

# A MULTI-MODEL DOCKING EXPERIMENT OF DYNAMIC SOCIAL NETWORK SIMULATIONS

Jin Xu Yongqin Gao Jeffrey Goett Gregory Madey

Dept. of Comp. Science  
University of Notre Dame  
Notre Dame, IN 46556

Email: {jxu1, ygao1, jgoett, gmadey}@nd.edu

September 2, 2003

## ABSTRACT

Axtell, Axelrod, Epstein and Cohen [26] describe a docking or alignment process and experiment for verifying simulations. By comparing simulations built independently using different simulation tools, the docking or alignment process may discover bugs, misinterpretation of model specification, and inherent differences in toolkit implementations. If the behavior of the multiple simulations are similar, then verification confidence is increased. North and Macal [24] reported on such experiment using Mathematica, Swarm and RePast to simulation the Beer Distribution Game (originally simulated using system dynamics simulation methods).

In this paper, we present the results of docking a Repast simulation and a Java/Swarm simulation of four social network models of the Open Source Software (OSS) community. Data about the SourceForge OSS developer site has been collected for over 2 years. Developer membership in projects is used to model the social network of developers. Social networks based on random graphs, preferential attachment, preference attachment with constant fitness, and preferential attachment with dynamic fitness are modeled and compared to collected data. We describe how properties of social networks such as degree distribution, diameter and clustering coefficient are used to dock Repast and Swarm simulations of four social network models. The simulations grow “artificial societies” representing the SourceForge developer/project community. As a by-product of the docking experiment, we provide observations on the advantages and disadvantages of the two toolkits for modeling such systems.

**Keywords:** dynamic social network; docking; agent-based modeling; open source software

## INTRODUCTION

Agent-based Modeling has become an attractive computational methodology in recent years. Its popularity results from the fact that it allows for complex systems to be simulated in a relatively

straightforward way. Unlike traditional mathematical simulation tools, agent-based modeling simulates artificial worlds based on components called agents and defines rules to determine the interactions of agents. Although agent-based modeling is used commonly in simulations, it is not guaranteed to provide an accurate representation of a particular empirical application [3]. In this context, Axtell et al claimed “It seems fundamental to us to be able to determine whether two models claiming to deal with the same phenomenon can, or cannot, produce the same result” [26].

There are three ways to validate an agent-based simulation. The first way is to compare the simulation output with the real phenomenon. This way is relatively simple and straightforward. However, often we cannot get complete real data on all aspect of the phenomenon. The second way compares agent-based simulation results with results of mathematical models. The disadvantage of this way is that we need to construct mathematical models which may be difficult to formulate for a complex system. The third way is by docking with other simulations of the same phenomenon. Docking is the process of aligning two dissimilar models to address the same question or problem, to investigate their similarities and their differences, but most importantly, to gain new understanding of the question or issue [4].

Axtell, Axelrod, Epstein and Cohen describe a docking or alignment process and experiment for verifying simulations [26]. By comparing simulations built independently using different simulation tools, the docking or alignment process may discover bugs, misinterpretations of model specification, and inherent differences in toolkit implementations. If the behaviors of the multiple simulations are similar, then validation confidence is increased. North and Macal reported on such an experiment using Mathematica, Swarm and RePast to simulate the Beer Distribution Game (originally simulated using system dynamics simulation methods) [24]. In Louie and Ashworth [2], docking is done by comparing results of the canonical Garbage Can model with those of “NK Model”. Xu and Gao used Repast and Swarm to dock a random network model of the Open Source Software phenomenon [17]. Although the above docking experiments show the importance and advantages of docking, there are only a few docking studies and none have used topological properties of social networks as docking parameters.

In this paper, we present the results of docking a Repast simulation and a Java/Swarm simulation of four dynamic social network models of the Open Source Software (OSS) community. The results reported in this paper are part of a study of the Open Source Software [8–15, 17]. Data about the SourceForge OSS developer site has been collected for over 2 years. Developer membership in projects is used to model the social network of developers. Social networks based on random graphs, preferential attachment, preferential attachment with constant fitness, and preferential attachment with dynamic fitness are modeled and compared to collected data. We describe how properties of social networks such as degree distribution, diameter and clustering coefficient are used to dock Repast and Swarm simulations of four social networks. The simulations grow “artificial societies” representing the SourceForge developer/project community. As a by-product of the docking experiment, we provide observations on the advantages and disadvantages of the two toolkits for modeling such systems.

The rest of paper is organized as follows. The second section provides background on our Open Source Software (OSS) study and simulation. The following section discusses docking simulations using Swarm and Repast. Experimental results and comparisons are given in the fourth section. The fifth section presents conclusions and discussions.

## SOCIAL NETWORK MODEL

Social network theory is a conceptual framework through which we view the OSS developer movement. The theory, built on mathematical graph theory, depicts interrelated social agents as nodes or vertices of a graph and their relationships as links or edges drawn between the nodes [28]. The number of edges (or links) connected to a node (or vertex) is called the index or degree of the node.

Of special interest are the evolutionary processes and associated topological formation in dynamic growing networks. Early work in this field by Erdos and Renyi focuses on random graphs, i.e., those where edges between vertices were attached in a random process (called ER graphs here) [1]. However, the distributions of index values for the random graphs do not agree with the observed power law distribution for many social networks, including the OSS developer network at SourceForge. Some other evolutionary mechanisms include: 1) the Watts-Strogatz (WS) model [29], 2) the Barabasi-Albert (BA) model with preferential attachment [19, 20, 25], 3) the modified BA model with fitness [1, 18], and 4) an extension of the BA model (with fitness) to include dynamic fitness based on project life cycle reported in [11, 13–15]. The WS model captures the local clustering property of social networks and was extended to include some random reattachment to capture the small world property, but failed to display the power-law distribution of index values. The BA model added preferential attachment, both preserving the realistic properties of the WS model and also displaying the power-law distribution. The BA model was extended with the addition of random fitness to capture the fact that sometimes newly added nodes grow edges faster than previously added nodes (the “young upstart” phenomenon).

The Open Source Software (OSS) development movement is a classic example of a dynamic social network; it is also a prototype of a complex evolving network. Prior research suggests that the OSS network can be considered a complex, self-organizing system [1, 7, 16]. These systems are typically comprised of large numbers of locally interacting elements.

The Open Source Software community can be described as a dynamic social network. In our model of the OSS collaboration network, there are two entities – developer and project. The network can be illustrated as a graph. In this network, nodes are developers. An edge will be added if two developers are participating in the same project. Edges can be removed if two developers are no longer participating on the same project. The study of the OSS collaboration network can help us understand the evolution of the social network’s topology, the development patterns of each individual object and the impact of the interaction among objects to the evolution of the overall network system.

We use agent-based modeling to simulate the OSS development community. Unlike developers, projects are passive elements of the social network. Thus, we only define developers as the agents which encapsulate a real developer’s possible daily interactions with the development network. Our simulation is time stepped instead of event driven, with one day of real time as a time step. Each day, a certain number of new developers are created. Newly created developers use decision rules to create new projects or join other projects. Also, each day existing developers can decide to abandon a randomly selected project, to continue their current projects, or to create a new project. A developer’s selection is determined by a Java method based on the relative parameter and the degree of the developer.

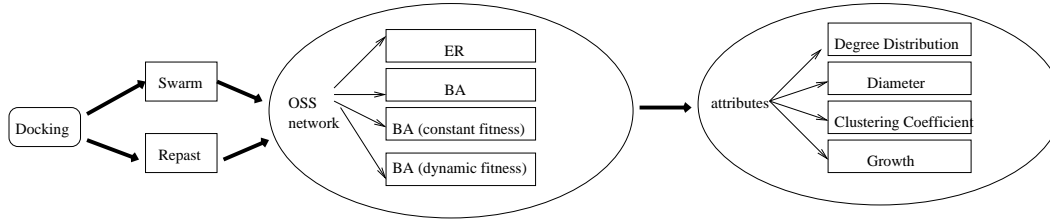


Figure 1: Docking Process

## DOCKING OSS COLLABORATION NETWORK SIMULATION

In this section, we describe the docking of our OSS collaboration network simulation by two agent-based modeling tools – Java Swarm and Repast. Simulation details are compared between these two models.

### The Docking Process

The docking process is an important stage of the OSS project [8]. The initial simulation was written using Swarm. There are several reasons why docking is necessary in this project. First, docking is used to test the correctness of the Swarm implementation. Second, docking provides the Repast version of the OSS simulation which we would like to use in our future research. Repast has several advantages in this project: it is written in pure Java which makes debugging easier; it provides us a graphical representation of the network layout; and most importantly, Repast 2.0 provides distributed running environment [5].

As shown in Figure 1, Swarm simulations and Repast simulations are docked for four models of the OSS network. Our docking process began when the author of the swarm simulation wrote the docking specification. Then, the Repast version was written based on the docking specification. Simulations are validated by comparing network attributes generated by running these two simulation models.

### Swarm

Swarm is a software package for multi-agent simulation of complex systems, originally developed at the Santa Fe Institute [21]. In the Swarm model, the basic unit is called an agent. Modelers can define a set of rules to describe the interaction of agents. Furthermore, Swarm also provides display, control and analysis tools.

Our swarm simulation has a hierarchical structure which consists of a *developer* class, a *modelswarm* class, an *observerswarm* class and a *main* program. The *modelswarm* handles creating developers and controls the activities of developers. In *modelswarm*, a schedule is generated to define a set of activities of the agents. The *observerswarm* is used to implement data collection and draw graphs. The *main* program is a driver to start the whole simulation.

The core of a swarm simulation consists of a group of agents. Agents in our simulation are developers. Each developer is an instance of a Java class. A developer has an identification id,

a degree which is the number of links, and a list of projects participated by this developer. Furthermore, a developer class has methods to describe possible daily actions: create, join, abandon a project or continue the developer's current collaborations. A separate Java method models each of the first three possibilities. A fourth method encapsulates a developer's selection of one of the three alternatives. Here, three model parameters appear. Each represents the probability of one of the three developer activities. Comparison of a randomly generated number to these probabilities determine which behavioral method the agent will enact.

## Repast

RePast is a software framework for agent-based simulation created by Social Science Research Computing at the University of Chicago [27]. Like Swarm, RePast provides an integrated library of classes for creating, running, displaying, and collecting data from an agent-based simulation [6]. In addition, RePast is written in pure Java which has better portability and extensibility than Swarm. Furthermore, RePast provides some different library packages which provide features such as network display, QuickTime movies and snapshot.

Our RePast simulation of OSS developer network consists of a *model* class, a *developer* class, an *edge* class and a *project* class. The class structure of the simulation is different from that of the Swarm simulation. This is due in part to the graphical network display feature of Repast. The model class is responsible for creation and control of the activities of developers. Furthermore, information collection and display are also encapsulated in the *model* class. The *developer* class is similar to that in Swarm simulation. An *edge* class is used to define an edge in OSS network. We also create a *project* class with properties and methods to simulate a project.

## EXPERIMENTAL RESULTS

This section describes docking of Repast and Swarm simulations on four OSS network models – ER, BA, BA with constant fitness and BA with dynamic fitness, and then presents the results and comparisons.

### Docking Procedure

Our docking process sought to verify our Repast migration against the original Swarm simulation. To do so, the process began with a comparison of degree distribution between corresponding models. Upon finding differences, we compared each developer's actions.

The first attempt at docking compared the degree distributions between these two simulations. The Swarm simulation used its built in random number generator. The Repast simulation used the COLT random number generator from the European Laboratory for Particle Physics (CERN). From a graphical comparison of degree distribution for projects and developers over multiple runs of Swarm and Repast, we observed systematic differences between the two simulations' outputted data. Over one subdomain of the developer degree distribution, Swarm had a higher count than Repast. Over another subdomain, Swarm had a lower count. The next step in the docking process determined that the random number generators did not cause this systematic difference. We ran

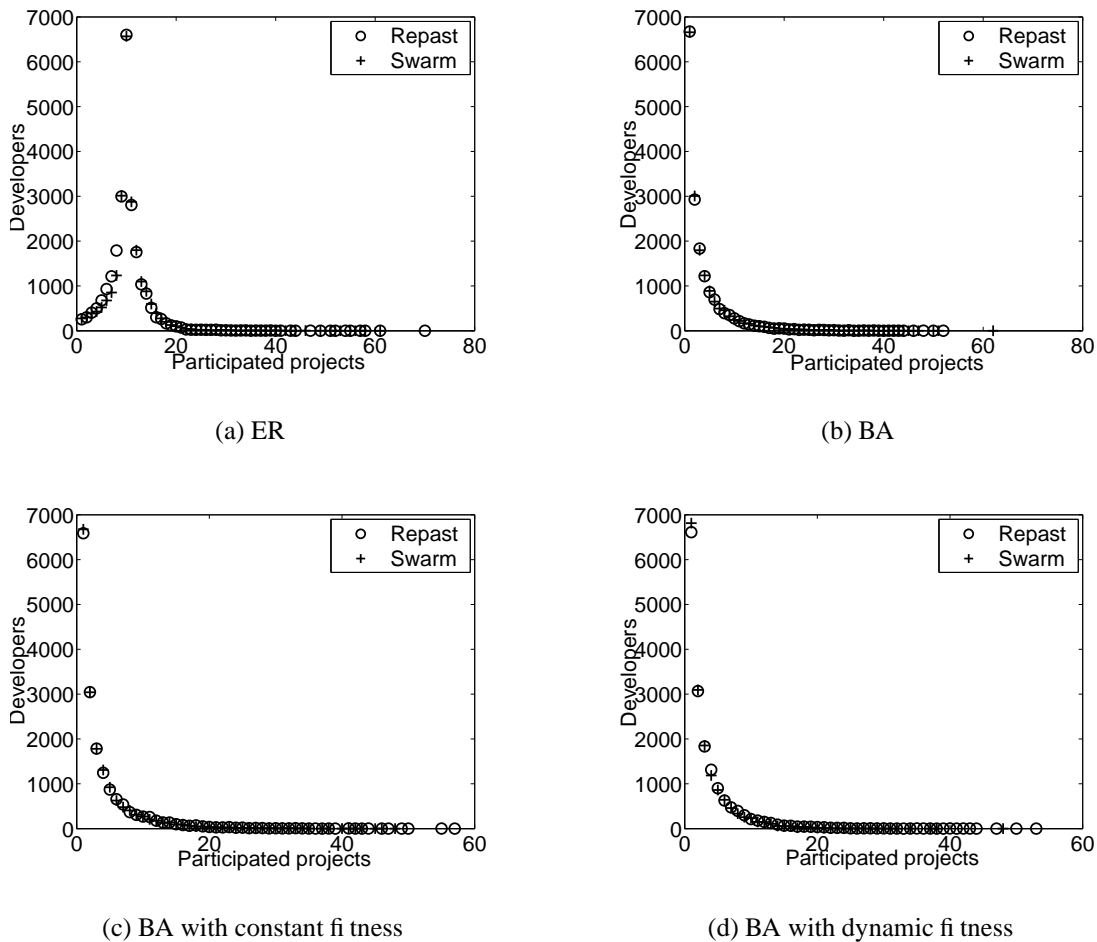


Figure 2: Degree distribution

the two simulations using the exact same set of random numbers: each simulation used the same random number generator with the same seed. The developer and project degree distributions from these runs, however, revealed similar systematic differences between the two simulations.

To determine the exact reasons for this difference, we had the simulations log the action that each developer took during each step. Comparing these logs, two reasons for the differences emerged.

First, we determined that one simulation would occasionally throw an SQL Exception (we store simulation data in a relational database for post-simulation analysis). To recover from such an error, the simulation does not log the developer's action: it moves on to the next developer. Since the developer's previous actions affect its future actions, one error can cause more discrepancies between the two simulations at future time steps. We found the cause of this error to be a problem with the primary keys in the links table of our SQL database (this is a programming bug). Further inspection of the data logs showed that each simulation's data snapshots that are used in analyzing macroscopic graph properties were out of phase by one unit time. Even if the corresponding

simulation ran identically, this extra time step prevented the outputted data from matching. We found that the Swarm scheduler begins at time step 0 while the Repast scheduler begins with time 1. Thus, when snapshots were logged at time step 30, Swarm had actually performed one extra time step.

With these two problems corrected, the corresponding logs of the developers' actions matched. Using the same sequence of random numbers, the Swarm and Repast simulations produced identical output.

## Comparisons of OSS Parameters

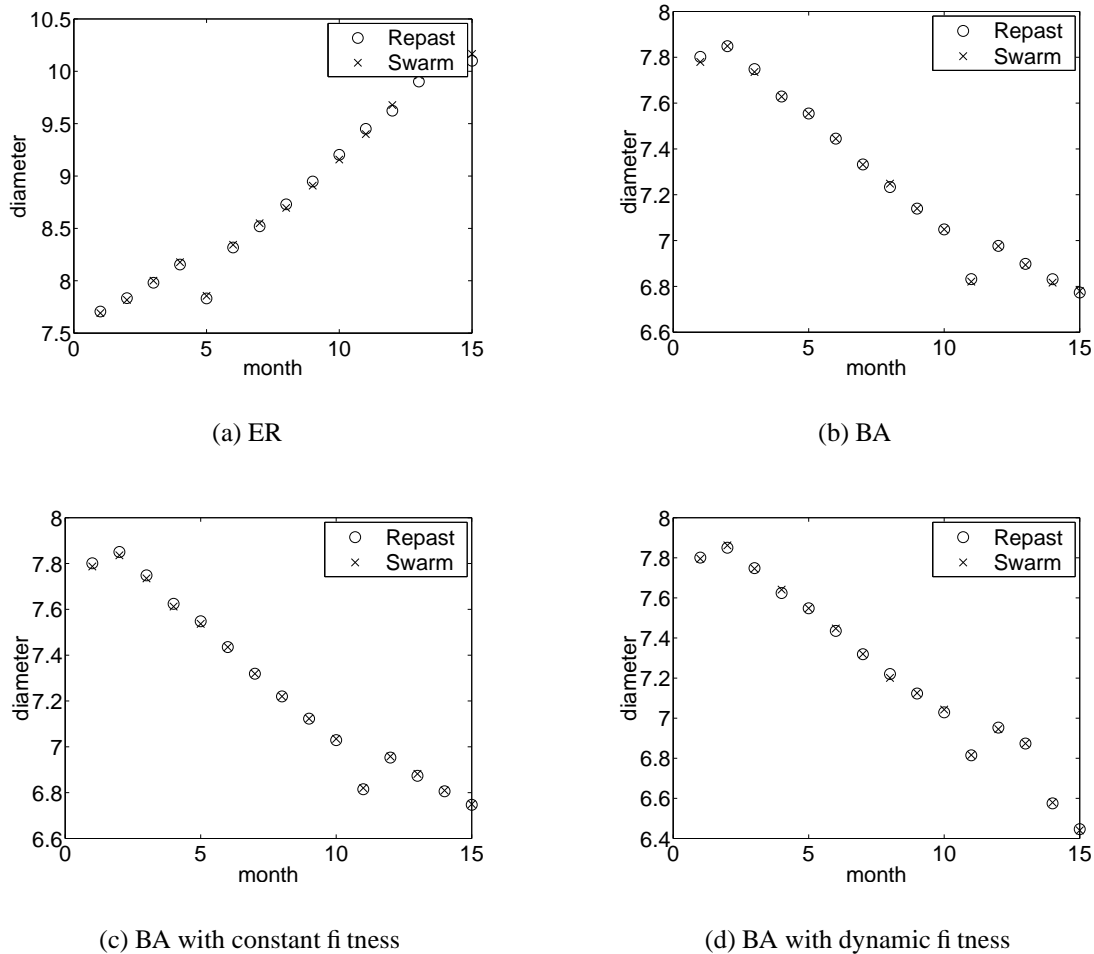


Figure 3: Diameter

Degree distribution, diameter and clustering coefficient are frequent attributes used to describe a network [22, 23] and have been used ever since the foundation of random network theory. In order to study the validity of our simulation, we compared these attributes in Swarm and Repast

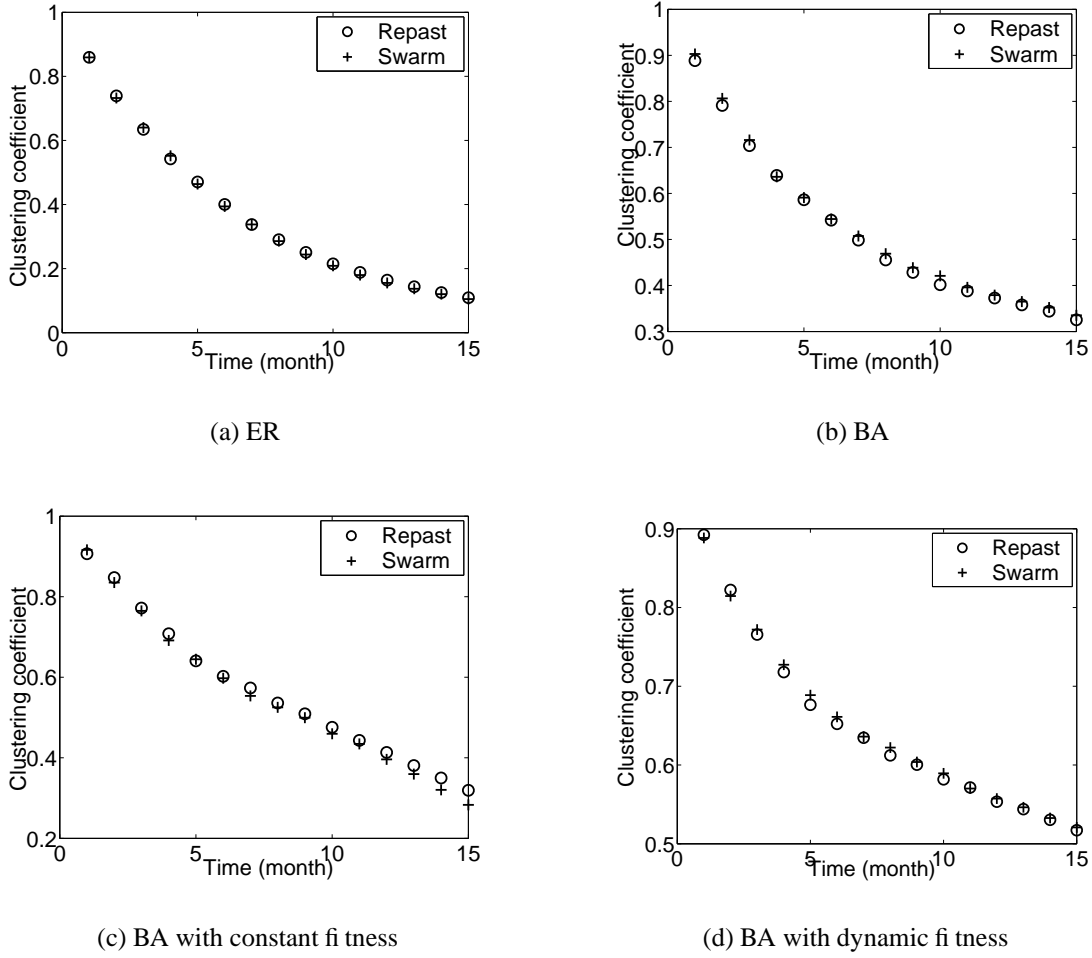


Figure 4: Clustering coefficient

simulations. We observed matches on these attributes between corresponding Swarm and Repast models, which indicates a clean docking.

Degree distribution  $p(k)$  is the distribution of the degree  $k$  throughout the network. The degree  $k$  of a node equals the total number of other nodes to which it is connected. Degree distribution was believed to be a normal distribution, but Albert and Barabasi recently found it fit a power law distribution in many real networks [25]. Figure 2 gives developer distributions in four models implemented by Swarm and Repast. The  $X$  coordinate is the number of projects in which each developer participated, and the  $Y$  coordinate is the number of developers in the related categories. From the figure, we can observe that there is no power law distribution in ER model. The distribution look more like the mathematically proven normal distribution. Developer distributions in the other three models match the power law distribution. However, there are slight differences between Swarm results and Repast results. We believe this difference is caused by different random generators associated with RePast and Swarm.

Degree distribution  $p(k)$  is the distribution of the degree  $k$  throughout the network. The degree

$k$  of a node equals the total number of other nodes to which it is connected. Degree distribution was

The diameter of a network is the maximum distance between any pair of connected nodes. The diameter can also be defined as the average length of the shortest paths between any pair of nodes in the graph. In this paper, the second definition is used since the average value is more suitable for studying the topology of the OSS network. Figure 3 shows the evolution of the diameter of the network. We can see that Repast simulations and Swarm simulations are docked. In the real SourceForge developer collaboration network, the diameter of the network decreases as the network grows. In our models, we can observe that ER model does not fit the SourceForge network, while other three models match the real network.

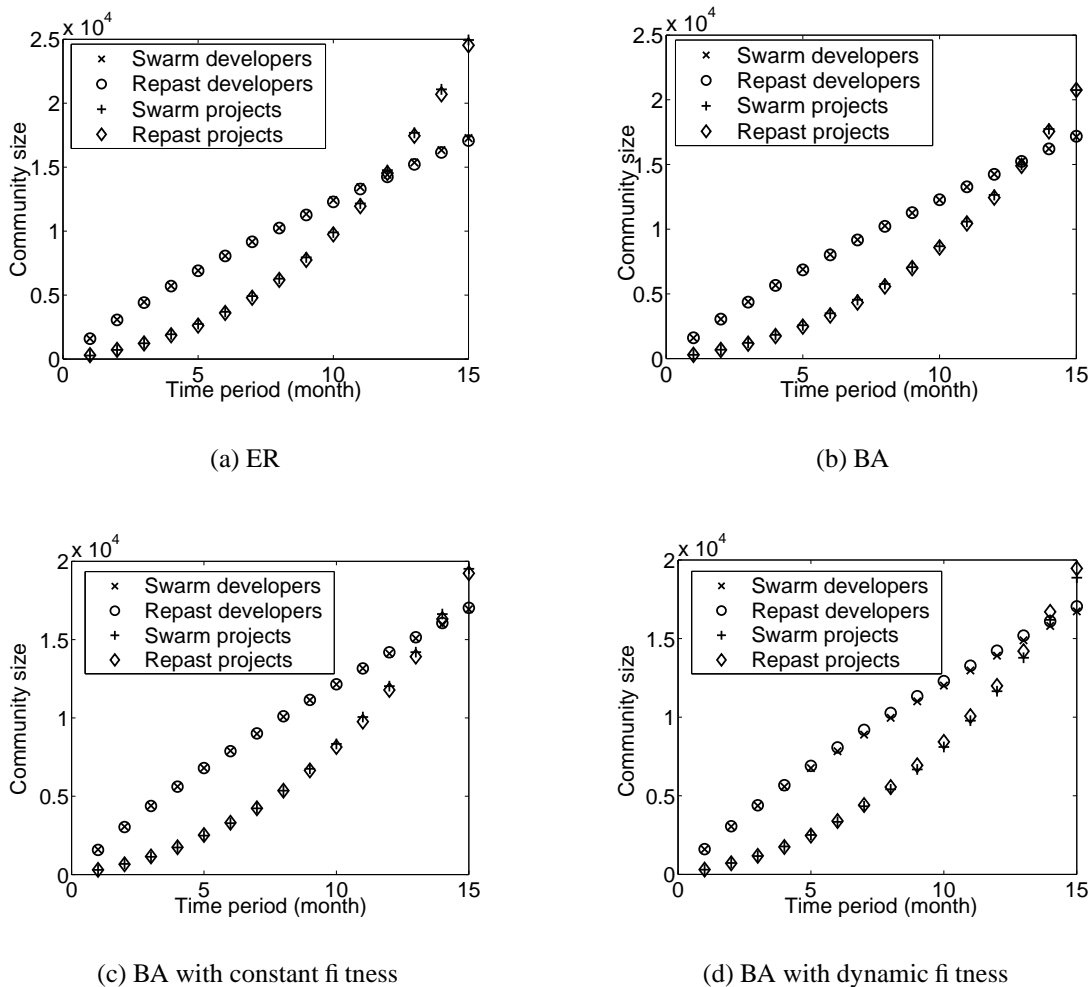


Figure 5: Community size development

The neighborhood of a node consists of the set of nodes to which it is connected. The clustering coefficient of a node is a fraction representing the number of links actually present relative to the total possible number of links among the nodes in its neighborhood. The clustering coefficient

of a graph is the average of all the clustering coefficients of the nodes represented. Clustering is an important characteristic of the topology of real networks. So the clustering coefficient, a quantitative measure of clustering, is also an attribute we investigated. Clustering coefficients for the developer network as a function of time is shown in Figure 4. All models are docked very well. We can observe the decaying trend of the clustering coefficient in all four models. The reason is that with the evolution of the developer network, two co-developers will less likely join a new project together because their participated projects are approaching their limits.

Figure 5 shows the total number of developers and projects relative to the time period in four models, which describe the developing trends of size of developers and projects in the network. The size of developers and projects are almost the same for Swarm and RePast simulations.

## Conclusion

This paper discusses validation of agent-based simulation using the docking process. It describes four simulation models of OSS developer network using Swarm and RePast. Properties of social networks such as degree distribution, diameter and clustering coefficient are used to dock Repast and Swarm simulations of four social networks. Experimental results show that docking two agent-based simulations can help validate a simulation. This paper showed that a docking process can also be used to validate a migration of a simulation from one software package to another. In our case, the docking process helped with the transfer to Repast to take advantages of its features. Repast simulation runs faster than Swarm simulation because RePast is written in pure Java while Swarm is originally written in Object C which may cause some overhead for Java Swarm. Furthermore, RePast provides more display library packages such as network package which help users to do analysis.

*This research was funded in part by the NSF Award-0222829, from the Digital Society & Technologies Program, CISE/IIS.*

## References

- [1] Barabasi A-L. *Linked: The New Science of Networks*. Perseus, Boston, 2002.
- [2] Michael J. Ashworth and Marcus A. Louie. Alignment of the garbage can and nk fitness models: A virtual experiment in the simulation of organizations. [http://www.casos.ece.cmu.edu/casos\\_working\\_paper/GCandNK\\_Alignment\\_abstract.html](http://www.casos.ece.cmu.edu/casos_working_paper/GCandNK_Alignment_abstract.html), 2002.
- [3] Robert Axelrod. Advancing the art of simulation in the social sciences. *Simulating Social Phenomena*, ed. Rosaria Conte, Rainer Hegselmann, and Pietro Terna., pages 21–40, 1997.
- [4] Richard Burton. *Simulating Organizations: Computational Models of Institutions and Groups*, chapter Aligning Simulation Models: A Case Study and Results. AAAI/MIT Press, Cambridge, Massachusetts, 1998.
- [5] Nicholson Collier and Thomas R. Howe. Repast 2.0: Major changes and new features. In *Seventh Annual Swarm Researchers Conference (Swarm2003)*, University of Notre Dame, IN, 2003.

- [6] Nick Collier. Repast: An extensible framework for agent simulation. <http://repast.sourceforge.net/projects.html>.
- [7] Faloutsos P. Faloutsos C. Faloutsos, M. On power-law relationships of the internet topology. In *SIGCOMM'99*, pages 251–262, Cambridge, MA, 1999. ACM.
- [8] Free/open source software development phenomenon. <http://www.nd.edu/oss>.
- [9] Madey G., Freeh V., and Tynan R. The open source software development phenomenon: An analysis based on social network theory. In *Americas Conference on Information Systems (AMCIS2002)*, pages 1806–1813, Dallas, TX, 2002.
- [10] Madey G., Freeh V., and Tynan R. Understanding oss as a self-organizing process. In *The 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering (ICSE2002)*, Orlando, FL, 2002.
- [11] Madey G., Freeh V., Tynan R., and C. Hoffman. An analysis of open source software development using social network theory and agent-based modeling. In *The 2nd Lake Arrowhead Conference on Human Complex Systems*, Lake Arrowhead, CA, USA, 2003.
- [12] Madey G., Freeh V., Tynan R., Gao Y., and Hoffman C. Agent-based modeling and simulation of collaborative social networks. In *Americas Conference on Information Systems (AMCIS2003)*, Tampa, FL, 2003.
- [13] Y. Gao. Topology and evolution of the open source software community,. In *Seventh Annual Swarm Researchers Conference (Swarm2003)*, Notre Dame, IN, 2003.
- [14] Y. Gao, V. Freeh, , and G. Madey. Conceptual framework for agent-based modeling,. In *North American Association for Computational Social and Organizational Science (NAACSOS 2003)*, Pittsburgh, PA,, 2003.
- [15] Y. Gao, Freeh V., and Madey G. Analysis and modeling of the open source software community,. In *North American Association for Computational Social and Organizational Science (NAACSOS 2003)*, Pittsburgh, PA,, 2003.
- [16] Adamic L. A Huberman, B. A. Growth dynamics of the world wide web. *Nature*, 401:131, 1999.
- [17] Xu J. and Gao Y. A docking experiment: Swarm and repast for social network modeling. In *Seventh Annual Swarm Researchers Conference (Swarm2003)*, Notre Dame, IN, 2003.
- [18] Barabasi A. L., Jeong H., Neda Z., Ravasz E., and Schubert A. amd Viscek T. Evolution of the social network of scientific collaborations. [xxx.lanl.gov/arXiv:cond-mat/0104162v1](http://xxx.lanl.gov/arXiv:cond-mat/0104162v1), April 10, 2001.
- [19] Barabasi A. L. and Albert R. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
- [20] Barabasi A. L., Albert R., and Jeong H. Scale-free characteristics of random networks: The topology of the world wide web. *Physica A*, pages 69–77, 2000.
- [21] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. <http://www.santafe.edu/projects/swarm/swarmdoc/swarmdoc.html>.
- [22] M.E.J. Newman. Clustering and preferential attachment in growing networks. *Physics reviews*, 64(025102), 2001.

- [23] M.E.J. Newman. Scientific collaboration networks: I. network construction and fundamental results. *Physics reviews*, 64(016131), 2001.
- [24] Michael North and Charles Macal. The beer dock: Three and a half implementations of the beer distribution game. <http://www.dis.anl.gov/msv/cas/Pubs/BeerGame.PDF>.
- [25] Albert R., Jeong H., and Barabasi A. L. Diameter of the world wide web. *Nature*, 401:130–131, 1999.
- [26] Axtell R., R. Axelrod, J. Epstein, and M. Cohen. Aligning simulation models: A case study and results. *Computational and Mathematical Organization Theory*, 1(2):123–141, 1996.
- [27] Repast home. <http://repast.sourceforge.net/>.
- [28] Wasserman S. and K. Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, Cambridge, UK, 1994.
- [29] Strogatz S. H. Watts, D. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998.