A Rollback Conflict Solver for Integrating Agent-based Simulations

(Extended Abstract)

Dhirendra Singh
RMIT University, Melbourne, Australia
dhirendra.singh@rmit.edu.au

Lin Padgham
RMIT University, Melbourne, Australia
lin.padgham@rmit.edu.au

ABSTRACT

We present a framework we have developed, `OpenSim`, for integrating disparate simulation components, particularly agent-based models (ABMs) that are increasingly being used for modelling complex social systems, into a single simulation. In this paper we focus on the integration of existing ABMs that have been independently developed and validated. The key challenge is to make as few modifications as possible, while still faithfully producing what would be obtained if the models and their interactions had been rebuilt as a single system from scratch. Our proposed mechanism for doing this, improves substantially on previous options.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of simulation—*Combined, Distributed*

Keywords

Simulation Integration, Agent-based, Conflict Resolution

1. INTRODUCTION

Agent based simulations are popular for modelling complex social systems involving human behaviour which cannot readily be abstracted into mathematical formulae. We propose a mechanism for component based integration of existing simulations, which can be difficult due to the complex ways in which components interact. Examples of the kinds of integrations we are interested in include coupling MATSim (a traffic flow simulator) with UrbanSim (a simulation system for urban planning and development) in [4], and the Phoenix fire simulator with MATSim in [6].

While some frameworks to support integration do exist, such as the High Level Architecture (HLA) used in defence simulations [2] and the Open Modeling Interface (OpenMI) [1] used in the domain of water management, these do not provide adequate support for managing conflicts that can arise when combining existing ABMs that, inevitably, individually represent and update aspects of the environment that are now essentially "shared" in the contect of a global simulation. In contrast to our previous approach of [5], rather than have a conflict resolver manage interactions *ex ante* on a continuous basis within a single timestep to avoid such update conflicts, we now detect conflicts *ex post* at the end of a timestep, and the conflict resolver then directs some conflicting components to re-run their current time step under changed conditions. This approach has the advantage of requiring less change to component code than previously in [5], while at the same time addressing some identified problems inherent in that mechanism.

[1] `http://www.openmi.org`

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

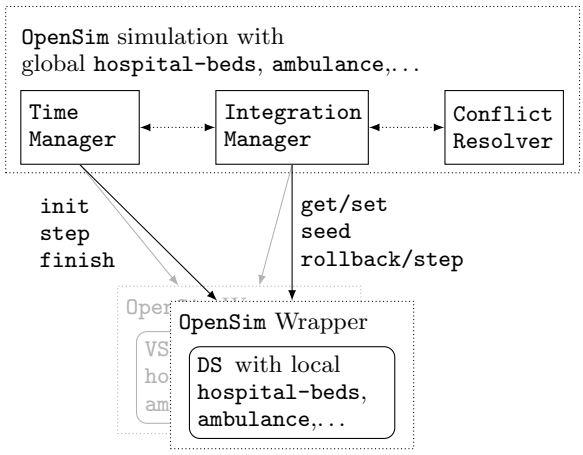

Figure 1: Architecture of an `OpenSim` simulation, here showing the example integration of `DS` and `VS`.

2. OPENSIM FRAMEWORK

Our framework, `OpenSim`, provides an *interface* that allows concepts within individual components to be linked together (via shared variables), as well as *infrastructure* for merging the values of these shared variables at each timestep (via an `Integration Manager` (`IM`)), resolving conflicts (via a `Conflict Resolver` (`CR`)) from incompatible updates to shared variables, and progressing the simulation time (via a `Time Manager` (`TM`)). The process of integration using `OpenSim` basically involves writing component *wrappers* that implement the interface functions of Table 1. The progression of logical time is managed similarly to [1, 5] via the `TM`. Each simulation component sends a request to the `TM` that then progresses the simulation to the earliest logical time requested.

Figure 1 shows the architecture of a global simulation composed of an existing disaster management simulation (`DS`) and virus simulation (`VS`), to explore the impact of two simultaneous stressors - a natural disaster and a disease epidemic - on a shared hospital system. Here `hospital-beds`

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 granularities. 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 convertors 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 1: `OpenSim` component model wrapper interface functions

and `ambulance` are shared variables common to both the `DS` and `VS`, and are linked together via the `OpenSim` wrappers. It is important for us, like in [5], that both modules have the ability to modify the shared variables (`hospital-beds` and `public-funds`) in any given timestep. As explained in detail in [5], the approach of HLA, which requires ownership by a single component for a whole timestep is unsatisfactory.

The simulation progresses from one logical time to the next, controlled by the `TM` via `step` interface calls to the components. At the end of a step, the `IM` compares the pre and post values of the shared variables, via `get` calls, and checks for update conflicts, such as if the `DS` and `VS` both updated the `hospital-beds`in a way that when their changes were combined the number of remaining `hospital-beds`was less than zero. If there are no conflicts, it synchronises the components with the final values of the shared variables, via `set` interface calls, and informs the `TM` that the simulation can now be progressed to the next logical time. If on the other hand, there are conflicts, it works with the `CR` to resolve them first. A resolution basically involves the `IM` resetting the conflicted components to their initial state at the beginning of the time step, via `rollback` interface calls, adjusting the *perceived* values of the shared variables, via `set` calls, in such a way that the original conflicts cannot occur, and re-stepping the components once more. 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.

Conflicts over shared variables are treated differently based on whether the resource can only be updated by one component at a time (*serially accessible*), or whether simultaneuos writes are allowed (*concurrently accessible*). This is essentially what [3] refers to as *exclusive* vs *cumulative* use. Serial access to resources is achieved in `OpenSim` via the `getInUse` and `setInUse` component interface functions. Implementing serial access however does require some changes to the internal logic of the components as they must now explicitly check the status of variables and decide what do if they are unavailable. Serially accessible variables do not get update conflicts. In HLA, all shared resources are accessed serially.

Conflicts in concurrently accessible resources arise from over-use. The `CR` resolves conflicts using one of four user-configured resolution policies: *Equal allocation* resolves the over-use by allocating the resource equally amongst all using components; *Proportional allocation* allocates in the same proportion as initial conflicted use; *Priority allocation* resolves by allocating the resource in priority order, to the full amount of conflicted use, until depleted; *Custom*: uses a custom user-provided function.

When conceptually the "same" individual agent is represented in each component, the state of the agent can basically be captured as a combination of serially and concurrently accessible shared variables. Additionally, we must consider what actions they are doing in each component, and whether or not these are compatible. For example it should not be possible for a doctor agent to simultaneously be treating a virus patient at the hospital in the `VS`, and going with the ambulance to the disaster scene in the `DS`. We are yet to extend `OpenSim` for specifying, recognising and resolving these types of conflicts.

3. REFERENCES

[1] R. M. Fujimoto. Time management in the high level architecture. *Simulation*, 71(6):388–400, 1998.

[2] IEEE 1516 (Standard for Modelling and Simulation High Level Architecture Framework and Rules) , 2000.

[3] R. Minson and G. Theodoropoulos. Distributing RePast agent-based simulations with HLA. *Concurrency and Computation: Practice and Experience*, 20(10):1225 – 1256, 2008.

[4] T. W. Nicolai, K. N. L. Wang, and P. Waddell. Coupling an urban simulation model with a travel model - a first sensitivity test. In *Computers in Urban Planning and Urban Mgmt*, number 11-07, 2011.

[5] D. Scerri, A. Drogoul, S. L. Hickmott, and L. Padgham. An architecture for modular distributed simulation with agent-based models. In *Proc. of Autonomous Agents and Multi-Agent Systems*, pages 541–548, 2010.

[6] D. Scerri, S. Hickmott, K. Bosomworth, and L. Padgham. Using modular simulation and agent based modelling to explore emergency management scenarios. *Australian Journal of Emergency Management (AJEM)*, 27:44–48, July 2012.