Interactive Environment to Support Agent-Based Graph Programming in MASS Java

Daniel Blashaw

Autumn 2020: CSS 595 - Term Paper

Contents

1 Introduction

The Multi-Agent Spatial Simulation (MASS) library has been developed with the intent to provide an intuitive infrastructure for researchers of all disciplines and computing experience to create Agent-Based Model (ABM) simulations that execute in a parallel fashion using multiple computing nodes. In ABMs, agents are given specific rules which guide the agents' interaction with the simulation space, considered as "places" in MASS, as well as with other agents in the system during program execution. These ABMs are then useful wherever researchers would like to observe the dynamic and collective effect of the agents on the simulation space, such as in social sciences, bioinformatics, traffic simulations, and economics. However, ABMs can be challenging to conceptualize for those just starting out, and the non-deterministic nature of most simulations magnifies this difficulty. To that end, the work described in this document seeks to create an interactive environment that helps lessen the learning curve when building or modifying graph-based simulations.

This work is to build upon the foundations laid by Justin Gilroy and Nasser Alghamdi, two recent graduate students, to develop a new way for users to code, debug, and execute their graph-based simulations. Specifically, the motivation of this work is to give developers a tool which allows them to step through simulation execution line-by-line and visually inspect the movement of agents over

the graph structure. Further, because the new visualization tool will be executed on a single machine and not on a cluster, we must consider solutions on how to visualize a graph that may be too large to fit on a single computing node.

2 Background

2.1 Interactive MASS

Interactive MASS (InMASS) was developed during Nasser Alghamdi's graduate study and enabled MASS simulations to run through Java's JShell interface. JShell behaves as a Read-Evaluate-Print Loop (REPL) that allows Java code to be entered from a command-line interface and be executed line-by-line, like Python. Further, Nasser built several functions that leverage this line-by-line evaluation to allow the user to checkpoint a running application, test new code, and rollback to the checkpointed state without the need to halt execution and then recompile, distribute, and reinitialize the program on the cluster [3]. This functionality has fundamentally changed the way developers can interact with MASS, and the features are going to play a central role in my work to create an interactive tool for graph simulations.

2.2 MASS with Graphs

Justin Gilroy's research yielded the necessary infrastructure for MASS Java to support graph structures without the need for additional graph code from the programmer. This was accomplished by enabling: the construction and maintenance of graph vertices and edges with new API calls; the import of graphs from HIPPIE [7] and MATSim [6] file types; and the integration of third-party bioinformatics software, Cytoscape, for graph visualization [2]. This work eliminated the need for custom graph classes and provided the foundations of the MASS-Cytoscape interface by adding the first two Cytoscape plugins: *import-network* and *export-network*.

2.3 Cytoscape

Cytoscape is an open-source platform originally designed for visualization of biomolecular interaction networks that has been adopted by a wide variety of disciplines for general graph visualization [4]. DSLabs has chosen to interface Cytoscape with the MASS Java library for graph support because of Cytoscape's extensive file support for importing graph structures as well as its ability to handle dynamic manipulation of an existing graph [1]. Importantly, Cytoscape is also based on the OSGi framework which introduces some additional things to consider when designing the new application for MASS.

Open Service Gateway Initiative (OSGi) is a Java-based component system in which each component, also referred to as a "bundle", is a self-contained piece of functionality that can be started, stopped, and removed without impacting other bundles [8]. This modularity allows for each bundle to be maintained independently from other bundles in the OSGi container. The OSGi container then is the instance of the OSGi framework that provides an environment in which the bundles operate. In this case, the OSGi container is included with the Cytoscape application. In the event a bundle needs to access functionality of another registered bundle, then it may do so by obtaining a reference to the other bundle through the OSGi service API.

3 Progress

As mentioned in the introduction, the objective of this research is to develop a new interactive tool that enables the visualization and incremental inspection of graph-based simulations. Toward this objective, there are five main deliverables within the scope of this research:

1. Extend MASS to natively support tracking and reporting of agent data.
2. Extend MASS-Cytoscape integration to include flow of agent data to Cytoscape.
3. Create unified control panel that enables visualization of agent data in Cytoscape.
4. Integrate InMASS into the Cytoscape control panel to facilitate simulation control.
5. Modify MASS-Cytoscape communication to allow for query and visualization of subgraphs.

In this document, we will review the progress toward the first three deliverables, with the latter two items being the focus of future quarters.

3.1 Agent History (Refactored)

During my independent study quarter, in Spring 2020, I began the work to add agent tracking into MASS by adding functionality to *register* and *delist* agents from the tracking feature as well as the ability to *get* the results back to the main program. This implementation met the specified requirements but did so in some nonobvious ways that made the implementation difficult to extend and the data awkward to consume. For these reasons, my initial work this quarter was to refactor this feature into something more manageable and user friendly.

The most significant refactoring is the logic for recording and reporting the agent history information has been encapsulated into the *AgentHistory* class. Each *Place* instance maintains its own *AgentHistory* object that is called each time an agent visits the place. Then, when there is a call to *get* the agent history information, each *Place* returns its local *AgentHistory* instance and those are then merged by the MASS host computing node into a single all-inclusive model that gets returned to the user.

Further, the data structure for agent history information is now encapsulated into the *AgentHistoryModel* class. Each *AgentHistory* instance maintains a Hashtable of *AgentHistoryModel*s, and each *AgentHistoryModel* represents the data of a single agent including the *agent name*, whether the agent is *alive* and/or *successful*, and the *full visitation history* for the agent. (At this time, the *alive* and *successful* fields are just placeholders for future extension as there is currently no way for the simulation program to modify these values.)

See **Appendix I** for code examples of new and updated Agent History features.

3.2 Cytoscape Services

Figure 3.2 illustrates the current data flow in the MASS-Cytoscape integration. The Cytoscape side of the graphic shows all details regarding the bundles and the tables they manipulate, whereas the MASS side has been abridged to only show the more immediate parts of the interaction with the more core aspects of MASS have been lumped into the "MASS Communication Layer."

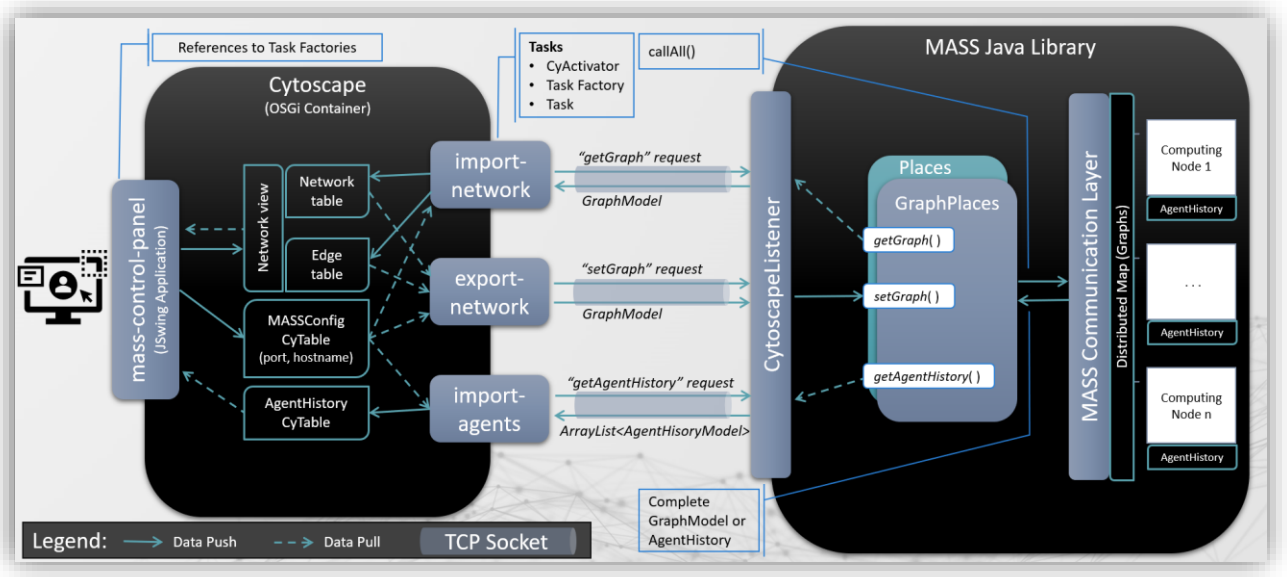


Fig. 3.2 - MASS-Cytoscape Data Flow Diagram

The current architecture seeks to make use of the OSGi framework by encapsulating each piece of Cytoscape functionality in its own bundle and allowing the bundles to "communicate" with one another by manipulating the data in the local CyNetworks and CyTables[1]. Additionally, this improves extensibility of this work by allowing future developers of MASS to reuse the service bundles in their own applications.

There are currently three developed services (*import-network, export-network,* and *import-agents*) that handle all communication with the MASS cluster. These bundles are comprised of three main classes: the *CyActivator* class, which is responsible for starting the bundle and securing references to other required OSGi services; the *TaskFactory* class, which is responsible for creating tasks and is the part of the bundle registered for invocation by other components; and, the *Task* class, which contains all the logic required to carry out the bundle's function.

When invoked, each of these services communicates with the CytoscapeListener class in MASS through a basic networking socket by first sending one of three supported calls: *getGraph*, *setGraph*, or *getAgentHistory*. The CytoscapeListener then handles these requests by calling the associated functions in the *GraphPlaces* class (an extension of the *Places* class) to either retrieve or set the graph and agent data in MASS.

In addition to graph data retrieval, the *import-network* service has the very important function of creating the "Network View" responsible for determining how the network is displayed in the Cytoscape GUI. This network view can assign formatting for all node or edges by setting default formatting, but, more importantly, can also map columns from the node and edge tables to specific visualization attributes so each node or edge is formatted independently. For example, the current implementation has a "edge_width_value" column in the edge table that the network view uses to inform the visualization of how thick each corresponding edge should be. Importantly, when the

[1] CyNetwork and CyTable are the basic data structures available in Cytoscape. CyTables are simple standalone data structures with rows and columns like what you would find in Microsoft Excel. Whereas CyNetworks have reference to other tables that are specifically for Nodes, Edges, and visualization settings.

network view is mapped to columns of data, the visualization will dynamically adjust if the values of the mapped columns should change.

3.3 MASS Control Panel

The Mass Control Panel is another bundle but one that simply starts a JSwing application panel and then waits for user input. This panel is designed to provide a unified experience for the user to in Cytoscape by employing the other service bundles to exchange data with MASS and then using the retrieved data to inform the UI. To update the visualizations, the MASS Control Panel can modify the columns that are directly mapped to attributes in the network view.

A benefit of this architecture is that all data is stored in the Cytoscape client so the user can inspect the status of the simulation without unnecessary and costly network communication. Additionally, when the simulation progresses and new data is needed, the MASS Control Panel user can selectively refresh their data in Cytoscape, whether that is graph data, agent data, or both.

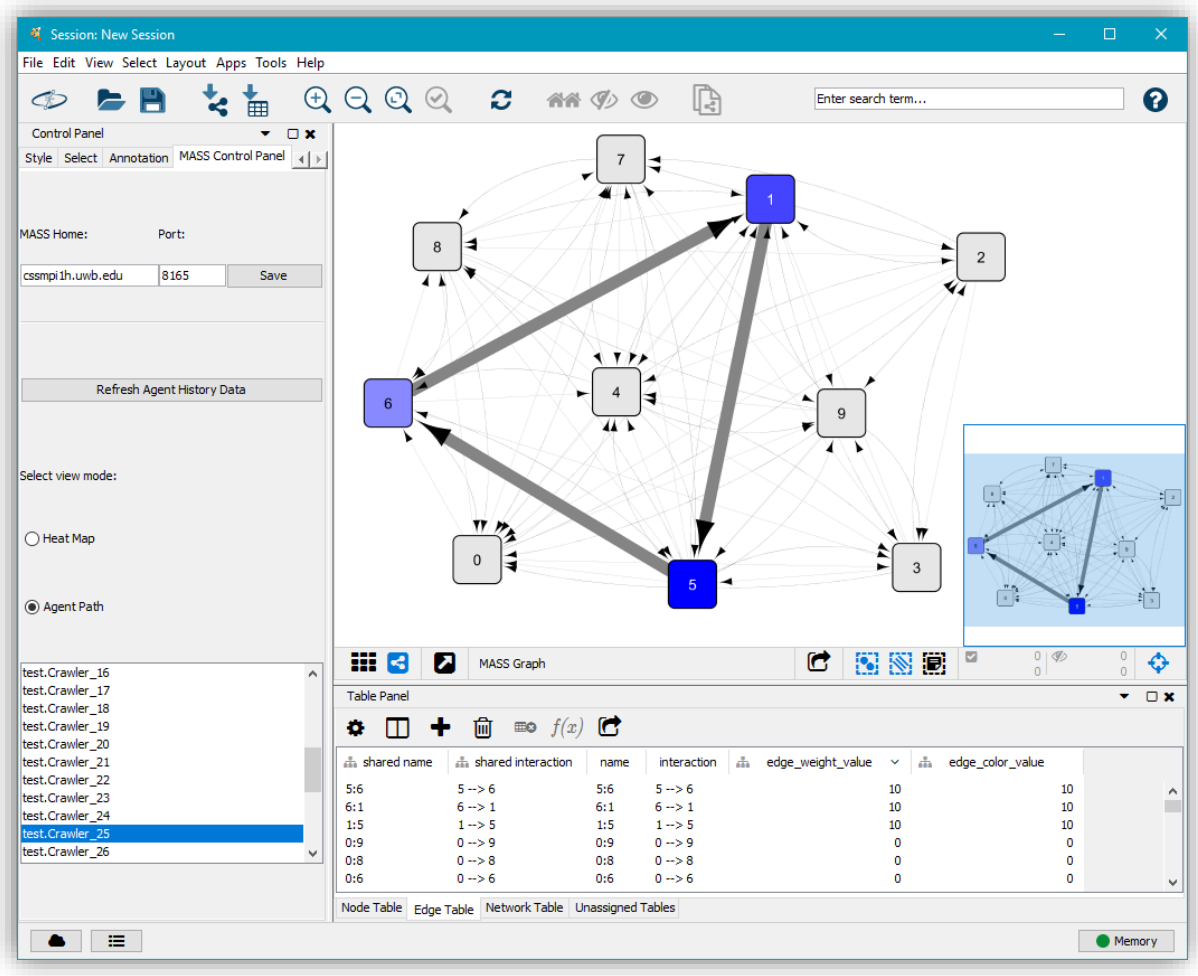


Fig. 3.3 – MASS Control Panel displaying Agent Path visualization

There are currently two visualizations available, *Agent Path* (shown in **Figure 3.3**) and *Agent Heat Map.* When selecting either of these visualizations in the MASS Control Panel, the application will first retrieve the corresponding agent information and then use that data to update the fields

mapped to the network view. In **Figure 3.3**, for example, we can see the edge_weight_value and edge_color_values are set to 0 for all edges except those that are on the selected agent's path. Further, in the *Agent Path* visualization shown in **Figure 3.3,** higher values in these mapped fields translates to either darker vertices or thicker edges such that vertices visited more recently are darker and edges traversed more frequently are thicker.

At this point, both visualizations are achieved using just the edge weight and node color attributes. That said, there are other options available in the network view that could be used to further refine the user's experience in Cytoscape. For example, in the Heat Map visualization, we could use tooltips for displaying a list of agents that are currently on a given node or, similarly, we could update the node labels to show the count of agents next to the node name (e.g., "Node 5 [12 agents]").

See **Appendix II** for additional screenshots of MASS Control Panel and visualizations.

3.4 MASS Configuration

Due to the self-contained nature of OSGi bundles, one obstacle that needed to be addressed was how to update MASS host and port information across all the service bundles without the need to modify code. Pushing the data directly from the MASS Control Panel would not work because that would require passing parameters to the task factory, which goes against OSGi's core philosophies. Instead, the host and port information needed to be pushed to a common location, the "MASS_Configuration" CyTable, and then retrieved or modified by the various bundles, via the *MASSConfig* class.

On instantiation, the *MASSConfig* class tries to load current settings from the MASS_Configuration table. If the table does not exist, then it will be created with default values. Each task created by the service bundles begins by creating a new MASSConfig object to determine where to send the *get* or *set* request. Similarly, the MASS Control Panel uses the same MASSConfig object to update the MASS_Configuration table whenever there is new input from the user.

At this time only host and port information are included in the *MASS_Configuration* table, although the mechanism of shared information between bundles will be useful in future work. Particularly, when information may be needed from the user to determine how MASS responds to the request, as is the case in partial-graph streaming.

4 Future Work

The following items remain as gaps in the current implementation. Many features discussed in this paper are still in active development, so some of the items discussed below will be addressed prior to completion of my research. Others, however, will be outside of my current scope and instead represent opportunities for future extension by others.

1. A **Cytoscape Table Listener** is needed in the MASS Control Panel plugin to clean up the user experience, especially when refreshing Agent data. When data is refreshed, a task is created by the appropriate task factory and then the task is completed parallel to the rest of the program. This is great for user experience because it allows the user to maintain control of the program

while the task runs; however, this also means the data may not be available to the MASS Control Panel when needed.

For example, the MASS Control Panel would like to update the Agent Selection List whenever new agent data is retrieved from MASS. To do this, the MASS Control Panel first creates a new import agent task and then retrieves the current agent data from the local AgentHistory CyTable to update the Agent List. Unfortunately, because the import agent task processes asynchronously, the data in the local CyTable is usually not updated in time and, therefore, the Agent Selection List is repopulated with old information. To remedy this, there is a pop-up window that serves to delay the execution of the MASS Control Panel long enough for the agent data to refresh before to updating the Agent Selection List. Ideally, there would be a thread checking for changes to the AgentHistory CyTable that would then push new data to the Agent Selection List immediately following the table update.

2. **Network data table addressing** is currently inconsistent when the network data is refreshed. Specifically, sometimes when the network data is refreshed, the agent visualization code loses track of the appropriate network tables. This can currently be remedied by refreshing the network data again but is certainly something that will need to be addressed prior to application release.

3. **Agent *Alive* and *Successful* attributes** have been added to the AgentHistoryModel class as well as in the Cytoscape data tables, but further work is needed to enable manipulation of these attributes from the simulation program in MASS. The implementation of these features would likely be a modification of the agent death and migration calls. Ideally, the simulation program would provide a logical set of parameters which determine agent successfulness so agents that meet those criteria could be marked automatically.

4. **Agent List filtering** in the MASS Control Panel would be a helpful addition, especially once the *Alive* and *Successful* attributes have been implemented into MASS. This feature would allow the user to down select large lists of agents into only those that are the most interesting for review.

5. **Automatic edge creation** needs to be either refined or removed. This feature was added for debugging so that if an agent reports to have traversed from node to node on a non-existent edge, then the reported edge will be added to the visualization. This was very helpful when programming the agent path visualization because the graph and agent datasets used for debugging are generated without reference to one another, making non-existent edges very common. On the other hand, if a user is trying to debug their simulation, and agents are traversing over non-existent edges, then it is fair to assume they would like at least some indication of the missing edge.

   One potential solution is to mark the added edge to be visually different from the others, for example with a unique color or a dashed line. Even then, the user may want the option of turning off what might be considered visual clutter.

In addition to the above, my work in upcoming quarters will address the two remaining goals of my work. First, I plan to update MASS-Cytoscape communication procedure to allow for subgraph streaming based on an *n*-neighbors approach, in which the user selects a central node and then MASS returns only the nodes and edges within *n*-degrees of separation. Finally, I will work to integrate InMASS which will allow the programmer fine-grain control over the execution of their simulation from the MASS Control Panel Cytoscape extension.

5 References

[1] University of Washington. MASS Java Manual, 2016. https://depts.washington.edu/dslab/MASS/docs/MASS%20Java%20Technical%20Manual.pdf

[2] Gilroy, Justin, "Dynamic Graph Construction and Maintenance," (Master's paper). University of Washington, Bothell, WA, 2020.

[3] Alghamdi, Nasser, "Supporting Interactive Computing Features for MASS Library: Rollback and Monitoring System," (Master's paper). University of Washington, Bothell, WA, 2020.

[4] Ono, K. (2018). Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization. Retrieved July 27, 2020, from https://cytoscape.org/

[5] Fukuda, M. (2020). Distributed Systems Laboratory (DSL). Retrieved July 30, 2020, from http://depts.washington.edu/dslab

[6] Horni, A., Nagel, K. and Axhausen, K.W. (eds.) 2016 The Multi-Agent Transport Simulation MATSim. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw. License: CC-BY 4.0

[7] "HIPPIE: Human Integrated Protein-Protein Interaction rEference" Cbdm01.zdv.unimainz. de, 2020. [Online]. Available: http://cbdm-01.zdv.unimainz. de/~mschaefer/hippie/hippie_current.txt. [Accessed: 29- July- 2020].

[8] Baeldung. "Introduction to OSGi." 14 Aug. 2019. Web. 17 Dec. 2020.

6 Appendix

Appendix I: Code Examples of Agent History refactoring

New Feature – Registering an entire class of agents to be tracked.
(Previously done iteratively by send one agent at a time or by sending an array of agents.)

```
places.callAll( places.AGENT_TRACE_REGISTER_CLASS, ( Object ) Agent.class.getName());
```

Updated – Extracting agent history information from the *AgentHistoryModel*.

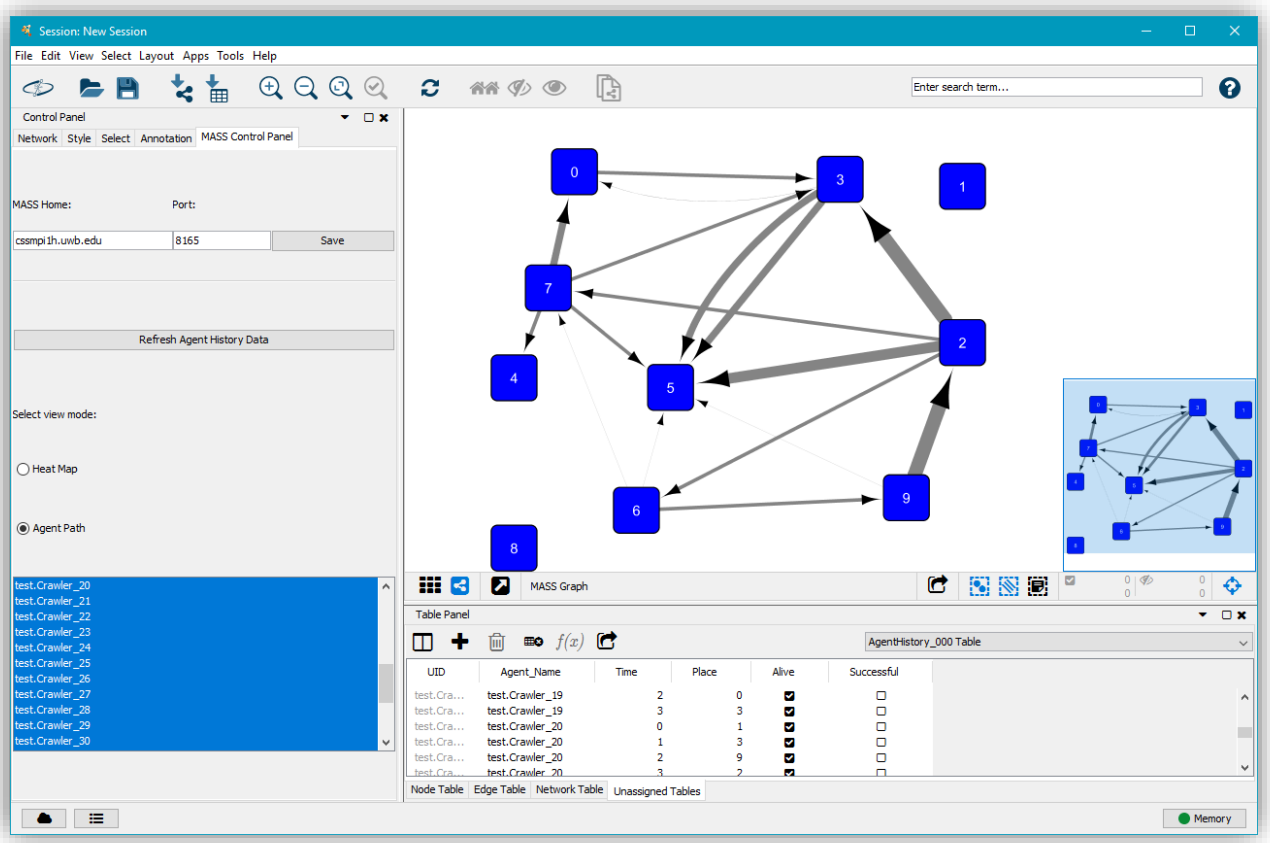
(1) This is a standard places.callAll()… note that the agentsCallAllObjs is just an empty array.
(2) Full AgentHistory information is returned at index 0 of the agentTraceResults array.
(3) Iteration through AgentHistoryModel and examples of available functions.

```
(1)  Object[] agentsCallAllObjs = new Object[ x * y ];
     Object[] agentTraceResults =
         ( Object[] ) places.callAll( places.AGENT_TRACE_GET, agentsCallAllObjs);

(2)  Hashtable<String, AgentHistoryModel> results =
         (Hashtable<String, AgentHistoryModel>) agentTraceResults[0];

(3)  Set<String> keys = results.keySet();
     for (String key : keys) {
        AgentHistoryModel myModel = results.get(key);
        for (AgentHistoryModel.Visit visit : myModel.getHistory()) {
           String name = myModel.getName();
           myModel.isAlive();
           myModel.isSuccessful();
           int thisTime = (int) visit.time;
           int thisPlace = (int) visit.placeLinearIndex;
           // do something...
        }
     }
```

Appendix II: Screenshots of Cytoscape – MASS Control Panel

Agent Path with Multiple Selection:

- Shows full history of selected agents
- Node color by recency of visit (darker = more recent)
- Edge width by popularity of route (thicker = more used)



Agent Heat Map:

- Shows density of agents at current positions
- Node color by count number of agents on node (darker = more agents)