, A., Sordo, F., Lafuente, A.: Martinez-Val, J.M., Munoz, J.: Application of computational fluid dynamics (CFD) codes as design tools, *5th Int. Conf. on ISFA* (2007)
- [13] HAZUS-MH: Multi-hazard Loss Estimation Methodology. User Manual (2003)
- [14] Cho, S., Huyck, C.K., Ghosh, S. Eguchi, R.T.: Development of a Web-based Transportation Modeling Platform for Emergency Response. *8th Conf. on Earthquake Eng.* (2006)
- [15] CAPARS: <http://www.alphatrac.com/PlumeModelingSystem>
- [16] Jefferson, D.: Virtual Time, *ACM Trans. Programming Lang. Sys.*, No. 3, 404-425, (1985)
- [17] Verkehr, A.: VISIM V3.6 Innovative Transportation (2001)

- [18] Cameron, G., Wylie, B., McArthur, D.: PARAMICS- Moving Vehicles on the Connection Machine, Conf. on High Performance Networking and Computer, 291– 300 (1994)
- [19] Cooper, L.Y, Forney, G. P.: The consolidated compartment fire model (CCFM) computer code application CCFM.VENTS - Part I: Physical basis. NISTIR 4342. (1990)
- [20] LTE System Level Simulator: <https://www.nt.tuwien.ac.at/>
- [21] Boer, C., Bruin, A., Verbraeck, A.: Distributed Simulation in Industry - a survey, part 3- the HLA standard in industry, Proc. of the 40th Conf. on Winter Sim, 1094-1102 (2008)
- [22] Huang, J., Tung, M., Hui, L.: Ming-Che Lee An Approach for the Unified Time Management Mechanism for HLA Source Simulation, Vo. 81 , Issue 1, 45-56 (2005)
- [23] Ramamritham, K., Calton, P.: A Formal Characterization of Epsilon Serializability, IEEE Transactions (1995)
- [24] Jalali, L., Venkatasubramanian, N., Mehrotra, S.: Reflective Middleware Architecture for Simulation Integration, ARM'09, Urbana Champaign, Illinois (2009)
- [25] Balasubramanian, V., Kalashnikov, D., Mehrotra, S., and Venkatasubramanian, N.: Efficient and scalable multi-geography route planning, EDBT'10, Switzerland. (2010)
- [26] Haas, L. M., Hernández, M.A., Ho, H., Popa, L., Roth, M.: Clio Grows Up: From Research Prototype to Industrial Tool, Proc. ACM SIGMOD, 805-810 (2005)
- [27] Pope, A.: The SIMNET Network and Protocols, Technical Report 7102, MA: BBN (1989)