# Quick Prototyping and Simulation with the INGENIAS Agent Framework

Jorge J. Gomez-Sanz
Facultad de Informática
Universidad Complutense
Madrid, Spain
Email: jjgomez@sip.ucm.es

Carlos Rodríguez Fernández
Facultad de Informática
Universidad Complutense
Madrid, Spain
Email: carlosrodriguez@computer.org

Juan Pavón
Facultad de Informática
Universidad Complutense
Madrid, Spain
Email: jpavon@fdi.ucm.es

***Abstract*—A major nightmare of software developers is having clients claiming the delivered software is not what they expected. Developers make an extensive use of prototypes to prevent such situations to occur. However, the role of simulations have not been studied enough. If an agent based simulation is intended, not all existing agent oriented methodologies are capable of it. It requires code generation capabilities, round-trip features, and components with predefined behaviors. This work introduces the ability of the INGENIAS Agent Framework to simulate MAS specifications and how this can be useful for for agile development of agent-based applications.**

## I. Introduction

A software development needs to verify the system under construction satisfies a client needs. Requirements engineering has contributed strongly to the solution by guaranteing the requirements are properly captured and processed. Software development processes have been modified to include the ideas of incremental development (segmenting the system functionality into coherent pieces and ordering them so that their sequential/paralell realisation leads to the final system), iterations (miniprojects focusing on the development of a concrete piece of functionality of the application). Expected products from this development process include several types of documents aiming to communicate with the client and tell them what is being done; and acceptance tests, to ensure the behavior of the application is the one expected. Despite these avances, prototyping seems to be a recurrent option.

Prototypes [1] have been extensively used as a effective mean of showing results to the client and for experimenting with part of the functionality the client demanded. However, the development of a prototype is incomplete without a proper environment that resembles the platform where the software will be deployed. Hence, essaying with different software architectures or showing a client how the software is expected to behave is expensive.

In this scenario, agent technology has been recognised as an affordable mean for producing prototypes and define simulations [2]. Nevertheless, actual prototyping and simulators are built ad-hoc. The facilities needed to facilitate rapid application development where a simulation is required are not present in most agent oriented methodologies. Remarkable exceptions are ADELFE [3] and PASSIM [4], which integrate simulations into their development cycles. However, as discussed in section VIII, more effective code generation and specification-code synchronization facilities are needed.

In this aspect, INGENIAS [5] can improve existing proposals. INGENIAS is an agent oriented software engineering methodology which follows the model driven development paradigm [6]. As a result, it considers the MAS specification as the main product of the development and provides tools to transform this specification into executable code. INGENIAS takes advantage of the INGENIAS Development Kit for producing fully functional systems [7]. Within this kit, there is a distinguished module, the INGENIAS Agent Framework which determines how to interprete the MAS specification using as target platform JADE. This module is complemented with the Code Uploader and the AppLinker modules that provide round-trip engineering features, i.e., they upload changes made in the code back into the specification.

This paper discusses how INGENIAS can make use of the simulation concept in order to develop a software system. The discussion bases in the features provided by the INGENIAS Agent Framework, which is introduced in section II. The way this software can be used to simulate is introduced in section III. The introduction of the case study concerns to section IV. The INGENIAS solution to the case study is made in section V. The simulation made in this case study itself is introduced in section VI and evaluated in section VII. The related works are presented in section VIII. Finally, section IX introduces the conclusions.

## II. The INGENIAS Agent Framework

IAF stands for the INGENIAS Agent Framework. It is a framework developed along several years that enables a full model driven development. This means that a developer can focus most of its effort in specifying the system, converting a great deal of the implementation in a matter of transforming automatically the specification into code. This IAF permits to combine the classic approach for coding applications with modern techniques of automatic code generation. The resulting system is almost fully operational, reducing the amount of work of the developer in an relevant degree. Each produced MAS works over the JADE platform. Hence, additional tools existing for this framework can be applied as well.

A MAS in the IAF is constructed over the JADE platform. The MAS can be distributed along one or several containers in one or many computers. To enable this feature, the IAF has means of declaring different deployment configurations.

The running MAS will be connected to several non-agent applications providing the basic services. Hence, if the MAS has to interact with a user, there will be GUIs producing events according to user actions, and defining actuators for agents. These GUIs will be specified as *applications* at the specification level.

An important feature of the IAF is the relevance of interactions, which are considered first class citizens during specification and coding. An *interaction* in runtime is called a *conversation*. The interactions according to the IAF have the main purpose of transferrring information from one agent to another. This information transfer is ruled by timeouts and initiation/colaboration conditions. Also, interactions can be aborted due to failures in the communication or simply because an agent did not answer within the timeout. Finally, the software code realising interactions consider cases where there may be several actors of the same time, i.e., supports the deliver of information to several recipients and the reception of the answer from several agents at the task level.

Tasks are important as well. An agent chooses to schedule a task for execution because the agent wants to attain a pursued goal. The tasks influence in the mental state by removing/adding information, starting conversations with other agents, or modify already existing conversations. Tasks support cardinality attributes associated to the inputs, so a task can use as input all instances of a certain information type or just a few.

Custom deployments permit the developer to define which types of agents will be used and what individual mental state will have during the start-up. This feature allows the developer to define different configurations of the system so that the developer can observe the behavior of the MAS under such conditions.

Testing is a recent addition to the IAF. A developer can define at the specification level what tests will be performed and to what configuration of MAS will be applied. The detailed definition of the test has to be handcrafted, though there are some software libraries that make this work easier.

## III. Building a simulation with the INGENIAS Agent Framework

In general, there are two approaches for simulation using the IAF: simulating the environment or simulating the environment and the application. In the first, MAS infrastructure is provided in order to represent different elements of the target runtime environment. Therefore, there is external software (the one recently developed and whose behavior is to be verified) and the environment built with a MAS. The external software would be docked to the MAS based environment by means of *application* entities. In the second, not only the elements of the application environment are provided, but parts of the system-to-be are included as well, reusing directly predededefined behaviors from the IAF.

### A. Simulating the internals of the application

The IAF provides built-in predefined behaviors which can be used to simulate parts of the application. They are introduced following:

- Task Raw simulation. The developer defines no specific code for tasks, assuming the default behavior of tasks in the IAF. This behavior consists in consuming/reading the information declared as input and producing all the outputs declared in the specification. All produced instances of information entities are empty. This means that if an information entity has attributes, its instances will see these attributes exist, though they have empty values. Also, when a task creates an instance of an interaction, i.e., a conversation, the collaborators will be chosen randomly among existing valid ones (a valid agent type or a valid agent role according to the interaction definition). If one collaborator is defined as having cardinality greater than one, then multiple agents satisfying the requirements from the concrete interaction specification are chosen and incorporated automatically.
- Application Raw simulation. The applications are wrappers for non-agent software. The default code produced for such instances of applications does nothing. Nevertheless, if the specification declared a certain event has to be produced from the application, then the IAF generates a GUI from which the user can trigger the generation of an instance of the expected event. This new instance would be incorporated automatically into the mental state of the agent owning the application. If the specification declares an application type is owned by certain agent type, then an instance of an application can be accesible only to an instance of the agent type or to multiple instances of the agent type (this is implemented by means of a singleton pattern [8]). This is useful to represent shared resources and communication through the environment. In the later case, one agent performs an action over the application that triggers an event which is passed to all agents owning the application.
- Interaction Raw simulation. For each agent capable of initiating a conversation, the IAF generates a basic GUI that can trigger this conversation without having a task launching it. The conversation may not progress if the corresponding information that should be delivered/received does not exist. Therefore, a simulation of the interactions must be accompanied with a specialized deployment where the initial mental state of involed agents satisfies the requirements of the interaction at the specification level.

Using these pre-defined behaviors, the developer models the internals of the application using an agent oriented methodology, like INGENIAS, and proceeds to observe how the resulting MAS behave.

The simulation can get closer to the actual intended software by customizing more the behavior of the elements:

- Adding custom code to the tasks in the MAS. Instead of the default behavior (consuming input entities and creating new ones in the output), a developer can code more concrete behavior, like using existing APIs from applications to perform actions or detail which collaborators a conversation will have. This new code is inserted into a code component entity and used to replace the code in the generated task. These changes are maintable throught he use of the code uploader module of the INGENIAS Development Kit, which migrates changes made to tasks into existing code components in the specification.
- Adding initialization/shutdown code for applications. This code permits to properly construct/shutdown the application knowing during creation/shutdown time which agent will be assigned to it. This way, an application acquires a reference to its agent or agents.
- Modifying the API of the application code and providing a body to the methods. An application can be coded adhoc for a concrete development or act as mediator [8] to some external software (e.g. a database or some existing GUI). The API is synchronized with the specification by means of the AppLinker module of the IDK. This module analyzes the generated code of already generated applications looking for changes in the API with respect to the API stored in the specification. Differences are merged automatically so that the API in the specification is the same. This way, a developer can either modify the specification and let the system regenerate the code, or modify the API in the code and upload the new methods to the specification.

The degree of customization is a decision of the developer. The resulting MAS can become the intented system or remain as a proof that the MAS specification is valid for the problem. In the first, case, the specification and the customization of the generated code would progress towards the final system. In the second case, the developers would decide to realize the specification into a different agent platform or just reuse the acquired knowledge to be used within another methodology (agent oriented or not).

The IAF recognises automatically generated code, manually maintained code, and a hybrid mixture of both. All of them exist into separated folders. Hence, a developer can work safely in the *src* folder, creating clasess regularly; delete safely the content of *gensrc* folder knowing that it can be completely generated from the specification; or customize the content of *permsrc* folder knowing that changes made will not be overwritten by sucessive code generation requests.

### B. Simulating the environment

Once key elements to be simulated are identified, a developer can represent them as agents or as applications. It is an agent when its behavior of the simulated entity can be captured with goals and tasks and the execution of the corresponding tasks corresponds to the achievement of goals. It is an application in other case. Being an application means an API is offered and that this API can be used within the tasks of the agent. Optionally, the application is expected to produce events in order to notify agents of changes. As explained in the previous section, these events can be asserted within a single or multiple owners.

The external software (software which already exists before the current MAS is developed) is wrapped into applications. Generated code for applications will act as a mediator [8] between the generated MAS and the external software. The initialization code for applications will be used to connect the mediator with the external software, while the shutdown code will do the opposite. These appliciations should offer the same API the external software does. If there are multiple APIs, a developer can choose to merge all of them into a single application or creating multiple applications holding each one separately.

Most likely, there will be agents representing the different types of users in the system, which could be humans or not. A developer will define a certain role capturing the generic behavior expected from that user. This behavior is supposed to be assumed by a agent or specialised by another role. Assuming there is a role with a task X having as input an entity type Y, the specialization can happen as follows. First, an agent can define a task XX having as input the entity Y. This will cause tasks XX will be executed instead of tasks X. Second, a new role extending the original role can be defined. This new role, can define a task XXX having as input the entity Y. This will cause tasks XXX are executed before the task X. In both cases, when task XX or XXX is executed, they, probably, will remove entity Y from the mental state, aborting this way any possible execution of previously scheduled X tasks.

With these two ways of capturing behaviors, a developer can create populations of users with varying behaviors. The resulting agents will perform actions over the existing applications, which are supposed to be connected to the software system currently under execution.

### C. Analysing a simulation run

Once the MAS is defined, it is time to perform different runs of the system. The user can inspect visually the results by means of the IAF default GUI. Nevertheless, it is convenient to make use of a more exhaustive and objective analysis by means of studying the system logs. The INGENIAS Agent Framework produces logs with the produced events in a system run, so that they can be inspected and accounted later on. Registered events are:

- A task has been scheduled. The id and type of the task, as well as the id of the agent, are provided
- A new piece of information is added/removed to/from the agent mental state. The id and type of the entity as well as the id of the agent are provided.
- A task has been executed. The id and type of the task, as well as the id of the agent, are provided

- A task has been aborted. The id and type of the task, the expected inputs that were missing, as well as the id of the agent, are provided
- A conversation has been started. The id of the conversation, the interaction type, collaborators, and the id of the launcher agent, are provided
- An agent decides to participate into an requested conversation. The id of the conversation, the interaction type, collaborators, and the id of the launcher agent, are provided
- A message has been delivered. The id of the message, its content, sender and receivers are provided

Each event is marked with a timestamp (24 hour format and milliseconds format) obtained from the same hardware clock. Therefore, this timestamp should not be used as a reference to compare logs produced into different physical machines.

By properly interpreting each task, the developer can produce graphics similar to those frequently found in conventional simulations. For example, a task Y is designed to peform a payment through paypal in ebay after a succesfull auction. By accounting the times this task was executed, and using timestaps, it can be generated a graphic of the number of finished auctions through a determined period of time.

## IV. The case study

Technological advances are increasing with time. The volume of scientic publications patents, research projects, technology news and related international standards of technology is in continuous increase. This makes available to researchers, R+D organizations, and industry in general, a huge amount of information to analyze for their projects and strategies. Technology Watch Systems are involved in processing of all information tech- nology environment to extract knowledge, such as identifying trends and changes. This case study focuses on the management of quality of information sources within a Technology Watch System.

Technological watch is a tool used wihtin *Competitive Intelligence*. Competitive Intelligence is the legal obtention, analysis, distribution of information about a competitive environment, including strong and weak points as well as the intentions of competitors [9]. Technological watch is an organized, selective and permanent process for information gathering scientific and technological information coming from in and out the organization; selecting it; analyzing it; distributing it; and commnunicating it; to convert it into knowldge supporting decision making activities with lower risk and being able to anticipate changes [10].

In an organization dedicated to R+D, clients of a Technological Watch are research groups. These researchers, generally, have already located relevant information sources to be watched. Therefore, researchers are a potential suppliers of information sources. If the system was feeded with bad quality information sources, the system would supply results with noise causing the analyses to be inadequate or wrong. This problem suggests the system should not accept all suggested information sources. In fact, giving more relevance to those researchers investing effort in providing good information sources would potentially increase the quality of the produced results. Similarly, controlling more bad suggestors allows to keep the quality degree.

The development of Technological Watch system follows recommendations from the UNE 166006:2006 EX [10]. This normative provides a set of requirements, but no APIs or formal definitions. One of the recommendations consists in qualifying information sources with some attributes, which are are highly related to the reputation and trust models well known in the agent literature. These attributes are supposed to be managed by humans, what means necessarily increasing the amount of work of operators.

Therefore, this case study to what extent reputation and trust models can be integrated in a Technological Watch System. The question is how such functionality can be integrated, i.e., are new components needed? is it enough with modifying the responsibilities of existing components?

This evaluation has required building a prototype dealing with three basic scenarios corresponding to the use case *Manage quality of information sources by means of reputation and trust models*:

- A low quality information source proposal made by a collaborator with low reputation in general
- A high quality information source proposal made by a collaborator with high reputation in general
- A low quality information source proposal made by a collaborator agent subject of bad reputation on behalf a supervisor agent which has witnessed past requests from the same agent.

The development made focuses in the first scenario.

## V. Defining with INGENIAS

In the long term, the developed MAS aims to discover what kind of agents are required in order to have a population representative of a real scenario. Also, As it is now, it serves to experiment with the necessary protocols for integrating a trust model in the information sources management mechanisms.

As figure 1 shows, there are four groups of agents in the organization responsible of technological watch services.

- **Collaborators.** They are agents which propose new information sources. These agents can be a human operator representative or, directly, an agent that does not require a human operator.
- **Supervisors.** They are resposible of deciding how to evaluate proposals from the collaborators.The evaluation itself is performed by agents belonging to the Test Team.
- **Test Team.** They assign a quality value to an information source prior to its incorporation into the system. They do this by pretending the source is already incorporated and starting to use it with some predefined queries. Also they can request human expert evaluations.
- **Operations Team.** They watch accepted information sources and other technological watch services. There are agents within this group who are in charge of inspecting accepted information sources. They maintain

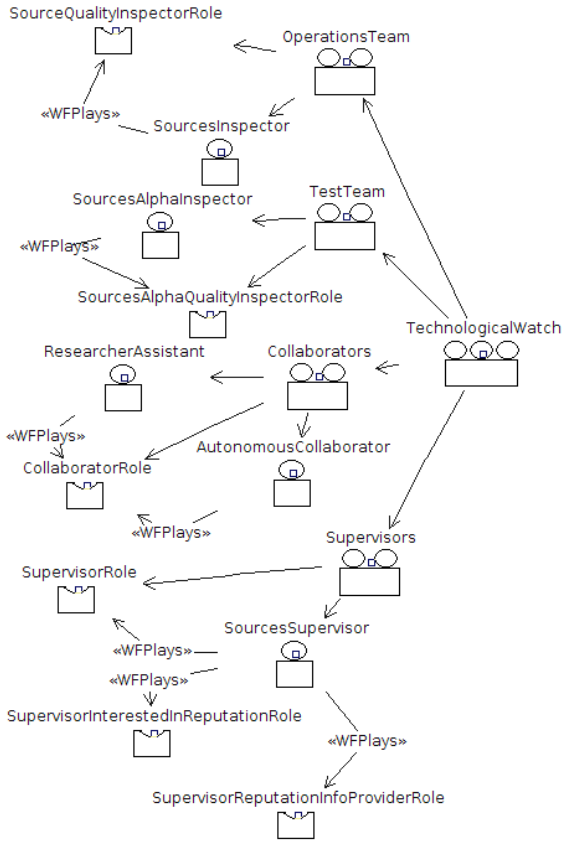


Fig. 1. Organization responsible of Technological Watch

updated information about the quality of the sources. This evaluation is used later to compute the trust degree of collaborators.

The MAS developed uses the REGRET trust model [11], making two simplifications. All supvervisors have a credibility of 1 and only reputation information from witness will be taken into account.

Focusing on a *SupervisorRole*, this role has as goal keeping the system with high quality information sources. The capabilities of this role are expressed as tasks (fig. 2):

- **ProcessReceivedProposalTask**. The agent can process proposal from collaborators.
- **AddSourceIntoSystemTask**. The agent can add accepted information source into the system. This task also includes the request of quality inspection for the accepted information source.
- **RequestAlphaQualityInspectionTask**. The agent can request alpha quality inspection to inspectors. "Alpha quality inspection" means making a quality inspection without adding the information source into the system to be watched.
- **ProcessAlphaQualityInspectionResultTask**. The agent can process the results of alpha quality inspections.
- **ProcessQualityInspectionResultTask**. The agent can process the results of quality inspections.

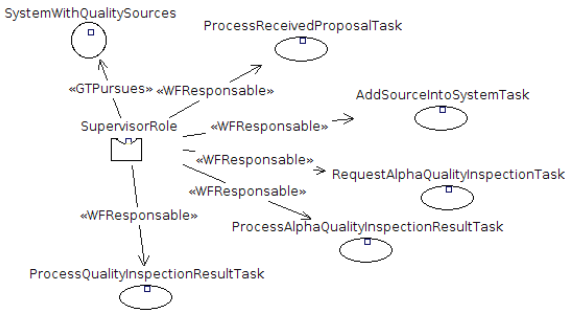


Fig. 2. Tasks assigned to a Supervisor

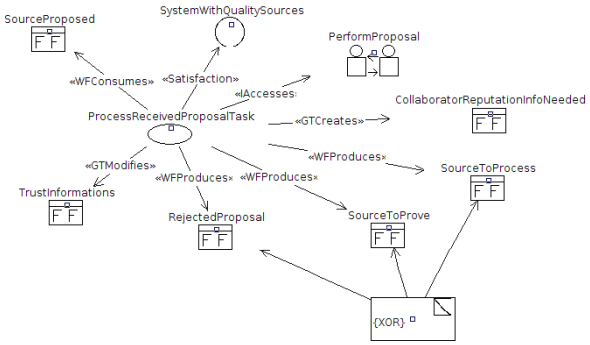


Fig. 3. Description of the task which process an information source proposed

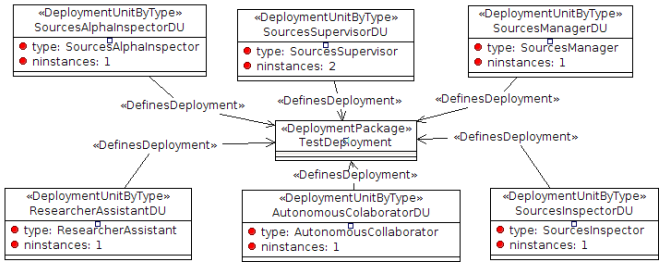


Fig. 4. Definition of the deployment for the test

Task *ProcessReceivedProposalTask* (fig. 3) is activated when the fact *SourceProposed* is found. The *SourceProposed* fact comes in the `proposal` message of the *PerformProposal* conversation. This task makes decisions about what filters apply to the proposal as follows:

- Reject the proposal because the collaborator has attempted too much to add information sources with low quality. The task produces the *RejectedProposal* fact to be sent as response in the `reject-proposal` speaking act of the *PerformProposal* conversation.
- Request an alpha quality inspection for the information source in order to obtain quality information in some criterias. It is because the supervisor doesn't trust in the collaborator about the quality (in some criterias) of information sources which he usually proposes. The objetive behind is to apply the selected filters, that is, *If the information source has the quality value (in a specific criteria) less than the minimum quality value permited (in*

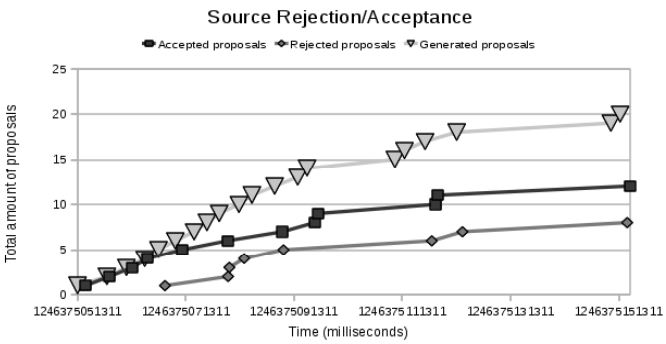


Fig. 6. Information source acceptance/rejection ratio

*the specific criteria), then reject the proposal. Otherwise, accept the proposal*. The task produces the *SourceToProve* fact to indicate the request.

- Request the adding of the information source into the system to be watched. The proposal is accepted because the supervisor trust in the collaborator. The task produces the *SourceToProcess* fact to indicate the request.

The *TrustInformations* fact has the information about trusting of all collaborators who have made proposal to the supervisor.

## VI. Simulating

Figure 5 shows the GUI of the developed system where different actions can be triggered to run the basic scenarios. The log corresponds to these sequence of actions. First, the Collaborator makes a proposal of a information source with regular quality.The Supervisor #0 applies the filter, it's mean, request an alpha quality inspection to the Alpha Inspector. Also, The Supervisor #0 requests reputation information to the Supervisor #1, and the later response with the reputation information which has a high reliability. The Supervisor #0 accepts the proposal. Second, the Collaborator makes a new proposal with low quality. The Supervisor #0 doesn't trust (trust degree information with high reliability) in the Collaborator, then the agent decides to apply the filter to the proposal, and finds that the proposal has low quality and must be rejected.

The simulation of the system leads to a log of several megabytes of information. By filtering the content, and focusing in the production of *AcceptedProposal* entities, *RejectedProposal* entities, and *SourceProposal*, it is straightforward to obtain a list of events which tell when such entities are incorporated into each individual agent mental state. These entities are representative of a rejection, acceptance, and proposal of new information sources. Hence, accounting occurrences, one can determine the performance of the system. As figure 6 indicates, there are rejected sources and accepted sources. Hence, the rejection mechanisms are used. Nevertheless, it would be necessary to determine if the rejections should actually happen, something not accounted here.

## VII. Evaluation

After developing this prototype, a greater knowledge of the problem has been acquired. Using INGENIAS, generic information exchanges were depicted, detailing the information exchanged and having some wired code dealing with its transformation.

The development time was reduced to a minimum, one person for one weeks at full time (eight hours a day). Taking into account that the actual Technological Watch system has been developed for two years, this seems a reasonable price for having an accurate specification of the problem. Also, incorporating the produced system as an add on to the real system remains a possibility.

The simulation of the environment was straightforward to produce. Since the protocols were already established, it was known what information the system was expecting from the users. So, defining GUI agents interfacing with real human operators or agents pretending to act directly with the system was easy. From here, deciding the kind of agents to deploy and their specific features, was a matter of depicting an ingenias deployment entity.

The degree of reusability is not known yet. The model was concluded recently and it is currently under the evaluation of other partners in the project. Should the specification be accepted, the ideas woud be incorporated into production, modifying the current Technological Watch system being used.

## VIII. Related work

The current state of art of agent oriented software engineering methodologies shows only a little number of methodologies permitting to produce directly code and perform the kind of simulations made with INGENIAS. Methodologies like MaSE [12] or Prometheus [13] are capable of code generation. Nevertheless, they intend to produce fully functional systems everytime and do not conceive the use of simulations as part of the development. In the case of Prometheus, code generation is a recent incorporation so its effective use to produce MAS automatically is still under study (there is a plugin but the documentation for its use has not been update as of today). In the case of MaSE, code generation has been integrated since the methodology was born. Systems are specified almost completely from the tool. Nevertheless, the customization of the produced code is not as effective as in INGENIAS. A developer in INGENIAS will find several tools to synchronize the produced code with the specification. In MaSE, this possibility does not exist.

ADELFE [3] bases on several simulation platforms, one of the most recent is SeSAm. The behavior of agents within SeSAm is made by means of activity diagrams, permitting the developer to express a variety of possible agents. Nevertheless, the production of a SeSAm specification is achieved manually using ADELFE concepts. This complicates the synchronization of both the problem specification and the simulation, something that does not occur with INGENIAS and the IAF. Besides, the effort for defining an agent is lower in INGE-

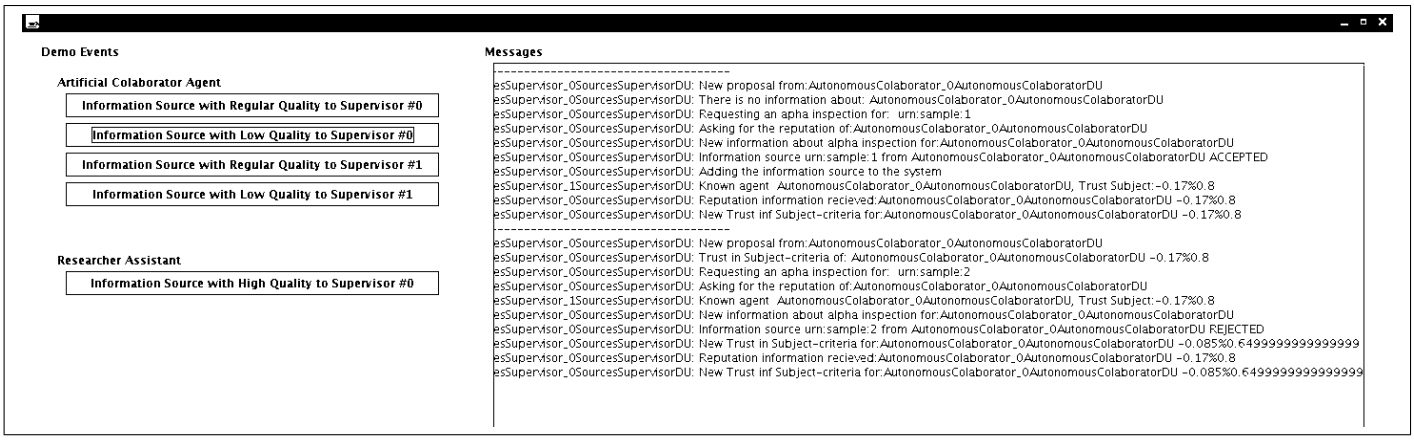


Fig. 5. Agent learns not to trust in another agent by direct experience and reputation

NIAS. Doing the same in SeSAm implies manually modifying the SeSAm agent template to incorporate the activities.

PASSIM [4] uses state-chart based simulation to validate and produce protypes. The formalism used to describe the system to simulate is Distilled StateCharts. The translation between design concepts and simulation concepts is semi-automatic. There is a first stage which produces the skeleton and a second stage that requires human intervention to refine the code. Simulation in PASSIM concerns the whole system. In the work introduced in this paper, the simulation generation is automatic and its content can concern the whole application or only its environment. Also, INGENIAS provides means to integrate with external applications, where PASSIM does not. INGENIAS simulation agents base on the BDI paradigm and are coded that way. In PASSIM, the coding corresponds to the statechart formalism. Like in SeSAm.

## IX. Conclusion

Prototyping and simulations are two ways of clarifying system requirements and experimenting with different approaches in a domain problem. Prototypes are expendable and simulations, depending on the support tool, are expendable as well. A simulation performed with SeSAm, for instance, cannot be used as a final product to be delivered to end users. Hence, an approach permitting prototyping, creating simulations, and reduce costs in producing both, would be welcome.

INGENIAS can provide such services. In INGENIAS, with the aid of the INGENIAS Development Kit and the INGENIAS Agent Framework, it is possible to experiment different configurations of a MAS investing little effort. Also, it is possible to create artificial environments where there are simulated human operators or external agents interacting with the developed system. The result of the experimentation is a MAS specification capturing the requirements of the client, whose interpretation can be inspected visually by the client; and a MAS obtained automatically from the specification, which can be used as prototype or as final system, depending on the needs of the development.

## Acknowledgment

We acknowledge support from the project *Agent-based Modelling and Simulation of Complex Social Systems (SiCoSSys)*, supported by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01. Also, we acknowledge the funding from the *Programa de Creación y Consolidación de Grupos de Investigación UCM-Banco Santander* for the group number 921354 (GRASIA group).

## References

[1] M. Schrage, "Cultures of prototyping," pp. 191–213, 1996.

[2] M. Luck, P. McBurney, and C. Preist, "A manifesto for agent technology: Towards next generation computing," *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 3, pp. 203–252, 2004.

[3] C. Bernon, M. P. Gleizes, and G. Picard, "Enhancing self-organising emergent systems design with simulation," in *ESAW*, ser. Lecture Notes in Computer Science, G. M. P. O'Hare, A. Ricci, M. J. O'Grady, and O. Dikenelli, Eds., vol. 4457. Springer, 2006, pp. 284–299.

[4] M. Cossentino, G. Fortino, A. Garro, S. Mascillaro, and W. Russo, "Passim: a simulation-based process for the development of multi-agent systems," *IJAOSE*, vol. 2, no. 2, pp. 132–170, 2008.

[5] J. Pavón and J. J. Gómez-Sanz, "Agent oriented software engineering with ingenias," in *CEEMAS*, ser. Lecture Notes in Computer Science, V. Maríık, J. P. Müller, and M. Pechoucek, Eds., vol. 2691. Springer, 2003, pp. 394–403.

[6] J. Pavón, J. J. Gómez-Sanz, and R. Fuentes, "Model driven development of multi-agent systems," in *ECMDA-FA*, ser. Lecture Notes in Computer Science, A. Rensink and J. Warmer, Eds., vol. 4066. Springer, 2006, pp. 284–298.

[7] J. J. Gómez-Sanz, R. Fuentes, J. Pavón, and I. García-Magariño, "Ingenias development kit: a visual multi-agent system development environment," in *AAMAS (Demos)*. IFAAMAS, 2008, pp. 1675–1676.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, January 1995.

[9] K. Conttrill, "Turnin Competitive Intelligence into Business Knowledge," *Journal of Business Strategy*, vol. 19, Julio/Agosto 1998.

[10] AENOR, "UNE 166006:2006 EX: Gestin de la I+D+i: Sistema de Vigilancia Tecnolgica," UNE, Final, 2006.

[11] J. Sabater, "Trust and Reputation for agent societies," PhD Thesis, Universitat Autnoma de Barcelona, 2003.

[12] J. C. Garcia-Ojeda, S. A. DeLoach, and Robby, "agenttool iii: From process definition to code generation," in *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009, pp. 1393–1394.

[13] L. Padgham, J. Thangarajah, and M. Winikoff, "Auml protocols and code generation in the prometheus design tool," in *AAMAS*, E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds. IFAAMAS, 2007, p. 270.