Society of Simulations: An Architecture for Integrating Heterogeneous Simulations

Alok Chaturvedi, Jay Gore, Sergei Filatyev
Purdue University, West Lafayette, IN 47907
{alok; gore; sfilatye}@purdue.edu

Chee Mun Foong, Brian Armstrong, Daniel Dilley, James Grahn
Simulex Inc., 3000 Kent Avenue, West Lafayette, IN 47906
{cfoong; barmstrong; ddilley; jgrahn}@simulexinc.com

Daniel Dolk
drdolk@nps.edu
Naval Postgraduate School
Monterey, CA 93943

Abstract

A society of simulations is a Dynamic Data Driven Application System (DDDAS) that facilitates modeling of complex, nonfunctional situations which occur in the real world. Precise models result from the emergent behavior of specialized simulations grouped together as individual members in a society. Each member simulation independently operates on its own understanding of reality. Members cooperate with each other to achieve the goals of the society while satisfying their own local goals. When members interact, portions of their models of reality coincide. The implementation of a shared reality enables interaction and coordination among diverse autonomous simulations.

A design and evaluation of the data exchange mechanisms needed to support the creation of a society of simulations is given. The construction of SoS is illustrated using the example of an emergency evacuation scenario. In the Emergency Evacuation Society, a simulation of fire is combined with simulations of synthetic human individuals to provide a precise model for the behavior of a small family in the event of a residential fire. The Emergency Evacuation Society reveals the strength of SoS in uniting a continuous-time, physics-based simulation with multiple event-based simulations. Comprehensive, lifelike situations can be modeled by applying the concept of SoS.

Keywords: Dynamic Data Driven Application System, system architectures, integration and modeling, parallel architectures, distributed architectures

---

1 This research was partially supported by the National Science Foundation’s DDDAS program grant # CNS-0325846 and the Indiana State 21st Century Research and Technology award #1110030618
2 Please direct all communications to Prof. Alok Chaturvedi, alok@purdue.edu
Society of Simulations: An Architecture for Integrating Heterogeneous Simulations

1 Introduction

Simulations are increasingly being used for analysis, training, and courses of action development for complex, real world problems such as emergency preparedness and response, global corporate strategy development and assessment, and military campaign planning for winning the hearts and mind of the population. Traditional requirements driven approaches for building such comprehensive simulations are inadequate. These problems require multi-disciplinary thinking; multi-scale temporal and spatial representations of forward and inverse problems; multiple analytical points of view; massive numbers of entities representing multiple sides with diverse interests; and the emergent behaviors and interactions among entities and between entities and the environment. Further, building such simulations from scratch is expensive (hundreds of millions to billions of dollars) and rarely gets completed on time and within budget [22]. As such there is renewed interest in approaches to integration and/or federation of diverse models to build simulations that match the complexities of their real world counterparts [18, 21]. Diversity of simulations implies diversity in terms of granularities (temporal, spatial, and otherwise), semantics of data and data formats, and/or points of view.

A variety of approaches have been proposed to integrate simulations. SIMNET is one of the earliest efforts to build large, distributed simulation by integrating many autonomous components [17]. SIMNET consists of a bus-like communication network upon which all messages are broadcast to all simulations. An autonomous simulation in SIMNET is not aware of which other simulations are connected to the network. Any simulator that can connect to the network can participate in the simulation. Moreover, simulators can be attached to and removed from the network dynamically during the execution of the simulation. SIMNET does not require a central controller that regulates the time management of the
simulators. Rather, each simulation synchronizes itself with every event that is published on the network. Each simulation in the network produces events that all simulations can consume.

The experiences with SIMNET led to the development of the High Level Architecture (HLA) [8, 10, 11, 12, and 13]. HLA defines an interface between a simulation and a set of management tools that enable interactions among arbitrary simulations. To connect different simulations (called federates in HLA), a Run Time Infrastructure (RTI) is used. The HLA standard specifies the interface between the RTI and constituent federates. An RTI is given only enough knowledge of the specifics of the federates in order to allow them to interact, such as the data a federate subscribes to is what a federate’s next time step will be. Federates are not aware of other federates connected in the same federation. Federates interact only through the RTI. Separating the design of federates from the implementation of the RTI facilitates reuse of federates and the RTI.

The time and data management mechanisms of the RTI have to be tuned in order to allow a federation to scale with respect to the number of federates. The time management relies on the computation of the Lowest Bound Time Stamp (LBTS), which is also used to enable parallel execution. Consequently, the computation of this value requires coordination among all federates. One approach to this limitation is to implement the LBTS computation as a repeated reduction [7, 19].

Data management is another area of significant overhead in HLA when the portion of data a federate consumes changes dynamically during the simulation. The HLA method of combining simulations relies on a push-based protocol wherein newly produced data is pushed onto the consumers resulting in production of duplicate and irrelevant messages. Duplicate messages occur when a federate is notified more than once of the same data, which occurs when the data manager decomposes the data into too fine of a granularity. Irrelevant messages result from a federate receiving an update that it does not need, which occurs when the data manager decomposes the data into too coarse of a granularity. A proposed remedy is to use a hybrid scheme which divides up the data space into a fine grid and then keeps track of the coarser portion of the grid that each federate produces and consumes [7].
The time and data management required to operate a federation limit the performance when the RTI assumes the existence of dependences among the federates at each time step, such as when the federates have no known lookahead values. To synchronize simulations while avoiding delays due to unnecessary blocking, an approach was developed to enable optimistic simulation [14]. Optimistic simulation occurs when a simulator is allowed to progress beyond a simulation time, even if it may violate a causality condition. Causality is violated when events are processed out of order.

Time Warp enables optimistic simulation by causing the simulators to roll back to a simulation time in the past when a causality violation occurs. A simulation periodically checkpoints its state and records the messages it sends in each period. When a simulator must roll back to a time, it will restore an old state prior to the roll back time and send out anti-messages (messages that cancel the effects of their counterparts) for every message it had already sent after the roll back time. Time Warp avoids much centralized time management by allowing each simulation to manage its own state and time.

The optimistic execution enabled by Time Warp comes at the expense of an overhead in space and time. Memory space is required to checkpoint and roll backs accrue execution time overhead. The space required to save each simulator's state continues to increase as more time steps are simulated.

To offset the memory use problem, a Global Virtual Time (GVT) is calculated and broadcast to each simulator. The GVT is the minimum time any future roll back can occur, similar to the LBTS calculation of HLA. It can be calculated by taking the minimum simulation time from every simulator and every message that is not yet processed.

Any checkpoint or recorded message before the GVT can be discarded or saved to disk because no roll backs will occur with a simulation time less than the GVT. In order to calculate an accurate GVT, a barrier synchronization is required. However, an approximate GVT value can be obtained using a distributed broadcast.
Besides the overhead required to enable rollbacks, Time Warp simulations face the challenge of instability. A simulation can become unstable if continuous rollbacks occur, causing the simulation to fail to complete. In response to the potential instability and to address the overheads involved with frequent checkpointing, Avril and Tropper propose a method of grouping Time Warp simulators into clusters, checkpointing and rolling back at the cluster level [2]. The goal of their techniques is to achieve a balance between memory use and execution time.

Creating a system in which diverse simulations can interact is a complex problem with the many interacting tradeoffs described above. The SIMNET approach is limited by the network bandwidth and the small, packet-like message size allowed. The performance of a HLA-compliant federation can suffer from delays that are not necessary for correct coordination of federates and from unexpected overheads in the RTI mechanisms. The optimistic approach used by Time Warp simulations potentially exhibits instability. Additionally, the Time Warp simulations communicate directly with each other, requiring the producers of messages to know who the consumers are.

A Society approach specifically addresses situations where the simulations exhibit dynamic characteristics and have heterogeneous requirements. A Society approach has been applied to integrating a behavior-based simulation modeling human behavior with an HLA federation of tactical military simulations with the goal of modeling reconstruction operations [3].

The rest of the paper is organized as follows: Section 2 describes a Society of Simulations approach to integrating heterogeneous systems. Section 3 contains a detailed case study illustrating how diverse simulations can be integrated using the Society of Simulations approach. In Section 4 a Society of Simulations approach is compared with related work in simulation integration. Future work and conclusions follow.
2 Definition of a Society of Simulations (SoS)

A Society of Simulations (SoS) is analogous to a society of people as both are loosely coupled constructs in which independent individuals contribute toward a single societal identity [6]. A society is an organized group of individuals who associate for common purposes. Likewise, autonomous simulations in SoS work together to achieve the common goal of modeling the system. Each simulation in SoS is an autonomously managed member which cooperates with other members to reach its personal goals. In the process of meeting its personal goals, a member contributes to societal goals. Satisfaction of societal goals emerges as all members progress towards their personal goals.

Consider the example of FireSim and HumanSim shown in Figure 1. FireSim models the physics of a fire and HumanSim models the behavior of people. Thus, the personal goal of FireSim is to model the spread of fire and gasses, taking into account the building layout. The personal goal of HumanSim is to model the movement and actions of an individual within an enclosed environment. A society can be constructed consisting of a number of FireSim and HumanSim simulations to model evacuations in the event of a fire. Therefore, each simulation has simulation-specific goals and all simulations contribute to the societal goal of modeling an evacuation.

SoS can be heterogeneous, consisting of diverse members. Heterogeneity can exist in terms of the data the members produce and consume as well as in terms of how the members advance their simulation times. Concerning heterogeneous data representations, members can have unequal temporal and spatial granularities, be from various fields of science, require different data, use different syntax, and apply different meanings to the same phenomenon. Despite their nonconformity, members contribute to the goals of the society and benefit from the
society's resources. A society consisting of FireSim and HumanSim members is an example of using a heterogeneous set of simulations, with the pooled resources of FireSim’s temperature, carbon monoxide, and soot density information as well as the open door and window events from the HumanSims.

Decoupling producers and consumers enables heterogeneous time management styles among members. Consequently, the time management mechanisms employed by producers do not dictate how consumers must manage their times. Members can be continuous-time, such as FireSim which steps through time at a one millisecond temporal granularity. Members can also be discrete event, as HumanSim members are. FireSim is also optimistic, meaning it can speculatively execute assuming no open door or window events will occur and roll back if such an event occurs for a time in FireSim’s past. HumanSim, on the other hand, is conservative, meaning it will not simulate to a future time until all events up to that future time have been posted.

SoS provides a means whereby independent members with heterogeneous representations of information and diverse methods for advancing a member’s internal simulation time can interact. Interactions among members result from aspects of the simulations’ models of reality that are shared. These shared aspects make up the Shared Reality component of a society which is accessible by all members. All interactions between members occur as members act on entities within Shared Reality. Shared Reality is distributed, enabling localized exchange of data between dependent members. Each member senses changes to the portions of the Shared Reality that are relevant to its execution. The regions of the Shared Reality that a member depends on can change during run time, such as when a HumanSim moves to a new location in the virtual building.
SoS approach consists of an information sharing mechanism, a framework of linking distributed component simulations to shared information, and a process of analyzing and assessing a simulation with respect to the overall goals of the society. These three components are referred to as Shared Reality, Members, and Liaisons\textsuperscript{3}. The following sections describe these components in detail.

2.1 Shared Reality

Shared Reality houses the shared aspects of a member’s models. Shared Reality does not manage how the members operate. As a result, simulations that adhere to diverse time and data management mechanisms can be linked to Shared Reality, enabling a society to be heterogeneous. Data within Shared Reality is coupled with semantic information necessary for a member to comprehend the relevance of the information to the member’s models. The semantic information does not dictate the format, meaning, or granularity of a consumer's inputs. Rather, data relevant to a member is acquired from Shared Reality, transformed into a format the member can digest, and perceived according to the semantics of the member before being used by the member.

What data a member requires and when to incorporate new data during its execution are characteristics of a member and, therefore, are governed by each member individually. The intelligence for transforming information within Shared Reality into a form a consumer can digest and for synchronizing a consumer with produced data is pushed from the data exchange mechanism of Shared Reality onto the linkages that connect members to Shared Reality. Consequently, Shared Reality is light weight in the sense that the overhead a member experiences when accessing Shared Reality is not due to the time management of the other

\textsuperscript{3} Please note that we referred to Liaisons as bridges in previous literature.
members in the society since Shared Reality does not manage the members. Nor is the overhead for accessing Shared Reality proportional to the number of members in the society since Shared Reality is not keeping track of the members that are accessing its data.

For example, FireSim and HumanSim members share open door events. When a HumanSim opens a door, the corresponding door in Shared Reality is opened. FireSim will sense changes to the building layout by querying Shared Reality. Upon finding that a door has been opened, FireSim will move the appropriate wooden obstacle in its representation of the door to the side. In this case, Shared Reality includes the shared aspects of the building layout, such as a door that has an attribute indicating that the door is in a closed or open state. FireSim interprets the meaning of the open door event into a move obstacle action.

Shared Reality is a persistent information space to accommodate different temporal granularities and execution mechanisms among the members. The traits of an entity in Shared Reality can be queried across time. A member requiring data for a time within two posted times can interpolate to find midpoints. For example, a FireSim may post temperature data to Shared Reality once for each simulated second. If a HumanSim queries Shared Reality for temperature data at a half-second temporal granularity, it can assume that the temperature at the start of a second does not change within the second. Otherwise, depending on the precision required by the HumanSim, the temperature values can be interpolated. The specific techniques used to transform data in Shared Reality into a form that meets member-specific needs are performed by the Liaisons, described in Section 2.3. Shared Reality offers historical data (such as temperature values for previous time steps) as needed.

Additionally, distributed development of individual members is possible because a member’s design is separated from the data exchange mechanism. Extensions to a member’s
design do not require changes to the design of Shared Reality. Consider extending HumanSim to incorporate the lack of oxygen in its health model. To do so, HumanSim gathers the concentration of other gasses (concentrations of carbon monoxide and carbon dioxide are produced by FireSim) from Shared Reality to determine its oxygen intake. This extension to HumanSim does not require any changes to the design or configuration of FireSim.

In essence, Shared Reality decouples the producers and consumers of data. This added layer of abstraction between producers and consumers consists of linking members to data types as opposed to directly linking producers to consumers. Shared Reality is relieved of the burden of maintaining a map of producers to consumers, easing not only the implementation of Shared Reality but also the development of links between the members and Shared Reality, referred to as Liaisons.

2.2 Members

Members can be stand-alone simulations, simulations built specifically for a Society, and other components such as visualizations and user interfaces. Before designing a Liaison, members are examined for their potential contributions to the Society. Each input and output of a member is characterized by three orthogonal components: syntax, granularity, and semantics. In Figure 1, two examples are given to show how different members can represent the same information in manners that differ in any or all of the three components. Syntax refers to the data structure and type the member uses to produce the input or consume the input.

Granularity indicates how multiple instances of a member’s input or output relate to each other. A granularity will exist for each dimension of the data, such as a spatial or temporal dimension, as well as whether the entity is an individual or aggregated representation. For
example, a member input that takes a crowd will be aggregating the individual people within the crowd into a coarser grained construct representing the crowd as a whole.

The semantics indicate the meaning of the member’s input or output. A door in a building layout means a wooden obstacle to a FireSim and a removable blockage on a route to a HumanSim.

![Figure 1. The Three Components of Information in Shared Reality.](image)

Information in Shared Reality can differ in any one or all three of the components. Heat and temperature are two terms used to represent the same entity that differ only in their semantic representation. An opening of a door by a HumanSim at a 1 m³ spatial granularity differs from a move obstacle event in FireSim, which represents its obstacles at a 0.1 m³ spatial granularity. However, the syntax of the two events is the same: (open/close, location).

In order for the Society to operate as a whole and meet its intended goals, the inputs of all types of members involved must be satisfied. The role of a type of member with respect to a Society and any additional types of members that may be needed to support a member’s inclusion become evident when assessing the Society as a whole. This analysis is performed when the Society is initially set up. Society-wide analysis is performed using the types of
members that will participate in the Society instead of linking specific instances of members since the data exchange that occurs at run time emerges as an artifact of how the members sense data that satisfy their input dependences.

2.3 Liaisons

Each member in a Society accesses Shared Reality through a member-specific Liaison. A Liaison consists of the intelligence needed to interact with and control a member and to interact with the rest of the Society. A Liaison is configured to use member-specific mechanisms—initializations, inputs, outputs, and control mechanisms. In this way, the same member can be used in different Societies and be continuously developed without being forced to address Society-specific characteristics, enabling reuse and distributed development.

Considering the three components of representing information illustrated in Figure 1, the Liaisons perform the necessary transformation of syntax, conversion of granularity, and translation of semantics between a consuming member’s representation and the various representations of data available in Shared Reality. Transformations between syntaxes are straightforward data type or data structure conversions, such as from a floating-point to an integer number or from a list to an array.

Conversion of granularity can use common techniques to reduce or curve fit discrete data, depending on the type of data and how it is being used. Common techniques to reduce data include utilities to sample, average, and sum, for example.

Semantic translation is necessary when exchanging data between independently developed members since the same data type can be used to describe different information or the same information is implemented using different data types by diverse members. For example,
FireSim and HumanSim both interact with the building layout, but changes to the layout have different meanings to the two types of members. When a door is moved, a route becomes available to HumanSim. The same event to FireSim means a change to airflow in the building, and a movement of a wooden object.

By separating syntax, granularity, and semantics, methodologies can be applied to each component separately. Various syntax transformation tools can be reused for data with different semantics or different granularities. Likewise, semantic-matching tools can be used with different representations of the same information without requiring the tools to address the implementation details of granularity conversion and syntax transformation.

All conversions and transformations are performed by the Liaisons of the consuming members. Producing members create data within Shared Reality without accommodating any representations used by other members. By doing so, the overhead of accommodating the various member formats is performed by each consuming member asynchronously. A Liaison performs the following tasks:

- **Synchronizes the member with data the member depends on.** For example, HumanSim members are conservative simulations that will simulate up to the time steps of any input data available. A Liaison for a HumanSim instance will wait until all data is available in Shared Reality for a time step before forwarding the data on to the HumanSim. As a result, HumanSim members wait on FireSim outputs. When a HumanSim is being rescued, its Liaison will also wait for its rescuer to post a new location.

- **Starts, stops, restarts, and checkpoints a member.** For example, FireSim can be stopped and restarted from a checkpointed time. This enables FireSim to be run optimistically, meaning it can simulate into the future, assuming no open door or window events will occur. If such an event occurs for a time in FireSim’s past, FireSim’s Liaison will stop FireSim, determine the latest time step before the event for which a checkpoint was saved, and restart FireSim.

- **Gathers data from Shared Reality, transforms its syntax, converts its granularity, and translates its semantics.**

- **Places the member’s outputs into Shared Reality coupled with semantic information describing the syntax, granularity, and semantics of the data.**
3 An Illustrative Example: The Evacuation Society

To illustrate how the components of SoS enable heterogeneous members to cooperate to meet societal goals, consider modeling the influence of decision-making on the outcome of evacuations. This Society will be referred to as the Evacuation Society. The simulations are analyzed and the Society components are configured to enable distributed development and reuse.

Several simulations exist that can contribute to the Evacuation Society: HumanSim and Fire Dynamics Simulation (FDS). HumanSim is built to model a person’s implementation of intentions within a physical environment, such as moving around obstacles, interacting with other people, routing to and searching for a location. FDS is a stand-alone combustion simulation provided by the National Institute of Standards and Technology (Building and Fire Research Laboratory, 2004). It models the spread of fire and the flow of gasses at fine detail. FDS also gives an approximation of smoke. FDS has been used to reenact deadly fires in an effort to gain a deeper understanding of how to prevent fatalities in the future [15]. FDS is included in the Evacuation Society under the label FireSim.

The Evacuation Society is constructed with the future goal of modeling emergency evacuations of entire office buildings or places where people gather, such as the evacuations of the Cook County office, Chicago nightclub, and the Rhode Island Nightclub [1, 4, 9, 16]. In order to do so, accurate models of crowding will have to be employed, which are not included in the simplified version of HumanSim used for this example. Additionally, multiple instances of the Evacuation Society may cooperate with other Societies to model emergency evacuations within a city context. The Evacuation Society should be constructed in such a way as to allow these future extensions with minimal changes.
3.1 Analysis of HumanSim

HumanSim models one person’s implementation of his or her intentions. Such an artificial human can sense fires, open doors and windows, rescue other people, and escape. HumanSim also has a health dimension which can be influenced by inhaling toxic carbon monoxide and exposure to extreme heat. The inclusion of HumanSim in the Evacuation Society allows us to quantify the effectiveness of an evacuation in the presence of life-threatening danger.

To enable HumanSim to interact with its virtual environment—opening doors, rescuing people, and routing an escape path—HumanSim is endowed with senses of sight and smell. A HumanSim can see smoke or another member representing a human. Smoke alerts HumanSim to the presence of a fire. HumanSim members can smell smoke as well.

The health of a HumanSim uses a formula to determine the cumulative negative effect of carbon monoxide on a person, taking into account the person’s size. Carbon monoxide reduces the ability of a person’s blood to take oxygen. Also, the concentration of carbon dioxide and carbon monoxide reduces the amount of oxygen available to a HumanSim. Therefore, a HumanSim requires information of the concentration of gasses in the air it is breathing.

3.1.1 HumanSim Initialization

To initialize a HumanSim, a Liaison must provide the information in Table 1. Each HumanSim is given a unique name by an initialization process that starts the members. This name will be used by other members to refer to the HumanSim.

<table>
<thead>
<tr>
<th>Name</th>
<th>building layout</th>
<th>state</th>
<th>location</th>
<th>relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>unordered set of 3D entities</td>
<td>name and string</td>
<td>name and location</td>
<td>name and relation</td>
</tr>
</tbody>
</table>
### Granularity

<table>
<thead>
<tr>
<th>Granularity</th>
<th>0.2 m$^3$</th>
<th>individual</th>
<th>1 m$^3$</th>
<th>individual</th>
</tr>
</thead>
</table>

### Precision

<table>
<thead>
<tr>
<th>Precision</th>
<th>0.2 m$^3$</th>
<th>precise</th>
<th>1 m$^3$</th>
<th>precise</th>
</tr>
</thead>
</table>

### Range

<table>
<thead>
<tr>
<th>Range</th>
<th>unknown limit on # of entities, each within the boundaries of the building</th>
<th>state enumeration: asleep, awake, ill, unconscious, dead</th>
<th>any 3D location</th>
<th>stranger, parent, child, sibling</th>
</tr>
</thead>
</table>

### Ontology

<table>
<thead>
<tr>
<th>Ontology</th>
<th>set of obstacles and openings at specific locations in a frame of reference</th>
<th>personal state</th>
<th>where the HumanSim is</th>
<th>relationship between the named member and “me”</th>
</tr>
</thead>
</table>

---

A HumanSim can have relationship ties to other members that represent humans, such as ties with other members it perceives as its children.

#### 3.1.2 Analysis of HumanSim Inputs

After initialization, the inputs in Table 2 and Table 3 will have to be satisfied.

<table>
<thead>
<tr>
<th>Name</th>
<th>CO</th>
<th>air</th>
<th>temperature</th>
<th>odorous smoke</th>
<th>visible smoke</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>density and location</td>
<td>density and location</td>
<td>temperature and location</td>
<td>boolean and location</td>
<td>boolean and location</td>
</tr>
<tr>
<td>Precision</td>
<td>Unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>Ontology</td>
<td>…</td>
<td>…</td>
<td>…above scalding</td>
<td>…that can be smelled</td>
<td>…that can be seen</td>
</tr>
<tr>
<td>Time-Controlling</td>
<td>True</td>
<td>true</td>
<td>true</td>
<td>true if not alerted to a fire</td>
<td></td>
</tr>
</tbody>
</table>

The HumanSim inputs in Table 2 have the following characteristics in common:

- frequency of one second,
- granularity and precision of 1 m$^3$,
- range of non-negative values,
- all values mean the maximum within the spatial granularity with the provided location as the origin of the 1 m$^3$ cube,
- density is in kg/m$^3$,
- all inputs must be non-speculative values.
The fire-related information can influence the health of a HumanSim. Therefore, the desired values are the maximum concentrations and temperature within the spatial granularity (indicated by “…” in the table). Only temperatures above scalding (approximately 60°C depending on the age of the individual) are considered. Only soot density levels that rise above a threshold (the approximate density that can be sensed by smell) are sensed by a HumanSim. The precise density of soot is not significant. Once smoke is sensed by smell or by less sensitive sight, the HumanSim is alarmed to the danger of a fire. Smoke is sensed only if the HumanSim is not yet alerted to a fire. Once alerted, only the harmful gasses and temperatures are sensed.

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>location</th>
<th>grab event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>name and string</td>
<td>name and string</td>
<td>name and location</td>
</tr>
<tr>
<td>Precision</td>
<td>Exact</td>
<td>1 m³</td>
<td>unique name</td>
</tr>
<tr>
<td>Range</td>
<td>state enumeration</td>
<td>any 3D location</td>
<td>all named members any 3D location</td>
</tr>
<tr>
<td>Ontology</td>
<td>visible indication of named human’s state</td>
<td>location of the named human</td>
<td>the named human is rescuing “me” at location</td>
</tr>
<tr>
<td>Time-Controlling</td>
<td>False</td>
<td>true if being rescued by the named member</td>
<td>false</td>
</tr>
</tbody>
</table>

HumanSims can also interact with members that represent humans. The common characteristics of these inputs are:

- a frequency of one second,
- a granularity of an individual, spatial granularity of 1 m³,
- all inputs must be non-speculative values.

When one member is grabbing a HumanSim to rescue it, the rescuer provides a name and a location. If the named HumanSim is not near the location (within 1 m³), the event is ignored. HumanSims can exit the building layout, so negative locations are allowed. A HumanSim is dependent on a named member’s location only when being rescued by the member.
A HumanSim will sense the state of a human member before attempting to rescue the other member. If the other member is not perceived as alive, no rescue is performed. A HumanSim senses the locations of any human members who are visible within a three-meter radius. The locations of other members are important only to avoid colliding with them and to determine where a human member is for the purpose of rescuing it.

3.1.3 Analysis of HumanSim Outputs

HumanSim can also produce the outputs in Table 4. Common characteristics of all HumanSim outputs:

- granularity of individuals, spatial granularity of 1 m³,
- cannot be rolled-back.

Doors and windows can be opened by an open event.

Table 4: HumanSim Outputs.

<table>
<thead>
<tr>
<th>Name</th>
<th>open</th>
<th>grab</th>
<th>state</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>obstacle</td>
<td>name and location</td>
<td>name and string</td>
<td>3D point</td>
</tr>
<tr>
<td>Precision</td>
<td>unique name</td>
<td>precise</td>
<td></td>
<td>1 m³</td>
</tr>
<tr>
<td>Range</td>
<td>N/A</td>
<td>all named members</td>
<td>state enumeration</td>
<td>any 3D location</td>
</tr>
<tr>
<td>Ontology</td>
<td>open a door/window</td>
<td>grab the named human for the purpose of rescuing</td>
<td>personal state</td>
<td>where the HumanSim is</td>
</tr>
</tbody>
</table>

3.2 Analysis of FireSim

FireSim is analyzed independently from HumanSim. The original version of FireSim (FDS version 3 from NIST) did not incorporate changes to the building layout when FireSim was restarted from a checkpoint. FireSim was modified to reread the building layout file and incorporate the new appear/disappear times for obstacles in the file, such as when a door is moved.
**Analysis of FireSim Inputs.** Changes to the building layout occur by objects appearing or disappearing. FireSim will read in the entire building layout file described in Table 5 each time it restarts. FireSim’s Liaison will need to manage changes to the building layout file.

**Table 5: Inputs of FireSim.**

<table>
<thead>
<tr>
<th>Name</th>
<th>building layout</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>unordered set of 3D entities</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>1 s</td>
</tr>
<tr>
<td><strong>Granularity</strong></td>
<td>0.1 m³</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.1 m³</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>unknown limit on # of entities each within boundaries of the building</td>
</tr>
<tr>
<td><strong>Ontology</strong></td>
<td>set of obstacles and openings at specific locations in a frame of reference</td>
</tr>
</tbody>
</table>

**Analysis of FireSim Outputs.** Table 6 describes the possible outputs that FireSim produces which are relevant to the Evacuation Society. All output values of FireSim have the following common characteristics:

- type is a 3D matrix of floating-point numbers,
- spatial granularity is 0.1 m³, temporal granularity is 1 s,
- range of non-negative values,
- concentration is in kg/m³,
- can be rolled-back.

**Table 6. Outputs of FireSim.**

<table>
<thead>
<tr>
<th>Name</th>
<th>CO</th>
<th>CO₂</th>
<th>temperature</th>
<th>soot density</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>Unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>approximation</td>
</tr>
<tr>
<td><strong>Ontology</strong></td>
<td>concentration in each block of the 3D grid</td>
<td>concentration in each block of the 3D grid</td>
<td>average in each block of the 3D grid</td>
<td>concentration in each block of the 3D grid</td>
</tr>
</tbody>
</table>

In order to restart FireSim from a specific checkpoint, multiple checkpoints must be saved and timestamped with the simulation time that they represent. This intelligence is placed within FireSim’s Liaison.
FireSim operates at the precision of a millisecond but checkpoints at the frequency of one simulation second. FireSim produces outputs and can sense changes to the building layout at its checkpoint frequency.

3.3 Configuration of Shared Reality

All outputs from HumanSim and FireSim become the entries for Shared Reality. The information that describes the three components of each type of entry (syntax, granularity, and semantics) is placed in Shared Reality by the Liaisons when a member joins a Society.

The FireSim outputs are grouped into 1 m$^3$ cubes (as opposed to one entry for each 0.1 m$^3$ cube) to make the number of entries in Shared Reality easier to manage for small experiments. For example, the following Shared Reality entry is produced to hold carbon monoxide information:

```
carbon monoxide
  10x10x10 matrix of density [units=kg/m$^3$]
  origin of matrix [x,y,z, units=meters]
  time [units=seconds]
```

*Design and Customization of a HumanSim’s Liaison.* The HumanSim Liaison provides the following functionality to HumanSim members:

- All sensing of Shared Reality is performed at the frequency of one simulation second, which is the sensing frequency of HumanSim.
- Sense fire related information.
  - *Query Shared Reality for odorous smoke at the location of the synthetic human’s head.* The Liaison only queries for smoke if the fire has not been perceived yet. This query takes into account the location and height of the HumanSim. The Liaison will filter out a soot density value that is below what can be smelled by a human.
  - *Query Shared Reality for visible smoke that has not already been recorded.* Shared Reality has precise soot density values. The HumanSim Liaison will filter out soot density that is not perceptible. The locations of smoke are provided to the HumanSim’s sight sensor which determines if any smoke is visible or blocked by
obstacles in the building layout. The Liaison keeps track of which locations are over the visibility threshold and only forwards the new locations to its member.

- Sense information relating to humans.
  - Query Shared Reality for any grab events of this HumanSim. When another member grabs this HumanSim, the HumanSim’s Liaison determines if the location of the HumanSim is close enough to the location of the grab event. If so, it will inform the HumanSim that it is being rescued and will query Shared Reality for the rescuer’s location from then on.
  - Query Shared Reality for the state and location of other humans within a three meter radius of the HumanSim. The Liaison provides the human’s location to its member. If the human is not blocked by obstacles, the member can sense the human’s state. This query is only performed when the member is moving or seeking to rescue someone.
- If the HumanSim dies, no sensing of Shared Reality will be performed.
- Produce HumanSim outputs within Shared Reality.

Design and Customization of a FireSim’s Liaison. The following functionality was encoded in FireSim’s Liaison:

- Produce FireSim outputs within Shared Reality.
  - Read the FireSim output files and decompose the 3D matrix into 1 m³ cubes.
  - Produce one set of 1 m³ cubes for each data type (CO, CO₂, temperature, soot density).
  - Calculate the location of each cube and create a Shared Reality entry.
- Sense changes to the building layout in Shared Reality.
  - Changes in this example come from open door and open window events.
  - Transformation of syntax and translation of semantics
    - Doors and windows are identified by location. The FireSim Liaison will locate the corresponding obstacle and remove it.  
  - Conversion of granularities
    - The sensing is performed at the top of every second, when FireSim is checkpointing, before rereading the building layout file.
    - Upon sensing a change in building layout, the FireSim Liaison will incorporate the change in building layout in the building layout file. If

---

4 To model a door that swings open requires more than the FireSim building layout can handle. Both the closed door and opened door can be placed within the layout. Then, the Liaison will make the opened door appear and the corresponding closed door disappear when the door is opened. This precision did not add enough to the simulation’s results to warrant the effort.
FireSim has already simulated past the event’s timestamp, FireSim’s Liaison will roll FireSim back to a time equal to or before the event’s timestamp.

- Checkpoint and roll back FireSim.
  - FireSim’s Liaison will save copies of the restart file for each simulation second and append the simulation timestamp to the filename.
  - To roll back FireSim, the Liaison stops FireSim, discards the checkpoints with times greater than the event’s timestamp, and restarts FireSim from the latest checkpoint before the event.

### 3.4 Analysis of the Evacuation Society

To simplify the example, all members that produce information that FireSim consumes are assumed to be within one second of FireSim’s simulation time. Otherwise, the FireSim Liaison will have to undo data that was previously produced within Shared Reality when a roll back to a previous timestamp is required. The possibility of “undo data” also requires the calculation of when data produced by FireSim will never be rolled back. The issues surrounding optimistic simulation will be addressed in a future paper.

The potential dependences between members are illustrated in Figure 2. In this scenario, FireSim produces its data for simulation time of 1. This data is used by the HumanSim members to advance their time up to a simulation time of 2. When HumanSim opens a door during the period of time from 1 to 2, this event is given a time of 2. FireSim’s Liaison will sense the open door event and roll back to the time of 1. If a HumanSim is being rescued, the rescuee will wait for FireSim data and also for its rescuer to post its location. A rescuee will post its location to match that of its rescuer for a time of 2.
No deadlocks occur since HumanSims are driven by FireSim data and FireSim does not wait on HumanSim data. Rather, FireSim optimistically executes ahead. HumanSims do not get in a deadlock with each other since a HumanSim is either independent of the other HumanSims or is being rescued by one. If a HumanSim is being rescued, it will not grab another HumanSim to rescue it.

Figure 2. Dependencies between Members in an Evacuation Society. This illustration shows how HumanSims wait on data that is produced by FireSim and how HumanSims can, in turn, influence FireSim. The dependence arcs are labeled with simulation timestamps.

3.5 Results of the Illustrative Example

The Evacuation Society is not hard coded for a specific scenario or for a preconceived outcome such as a happy ending. Rather, the Society approach forces the modeling of an evacuation down to the individual human level in this case, with the result that the relative success of an evacuation is an emergent property. As the FireSim and HumanSims fulfill their
personal goals of modeling a fire’s spread across materials and the behavior of individuals in an emergency, the evacuation unfolds. Since the Evacuation Society is not scripted, fires and humans can be placed anywhere within the virtual environment and new story lines emerge. For example, configuring the Evacuation Society with a two-floor townhouse, multiple family evacuation situations can be modeled. The setup of several scenarios is shown in Figure 3.

Example 1 begins with an electrical fire in the infant’s bedroom while the parents and children are asleep. Running the Example 1 configuration resulted in the parents sensing the smoke, waking up, grabbing the children, and escaping with no damage. Example 2 begins with the parents downstairs. The door to the baby’s room is open and the two children are upstairs in the room with the fire. Even though the door is open, the parents do not sense the smoke until it is too late to rescue the children.

Though these examples are very rudimentary in terms of the psychological and physiological models, the Evacuation Society demonstrates how SoS is constructed with diverse simulations and how interactions among diverse members are made possible through the use of Liaisons and Shared Reality.

4 Categorization of Approaches to Integration

Integration of heterogeneous simulations draws from simulation integration, composition of heterogeneous components, and distributed computing. Eugster et. al. characterize publish-subscribe and related mechanisms by space, time, and synchronization decoupling [5]. Extending the work of Eugster et. al. for addressing simulation integration and the support for heterogeneity reveals the key differences in theory among simulation integration approaches.
Figure 3. Multiple Configurations of Townhouse (a) Example 1, (b) Example 2. In Example 1, the family is asleep. In Example 2, the parents are downstairs and the kids are upstairs awake.

4.1 Classification of Simulation Integration

Simulation integration approaches can be classified according to the following characterization:

1. Run-Time Autonomy—How autonomous are the run times of the members?
   The following categories address the amount of society-wide, centralized coordination that is required in order to integrate simulations for a given approach.
   a. data access intelligence—Where does the intelligence reside to determine what data to provide to which members? Within the middleware or within the members?
   b. event ordering intelligence—Where does the intelligence reside to ensure that events are processed in order? Within the middleware or within the members?
   c. coordination of publishers and subscribers—How do publishers and subscribers coordinate? By point-to-point messaging using society-wide, unique member IDs? By specifying the data type produced or consumed? By describing the semantics of information produced or consumed?
   d. initialized or adaptive—Is there a society-wide configuration established during initialization or does the society adapt as members join and leave?
e. synchronous or asynchronous—Is the communication between members synchronous (akin to communicating by telephone) or asynchronous (akin to communicating by email)?

f. simultaneous or disjoint run times—Do members have to be running concurrently in order to exchange data?

2. Heterogeneity

   a. heterogeneous syntax
   b. heterogeneous granularities
   c. heterogeneous semantics

Category 1.a characterizes the required data management within a simulation integration approach. If a centralized data manager is required, any change to a member’s consumption pattern will have to be coordinated with the manager. A data manager can be used to reduce the amount of unnecessary data being broadcast to subscribers by keeping track of the consumption patterns of members. Such data management mechanisms prevent member autonomy and limit the scalability of the system.

The choice of a time management mechanism can prevent member autonomy and limit the scalability of the system. Category 1.b addresses the component in which the time management intelligence resides. The time manager for a member is the mechanism that answers whether the member can advance its simulation time or not. Centralized schemes construct a time manager within the middleware. The time manager uses knowledge of each member’s current simulation time and each member’s potential outputs to order all messages being routed to each member.

Simulations that exchange data must collaborate using some form of common knowledge. Category 1.c addresses the run-time coordination that is necessary for exchanging data. This is similar to space decoupling in [5]. The coordination is either by a unique member identifier (such as HumanSim#3), the data type of the data being exchanged (such as “a double
named CO_volume”), or the information being shared (such as “density of toxic gas, carbon monoxide, throughout a volume”).

When simulations are linked based on data type, the designers of all members must agree on a type naming convention and on the granularity of the shared data. Both the required, pre-simulation coordination and the agreement on the granularity of specific data types hinder autonomy of members. Synchronous communication, (category 1.e), described in [5] under the synchronization decoupling category, requires a channel to be established between communicating members at run time. On the other hand, asynchronous communication enables production and consumption to occur at different times by storing communication in an intermediary location.

Category 1.f addresses the necessity of coordinating the run times of members that are interacting, which can prevent member autonomy. This category is described as time decoupling in (Eugster et. al., 2003). Communication may be asynchronous and still require actions by both the producer and consumer in order to transmit the messages between them, for example, when the intermediary storage of messages are buffered within the consumer. In this case, if the consumer is not available, the message will not be delivered to the consumer.

Category 1.d addresses a subtle issue that becomes important when new members may be added after a society’s initialization. If initialization of SoS requires knowledge of all members involved, adding new members at later points in the run time of the society requires a reconfiguration of the society.

How heterogeneous SoS can be is determined by how information is shared among members. If members are required to use the same data format when sharing information, then the society is syntax-homogeneous, that falls under category 2.a. A syntax-homogeneous
approach requires coordination in the design of all members, preventing distributed development. Enabling the same information to be represented in different formats implies that data transformation is performed on data produced before being consumed.

Forcing all members to represent data according to the same granularities limits the heterogeneity possible in a society, addressed under category 2.b. When a data type shared by members is represented in a single granularity, a consuming member is forced to abide by the granularity characteristics of the producer. Allowing multiple granularities of access to the same data implies that a granularity conversion of data is performed on data produced before being consumed.

To allow distributed development of members, no coordination of what data types refer to what information can be done prior to run time. Rather, members inform others of information they produce by describing the semantics of the information, which is covered under category 2.c. Most integration systems assume the same semantics are used by all of the members, a semantics-homogeneous approach. To facilitate a semantics-heterogeneous society, a common, domain-specific ontology is provided by a society. Members then extend the common ontology to inform others of the semantics of data they produce and consume. Such a semantics-decoupled approach implies the ability to translate between multiple ontologies.

**Table 7.** Classification of Simulation Integration Approaches with Respect to Autonomy and Heterogeneity. The columns correspond to the categories for classifying simulation integration approaches.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Autonomy (1)</th>
<th>Heterogeneity (2)</th>
</tr>
</thead>
</table>

28
4.2 Classification of Related Approaches

In Table 7, many simulation integration and component integration approaches are classified using the classification method described above.

*Data Access Intelligence.* Data access intelligence can be within the producer, consumer, both producer and consumer, or in the middleware (MW). For example, producers and consumers are autonomous in SIMNET. Producers broadcast all messages to every other member. It is the job of the consumers to filter out the data that does not apply to them. High Level Architecture (HLA) uses a data manager within the middleware with the goal of only transmitting to a consumer only the data that is useful for that consumer. To initialize an HLA federation, each federate informs the data manager of its resources and needs through a Simulation Object Model (SOM) [18, 10]. The data manager uses this information to map producers to consumers. Changes to the publisher-subscriber map at run time require barrier synchronization across all federates.

In contrast, interactions among members of SoS are not pre-defined before the society is initialized. The links between producers and consumers emerge as members access common data in Shared Reality at run time. Consuming members determine what data they need from Shared Reality. As a result, if the data a member needs changes while the simulation is running, no
management mechanisms within Shared Reality have to be invoked; the member simply accesses different data.

*Event Ordering Intelligence.* Proper event ordering must be ensured either by the producer, consumer, both producer and consumer, or the middleware. In the simulation integration approaches, the RTI mechanism in the HLA approach manages the ordering of events through by delaying when messages are forwarded to consumers. Simulations request from RTI the ability to advance their times. RTI will hold events for a consumer until it can ensure that no new events will be produced out of order.

In contrast, SoS approach delegates to each member the task of coordinating the member’s internal simulation time with that of any produced data. No mechanisms are required within Shared Reality to manage the members’ advancement of time.

*Coordination between Producers and Consumers.* Many component integration approaches identify the recipient of a message by a characteristic of the recipient, such as by a unique ID. In HLA, the set of all data types to be shared is provided to the federation manager through a configuration file (the Federation Object Model) before federation initialization. The federation manager determines which of the federates a federate will communicate with.

The need for autonomous members is noted in [20] with respect to how optimal *usability* is determined by each user (which, in this case, is a simulation member) based on what the user perceives is important. In SoS, consumers search for data available in Shared Reality by specifying a particular data type. In future work, SoS will be extended to enable members to coordinate based on semantics of the information the data represents. Sonnewald [20] describes the need for semantics-based matching as “logical unification”.
*Adaptive Configuration.* Only SoS approach addresses the need for adaptive configuration of a Society. Other approaches, such as HLA, allow federates to join post initialization but require restarting the federation. In a Society approach, Shared Reality is not aware of what members exist. Therefore, adding a new member does not require coordination with a Society-wide manager.

*Synchronous or Asynchronous Data Exchange.* If data is forwarded to a consumer as soon as it is produced, the data exchange synchronizes the producer and consumer. The consumer is forced to receive the data at the production frequency of the producer.

SoS approach uses a data exchange mechanism such as JavaSpaces to implement Shared Reality. Shared Reality provides persistent data for multiple time steps, thus enabling producers to produce data for different time steps and consumers to consume data at different granularities. In contrast, an HLA approach interrupts the consumers with new data from producers and is therefore labeled “synch” in Table 7. The consumers are synchronized with the producers’ production frequency, though the consumers may not process and incorporate the newly produced data until they are at a stopping point.

*Simultaneous or Disjoint Run times.* The last column under Autonomy indicates whether the producer and consumer must both be running concurrently or if their run times can be disjoint. If the exchanged data is not stored in a persistent buffer both parties must be available, as is the case for all but the Society of Simulations approach. These are labeled with “sim” to indicate that producer and consumer must be executing simultaneously. Otherwise, the approach is labeled “disj” to indicate disjoint run times are possible.

*Syntax-Heterogeneity.* The columns under Heterogeneity indicate if the approach assumes homogeneous data syntaxes, granularities, and semantics for the data exchanged among simulations. Homogeneity is labeled with “homo”. Only SoS approach allows members to use a
different syntax for the same data. An HLA approach specifies a common syntax among all federates using a FOM.

*Granularity-Heterogeneity.* Granularity differences among diverse simulations can occur along any dimension of shared data, even the time dimension. SoS approach enables diverse temporal granularities among the members by using an on-demand data exchange mechanism—a consumer asynchronously queries Shared Reality for desired data at the consumer’s sensing frequency (the temporal granularity at which the consumer incorporates new data). Other approaches, such as HLA, use an interrupt-protocol that require consumers to at least buffer the data at the production frequency of the producers, thereby forcing a common temporal granularity of data access among simulations.

*Semantics-Heterogeneity.* The Society of Simulations approach described in this paper does not address multiple ontologies. The approaches described in this paper assume that all simulations have a common understanding of all events, such as an Open Door event. A semantics-heterogeneous approach would enable consumer members to discover new data by matching a semantic description of the data with their own ontologies, and then to automatically digest the new type of data appropriately. A semantics-heterogeneous Society approach that uses ontology-matching will be presented in a future paper.

5 Conclusion

Integration of heterogeneous simulations is necessary to address broad problems and to refine models with other validated models. The challenge of integration in the presence of heterogeneity results from the fact that diverse models correspond to diverse implementation methodologies. Diversity exists in the types of information the models produce and consume, the
granularities at which the models interact with the information, and the format of the data used to represent the information.

SoS approach enables solutions to multi-domain problems while facilitating distributed development and independent design of domain-specific simulations. Collaboration among heterogeneous simulations is likened to members of a society working together to achieve common goals. Simulations are regarded as members of SoS. SoS enables distributed development since members are autonomously managed. Changes to members do not require changes to the design of a Society, making distributed development possible. Autonomous management is enabled by linking members to information instead of to other members.

Heterogeneity results from allowing independent development of member designs. Interactions among heterogeneous members are enabled through intermediary Liaisons. Liaisons are configured to specific members and can interact with any type of information in Shared Reality. The example of an Evacuation Society of Simulations illustrates how a Society is constructed, starting with analysis of each member with respect to the goals of the Society. The outputs of members determine the information that can be shared through Shared Reality. The Liaisons are configured to communicate with and control members, and to interact with the rest of the Society on behalf of the members.

Many approaches to simulation integration have reuse, heterogeneity, and scalability as goals. However, the underlying data exchange mechanisms hinder these goals. A classification of simulation integration approaches clarifies the uniqueness of SoS approach and illustrates why both heterogeneity and simulation integration are not solved by existing techniques. Publisher-subscriber approaches fail to enable autonomy because of the centralized management mechanisms required to link and synchronize
the members in the context of integrating simulations. SoS approach allows simulations to cooperate yet remain autonomous, an inherent and scalable approach for linking heterogeneous simulations.

7. References


[23] Building and Fire Research Laboratory, 2004