MULTI AGENT BASED SIMULATION FOR DATABASE SECURITY: A FRAMEWORK

Marco Remondino
Department of Computer Science
University of Turin
10149 Turin, Italy
E-mail: remond@di.unito.it

ABSTRACT

Agent Based Simulation is successfully applied to enterprise modeling and social sciences, and is considered as a third way to represent models, alternative to the verbal argumentation and the strictly mathematical approach. The advantage over the other two is its high portability on a computer, in order to be executed, and its flexibility, which makes it the optimal tool to represent complex systems. The purpose of this paper is to discuss the possibility of using the Multi Agent paradigm to simulate a database, in regard to the security policy applied; databases are very complex objects, and thus an Agent Based approach should allow to exploit the interactions among users and the results deriving from a particular security policy. Besides, the necessity of granting a certain security level for data access often compromises the efficiency of data retrieval, and thus the optimal balance of the two is a difficult task to accomplish. The creation of an Agent Based simulation, which models a database security environment, allows what-if analysis and case study, at the variation of defined rules and parameters, without trying to change the security policy in the real environment. An operative framework with rules, to be used for the creation of a model representing a generic database, subject to a Discretionary Access Control policy, is then proposed and studied, in order to simulate the effect of security rules, through the modification of some initial parameters.

INTRODUCTION

According to (Ostrom, 1988), simulation can be considered a third way to represent social models; in particular, it can be a powerful alternative to other two symbol systems, the verbal argumentation and the mathematical one. Simulation has a great advantage over the other two, which is its high portability on a computer, through a program or a particular tool. In particular, Agent Based Simulation is optimal for modeling complex systems, which couldn't be ported to a computer in any other way. This approach allows to observe the emergence of complex behaviour, through the creation and study of models, known as Artificial Societies. Thanks to the ever increasing computational power, it has been possible to use these concepts to create software models, based on intelligent agents, which aggregate behaviour is often difficult to predict just studying the single parts, without considering the interaction among them and with the environment.

In this paper we'll propose an operative framework for the creation of a Multi Agent Based Model of a generic database, to simulate the security rules applied to it, and to verify the various effects they have on efficiency, time and data corruption, by modifying some core parameters. A database is always a very complex object, managed by many different rules, hence the idea of simulating the security environment using a Multi Agent Based Model. Besides, the necessity of granting a certain security level for data access often compromises the efficiency of data retrieval, and thus the optimal balance of the two is a difficult task to accomplish. The creation of an Agent Based Simulation, which models a database security environment, can give answers to what-if situations, at the variation of defined rules and parameters, without trying to change the security policy in the real environment.

MULTI AGENT SYSTEMS

A software agent can be described as a flexible system, capable of dynamic, autonomous actions, in order to meet its design objectives, that is situated in some environment. The main features for a software agent are: situatedness, that is ability to perform actions according to a particular input received from outside, and which can, in turn, change the environment itself; autonomy in performing actions, without intervention of humans; flexibility and adaptability. Some particular agents can also be proactive, which means they are goal-directed, and social, in the way they can interact with other artificial agents, robots, and humans. Such an intelligent agent can be referred to as a Belief-Desire-Intention (BDI) one. There are many agent based paradigms that can be applied to simulation:

- Symbolic: highly structured agents, described through expressions of modal logic. This paradigm is perfect when there is a single agent, which must interact with the environment, but it's not versatile when used to simulate big communities
- Sub-symbolic: simple agents, which can be described through metaphors. Here the stress is on interaction and cooperation and not on the single entities. A multi-agent context of this kind allows the emergency of complex behaviour and self-

Proceedings 18th European Simulation Multiconference
Graham Horton (c) SCS Europe, 2004
ISBN 3-936150-35-4 (book) / ISBN 3-936150-36-2 (CD)

organization. Intelligent behaviour is a product of the interaction among agents and environment, and of the interaction among many simple behaviours. It can be really hard to describe the real world under every aspect: on the single agents can thus be defined some fundamental macro-actions, which allow cooperation with the environment and with other agents. The concept of Multi Agent System for Simulation of Complex Systems is thus introduced: the single agents have a very simple structure. Only few details and actions are described for the entities: the behaviour of the whole system is a consequence of those of the single agents, but it's not necessarily the sum of them. This can bring to unpredictable results, when the simulated system is studied.

- Hybrid Architectures: at the lower levels, we find reactive agents, like the ones described above, while at the upper levels there are more complex and structured agents. In this way, we can combine reactive capabilities with planning.

DATABASE SECURITY

A Data Base Management System (DBMS) is defined as a software package, designed to store and manage databases, which are very large, integrated collections of data. A DBMS allows to reach the following objectives:

- Data independence and efficient access
- Reduced application development time
- Data integrity and security
- Uniform data administration
- Concurrent access, recovery from crashes

It is then obvious that one of the fundamental goals of a DBMS is to reach a security level which could prevent users with no specific grants to read data. It's also very important to reach a satisfying level for data integrity, by preventing users without an authorization to modify them. On the other side, it's necessary to reach an high efficiency for data retrieval, when the users have the specific rights. A security policy applied to a database must specify who is authorized to do what, and a security mechanism allows to enforce a chosen security policy. There are two main mechanisms at the DBMS level: Discretionary Access Control (DAC) and Mandatory Access Control (MAC).

The former is based on the concept of access rights or privileges for objects (i.e. tables and views), and mechanisms for giving and revoking users privileges; in this model, the creator of a table or a view automatically gets all privileges on it. The DMBS keeps track of who subsequently gains and
loses privileges, and ensures that only requests from users who have the necessary privileges, at
the time the request is issued, are allowed. The fundamental command, in this paradigm, is GRANT:

GRANT privileges ON object TO users [WITH GRANT OPTION]

In this way, the specified users get the privileges on the object belonging to the DB; usually, the privileges are the following ones:

- SELECT: Can read all columns (including those added later via ALTER TABLE command).
- INSERT(col-name): Can insert tuples with non-null or non-default values in this column. Similarly, UPDATE.
- INSERT means same right with respect to all columns.
- DELETE: Can delete tuples.
- REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.

If a user has a privilege with the GRANT OPTION, he can pass it on to other users, in turn with or without passing also the GRANT OPTION. Privileges can of course be lost, through the REVOKE command; if a user looses his privileges on an object, also the ones who had them from him will lose them. A user can receive the same privileges from different subjects and, in this case, he would loose them only if all these users loose those privileges on the object.

While in SQL-92, privileges are assigned to authorization ids, which can denote a single user or a group of users, in SQL:1999, and in many current systems, privileges are assigned to roles, that can then be granted to users and to other roles. This approach reflects how real organizations work and illustrates how standards often catch up with de facto standards embodied in popular systems.

Differently from the model described above, the MAC is based on system-wide policies that cannot be changed by individual users. Each object in the database is assigned a security class and each subject, user or user program, is assigned a specific clearance for a security class. The rules based on security classes and clearances govern who can read or write which objects. The MAC was born to overcome a typical flaw of the discretionary system, known as Trojan Horse. In fact, user A could create a table, on which he has all the privileges, and then can grant to user B the INSERT privileges on it. User B has privileges on, and thus can access, another table, containing secret data, which are forbidden to user A; then, user A modifies the code of an application program used by user B to additionally write those secret data to the newly created table, and so user A can now access these secret data. Bell-LaPadula model defines the main rules for the management of MAC. In this model we find:

- Objects (e.g., tables, views, tuples)

- Subjects (e.g., users, user programs)
- Security classes: Top secret (TS), secret (S), confidential (C), unclassified (U)
- An order for the classes: TS > S > C > U
- Each object and subject is assigned a class:
  - Subject S can read object O only if class(S) class(O) (Simple Security Property)
  - Subject S can write object O only if class(S) class(O) (*-Property)

The main idea is to ensure that information can never flow from a higher to a lower security level. This, obviously, avoids the Trojan Horse problem. The MAC rules are usually applied in addition to any discretionary controls that are in effect.

AN AGENT BASED MODEL FOR DB SECURITY

In the following description of the model, we will represent a database organized according to the DAC, which is simpler and easier to port to a programming language. The use of an Object Oriented Language (OO) is assumed; this kind of languages (C++, Java) allows the creation of many independent objects, without having to write specific code for each of them. Besides, in an OO language, there are proprieties such as inheritance and polymorphism, useful for this model.

We can think of a set of agents, which are the users of a database, organized into a hierarchy; in general, a community of agents which can access data according to specific rules. When the single agent needs a datum, he first looks for it and, if he can't access it directly, he asks other agents, who have the specific permission on it. Hierarchy and proximity relations are defined among the agents: there are n levels, organized into a pyramid. Level 1 is the upper one, while Level n is the bottom. Besides, on the same level, proximity relations can be defined: a list could also exist, called Project Colleagues, containing the IDs of the agents working on the same project, so that if one of those creates a table or an object, the other agents will automatically have the access to them. Data inside the database are modelled as very simple objects, which have: a unique number, so that they can be called and retrieved; an identifier, signalling if the datum is corrupt or fine; the ID of their creator. Besides, there is a variable, associated with the datum, which signals whom accessed it for the last time; this is used to keep track of whom damaged it, if the datum is not fine anymore. Each user agent has his own unique ID, representing the name of the subject; a number, identifying the level to which he belongs and a list of the privileges on data; each element of the list is an array whit the following elements:

1) datum number (code)
2) read privilege (yes/no)
3) ID of the subject that granted the privilege at point 2
4) write privilege (yes/no)
5) ID of the subject that granted the privilege at point 4
6) delete privilege (yes/no)
7) ID of the subject that granted the privilege at point 6

Besides, each agent has another list, containing the privileges he granted to others. Again, each element of the list is an array, with these elements:

1) datum number (code)
2) IDs of the subjects to whom read permission has been granted
3) IDs of the subjects to whom write permission has been granted
4) IDs of the subjects to whom delete permission has been granted

Obviously, an agent can't grant a privilege on a datum if he hasn't got it himself. When a subject creates an object, an array is automatically inserted in his list, with the new datum number (code) and all the privileges on it. Besides, Colleagues List described above could be implemented and in this case, when an object is created, all the subjects in this list will automatically have the privileges on it. Each user has also an unreliability index, which is increased each time he damages data, after a write operation. When an agent must complete an operation on a certain datum, he tries to access it directly, by looking in its list if he has the needed privileges on it. If he has the privileges, he access the datum in a single time unit, t. This process is shown in Figure 2: the user B, belonging to the third level of the pyramidal hierarchy, wants to access datum_8, in the central database. When he contacts the DBMS (1), the datum attributes (2a) are compared to the list of the privileges of the user (2b). If the user has the needed privileges, he can immediately access the datum (3), and the operation is finished in a time t.

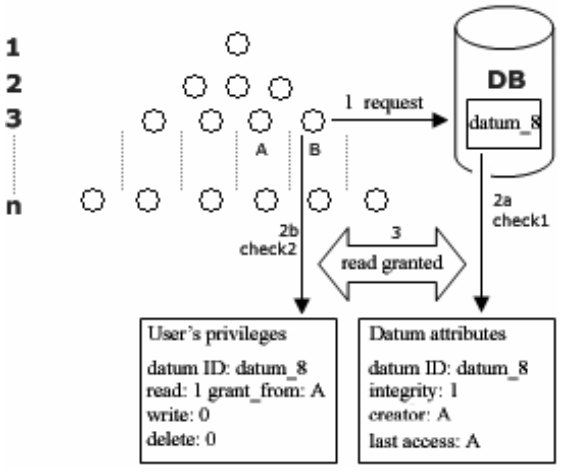

Figure 1 – access attempt, with privileges

If ha hasn't got the needed privileges, he asks the datum who its creator is; this requires a time equal to that of retrieving the datum directly, that is t; as a reply, the creator ID is returned. According to the adopted security policy, which in the simulation can be changed by the user, the agent will ask for the grant directly to the creator or, in the most inflexible case, he will have to move up in the hierarchy, asking for the grant to an user at the lever which is immediately upper, and so on, till when he meets one that has the needed privileges. In the worst case he will need to go all the way up to the creator; obviously, each request will consume some time, which can be considered equal to t/2. When the user meets an agent with the required privileges, he asks him to pass them to himself, consuming again a time t/2.

According to the inflexibility of the security policy, selectable before the simulation starts, and according to the unreliability index of the subject, the privileges will or won't be granted. If they are granted, the user will be able to access the datum, in a time t; again, according to the security policy, the privileges can be kept by the user or can be immediately revoked after the operation has been completed. If the privileges are not granted, the user which owns them will access the datum on behalf of the requesting agent in a time 2t. Of course, if the same user has to access the datum again, he will have to pass again through all these steps. In Figure 3, we show an example of the described case: user B must access datum_8 (1), but after verification (2a and 2b), he realizes he can't do that directly. The datum then returns the ID of its creator, that is user A (3), who is on the same level as B can thus be contacted directly (4) by B.

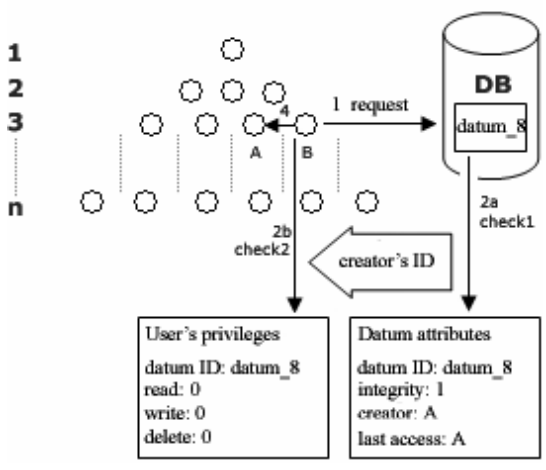

Figure 2 – access attempt, without privileges

With some simple calculations, we see that in the most inflexible situation, if the agent accessing the datum is not its creator, the access time, for each operation, is:

$$3t + t\left(\frac{\Delta + 1}{2}\right)$$

where Δ is the distance between the level of the user requesting privileges and the creator of the object that is being accessed. This time must be compared to time 3t, needed for data access in the most flexible situation, in which privileges are always granted. When an agent has the write privilege on a datum, there is a probability function which determines the possibility that this operation compromises its integrity. During the write operation, the user could modify the flag variable of the datum from 1 to 0.

The next time this datum will be necessary, the problem will emerge: this will increase the index of system unreliability and will waste a time 2t, to restore the datum. The user who damaged it, whose ID is stored in the Last Access variable of the datum, will have its personal unreliability index increased and, according to the security policy adopted, will loose or not the privileges on that datum. When a user looses his privileges on a certain datum, also the agents who had the privileges from him will loose them. It could be possible that an agent had received the same privileges from more than one user: in this case, he will keep the privileges, unless all the granting agents loose them. The need of the users to access data is controlled by random functions, and so it the probability function for data corruption.

This probability increases with the growth of the delta between the level of the creator and that of the user accessing the data. During the execution of the simulation, two real-time graphs will be created: one will represent the average time for data retrieval; the other one will represent the general unreliability index of the system, derived from the average of corrupted data. By varying the security policy, through the initial parameters, it will be possible to compare different situations, after the same number of simulation steps and with the same random seed. The security policy affects the probability for an agent to grant the privileges to another user, on certain data. A probability equal to 0 means that no agent will receive the privileges, so that only the owners can access the data they created. In this case, the general unreliability index will be very low, but the time for data retrieval will reasonably be very high. A probability equal to 1 means that the privileges are always granted, no matter who asks for them: of course this will bring to an opposite situation. The intermediate cases, i.e. a probability between 0 and 1, are the most interesting and difficult to predict, but also the most useful to model real situations.

Also the creation of new data is managed in a random way, and at the beginning of the simulation some steps

will be dedicated to this activity. We can think of a number of data with an inverse proportion in respect of the level or, in alternative, we can put in the simulation the exact situation that we observe in the organization we want to model.

The last case that needs to be considered is the request of privileges on certain data by an user who is at an upper level than their creator; there could be an automatic grant or, if we want to be more realistic, the request could be addressed directly to the creator, without needing to move down in the hierarchy, but using the rules defined in the security policy, that consider the personal unreliability of the requesting agent.

A BASIC IMPLEMENTATION

While the research is still in progress, we created a working example of the "worst possible case", to show that the simulation is feasible and that Agent Based Technology can be successfully applied to database security simulation. In this first implementation, we had to simplify several rules, when compared to the ones described in the previous paragraphs; these will be added in future implementations of the model.

The implementation is an agent based simulation realized with Java Agent-Based Simulation library (http://jaslibrary.sourceforge.net), by Alessandro Cappellini (cappellini@econ.unito.it). This simulation can represent an ordinary un-secure database, or a normal data warehouse, (e.g. a "soho" fileserver and its directories); in fact we haven't got any restriction, and everybody can access everyone's data. This is not the only simplification introduced in this first implementation: there is just a single level of agents (no hierarchy); just two basic operations are defined on data (read and write); no new data are created; damage data are not repaired; no access policy is defined.

The results are quite straightforward, since even if the probability to corrupt data is very low, sooner or later the database will collapse (100% failures, as shown in Figure 4).

Though, this simple model is just the basis to implement all the other rules described above. At each "tick" the agents access a random datum inside the db; they can access the datum to read it (probability 70%) or to modify it (probability 30%). They check the owner of the datum and its integrity; if the datum is corrupt, the access fails. When they access the datum to modify it, there's a probability (10%) that they can corrupt it.

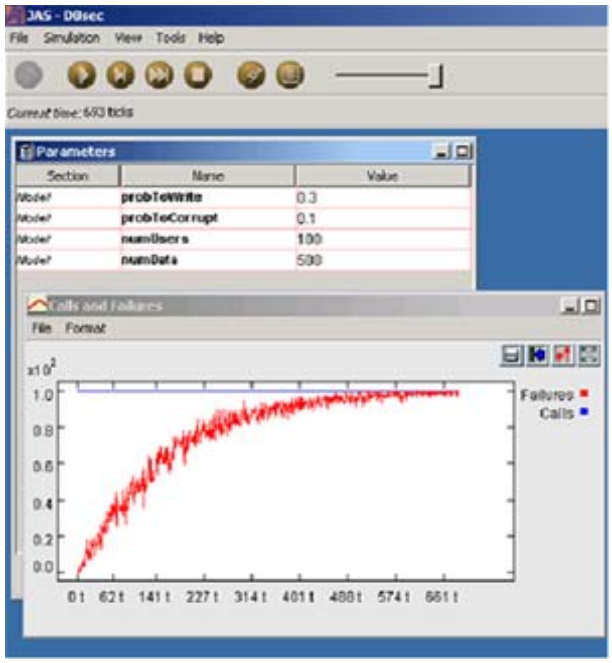

Figure 3 – The Basic Case being Simulated

CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have described a complete framework with rules, for the creation of a software Agent Based Model, to be used to simulate the behaviour of a DBMS, with regards to database security under a DAC. Three are the fundamental features, considered for the simulation: data integrity, time for retrieval and flexibility of the security policy used. While the first two are the dependant variables, the last one is the independent variable, which is the one that can be set by the user, through some parameters, before the simulation starts. The main purpose of this work is to demonstrate the feasibility of an agent based software simulation of database security, which would allow a what-if analysis for designers and users. A simplified working model is then shown, in which many rules described in the theoretical framework are not yet implemented. This is already interesting, since it shows that an agent based model of a database is indeed feasible. Since the research is still in progress, we intend to further develop the simulation tool, by adding all the features described in the theoretical framework, thus converting it into an operative model of a real database.

REFERENCES

Bertino E. et al. 2001. "Intelligent Database Systems", Addison Wesley Professional

Huhns, M. and Singh, M. 1997. "Readings in Agents", Morgan Kaufmann

Gilbert, N. and Terna, P. 2000. "How to build and use agent-based models in social science", Mind & Society 1, 57-72

Gilbert, N. and Troiztsch K.G. 1999. “Simulation for the Social Scientist”, Open University Press

Khalil E.L 1996. "Social Theory and Naturalism" in Khalil E.L. and Boulding K.E. [Eds.] 1996. "Evolution, Order and Complexity". Routledge Publ., London

Ostrom T. M. 1988. "Computer simulation: The third symbol system", Journal of Experimental Social Psychology, 24:381-392.

Ramakrishnan R. and Gehrke J. 2000. “Database Management Systems” (2nd Edition), McGraw-Hill

AUTHOR BIOGRAPHY

MARCO REMONDINO was born in Asti, Italy, and studied Economics at the University of Turin, where he obtained his Master Degree in March, 2001 with 110/110 cum Laude et Menzione and a Thesis in Economical Dynamics. In the same year, he started attending a PhD at the Computer Science Department at the University of Turin, which will last till the end of 2004. His main research interests are Computer Simulation applied to Social Sciences, Enterprise Modeling, Agent Based Simulation and Multi Agent Systems. He has been part of the European team which defined a Unified Language for Enterprise Modeling (UEML). He is also participating to a University project for creating a cluster of computers, to be used for Social Simulation.

ALESSANDRO CAPPELLINI, author of the basic implementation in JAS, is a PhD Student in Simulation, University of Turin, Italy.