Interoperability of Multiple Autonomous Simulators in Integrated Simulation Environments

Leila Jalali,
Sharad Mehrotra
Nalini Venkatasubramanian
Department of Computer Science
University of California, Irvine
949-351-8778, 949-865-5898, 949-824-5975
ljalali@ics.uci.edu, nalini@ ics.uci.edu, sharad@ ics.uci.edu

Keywords:
Simulation Integration, Interoperability, Data Consistency, Synchronization, Metamodels.

ABSTRACT: *In this paper, we propose a methodology for integrating multiple autonomous pre-existing simulation models into an integrated simulation environment. 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 modelled 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 simulation 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. Unlike the significant code rewrite required in the HLA case, our 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. In our methodology, integration of different simulators is ultimately achieved by using the meta-models for specifying the properties of the different simulators and reasoning about the interactions among them. To ensure the correct interoperability of the concurrently executing simulation models, time synchronization and data consistency are critical problems that must be addressed. Using concepts from multidatabases systems and transaction processing we model the integrated simulation execution as sequences of actions and capture dependencies across them; we express synchronization as a scheduling problem where the goal is to generate schedules that meets the dependencies without loss of concurrency.
We evaluate our proposed methodology and techniques 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).

1. Introduction

Simulation have been used widely to study a variety of real world problems which are too costly or impractical to do in real life. Simulation is cheaper, quicker, and enables what-if analyses for better system design [4, 7]. In this paper we aim to describe a methodology for simulation integration that supports the interoperability of multiple existing simulation models. The complex process of integration is decomposed in several phases, and for every phase several tasks are specified, with the strategies to be followed. The methodology has been designed in order to maintain simulators' autonomy (i.e. retain their internal representations of time/state etc.), thereby avoiding the need for rigid common interfaces across simulators.

Building complex simulations to understand the joint effect of multiple phenomena (spread of hazardous material as a result of an earthquake, impact of obesity on health issues in a society) is very useful. For example, 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 have been developed in the domain of emergency response, e.g., loss estimation tools (HAZUS [13], INLET [14], CAPARS [15], CFD [12]), fire simulators (CFAST [10], CCFM [19]), evacuation simulators (Drillsim [9], SDSS [25]), transportation simulators (VISISM [17], PARAMICS [18]) etc. Since these simulators are typically developed by domain experts who have an in-depth understanding of the phenomena being modelled and their evolution (e.g. hazardous materials spread via plumes), they are designed to be executed and

evaluated independently. Consider 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 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). An integrated and concurrent execution of the two simulators is essential to understand the adverse impacts caused such as increase in evacuation times or increased exposure to undesired particulates. Such what-if analyses can enable intelligent decision making to improve the outcome of the response.

Integrated simulation environments are also useful in the healthcare domain where understanding complex health issues requires combining multiple simulation models including agriculture, transportation, distribution, climate, and social, economic, and physical systems in which people and food operate. It has been argued that combined simulation results in a more accurate representation of population health; this, in turn, can help devise meaningful health care policies for the future [20, 27]. Similarly, in the hydro-electric power generation domain, we need multiple simulators that model meteorology (weather), hydrology (water cycle) and power generation in order to get an estimate of how much hydro-power can be generated by the power plant. A key limitation of simulation models in various domains is that they are typically designed to be executed and evaluated independently. We aim to integrate different expert simulation models that each capture relevant aspects of the real world. This will allow decision makers to explore alternatives and conduct low cost experiments in an integrated, interactive environment.

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 [1] 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 aim to build integrated simulation environments by creating a loosely coupled federation of pre-existing simulators. Unlike the significant code rewrite required in the HLA case, our 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. In order to participate in the federation, wrappers are written for each simulator that enable us to intercept and control the execution of individual simulators in order to ensure effective and correct composite simulation models. Unlike the case with HLA, the creation of such wrappers is a comparatively minor task as our experience working with multiple simulators (including those developed independently by different teams) has indicated.

This paper proceeds as follows. In Section 2, we discuss some background efforts in the area of simulation integration, our reflective architecture, RAISE, and the main challenges in our research. 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. Background on Simulation Integration

Simulation integration has been studied in two domains – (a) military command-and-control and (b) games and virtual environments. DoD has promoted the development of distributed simulation standards to provide a common framework in which simulators can be integrated. These include standards such as SIMulator NETworking (SIMNET) [32], Distributed Interactive Simulation (DIS) [11], Aggregate Level Simulation Protocol (ALSP) [26], High Level Architecture (HLA) [1]. 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 in the United States – it aims to promote interoperability, scalability, and reusability between simulators. HLA 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. Additionally it requires the participants to agree on a common interpretation of data that is produced and exchanged between them.

Recently simulation integration methods have been used in the game community [5, 29, 30], primarily to support interoperability. As in the case with the HLA architecture, solutions here are prescriptive - they force the developers to provide a particular functionality or to conform to specific standards 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 (third-party) simulators since they often require modification of the original source code.

1.1. RAISE: Reflective Architecture for Integrated Simulation Environments

In our prior work [31] we used a reflective middleware architecture to provide a principled, yet flexible approach to support the development of integrated simulation platforms. Figure 1 illustrates RAISE, our two-level reflective architecture for integrated simulation environments. In RAISE, 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 – i.e. what we intend to design and develop is a *meta-simulation*.

The base level consists of various pre-existing simulators that need to be integrated; each has its own databases, models, source code, and etc. Meta-level captures relevant aspects of each simulator in a meta-model as well as supports for data adaptations and exchange across simulators. By using the metamodeling capability the model elements that need to be integrated can be extracted. 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) – 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.

We aim to build metasimulations by creating a loosely coupled federation of pre-existing simulators. In order to participate in the federation, wrappers are written for each simulator that enable us to intercept and control the execution of individual simulators in order to ensure effective and correct composite simulation

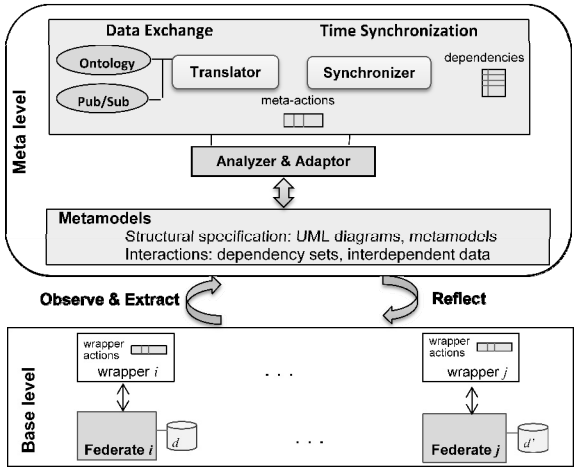


Figure 1. RAISE: Reflective Architecture for Integrated Simulation Environments.

models. The wrapper determines the external actions for which the simulator needs to communicate with meta-level and sends them to meta-level. Upon receiving such actions from a simulator, the meta-level generates meta-actions to notify its dependent simulators. Unlike the case with HLA, the creation of such wrappers is a comparatively minor task as our experience working with multiple simulators (including those developed independently by different teams) has indicated.

1.2. Using Concepts from Multidatabase Systems in RAISE

When we examine the nature of our problem and challenges, we note properties of simulation integration which parallel certain concepts in multidatabase systems and transaction processing. In multidatabase systems the individual database management systems need to be integrated in a single unified database environment while they desire to preserve the autonomy of the local database management systems [2]. Interesting parallels also exist between the concepts of simulation execution synchronization and transaction serializability [6]; and simulation time and data currency [3].

Using concepts from serializability theory, we model metasimulation as sequences of actions (time steps in time stepped simulators or events in event based simulators) and capture dependencies across them. Typically simulators execute their set of actions independently from the beginning to the end of simulation in an uncoordinated way. To participate in a metasimulation, each of these simulators needs to be modified, i.e. the introduction of integration points at which the simulations needs to stop processing its

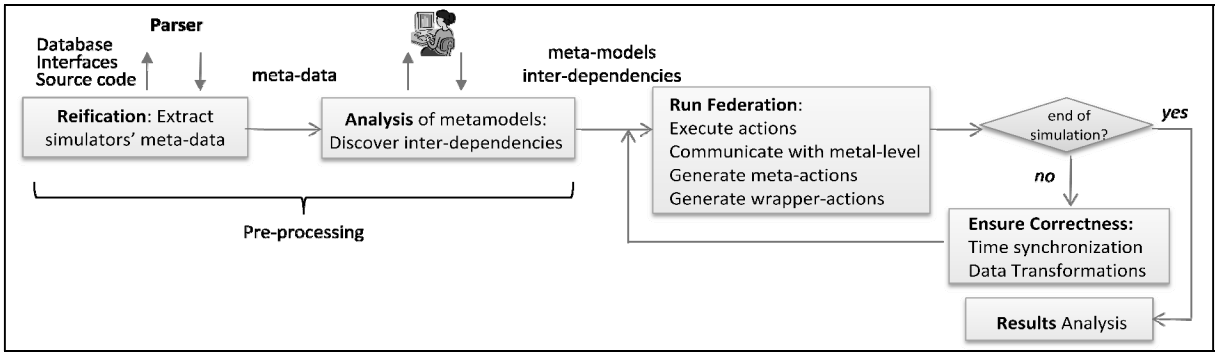


Figure 2. The Basic Steps of Methodology.

actions and communicate with the meta-level in order to be synchronized and integrated with other simulators. Simulators are interfaced with meta-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. Upon receiving such actions from a simulator, the meta-level generates *meta-actions* to notify its dependent simulators. Additionally, in the scheduling mechanisms the meta-meta-level synchronization methods uses the time stamps associated with meta-actions to preserve causality in the concurrent execution of multiple simulators.

Our goal is to come up with sequence of actions from different simulators that can be executed with minimum synchronization overhead such that dependencies are preserved. Like in multidatabases, there are several challenges in enabling seamless federation of simulation platforms. In particular, time synchronization and data consistency are critical problems that must be addressed in our framework to ensure the correct interoperability.

3. 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. We assume that the inputs to the integration process is a set of simulation models (federates) $\{S_1, S_2, \ldots, S_n\}$, each federate is a three-tuple $S_i =< T_i, A_i, D_i >$ for which T_i is the type of the simulator. In this paper we consider two types: time stepped simulators or event based simulators; $T_i \in \{TS, EB\}$, where *TS* and *EB* correspond to time stepped and event based simulators respectively. A_i is the set of actions that is $A_i = \{a_k |\ k = 1, \ldots, n\}$. An action a_k captures changes that occur in a clock tick (or a step) in a time stepped simulator or the execution of an event in an event based simulator. Finally, D_i is the set of data items that the simulator reads or updates (state variables).

The state, $\{\Phi_i\}_{ts}$ of the simulator S_i is the snapshot of its data items D_i and their values. A state maps every data item $d \in D_i$ to a value v, where $v \in Dom(d)$. Thus a state can be expressed as a set of ordered pairs of data items in D_i and their values, $\{\Phi_i\}_{ts} = \{(d, v) |\ d \in D_i,\ v \in Dom(d)\}$. Associated with each state is a timestamp, *ts*. Actions trigger state change; we use the notation $\{\Phi_i\}_{ts}\ a_k\ \{\Phi_i\}_{ts'}$ to indicate that when an action a_k in simulator S_i executes from a state $\{\Phi_i\}_{ts}$, it results in a state$\{\Phi_i\}_{ts'}$. The timestamp of the state is changed after the execution of action. In a time stepped simulator it advances by Δt, $\{\Phi_i\}_{ts}\ a_k\ \{\Phi_i\}_{ts+\Delta t}$. In an event based simulator it advances to the time stamp of the event e_m that is executed, $\{\Phi_i\}_{ts}\ a_k\ \{\Phi_i\}_{ts+e_m(ts)}$, where $e_m(ts)$ is the timestamp advanced by e_m.

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. Step 1: Extract and Describe Simulators' Metadata

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 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. In RAISE 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 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.

3.2. Step 2: Analysis of metamodels and discover inter-dependencies

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: $S_i \leftarrow S_j =< d \in D_i, d' \in D_j, f, R >$.

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. Each data item has a value at any given state, $(d, v) \in \{\Phi_i\}_{ts}$ and $(d', v') \in \{\Phi_j\}_{ts'}$. 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.

R specifies the *relaxations* for this dependency which describes the amount of deviation allowed from the ideal behavior. Ideally, dependencies need to be reflected from one simulator into another as soon as update in one simulator becomes valid in another. However, in most of applications, ideal behavior results in unnecessary synchronization overhead and loss of concurrency among simulators in the integrated simulation. Therefore we relax the dependencies (using *R*) that capture the extent to which simulators can deviate from ideal behavior. Consider a dependency between two data items in two simulators $S_i \leftarrow S_j$. Each data item has a value at any given state, $(d, v) \in \{\Phi_i\}_{ts}$ and $(d', v') \in \{\Phi_j\}_{ts'}$. Now we consider three types of deviations below:

- **Time** *(t-bound)*: *t-bound* works as the delay condition which states how much time the

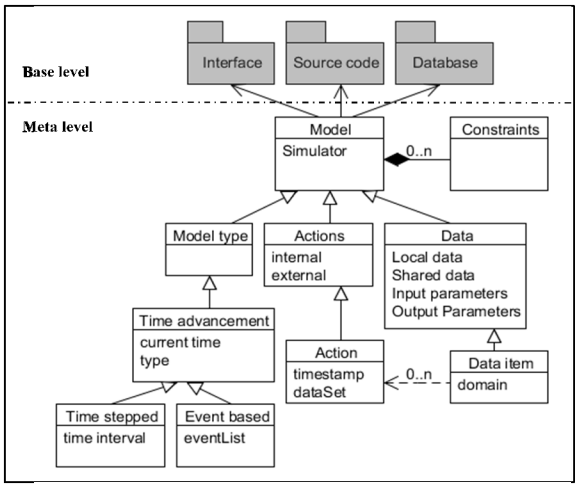


Figure 3. Basic Metamodel.

consumer can use a value behind the new update of the supplier. At any time, if $< ma(d, d'), t_m >$ is the meta-action posted as a result of an update by S_j and ts is the current time in S_i then: $ts - t_m \leq$ *t-bound*.

- **Value** *(v-distance)*: Let v be the value of d updated by S_i and v' be the value of d' updated by S_j, we consider the difference between the values of two data item using a user defined distance function $distance(v, v')$ [23]. *v-distance* is the maximum amount of distance permitted between the values at any time: $distance(v, v') \leq$ *v-distance*.
- **Number of changes** *(n-update)*: captures the maximum number of updates on d' before they become reflected on d. In other words, at any time, the consumer, S_i, can skip at most *n-update* of d' that have been performed by the supplier, S_j.

We use a logical expression to represent the relaxations as conjunction and/or disjunction of the above three deviation types: *t-bound*, *v-distance*, *n-updates*: $R = ($*t-bound=t* $[\wedge \mid \vee]$ *n-update=n* $[\wedge \mid \vee]$ *v-distance=v*$)$, T_p , $n \epsilon \mathbb{N}$, and $v \epsilon Dom(d)$. As an example, consider we want to express the relationship between d, the person's health condition, in an evacuation simulator S_1 [9] and d', the smoke level, in a fire simulator S_2 [10]: $S_1 \leftarrow S_2 =< d \in D_1, d' \in D_2, f, R >$, where $v = f(v') = \frac{1}{v'}$ and $R =$ (*t-bound=5* $\wedge$ *n-updates=2*). f is the dependency function that implies that if the smoke level increases, then the health condition will be decreased. Relaxations, *R*, indicates that we assume the smoke level will not affect the health condition immediately, but instead, in less than 5 *s*. Therefore, we express this relationship using *t-bound=5* which implies that the smoke level from the fire simulator must be reflected to health conditions in an evacuation simulator with at most a 5*s* delay to be consistent. Additionally *n-update=2* indicates that there should be at most two updates on smoke levels in S_2 before it is reflected into S_1.

3.3. Step 3: 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 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.

Let $S_i \leftarrow S_j =< d \in D_i, d' \in D_j, f, R >$ be a dependency, S_j in state $\{\Phi_j\}_{ts}$ performs an action a_k which changes its state and updates d' to its new value v' such that $(d', v') \in \{\Phi_j\}_{ts'}$ ($\{\Phi_j\}_{ts}$ a_k $\{\Phi_j\}_{ts'}$). S_j posts the external action and the current state's timestamp, $< a_k, ts' >$, to the meta-level. The meta-

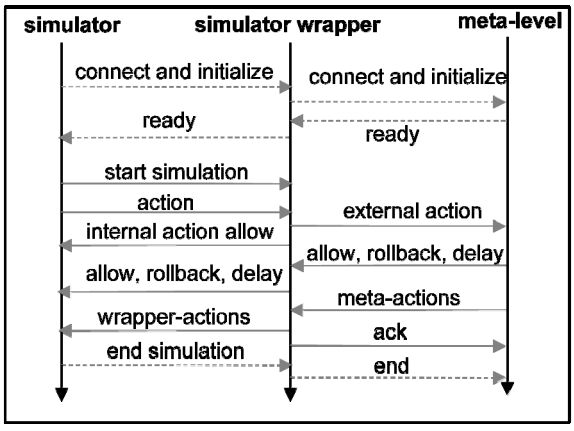


Figure 4. Base-level federates, wrapper, and meta level interactions (step 3).

level generates a meta-action $<ma(d,d'),ts'>$ and posts it to a meta-action queue. Note that associated with this meta-action is the timestamp ts', which is the timestamp of the state of S_j which contains (d',v'). In its next iteration, S_i's wrapper checks with the meta-level for corresponding meta-actions. Let the current time of S_i be t. The meta-level synchronizer will determine all meta-actions that need to be reflected to S_i in its current state $\{\Phi_i\}_{ts}$ that is, $d \in D_i$ and $ts' \leq t$. Note that actions that correspond to the future state, i.e., $t < ts'$, will not be reflected into S_i immediately and will be deferred until the simulator S_i advances to the time when the action is applicable. Once S_i's wrapper receives the meta-actions that need to be reflected to S_i, it generates wrapper-actions, $wa(d,d')$, to actually reflect the updates into the simulator.

3.4. Step 4: Ensure Correctness

Time synchronization and data consistency are critical problems that must be addressed to ensure the correct interoperability of the concurrently executing simulation models. If there is a dependency between S_i and S_j (in which S_j is a supplier and S_i is a consumer), every time S_j performs an actions that updates the interdependent data (external action), it posts the action to meta-level. Then meta-level notify S_i about the external action. Our goal is to generate a sequence of actions that preserve the dependencies between simulators (e.g. causality and correct data exchange). In the following we discuss the details of our approach for federation time synchronization and data consistency.

3.4.1. 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, 28]. 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 meta-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.

To ensure time synchronization, we use a scheduling approach that combines the benefits of both the optimistic and conservative strategies. Our approach takes account of the fact that scheduling strategies may become more (or less) effective as a federation 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. Given an action a_k by a simulator S_i, the meta-synchronizer can either delay the action or allow its progress (even though such a progress may result in an abort). In the conservative strategy, there might be some delayed actions, which if allowed to proceed, may not necessarily result in aborts (these are unnecessary delays). We consider an action as safe to execute if it can be scheduled without the potential danger of abort or without being delayed. When the hybrid scheduling strategy encounters unsafe actions, the meta-synchronizer needs to make a decision on whether to abort the action or delay it. To make such a decision it estimates the probability of an abort and the probability of unnecessary delay if an action is unsafe.

For each simulator, S_i, the meta-synchronizer maintains the following statistics that will be used by the hybrid strategy for decision making: dl_{gd}= the number of delayed actions that would have resulted in an abort if not delayed, i.e. the delay resulted in a good outcome, dl_{bd}= the number of actions that were delayed but the delay was unnecessary, i.e. the delay resulted in a bad outcome (i.e. longer processing), os_{bd}= the number of actions that were optimistically scheduled that lead to abort; i.e. the optimistic policy caused a bad outcome, and os_{gd} = the number of actions that were optimistically scheduled that did not lead to abort; i.e. the optimistic policy resulted in a good outcome. Given the above statistics, the meta-synchronizer can make an informed decision about an action based on the expected cost of delay (EC_D) and the expected cost of abort (EC_A) by estimating the probabilities of the outcome if the action is unsafe. Let p refer to the probability of the delay being unnecessary if the action is unsafe, C_D be the cost of unnecessary delay, and C_A be the cost of abort. The following are the expected costs:

$$EC_D = p(unnecessary\ delay|conservatively\ scheduled)\ .\ C_D$$
$$\cong \frac{dl_{bd} + os_{gd}}{dl}\ .\ C_D = p\ .\ C_D$$
$$EC_A = p(abort|optimistically\ scheduled)\ .\ C_A$$
$$\cong \frac{os_{bd} + dl_{gd}}{os}\ .\ C_A = (1-p)\ .\ C_A$$

For a given action, if the expected cost of the delay is less than the expected cost of an abort, $EC_D < EC_A$, it will be delayed: $C_D < \frac{(1-p)}{p}\ C_A$. Otherwise, the action will be aborted. We consider C_D, the cost of unnecessary delays, as the average time for

delay which is calculated by monitoring and adding all the delay times, dt, of actions that are delayed unnecessarily ($a_i \in A_{du}$) and dividing it by number of actions that were delayed unnecessarily: $C_D = \frac{\sum_{a_i \in A_{du}} dt}{dl_2}$. The cost of abort, C_A, is the estimate of the amount of time the simulator needs to redo its work for which we decrease the time the simulator spent in delay (the blocking time), t_{dl}, from the its total execution time, t_{exe}: $C_A = t_{exe} - t_{dl}$. Note that this strategy is self adaptive. Initially, the cost of abort is small, so the strategy will prefer to be optimistic. However, as the simulator proceeds, it will become increasingly conservative.

3.4.2. 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 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. Due to the space limitation we do not discuss the details of data translators in this paper.

4. Prototype Implementation

In this section, we discuss the implementation of a prototype metasimulation system. Figure 5 shows different modules in the prototype system. 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 (which includes the meta-models, inter-dependencies, consistency management modules). 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. (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

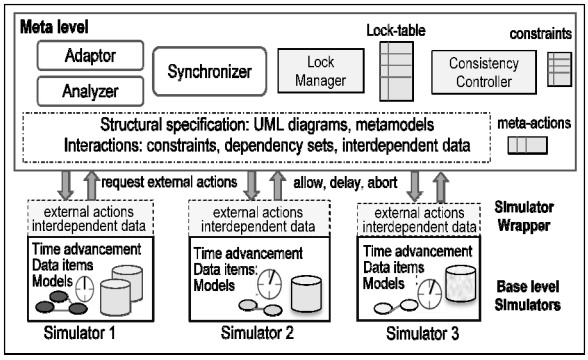


Figure 5. Metasimulation framework.

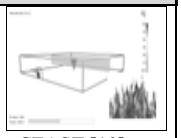
exchange and adapts information that is passed between simulators through the design of wrapper modules for each simulator.

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 metasynchronizer 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*.

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

Table 1. Three real-world simulators.

Evacuation Sim.	Communication Sim.	Fire Sim.
		
♦ DrillSim [9] ♦ Time stepped ♦ Open source, agent based (in Java) ♦ Parameters: health profile, visual distance, speed of walking, num. of ongoing call, etc. Output: num. of evacuees, injuries, etc.	♦ LTESim [31] ♦ Event based ♦ Open source (in Matlab) ♦ Parameters: num. of transmit and receive antennas, uplink delay, network layout, channel model, bandwidth, frequency, receiver noise, etc. Output: pathloss, throughput, etc.	♦ CFAST [10] ♦ Time stepped ♦ Black-box (no access to source) ♦ Parameters: building geometry, materials of construction, fire properties, etc. Output: temperatures, pressure, gas concentrations: CO2, etc.

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 [24] 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. Metamodels of three simulators are shown in Firgure 6. In the second step we analyse the metamodels to discover the dependencies. The following are the examples of dependencies among simulators:

- A harmful condition in CFAST can affect an individual's health in Drillsim.
- Agents in Drillsim can communicate information on the fire and its location – this will prevent agents from entering dangerous areas which might increase the number of ongoing calls (people talk about the crisis) in Drillsim.
- Harmful conditions in CFAST can affect the evacuation process in Drillsim e.g. increase walking speed which maps to user speed in LTEsim.
- 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.

Table 2. Dependencies between different simulators.

Dependencies $S_1 \leftarrow S_2$: $c_r(\boldsymbol{d} \in \boldsymbol{D_1}, \boldsymbol{d}' \in \boldsymbol{D_2}, \boldsymbol{f}, \boldsymbol{R})$
$c_1(health \in Drillsim, smoke \in CFAST, f_1, tbound = 5ms)$
$c_2(visualDistance \in Drillsim, smoke \in CFASTe, f_2, tbound = 7ms)$
$c_3(walkingSpeed \in Drillsim, fireLevel \in CFAST, f_3, nupdates = 1)$
$c_4(cellCoverage \in Drillsim, throughput \in LTESim, f_4, tbound = 1min)$
$c_5(numofUsers \in LTESim, ongoingCalls \in Drillsim, f_5, tbound = 1\,min \wedge nupdates = 5)$
$c_6(receiverNoise \in LTESim, ongoingCalls \in Drillsim, f_6, tbound = 2\,min \wedge nupdates = 30)$
$c_7(bldgGeo \in Drillsim, numAntenna \in LTESim, f_7, nupdates = 1)$
$c_8(userSpeed \in LTESim, walkingSpeed \in Drillsim, f_7, vdistance = 3m)$

- Pathloss in LTEsim can be used to determine connectivity/coverage in Drillsim.
- Information on building layout from CFAST and Drillsim can determine the number of transmit and receive antenna required in LTEsim.
- Number of evacuees from Drillsim determines the number of users in LTEsim.
- Number of evacuees in Drillsim can affect receiver noise in LTEsim.

In our current implementation, several such dependencies specified (the actual number of dependencies required was in the range of 10-50 for most situations). Table 2 shows some dependencies using our notation.

5. Experimental Evaluations

Our experiments are based on the case study described above where we integrate 3 real world simulators. In our first set of experiments, we implemented three different meta-scheduling techniques for synchronization across the three simulators: conservative scheduling (*CS*), optimistic scheduling (*OS*), and the hybrid scheduling (*HS*). Our goal is to evaluate our approach (*HS*). We consider wait time before each rollback to be 0.5*s*. In *HS* we considered the statistics of the most recent 50 actions in each simulator in order to calculate the expected costs of abort and delay. Every measurement in our results is an average of 5 runs.

We studied the synchronization overhead and the total execution time using 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 and in *HS*, the synchronization overhead is calculated by adding the delay times and the rollback times. Figure 7-a and 6-b illustrate the average synchronization overhead per time step and average execution time per time step during different phases of execution (i.e. for different numbers of actions). This is a more meaningful measure than total overhead/time since our experience indicates that simulation times can vary significantly based on the decision support process for which it is used. In our base case, the number of dependencies betweens simulators is 50 (a reasonably large number for our case study). The synchronization overhead in *HS* is much lower as compared to *CS* and *OS* during later phases of execution. This is due to the high blocking time in *CS* and rollback time in *OS*. In *HS*,

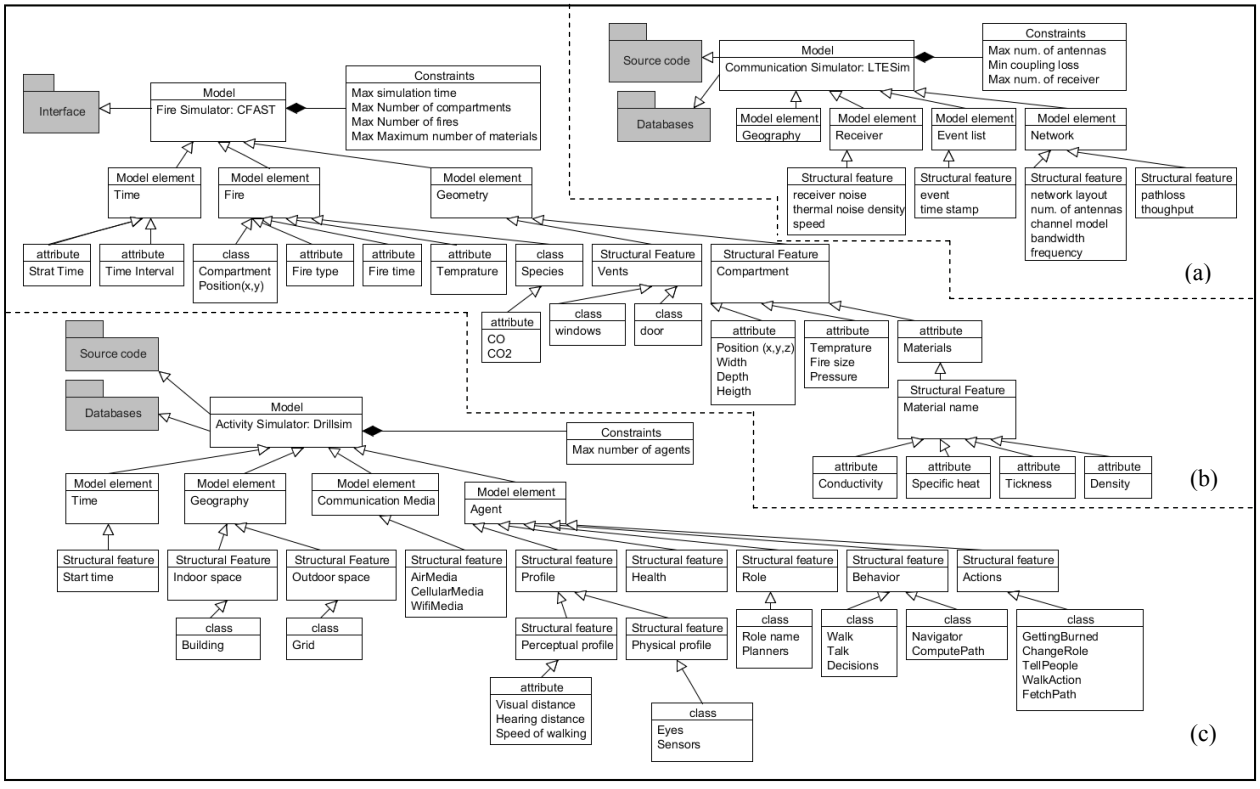


Figure 6. Metamodels of (a) LTEsim, (b) CFAST, and (c) Drillsim.

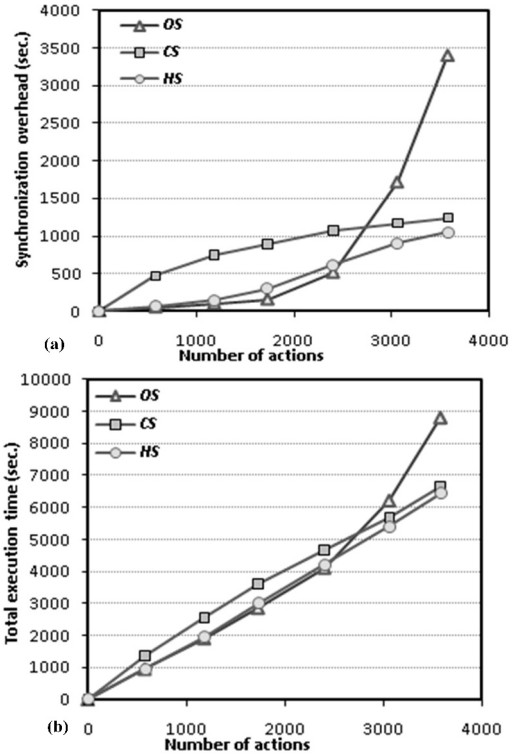


Figure 7. (a) Average synchronization overhead (b) Total execution time vs. the number of actions

initially, the cost of abort is small, so the strategy will lean towards being optimistic and is hence closer to *OS* in performance. However, as the number of actions increases, *HS* becomes increasingly conservative and approaches the behavior of *CS*.

We also applied relaxations ($R = (t\text{-}bound{=}t \ [\wedge \mid \vee] \ n\text{-}update{=}n \ [\wedge \mid \vee] \ v\text{-}distance{=}v)$ in dependencies) to our techniques. Table 3 summarizes the results of hybrid scheduling with and without using relaxations when the number of dependencies is 50. Appling relaxations results in better performance for all three simulators. The following are some key observations from our experiments:

- *HS* exhibits superior overall performance to other approaches.
- Relaxations in dependencies always help into get better results in terms of synchronization overhead and total execution time.

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. 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 framework, we focused on the synchronization problem in metasimulations to accommodate multiple time management and advancement mechanisms implemented internally in participating simulators. The runtime infrastructure of HLA, for instance supports multiple time management mechanisms. The HLA time management services are strongly related to other services such as message exchange and attribute updates. Using these services can be very complex for a developer who must understand internal design issues of the individual simulators; this gets more complex as the number of simulators increase. In our framework, 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 framework 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.

Table 3. The results for synchronization overhead and total execution time (no. of dependencies=50)

Strategy	*HS*		*HSR*	
Metric	*synch. overhead*	*total exe. time*	*synch. overhead*	*total exe. time*
CFAST	388.283	2258.475	316.007	2118.918
DrillSim	332.526	2103.698	288.555	2089.155
LTEsim	315.112	2084.187	221.079	2023.039
Total	1045.921	6446.360	816.641	6231.112

6. Conclusion

In this paper, we present a methodology for integrating multiple autonomous pre-existing simulation models. Using concepts from multidatabase systems and transactions processing we modelled metasimulations as sequences of actions with dependencies across them and expressed the synchronization problem as a scheduling problem. We also defined relaxations that capture the extent to which simulators can deviate from ideal behaviour. Our techniques are implemented in a real metasimulation framework using meta-level modules that safely enables high degrees of concurrency among simulators. Finally, we evaluate our proposed techniques via 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.

References

[1] Kuhl, F., Weatherly, R., Dahmann, J.: Creating Computer Simulation Systems: An Introduction to the High Level Architecture, Prentice Hall, 1999.

[2] Y. Breitbart, H. Garcia-Molina, A. Silberschatz, Overview of Multidatabase Transaction Management, VLDB, Vol. 1, pp. 181-240, 1992

[3] Garcia-Molina, H., Wiederhold, G.: Read- Only Transactions in a Distributed Database. ACM transactions on Database Systems, 7(2), 1982.

[4] Sowell, B., Demers, A., Gehrke, J., Gupta, N., Li, H., White, W.. From Declarative Languages to Declarative Processing in Computer Games. CIDR, 2009.

[5] Smith, R.: Synchronizing Virtual Worlds, Vol. 3 in the simulation 2000 series, 2000.

[6] Garcia-Molina, H., Ullman, J. D, Widom, J.: Database Systems: The Complete Book, Prentice Hall, 2002.

[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, Vol47, pp. 44-45, 2009.

[9] Balasubramanian, V., Massaguer, D., Mehrotra, S., Venkatasubramanian, N., DrillSim: A Simulation Framework for Emergency Response Drills, Intelligent and Security Informatics (ISI), pp. 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] 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.

[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, pp. 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, pp. 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. National Institute of Standards and Technology, 1990.

[20] Maglio, P. P., Cefkin, M., Haas, P., Selinger, P.: Social factors in creating an integrated capability for health system modeling and simulation. SBP 2010. Springer, pp. 44-51, 2010.

[21] Boer, C., Bruin, A., Vebraeck, A.: Distributed Simulation in Industry - a survey, part 3- the HLA standard in industry, Proc. of the 40th Conf. on Winter Sim,, pp. 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, pp. 45-56, 2005.

[23] Ramamritham, K., Calton, P.: A Formal Characterization of Epsilon Serializability, IEEE Transactions, 1995.

[24] LTE System Level Simulator: https://www.nt.tuwien.ac.at/

[25] De Silva, F.N., Eglese, R.W.: Integrating Simulation Modeling and GIS: Spatial Decision Support Systems for Evacuation Planning, JORS 51(4), pp. 423–430, 2000.

[26] Weatherly, R., Seidel, D., Weissman, J.: Aggregate Level Simulation Protocol, Summer Computer Simulation Conference, Baltimore, pp. 953-958, 1991.

[27] M. Cefkin, S. M. Glissmann, P. J. Haas**,** L. Jalali, P. P. Maglio, P. Selinger, W. Tan, Splash: A Progress Report on Building a Platform for a 360 Degree View of Health**,** Proceedings of the 5td INFORMS Workshop on Data Mining and Health Informatics, 2010.

[28] Carothers, C.D., R.M. Fujimoto, R.M. Weatherly and A.L. Wilson: Design and implementation of HLA time management in the RTI Version F.0. Proceedings of the 1997 Winter Simulation Conference, pp. 373–380, 1997.

[29] Ling, Y., Zhang, M., Lu, X., Wang, W., Lao, S.: Model Searching Algorithm Based on Response Order and Access Order in War-Game Simulation Grid, Springer-Verlag Berlin Heidelberg, pp. 627-637, 2006.

[30] Rhalibi, A.E., Merabti, M., Shen, Y.: Improving Game Processing in Multithreading and Multiprocessor Architecture, Springer-Verlag, pp. 669-679 (2006)

[31] Jalali, L., Venkatasubramanian, N., Mehrotra, S., Reflective Middleware Architecture for Simulation Integration, Proc. of the 8th International Workshop on Adaptive and Reflective Middleware, 2009.

Pope, A.: The SIMNET Network and Protocols, Technical Report 7102, MA: BBN Systems and Technologies, Cambridge, Massachusetts, 1989.