ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-837

OpenSim: A framework for integrating agent-based models and simulation components

Dhirendra Singh and **Lin Padgham** [1]

Abstract.
The growing use of agent-based modelling and simulation for complex systems analysis has led to the availability of numerous published models. However, reuse of existing models in new simulations, for studying new problems, is largely not attempted. This is mainly because there is no systematic way of integrating agent-based models, that deals with the nuances of complex interactions and overlaps in concepts between components, in the shared environment. In this paper we present an open source framework, called OpenSim, that allows such integrated simulations to be built in a modular way, by linking together agent-based and other models. OpenSim is designed to be easy to use, and we give examples of the kinds of simulations we have built with this framework.

1 Introduction

When building simulations of complex interacting processes, it is often preferable to *reuse* tried and tested pieces, that have been independently validated and verified, as much as possible. For instance, in simulating the evacuation of a community during a bushfire, it may be desirable, or even required by users, to build the various pieces (such as traffic flow, fire spread, human decision making) using existing specialised platforms and models (such as MATSim[2] for traffic simulation, Phoenix RapidFire[3] for fire simulation), and then combine these together into a single simulation [12].

Such integrations are particularly challenging when agent-based components are included, due to the high levels of interactions with a shared environment, that must be appropriately managed. For instance, multiple components updating a shared resource in the same logical time can leave the resource in an inconsistent state. Standards like the HLA [3] disallow such cases by mandating that only one component be allowed to update a shared resource at any time (Rule 5 [3]). Such rules serve well where resources by nature can only be used exclusively, such as an ambulance shared by two disaster simulation components. However, they are too restrictive for resource types that can be updated concurrently, such as hospital beds being used by two disaster simulation components simultaneously, and often several times in the same time step by the numerous agents within the components. Here it is not sensible to give precedence to one kind of emergency, and indeed if sufficient hospital beds were available at any time then both should be allowed to use them [10].

The issue is further complicated when the resource being shared by simulation components, is an agent. Here, as well as ensuring that updates to the agent's state are consistent in each simulation step, the integration runtime must also ensure that the agent's actions are consistent, since they could have being performed in different components [13]. For instance, it should not be possible for a sick patient to be travelling to the hospital in a ambulance in one component, and family car in another, at the same time.

Other key issues come in the form of incompatibilities between models relating to the paradigm in use (such as discrete event, time-stepped, or mathematical), handling of simulation time (models may operate on different time-scales or not model time at all), sharing of data (independently developed models may differ in their internal representations of shared concepts requiring translation), and the runtime environment (the available simulation pieces may run in separate environments like Java, Python, Matlab, NetLogo, and so on). Existing integration frameworks and standards that address some of these challenges include the High Level Architecture (HLA) [3] and its predecessor Distributed Interactive Services (DIS) [5], CSIRO Common Modelling Protocol[4], Object Modelling System[5], as well as domain-specific integration platforms like OpenMI[6], CIEPI[7] and SISS[8]. However, other than our own early work with the BLOCKS framework [11], these do not deal with the nuances specific to combining agent-based models.

In this work, we present OpenSim, a framework for building integrated simulations that include agent-based components, and discuss example uses of it: a simulation of hospital utilisation during emergencies, a mine excavation and blending simulation, a taxi service simulation, and a bushfire evacuation simulation. Our core contributions with OpenSim are that it: (i) enables agent-based couplings, via runtime support for simultaneous updates to shared resources, and resolves outstanding issues identified with our earlier BLOCKS framework (ii) is a complete integration framework that can be used as an alternative to existing frameworks including the HLA and OpenMI (iii) is easy to use, designed with the philosophy that simple integrations should be simple to realise (iv) is an Open Source initiative, with a Java-based runtime implementation.

The remainder of the paper is set as follows. The next section describes the requirements for an integration framework for agent-based simulations, and discusses related works in this context, followed by, in Section 3, motivating examples of the kinds of integrations we are interested in. We then present, in Section 4, the details of our OpenSim framework. Section 5 concludes with a discussion on open issues and the road ahead.

[1] RMIT University, Australia, email: firstname.lastname@rmit.edu.au
[2] http://www.matsim.org
[3] http://www.bushfirecrc.com/news/news-item/mapping-bushfire-potential
[4] http://www.apsim.info/Portals/0/Documentation/Protocol%20Specification.pdf
[5] http://www.javaforge.com/project/oms
[6] http://www.openmi.org
[7] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6465211
[8] https://www.seegrid.csiro.au/wiki/Siss

Requirement	OpenSim	BLOCKS	HLA	OpenMI
Flexible runtime configuration of components	XML	Custom	FOM	XML/GUI
Support different operating environments	√	√	√	√
Combine event-based and time-stepped models	√	√	√	√
Runtime support for pipelined/chained models	√	Partial	×	√
Progression of simulation time	Model-driven	Model-driven	Model-driven	Controller-driven
Shared data mapping and translation	User, Inbuilt planned	×	User	User, Inbuilt
Support for serial and concurrent data updates	√	Concurrent-only	Serial-only	Serial-only
Configurable resource allocation policies	√	Limited	×	×
Shared agents (states)	√	Concurrent-only	Serial-only	Serial-only
Shared agents (actions)	Planned	×	×	×

Table 1. Comparison of existing simulation integration frameworks against the requirements of Section 2

2 Requirements for an Integration Framework

A suitable integration framework should provide a *methodology* for systematically building-up a simulation, as well as a *runtime infrastructure* for running the component models, coordinating simulation time, and synchronising data. In this section, we provide the key requirements for the runtime infrastructure.

Runtime configuration of components: Infrastructure should include provision for specifying the makeup of the simulation, its components and how to initialise them, shared data between components, user-configurable simulation parameters, initial configuration, logging and recording results, and so on. In HLA, the Federation Execution Data (FED) file is used to supply such information to the runtime infrastructure, while OpenMI uses compositions built using a visual interface for specifying modules and linking their inputs and outputs. In contrast, OpenSim uses the standard XML format which has the added benefit that it is widely supported by parsers and tools.

Support for different operating environments: The framework should support integration of models that run in different execution environments, as in [7] that combines the Java-based MATSim traffic flow simulator with the Python-based UrbanSim[9] urban development simulation. All existing integration frameworks support this requirement, typically by *wrapping* existing components to operate in a common runtime environment such as Java, e.g., Portico[10], or C#, e.g., FluidEarth[11]. Where HLA/OpenMI implementations link components via inter-process (network-based) couplings, OpenSim additionally supports in-process couplings. This means that components written in Java can be linked to the Java-based runtime directly, which can boost execution performance up to six times compared to inter-process coupling[12].

Support for different modelling paradigms: Components being integrated often operate in different paradigms like event-based, pull-driven, pipelined/daisy-chained, and time-stepped. For example, our work in [9] integrates the event-based Jack belief-desire-intention (BDI) reasoning platform[13] with the time-stepped RePast Simphony[14] agent-based simulation platform. Most frameworks support this requirement, typically via wrappers and inter-process communication. Where HLA allows time/event-based approaches, and OpenMI/FluidEarth allows a chained/pull-driven approach, OpenSim supports a combination of any of these approaches.

Data sharing between components: Simulation components should specify what data they publish and subscribe to. Since internal representations of the data may vary between models, this may require model-specific translation to convert data to/from the shared format. In simple cases translation is loss-less, e.g, when converting between Celsius and Fahrenheit. In many cases, however, conversion requires approximation, aggregation, or dis-aggregation. OpenMI provides built-in *adaptors* [1] for converting between spatial representations. OpenSim currently only supports user-specified translation functions via the wrapper, and we plan to add commonly requested adaptors based on user feedback.

Management of simulation time: The runtime should manage progression of simulation time without restrictions on how time is represented (if at all) in the components. Models should be free to choose the time granularity at which they execute, which may change dynamically, such as for event-based execution. OpenSim fully supports simulation time management, which is model driven, similar to the HLA. Differences in time granularity can require special attention when a model accesses outdated data from another model, for which approximation techniques like extrapolation/interpolation have been proposed for OpenMI [1], and is planned for OpenSim.

Management of shared resources: Models should maintain a common and consistent view of shared resources as the global simulation progresses. The HLA/OpenMI approach to restrict update access to one component in each time step is not desirable. Later work has separated shared access into two types, resources that can only be updated *serially* like in the original HLA, and those that can be updated *concurrently* [6]. Allowing concurrent access puts additional responsibility on the runtime to ensure consistency of updates in each simulation step. A specific issue is to prevent "overuse" of limited resources, for which two alternative mechanisms have been proposed: acting *ex-ante* before the fact [6, 10], or *ex-post* after [13]. Additionally, frameworks should provide policies for imposing "fairness" in the use of resources. This is specially relevant for agent-based simulations, where agents in a model may make several modifications to a shared resource in a single simulation step. OpenSim is the only runtime that fully supports this requirement.

Support for shared agents: Sharing agents between components is more complex than sharing aspects of the environment. Changes in agent state can be handled via shared serial/concurrent access. However, there may be actions that could be done by an agent in separate modules, that are conceptually inconsistent. For instance, a doctor cannot be simultaneously treating a patient at the hospital in one component, and going with the ambulance to the disaster scene in another. OpenSim is the only framework to support this requirement. Current support is partial: only agent state, not actions, can be shared.

[9] http://www.urbansim.org
[10] Portico reference implementation of HLA: http://www.porticoproject.org
[11] FuildEarth reference implementation of OpenMI: http://fluidearth.net
[12] Based on a direct comparison for the example hospital system integration discussed in Section 3.1.
[13] http://aosgrp.com/products/jack
[14] http://repast.sourceforge.net

3 Example Integrations and Architectures

In this section we give examples of three kinds of use cases we have identified, for which OpenSim can be used.

3.1 Integrating existing simulations

This example captures the motivation of using existing simulations, that have been independently validated and verified, and combining them into a single simulation for a new purpose. Here, it is preferable to keep changes to the models as small as possible. As a consequence, common aspects of the environment represented in both models, cannot be simply pulled out into the larger simulation, as that requires significant re-engineering of the underlying models. Instead, the integration framework should allow such "variables" to be suitably linked together, and their states synchronised, at relevant points during the simulation. Our work in [10, 13] has focused on such issues of synchronisation and conflict resolution for agent-based simulations. Importantly, these kinds of integrations are not possible in existing frameworks such as the HLA and OpenMI.

The example simulation we developed, to highlight these kinds of issues, showed how a shared hospital system would cope with simultaneous emergency situations. The integration challenge was to take two independently developed simulations of emergencies where hospitalisation of patients was being simulated, and combine them into a a larger simulation with a single shared hospital. The individual systems existed as two separate Repast time-stepped simulations: one simulating a natural disaster, the other the spread of a deadly virus. The key issues are that: (i) disaster and disease components should both have the option of allocating beds in any timestep. (ii) resources required together by a component, should be provided to the component together. So for instance, it is no use allowing the disease component to have a bed, but not public funds also required for the medication for a patient. (iii) conflict resolution should ensure that the combined bed allocation in any timestep does not exceed the number available. (iv) a shared ambulance agent is being controlled by both components and updates to state and actions must be managed carefully[15].

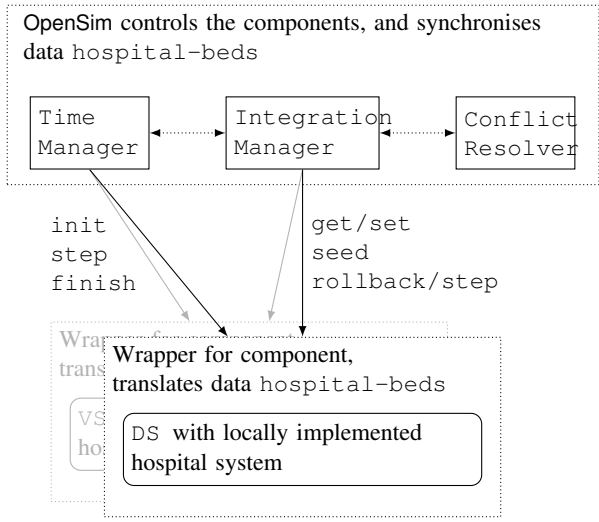

Figure 1. Architecture of an OpenSim simulation, here showing the example integration of the disaster simulation (DS) and viral spread simulation (VS) for the hospital system of Section 3.1.

3.2 Building simulations using existing platforms

It is often desirable to combine technologies that specialise in aspects being modelled, for a specific problem. These technologies can include domain-specific platforms like MATSim for traffic simulation, or general-purpose ones like BDI for reasoning agents. This use case relaxes many of the constraints of the previous one, since the system architect has some control over what concepts will being modelled and how they will be shared between platforms, and can design the application in a way that reduces data representation and conflict issues to some extent. The simulations we have built, that belong to this category, are:

- Bushfire evacuation simulation: was developed in consultation with the Country Fire Authority (CFA), and simulates a bushfire in extreme weather conditions in regional Australia. The fire was simulated by Phoenix RapidFire, as mandated by the CFA. The reasoning performed by the residents was implemented in the Jack BDI platform. The actions of agents, to drive somewhere, were simulated in MATSim. A visualiser showed the Breamlea landscape and roads, the driving agents, and the progressing fire. The BLOCKS runtime infrastructure was used to connect the various pieces together [12], and we are re-doing it now using OpenSim.
- Flooding response simulation: was developed in collaboration with local government council and Victoria State Emergency Service (SES), to analyse the potential for sandbagging in an urban area in Melbourne, Australia. It uses regional data including the local road network, buildings, demographics, and flood progression data from an earlier event. The actions of residents, such as driving to/from sandbagging depots, were implemented in Repast Simphony. The reasoning of the agents was captured using the BDI representation of goals and plans, derived from interviews with council, emergency services, and community members [16].
- Taxi service simulation: uses MATSim for simulating the movement of taxis, and the GORITE BDI platform, for the decision making of the taxi agents as well as the dispatching service. In this set up, some MATSim agents can be started with no fixed daily plans, representing taxi agents, and use BDI reasoning to decide which jobs to take, depending on the situation [8, 2].

These kinds of simulations often require a combination of time/event/pipelined couplings that cannot be achieved in other frameworks like HLA/OpenMI. For instance, the bushfire simulation includes the event-based Jack and time-based MATSim that have to be pipelined in order to achieve decision-action effect.

3.3 Developing simulations in parts

In the interest of modularity and maintainability, it is often desirable to build large simulations from ground up, but as a collection of smaller simulations. This scenario offers the most flexibility for designers, with full control over the development of the smaller simulations and their control and data interfaces.

We are building an example system in this manner, in collaboration with an industry partner, to capture the various processes and constraints in a mining operation. The simulation combines two smaller simulations that we are also developing: one to model the

[15] As per Section 2, OpenSim does not currently support shared actions.

[16] https://sites.google.com/site/rmitagents/emergency-management

Function	Description
`init()`	Initialises the model. Called once at the start of the simulation prior to any other call to the model.
`step()`	Progresses the model simulation by one simulation step. Different models may run at different time granularity. The `TM` controls the progression of logical time and this function is called on only the models scheduled to run in the current logical time.
`rollback()`	Reinstates the model to the precise state that existed prior to the last `step` call. As a result, a series of repeated executions `step` → `rollback` should result in identical start and end states. Model wrappers should implement this by saving the full state of the model at the start of every step, and restoring it when this function is called.
`setSeed(n)`	Set the seed to `n` for all pseudo-random number generators used by the model. As a result, the execution `setSeed(n)` → `step` → `rollback` → `setSeed(n)` → `step` should result in identical start and end states. Used by the OpenSim controller during conflict resolution to ensure that an execution sequence will converge to an acceptable resource allocation solution.
`getValue(v)`	Gets/sets the value of the simulation variable `v`. Models may differ in how the shared concept `v` is represented internally, and the `IM` uses converters for translating values from one representation to another.
`setValue(v,val)`	
`getInUse(v)`	Gets/sets the "in use" attribute of the shared variable `v`. The getter function returns `true` if the model is currently using v, for instance during a multi-step action. The setter is used by the `IM` to inform components when v is in use by some other component, and also when it is available for use again.
`setInUse(v,val)`	
`finish`	Called once at the end of the simulation to allow models to perform any final tasks then terminate gracefully.

Table 2. OpenSim component model wrapper interface functions

excavation process, and another to model the blending process for excavated material of varying quality.

This type of work-flow, i.e., building agent-based simulations by first developing smaller agent-based simulations of sub-systems and then combining them, is straightforward to achieve in OpenSim.

4 The OpenSim Framework

We now describe our integration framework, OpenSim, that is built with the requirements of Section 2 in mind, and which can be used for the kinds of integrations we have described in Section 3. OpenSim allows disparate simulation components to be combined into a single global simulation. The framework provides an *interface* for controlling the components in the simulation, and linking together the shared concepts, or *variables*[17], within the individual components, as well as runtime *infrastructure* for progressing the simulation and resolving and synchronising updates to any shared data.

The key modules in the OpenSim runtime infrastructure are: the `Time Manager` (`TM`) for progressing the simulation time, the `Integration Manager` (`IM`) for identifying conflicts in updates to shared variables as well as for merging the updated values of shared variables at the end of the timestep, and the `Conflict Resolver` (`CR`), for finding resolutions for flagged conflicts. Apart from that, the infrastructure also has a `Configuration Loader` (`CL`) for parsing the XML configuration and dynamically loading the specified components and shared variables at start-up.

OpenSim is the only framework that supports both concurrent and serial updates to shared resources (Table 1). Our scheme for managing concurrent updates operates *ex post*, by acting only when conflicts actually occur, and rolling back conflicted components to a saved consistent state prior to the step. In contrast, is the *ex ante* approach of avoiding conflicts in the first place [4, 6, 10]. Previously, Wang et al. have done some early investigation on including concurrent updates in HLA [14], however they do not provide details or show that this was achieved. Their subsequent work in [15], does not deal with concurrent updates, but does provide HLA-based infrastructure to ease the task of rolling back components that may have progressed optimistically, in systems where there can be no guarantee that all messages from prior timesteps have been received.

Our earlier work in [10] does provide an architecture to deal with the problem of concurrent update of shared resources, however, it requires significant modifications to the components being integrated. In that work, conflicts are prevented by mandating that components have all usage of shared variables pre-approved by the `CR`. This requires identifying all code within the model that modifies shared variables, and rewriting it in the form of actions which may be approved or failed by the `CR` as a single timestep progresses. The `CR` ensures that no component is ahead of other components in action requests involving shared variables, by more than one request. This allows the `CR` to ensure a modicum of fairness in processing requests, but results in potential idle time as one simulation has to wait for the requests of a potentially conflicting component. This approach also does not address issues of "fairness" when components request resources (e.g. funds) in widely varying amounts per request.

The OpenSim approach, which instead of relying on continuous requests to the `CR` to avoid conflicts, has a `CR` which identifies conflicts at the end of a time step. Then after deciding how this should be resolved, it requests one or more components to re-run the timestep with modified initial parameters. This approach enables integration with much less modification of components than [10], and is more efficient in cases where there is no actual conflict occurring. It also does not result in the anomalies of unnecessary failures, exhibited by [6].

4.1 Configuration

In building an integrated simulation using OpenSim a modeller starts by identifying the shared concepts, or variables, between the simulation components, such as the physical environment, resources, objects, and agents. The simulation components and their shared variables are then specified in a configuration file (XML format). In order for the simulation components to liaise with the OpenSim runtime, each must be first "wrapped" by implementing the interface of Table 2. Then, the combined simulation XML is given to the OpenSim

[17] Practically, concepts will be encapsulated in more complex forms such as data structures and classes, however a discussion based on simple variables will suffice here to highlight the concerns without loss of generality.

runtime infrastructure for execution.

Figure 1 shows a system level view of our example integration for the hospital system, where both the disaster and virus simulation components share the concept of a hospital. The local representation of a hospital system may well differ in each simulation, and for the purpose of combining, some common denominator needs to be identified, here `hospital-beds`. Note that the components do not necessarily need to model beds per se, rather only a general notion of hospital occupancy, that can be converted to the shared notion of `hospital-beds` via functions included in the wrapper implementation.

4.2 Time management

The management and progression of logical time is done in a manner similar to [4, 6, 10] via the `Time Manager` (`TM`). Each simulation component sends a request to the `TM` that then progresses the simulation to the earliest logical time requested. This setup allows for integrating components that operate at different time granularity, and also works for hybrid systems where some components may be time-stepped (these request time be progressed by a fixed duration) and some event-driven (these request time advance by variable duration).

Pipelined execution, where desired, is specified via the configuration file, such that the output of one component is fed to another, in the same timestep.

4.3 Simulation loop

Algorithm 1 shows a simplified version of the procedure for a single simulation step performed by the OpenSim infrastructure. The simulation loop starts with the `TM` collecting all the components that are scheduled to run in the current simulation step, and progressing them via `step` interface calls (lines 3–5). Then, the `IM` obtains the new values of the shared variables, via `get` calls, and checks for conflicts (line 6), i.e., when multiple components make incompatible updates to shared variables, such as if the disaster and virus simulations collectively use more beds than are available. If there are no conflicts, the `IM` updates the components with the final values of the shared variables via `set` interface calls, so that the shared variables are synchronised once again (lines 14–16).

If on the other hand, there are conflicts, the `IM` works with the `CR` to resolve them first (line 9). A resolution basically involves the `IM` resetting the conflicted components to their start state at the beginning of the time step, via `rollback` interface calls (line 10), adjusting the *perceived* values of the shared variables, via `set` calls (line 11), in such a way that the original conflicts cannot occur, and re-stepping the components once more (line 12). If the re-step causes new conflicts, then the `IM` consults the `CR` again to find a resolution (that doesn't undo the previous resolution), then resolves the conflicts in a similar way by rolling back, adjusting the shared variables, and re-stepping yet again. It does this repeatedly until all conflicts are resolved.

4.4 Shared Resources

Shared resources are specified in OpenSim via the XML configuration file. OpenSim differentiates between two kinds of resources: *serially accessible* resources that can only be used by one component at a time, such as a shared ambulance between the disaster and virus simulations, and *concurrently accessible* resources that lend themselves to simultaneous updates in the same time step. In [6], this is referred to as *exclusive* vs *cumulative* use.

Algorithm 1: `OpenSimStep`: the main simulation loop

```
   Data: The random generator seed s
   Result: Performs one simulation step
1  get components scheduled for this step;
2  shuffle the order in which components will be stepped;
3  foreach component do
4      set random seed to s;
5      step;
6  check conflicts using pre- and post-step shared vars' values;
7  if conflicts exist then
8      resolve conflicts by finding suitable re-allocation of values;
9      foreach component do
10         rollback the component;
11         set re-allocated values of shared variables;
12     OpenSimStep();
13 else
14     merge updates and calculate final shared values;
15     foreach component do
16         set final values of shared variables;
```

4.4.1 Serial access

Serial access resources cannot be changed concurrently by multiple components in the same logical time step. For instance in our hospital system example, the `ambulance` is this type of resource. It can only attend to one emergency call at a time. If both the `VS` and `DS` were to try and send the ambulance to different locations in the same time step for instance, then only one of those updates can be allowed. Moreover, the component that is given approval may *lock* exclusive access to the resource for several subsequent time steps in order to fully utilise it. For instance, in our example, it may take several time steps for a dispatched ambulance to arrive at a location and treat an injured person. The responsible component must be allowed exclusive access to the `ambulance` variables for that period. This scheme is not dissimilar to the HLA resource ownership scheme, where all resources are essentially accessed serially in this way.

Exclusive access to resources is achieved in OpenSim via the `getInUse` and `setInUse` interface functions. Component wrappers are responsible for implementing these functions. This requires some understanding of the underlying model logic, and possibly some changes to it, to determine resource availability before its use, and decide what to do when it is unavailable.

4.4.2 Concurrent access

As discussed, if the `IM` has detected an inconsistent state when integrating the results of the different components, the `CR` must determine how the conflict should be resolved. At a basic level, when this inconsistent state has to do with over-use of available resources, then the `CR` must decide how to allocate the resources to the different components. The way in which the `CR` resolves conflicts is configured by the integration engineer via a resolution policy. We support four different resolution policies as follows:

- *Equal allocation*: resolves the over-use by allocating the resource equally amongst all using components. For example, in our integration of the hospital simulations, if the available `hospital-beds`=10, the virus simulation (`VS`) used 6, and disaster simulation (`DS`) 10, then equal distribution would result

in both components seeing `hospital-beds`=5 when the conflicted step is re-run.

- *Proportional allocation*: resolves the over-use by allocating in the same proportion as initial conflicted use. So for the example above, proportional allocation would allocate 6/16 of 10 beds (3.75, rounded to 4) to `VS` and 10/16 (6.25, rounded to 6) to `DS`.
- *Priority allocation*: resolves by allocating the resource in priority order, to the full amount of conflicted use, until depleted. For the above, priority allocation in favour of `VS` would allocate 6 beds to `VS` and the remainder of 4 beds to `DS`. Note that this would require only `DS` to be re-run, as `VS` is allocated what it had used.
- *Custom*: resolves using a custom user-provided function.

There may be multiple conflicts, and it is important that all conflicts involving a component are resolved, and the start state modified appropriately, before the re-run of that component. When re-running a component, the execution will take a different path once the modified resource allocation is encountered. This may generate a new conflict, which will in turn need to be resolved by the same general process. Although this may require multiple iterations, it is guaranteed to terminate, if no component is allowed to be given the same start state more than once. In reality it is only under highly constrained circumstances, with highly interdependent components, that many re-runs may be required. This happens only when exploring limiting scenarios, where the runtime cost is justified as the purpose is to understand well what could happen in such extreme situations.

5 Discussion and Conclusion

This paper introduced our framework, OpenSim, for facilitating the runtime integration of simulation components including agent-based models. It supports a range of scenarios including combining existing models, combining general-purpose frameworks that model aspects of the problem, as well as building new simulations in a modular way from smaller simulations. It is simple to use: building a combined simulation requires implementing model wrappers with basic control and data access functions (Table 2), and specifying the makeup of the combined simulation via an XML configuration file.

OpenSim is the only framework to support concurrent and serial updates to shared resources, which is necessary for integrating agent-based simulations. Conflicts in concurrent updates are managed *ex ante* via a rollback-based mechanism. Additionally, it is a fully featured framework that can be used as a replacement for existing solutions like the HLA and OpenMI (Table 1). Currently, support for shared agents between components is partial, only agent states can be shared via serial/concurrent variables, and work is underway to include shared actions.

We have used, or are using, OpenSim in several projects. These include the integration of the disaster and virus Repast simulations for the hospital system, integration of BDI reasoning with MATSim for a taxi service simulation, a configurable bushfire evacuation simulation combining MATSim, BDI, and Phoenix RapidFire, for Australia, and the mine planning problem where the larger simulation is being built in a modular way as two sub-simulations.

OpenSim is being released under an Open Source license[18], with a Java-based runtime execution. This is the result of several years of work [10, 11, 12, 2, 13] on the central issues inhibiting modular integration including agent-based simulations. We believe that solving these challenges is necessary to enable reuse of a growing global repository of agent-based systems for the study of complex social systems.

[18] Release is planned for mid-2014.

ACKNOWLEDGEMENTS

This work is partly funded by ARC Discovery grant DP1093290, ARC Linkage grant LP130100008, and seed funding from RMIT University and SolveIT Software (now Schneider Electric). We would like to acknowledge the work of RMIT University students Arie Wilsher, Sutina Wipawiwat, and Thomas Wood, for development of the various simulation components used in integration, and help with evaluating existing frameworks.

REFERENCES

[1] Anthony M Castronova, Jonathan L Goodall, and Mehmet B Ercan, 'Integrated modeling within a hydrologic information system: an OpenMI based approach', *Environmental Modelling & Software*, **39**, 263–273, (2013). 2

[2] Qingyu Chen, Arie Wilsher, Dhirendra Singh, and Lin Padgham, 'Adding BDI agents to MATSim traffic simulator (Demonstration)', in *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1637–1638, Paris, France, (May 2014). International Foundation for Autonomous Agents and Multiagent Systems. 3, 6

[3] Judith S Dahmann, Frederick Kuhl, and Richard Weatherly, 'Standards for simulation: as simple as possible but not simpler the High Level Architecture for simulation', *Simulation*, **71**(6), 378–387, (1998). 1

[4] Richard M Fujimoto, 'Time management in the high level architecture', *Simulation*, **71**(6), 388–400, (1998). 4, 5

[5] Ronald C Hofer and Margaret L Loper, 'DIS today [Distributed Interactive Simulation]', *Proceedings of the IEEE*, **83**(8), 1124–1137, (1995). 1

[6] R. Minson and GK Theodoropoulos, 'Distributing RePast agent-based simulations with HLA', *Concurrency and Computation: Practice and Experience*, **20**(10), 1225 – 1256, (2008). 2, 4, 5

[7] T. W. Nicolai, L. Wang, K. Nagel, and P. Waddell, 'Coupling an urban simulation model with a travel model a first sensitivity test', in *Computers in Urban Planning and Urban Management (CUPUM)*, number 11-07, Lake Louise, Canada, (2011). See www.vsp.tu-berlin.de/publications. 2

[8] Lin Padgham, Kai Nagel, Dhirendra Singh, and Qingyu Chen, 'Integrating BDI agents into a MATSim simulation', in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, Prague, Czech Republic, (August 2014). 3

[9] Lin Padgham, David Scerri, Gaya Buddhinath Jayatilleke, and Sarah Hickmott, 'Integrating BDI reasoning into agent based modelling and simulation', in *Winter Simulation Conference (WSC)*, pp. 345–356, Pheonix, Arizona, USA, (December 2011). 2

[10] David Scerri, Alexis Drogoul, Sarah L. Hickmott, and Lin Padgham, 'An architecture for modular distributed simulation with agent-based models', in *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 541–548, (2010). 1, 2, 3, 4, 5, 6

[11] David Scerri, Ferdinand Gouw, Sarah L. Hickmott, Isaac Yehuda, Fabio Zambetta, and Lin Padgham, 'Bushfire BLOCKS: a modular agent-based simulation', in *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1643–1644, (2010). 1, 6

[12] David Scerri, Sarah Hickmott, Karyn Bosomworth, and Lin Padgham, 'Using modular simulation and agent based modelling to explore emergency management scenarios', *Australian Journal of Emergency Management (AJEM)*, **27**, 44–48, (July 2012). 1, 3, 6

[13] Dhirendra Singh and Lin Padgham, 'A rollback conflict solver for integrating agent-based simulations (Extended Abstract)', in *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1399–1400, Paris, France, (May 2014). International Foundation for Autonomous Agents and Multiagent Systems. 1, 2, 3, 6

[14] L. Wang, S.J. Turner, and F. Wang, 'Resolving mutually exclusive interactions in agent based distributed simulations', in *Proceedings of the 36th conference on Winter simulation*, pp. 783–791. Winter Simulation Conference, (2004). 4

[15] Xiaoguang Wang, Stephen John Turner, Malcolm Yoke Hean Low, and Boon Ping Gan, 'Optimistic synchronization in hla-based distributed simulation', *Simulation*, **81**(4), 279–291, (2005). 4