

APPLICATION ARCHITECTURE OF THE INTERNET SIMULATION MODEL: WEB WORD OF MOUTH (WOM)

Mahmoud Rafea^{*}, Fredrik Holmgren^{*}, Konstantin Popov^{*}, Seif Haridi^{*}, Stelios Lelis⁺, Petros Kavassalis⁺, Jakka Sairamesh[~]

^{*} Swedish Institute of Computer Science, SICS, Box 1263, SE-16429 Kista, Sweden

{seif, fredrikh, [mahmoud](mailto:mahmoud@sics.se)}@sics.se

⁺ Institute for Computer Science, Foundation of Research and Technology - Hellas, ICS-FORTH, Science and Technology Park of Crete, P.O.Box 1385, GR 711 10 Heraklion, Crete, Greece

slelis@ics.forth.gr and petros@rncp.mit.edu

[~] IBM Institute for Advanced Commerce, IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 jramesh@us.ibm.com

ABSTRACT

In this paper, we present an Internet simulation application: WoM. The WoM simulation aims at showing agglomerations of users in few web sites and at studying the different factors that lead to such a macro-structure. The paper focuses on the architecture of the multi-agent simulation environment. In this environment, a simulation application is constructed using four different components categories, namely: agent collections, a worker, a communication channels component, and a control component. An agent collection can be constructed from different kinds of Small World graph components. Small World graphs are used to represent the social network and web sites links. An agent collection scales to hundreds of thousands of agents. A worker component is configured from a set of behavior components to implement the behavior of agent collections. Simulation input/output is monitored through a communication channel component. The control component is responsible for configuration and initialization of the application components. The control component, also, invokes the worker at each simulation time step. For scalability, the environment architecture is designed to be suitable for distribution and parallelization.

KEY WORDS

Agents, Simulation, Internet, emergent behaviors

1. Introduction

The web Word of Mouth model (WoM) [1] is one of the behavioral models of iCities project for studying the emerging world of Internet and understanding the effect of various rules of interaction among Internet users. In effect, there is an analogy between web sites and physical cities, on one hand, and Internet users and city inhabitants, on the other. Also, there is an analogy between the services the web sites provide and physical products that people buy and sell in physical cities, where Internet users are considered the consumers.

Consequently, in the iCities project the term *infohabitants* is used to denote Internet users and web sites in order to alert the audience to the proposed concepts.

The analogy of Internet world to real world is the basis of creating the models of this project. The aim of those models is to prove the emergence of a macro-structure, i.e. agglomerations and other regularities, and to study the evolution of web sites and Internet users. The simulation approach is based on agents who represent *infohabitants* and who make individual decisions while interacting with each other. The WoM simulation aims at showing that the users agglomerate in few web sites, and at studying the different factors that lead to such macro-structure. The number of hits or visits, the site has, is the measure of these agglomerations.

The simulation environment is being implemented in Mozart [2], a multi-paradigm language with support for distribution and data-flow synchronization. One of the goals of the project is to provide a simulation environment that supports the development of large-scale simulations in the scale of 100K to 1M agents. Consequently, the challenges are:

- Optimizing code for both performance and memory usage. This certainly has a limit, and we have to consider distributed parallel processing to cope with such large numbers. It is expected that simulations will run on multiprocessor machines or to be distributed over a cluster of machines. Describing the distributed parallel processing in detail is out of scope of this article, but the design of the sequential implementation can be easily parallelized.
- Providing an architecture based on components. This architecture allows the configuration of a simulation application from a set of components. This is an essential requirement, not only, for experimenting with implemented models, but also for developing other simulation models. In effect, new simulation models can be developed by reusing component from implemented models.

In this paper, we will focus on the designed simulation architecture by showing how WoM is simulated. The WoM is briefly described in the next section. For complete description of the model, we refer to [1]. In section 3, the simulation environment is described. The application architecture is described in section 4. Also, we will show how this architecture is suitable for distribution and parallel processing.

2. Web Word of Mouth Model (WoM)

Users usually bookmark their favorite sites, the sites they visit frequently. They seek advice from friends, but also receive recommendations by chance, e.g., from the Media. Mainly, popularity of sites is self-reinforcing and is determined by a *historical process* in which referrals from friends have an effective influence in driving the growth of sites. Notice, also, that when Internet users visit a particular web site, it is plausible that while navigating within this site, they will be tempted to click on related or suggested links that drive to other sites. This means that the probability of an Internet user to decide to visit a particular site is proportional to the number of other sites pointing to it. The WoM adopts this view about how information is transmitted among Internet users and how this introduces an *information feedback* into the process whereby sites compete for market share. Consequently, the learning of users about sites depends on which sites others users have visited, so Internet-users are likely to learn more about popular sites than unpopular ones. Under certain circumstances this information feedback can cause popularity of sites to become self-reinforcing.

During simulation, individual agents choose sites to visit at each time step. But in which site they will stay, i.e., include it in their portfolio of favorite sites; this depends on a decision process. This process is based on their specific behaviors, which are simulated through encoding those agents so that they can perform the following actions:

- Have preferences for specific categories of content
- Participate in “local” social networks where they exchange information about sites that they have already visited and appreciated
- Surf from one site to another along structured navigation paths
- Maintain a portfolio of frequently visited sites but they only visit a number of sites from it at every step
- Have memory about visited sites and their perceived utility.
- Have an evaluation method in order to evaluate sites they visit and update their portfolio with sites with the highest perceived utility.

The number of sites and the number of users may increase during simulation. Sites have performance characteristic ri that determine collectively the quality of a site and have also a vector of specific categories of content sc from a

set of categories C . The value a user attributes to a particular site is determined by a utility function U . This utility function is based on user’s preferences uc , which is also expressed as a vector from the same set C , and a gauss-distribution random value g , which modifies the ri to represent the perceived performance. The utility function has the following expression:

$$U = F(sc \ uc) *(ri + g)$$

Where F is a function that calculates the inner product of the given vectors.

The process of sites selection starts early, at the beginning of simulation, when system is initializing by creating active agents. So, active agents select randomly a number of sites, from the currently available, to create their portfolio. The number of sites in the user portfolio is a simulation parameter. During each simulation time step, the sites selection process continues in the following way:

- A fraction of the active agents, ask other agents (friends and acquaintances within local social networks of relationships) to propose their favorite sites and visit them. This is called word of mouth information gathering mechanism.
- Agents also visit some sites from their portfolio.
- A number of other sites may be visited from surfing along the links of the already visited sites. This number is generated using a Poisson distribution. A set of 100 Poisson distributions of distinct means is randomly assigned to agents, where the mean’s range is a simulation parameter.
- The quality of a new site learned through the word of mouth information gathering mechanism and from surfing along links is, then, evaluated through the utility function U . However, for the most part, agents are loyal to their portfolio of favorite sites, and will only replace a site in the portfolio by a new site if that new site maintained a higher utility for a longer period of time.

3. The simulation environment

The simulation environment consists of categories of components. Components of the same category have the same interaction interface, which consists of a set of operations. Accordingly, the interaction interface of each category of components is determined on the basis of the needed interactions with other categories. A component is either simple or complex. A complex component is constructed from simpler components.

In this environment, a simulation application is constructed from four essential complex components that belong to different categories, and a parameters component, which is a special kind of component because it could be statically linked to the application or catered interactively through a graphical user interface. Namely, those components are:

- Agent collections
- A worker
- A communication channels component
- A control component
- A parameters component

The relationship between those application components is depicted in Figure 1. Notice that the parameters component is linked to all the application components. Other categories of components that are used to construct those essential components are: agent collection graphs, a set of behaviors, random number generators, statistics processing, and persistent store management. The architecture and functionality of those components will be described in the next section. New components can be coded provided that the interaction interface is consistent with the category they belong to.

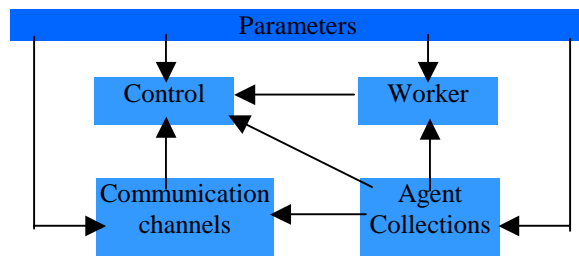


Figure 1: The relationship between the essential categories of components. The outgoing arrow represents an interaction interface provided to another component.

A simulation application experiment is designed by specifying the components together with the simulation parameters, e.g., specifying a component from the agent collection graphs category to each agent collection, the number of agents in each collection, the number of simulation time steps, etc. The control component dynamically loads the specified components and invokes the initialization operations. In this way, components are configured.

The monitoring and visualization of the simulation application is achieved by using what we call communication channels. The communication channels can be classified into: communication channels between the simulation agents themselves, and communication channels between the agents and the model designers. The latter channels will contain the states of simulation, which are the states of agents. In this way, model designers can verify, analyze, and visualize the simulation model. The model designers can determine which channels to monitor using specific fields of the simulation parameters.

4. The application architecture

4.1. Agents collections

In the WoM, agent collections are what we call the infohabitants. The first type of those infohabitants is the Internet users. The Small World graph [3] is used to model the social network of Internet users where each vertex (node) represents a user and the neighbors of this vertex represent his friends. It is also used to model Web sites, the second type, where neighbor vertexes represent the links to other sites. Each vertex is an agent within the collection it belongs to. The vertex number determines the identity of such agent, its state is stored in a data-structure, and its behavior is implemented by a set of operations. This kind of agent is called simple agent [4]. The architecture of the agent collection is illustrated in Figure 2.

Notice that the depicted architecture shows only the essential components. In WoM, another component is needed in both web sites and Internet users agent collections, namely: a random number generator.

The original small world graph is static, i.e., the graph does not expand. Two new dynamic small world graphs are designed and implemented. Those two graphs together with another kind of graph that is suitable only for web sites will be described in another article. Consequently, an application can use one of those graph components to construct the Internet users agent collection and the same or another graph component to construct the web sites agent collection. Recall that those graph components represent a category that is required by the agent collection. In WoM, there are, only, two categories of agent collections, namely, Internet users and web sites. Each category has one component, but other agent collections for both categories are, currently, being implemented to develop other simulation models.

The Mozart implementation is based on representing both the graph and the agent collection in a chunk data-structure. The chunk data-structure consists of two parts: an index and data. The vertex number indexes each agent in the collection, and also indexes the graph. The operations of the interaction interface of graphs are enumerated in Table 1. It should be remarked that id of vertexes of the graph is numbered in sequence. One of the graph implementations is based on having, initially, a continuous segment of vertex ids that represent agents who are active in the simulation. Vertexes or new agents are, then, added to the active agents according to a designed algorithm. The interaction interface contains operations to deal with this characteristic.

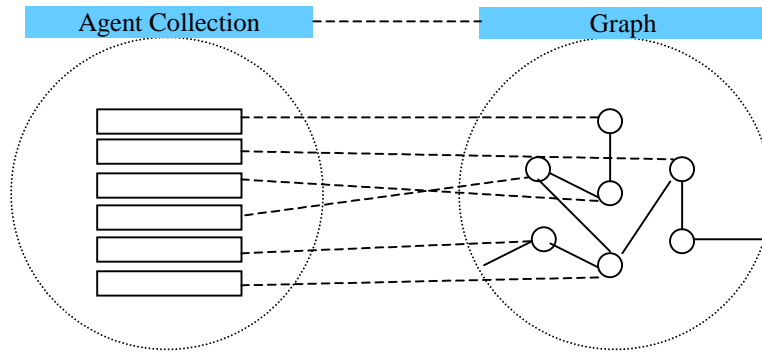


Figure 2: Agents architecture showing the relationship between graph vertices and the data-structure of agent collection

| Operation | Input | Output |
|------------------|---|--|
| Init | Record of total and initially active number of vertexes | Graph gets constructed |
| GetNeighbors | Vertex-id | Neighbor vertexes |
| GetActiveSegment | - | Ids of first and last vertexes |
| AddNodes | - | List of vertexes added to the graph |
| GetActive | - | List of current active vertexes |
| GetInDegree | Vertex-id | Number of edges directed to the vertex |

Table 1: The interaction interface of components that belong to the graph category

The operations of the interaction interface to the web sites and Internet users agent collections are enumerated in Table 2 and Table 3, respectively. Notice the similarity between operation names and functionalities, e.g., both GetLinks and GetFriends return the graph neighbor vertexes. Regarding the operation, the main difference between the two collections is the operations that are used by communication channels component, which are PortfolioDBSave and LoyaltyDBSave.

In the current implementation of WoM, the state of web site agent is static. Its Mozart record structure is of the following form:

```
site(
  categories: <Categories vector>
  performance:<uniform random value:Performance>
)
```

| Operation | Input | Output |
|-------------|---|--|
| Init | Record of total and initially active number of agents | Agents get constructed |
| GetRep | Agent id | Agent's record |
| GetLinks | Agent id | Graph neighbor vertexes |
| AddSites | - | List of ids of agents added |
| SitesDBSave | File name | Save the state of agents to a persistent store |

Table 2: The operations of the interaction interface of web sites agent collection

| Operation | Input | Output |
|-----------------|---|---|
| Init | Record of total and initially active number of agents | Agents get constructed |
| GetRep | Agent id | Agent's record |
| GetFriends | Agent id | Graph neighbor vertexes |
| AddUsers | - | List of ids of agents added |
| UsersDBSave | File name | Save the static state of agents to a persistent store |
| PortfolioDBSave | Stream | Save users' portfolios |
| LoyaltyDBSave | Stream | Save users' memory |

Table 3: The operations of the interaction interface of Internet users agent collection

The Mozart record structure of Internet user agent is of the form:

```

user(
  depth: <a handle to Poisson distribution random
        number function>
  lambda <uniform integer random number>
  categories: < user preferences of Categories vector>
  portfolio: <Dictionary containing vertexes numbers of
            sites agents>
  loyalty: < the user memory; initially an empty
           dictionary >
  loyaltySize: <a cell containing total number of sites the
              agent remembers
              >
)

```

Recall that those structures are indexed in the corresponding collection with the vertex id.

4.2. Worker

The simulation model describes the behavior of agent collections. The worker consists of two parts: a control loop and a set of behaviors, which are configured from behavior components. In each time step, the worker-control loops on each individual agent from a particular collection, invokes the operations of the interaction interface of behavior components, and update statistics data using one of the channels that are provided by the control component. The behavior components will use the current agent to interact with other agents, e.g., his friends, and his portfolio sites. Eventually, the states of

agents are updated and data is written to communication channels.

In WoM, the worker component loops on the Internet users agent collection. The behavior components of the WoM worker are depicted in Figure 3. The ellipse shapes represent the behavior component, and the rectangular shapes represent the individual agent's structures holding dynamic and static states. The sequence of actions is depicted as numbers from 1 to 5, in small rectangles. For each Internet user agent, the behavior sequence of actions is invoked exactly once per each time step. Some behavior components write to the communication channels. In such components, the needed channel is passed from the worker-control to those behavior components. The interaction interface of the WoM-worker consists of one operation, which is 'doStep'. The input parameters are the current simulation time step and a data-structure containing the channels.

It should be remarked that parallelization of WoM is based on running different instances of the worker in parallel, while distribution is based on partitioning the web sites and internet users agents into different collection. The challenges in such systems are: minimizing the communication to outside the processes, implementing an efficient communication mechanism among processes, and keeping the worker busy while waiting for data to arrive from the other processes. This will be the scope of another article.

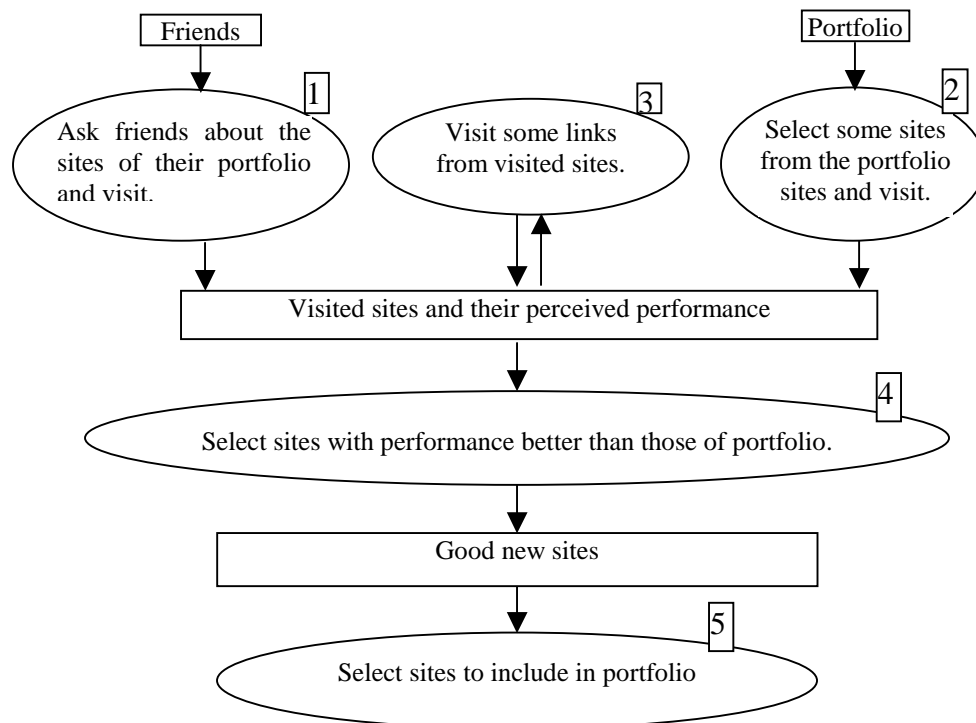


Figure 3: The behavior components and sequence of actions in WoM worker

4.3. Communication channels

Recall that the communication channels have been classified into: communication channels between the simulations agents, and communication channels that transfer the states of simulation to the model designers. The states of simulation can, further, be classified into: static and dynamic. The dynamic state means that a simulation state is stored for each simulation time step, while the static state means that one state need be stored because static state does not change during the whole simulation. An example of static state, in WoM, is the performance characteristic of web site agents, which are acquired when first created. Notice that new agents may be created during the simulation, but once created some of their state is static. Examples of dynamic state are a user agent portfolio and agent memory.

Consequently, the communication channels are of two types: static and dynamic. The static channel is created at the end of simulation. The control component invokes one operation in a communication channels component, which passes a different channel to each agent collection through a defined operation. This operation stores all static state data. The dynamic channel is created at the beginning of simulation. A separate channel is used to monitor each dynamic state, e.g., the channel that monitors the agent portfolio is separate from the channel that monitors his memory.

There are two implementation approaches for dynamic channels. The first approach is suitable for communication data in which the control component invokes an operation in a communication channels component that returns with all the needed channels. The control component passes those channels to the worker. The worker uses those channels accordingly. The second approach is suitable for dynamic states. The control component invokes a defined operation in a communication channels component, after the end of each simulation step, passing the current time step. This operation invokes another operation in each agent collection component passing the channel and the current time step. The dynamic channels are closed at the end of simulation.

Channels can be mapped directly to states so that the state data can be stored locally and need not be centrally processed. This is an important feature that is needed for the parallel-distributed version. In fact, storing the states of simulation constrained the design and implementation of the control component to minimize modifications when moving to the distributed version. A second issue is that this technique enables the model developer to use different user-interfaces for display for the content of the channels. A consequence of this is that the control component, in some sense, becomes generic. By generic, we mean that the implemented component can be used with more than one model of the project.

One of the WoM communication channels component contains a graphical user interfaces. This interface is illustrated in Figure 4. The user can enter the simulation parameters and monitor the agglomeration of users into sites through the graph. He can also stop and resume the simulation.

4.4. Control component

The control component is the main program. It consists of two components that are linked statically. The first component is responsible for configuration and initialization of different application components. The second component controls simulation time steps by invoking the simulation operation of the worker component in each new time step. The control component architecture is depicted in Figure 5. The synchronized sequence of actions is numbered from 1 to 6 and displayed in small rectangles.

It should be remarked that the first step in application initialization is to interpret the parameters file. If the simulation parameters are available, the parameters component interaction interface is created and made available to all application components. Otherwise, an operation in a communication channels component is invoked. Eventually, the parameters will be available and the parameters component interaction interface will be created.

5. Conclusion

In this paper, a design and implementation of Internet simulation application is presented. The design is based on an architecture that can be incrementally extended with new components. The simulation is based on simple agent collections. The described implementation represents an environment that can be used to both implement experiments with the presented WoM and develop other simulation models. An experiment of 50,000 Internet user agents, 50,000 web site agents, and 100,000 time steps takes less than one week to complete. The work is still in progress. New components are being developed to implement new agent collections and simulation models. The parallelization and distribution efforts have been successful, but the work is still under publication.

References

- [1] Stelios Lelis, Petros Kavassalis, Jakka Sairamesh, Seif Haridi, Fredrik Holmgren, Mahmoud Rafea, & Antonis Hatistamatiou, Regularities in the Formation and Evolution of Information Cities, The Second Kyoto Meeting on Digital Cities, October 19-20, 2001 Kyoto Research Park, Kyoto, JAPAN.
- [2] Mozart Consortium, The Mozart programming system, Available at <http://www.mozart-oz.org/>, 1999.

[3] Duncan J. Watts, *Small worlds: the dynamics of networks between order and randomness* (Princeton University Press, New Jersey, ISBN 0-691-00541-9, 1999).

[4] Joshua M. Epstein and Robert Axtell, *Growing artificial societies: social science from the bottom up* (The MIT Press, Cambridge, Massachusetts, ISBN 0-262-05053-6, 1996).

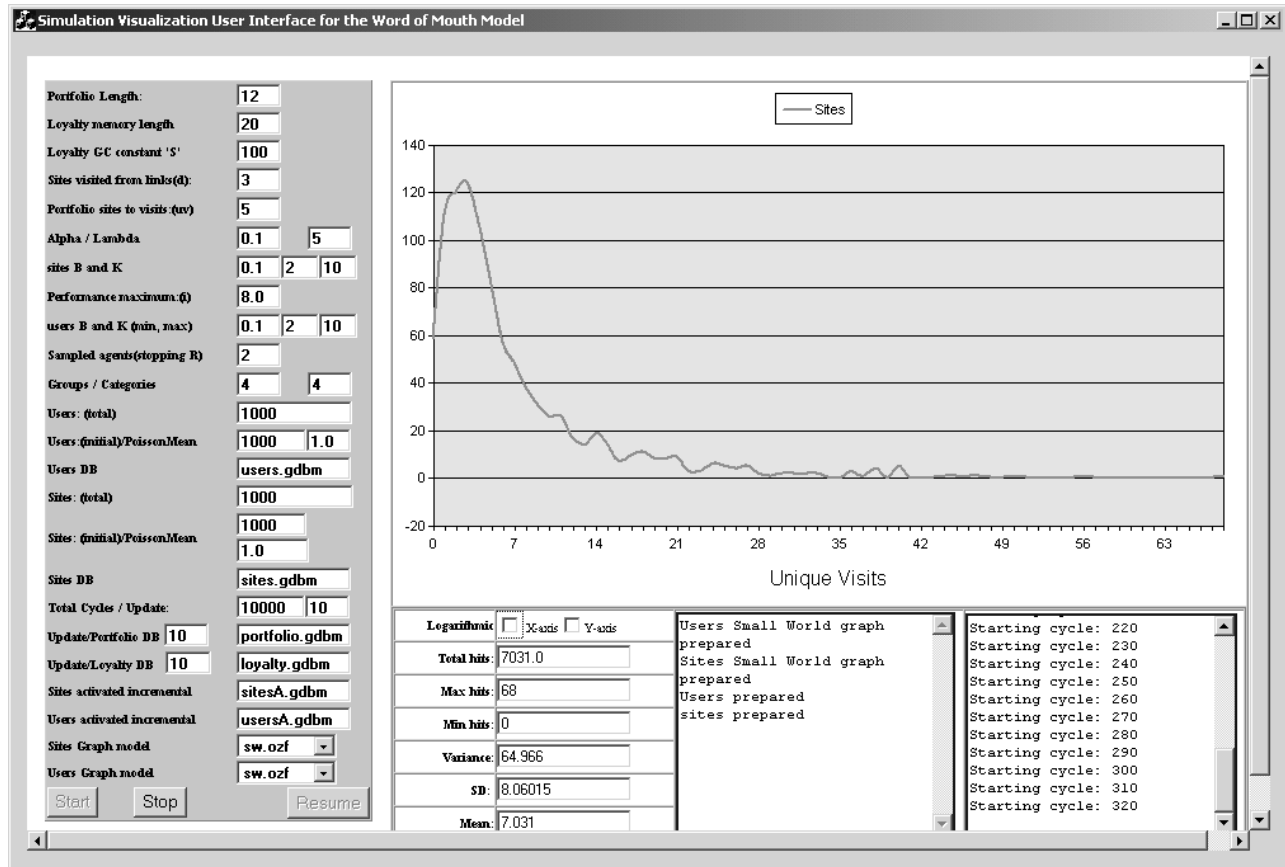


Figure 4: Graphical user interface for WoM

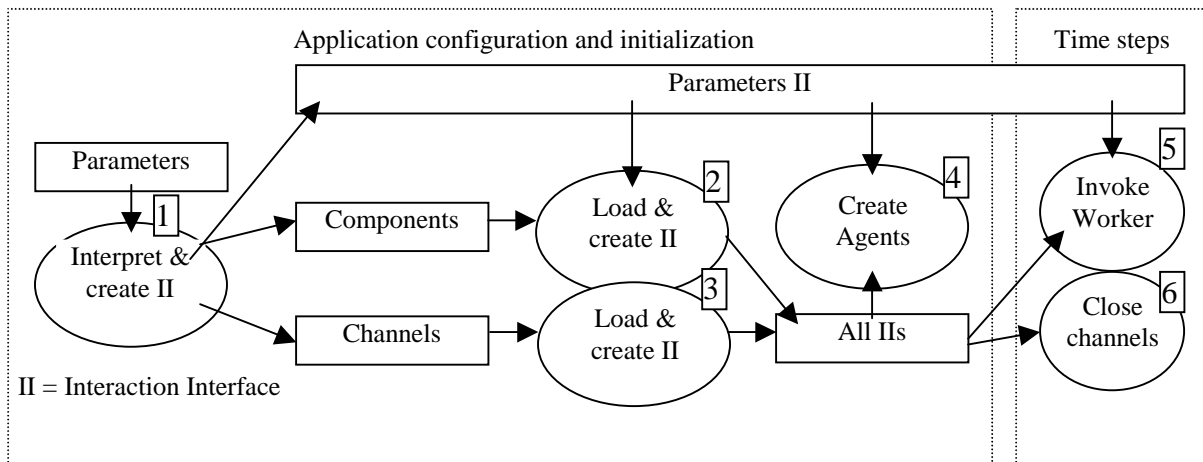


Figure 5: The control component architecture showing the operations sequence