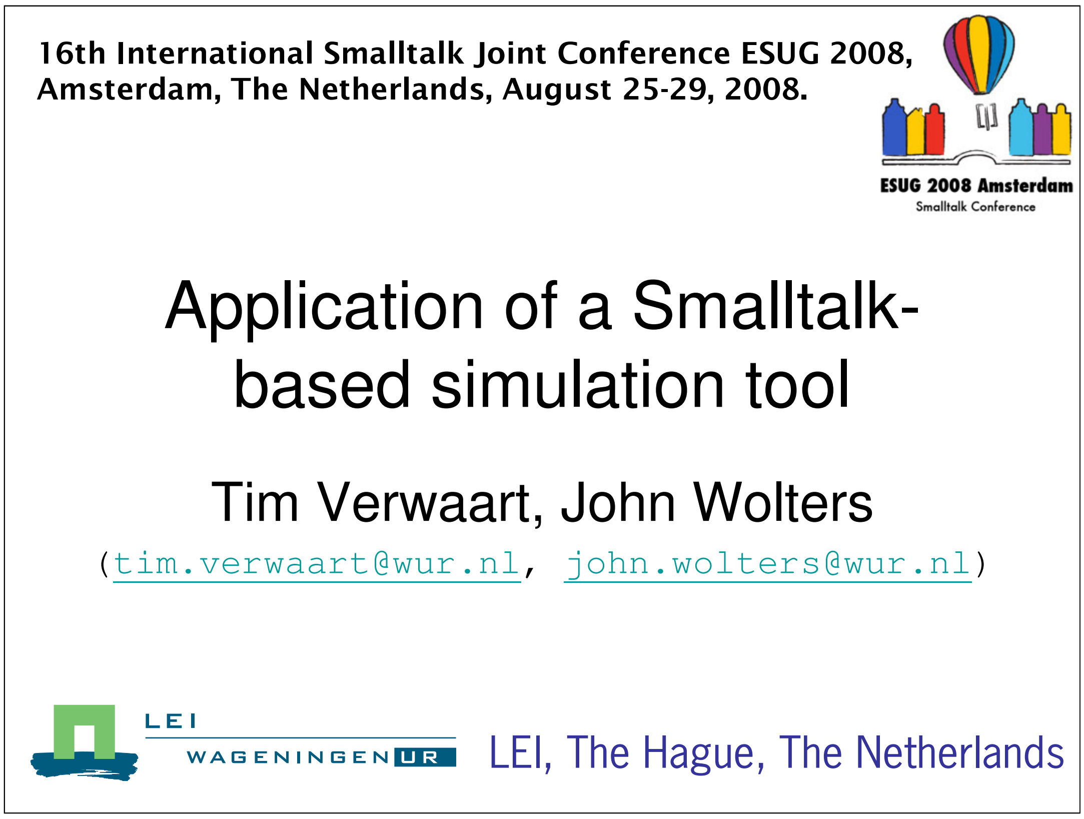


16th International Smalltalk Joint Conference ESUG 2008,
Amsterdam, The Netherlands, August 25-29, 2008.

ESUG 2008 Amsterdam
Smalltalk Conference

Application of a Smalltalk-based simulation tool

Tim Verwaart, John Wolters

(tim.verwaart@wur.nl, john.wolters@wur.nl)

LEI WAGENINGEN UR

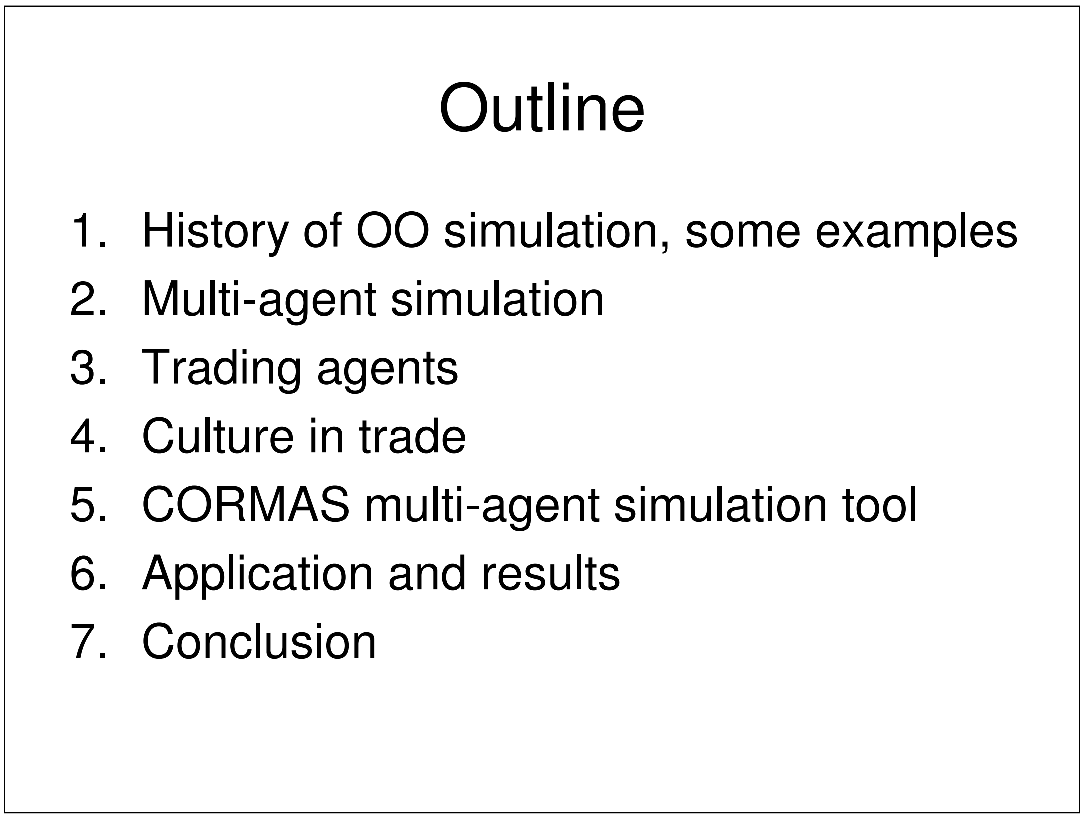
LEI, The Hague, The Netherlands

Abstract

From their origin almost half a century ago, object-oriented programming languages have been applied for simulation of real-world systems and the entities in the systems. The first object-oriented language, SIMULA, was designed as a language for discrete event simulation. SIMULA is as good as dead now, but Smalltalk lives. Numerous examples of simulations in Smalltalk illustrate its suitability for simulation of discrete entities that interact with an environment. This presentation refers to some examples of object-oriented simulations and in particular some successful Smalltalk applications.

A special branch of simulation is multi-agent simulation. Agents are software entities that can act autonomously, are goal-driven, responsive to their environment, and social, i.e. they are aware of the existence of other agents, can share information, and adjust their behavior in response to observed behavior and information of other agents. The natural application area of multi-agent simulation is the simulation of social processes. It can be used for two purposes. The first is to support group decision making in interaction with stakeholders, to show the effect of decisions on the behavior of social systems. The second application area are the social sciences by themselves, where multi-agent simulation models can be used to enhance our understanding of social processes.

This presentation discusses an application of CORMAS, a Smalltalk-based system for multi-agent simulation that has been broadly applied in the past. In the present project it is proven to be a strong tool for rapid development. The subject of the present simulation is: the effect that cultural differences have on trade in international supply chains. We will present how Smalltalk and CORMAS have helped us to rapidly create a model. We will discuss some advantages & disadvantages of Smalltalk for agent modeling.

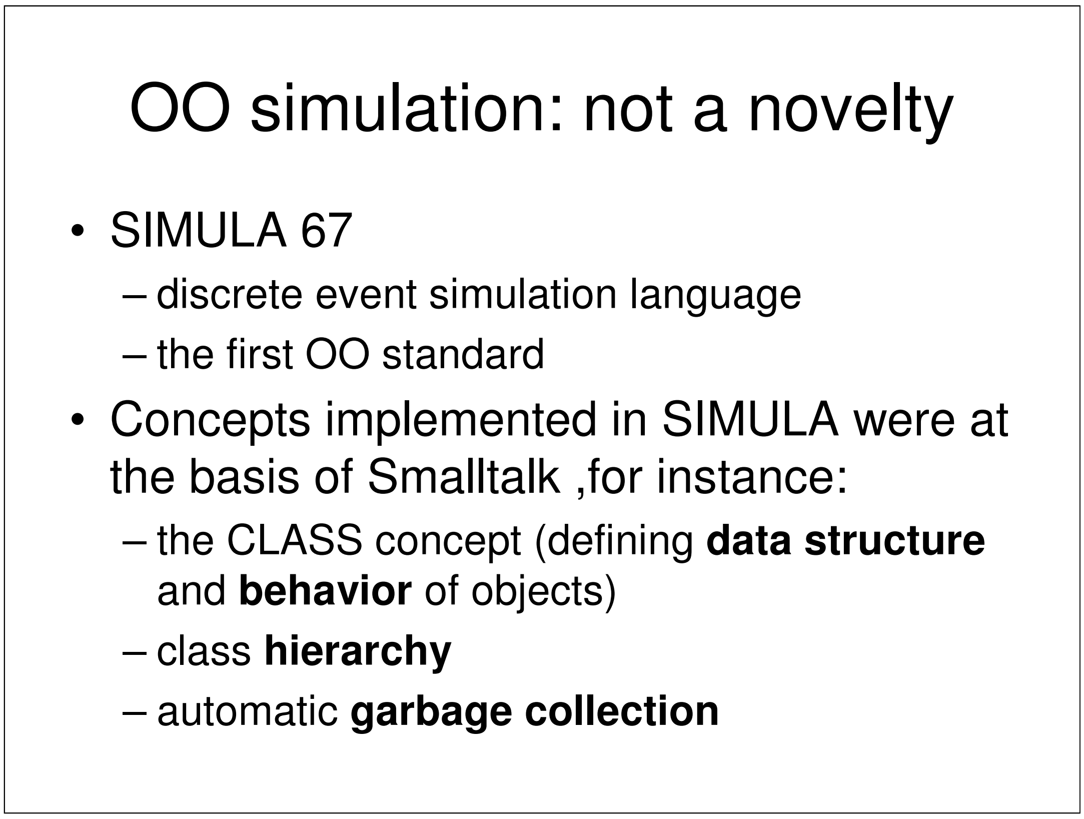


Outline

1. History of OO simulation, some examples
2. Multi-agent simulation
3. Trading agents
4. Culture in trade
5. CORMAS multi-agent simulation tool
6. Application and results
7. Conclusion

The outline of our presentation is as follows.

1. We refer to SIMULA as the origin of object-orientation and discuss a an example of simulation in SIMULA (carwash); we refer to the successful application of Smalltalk for ecological simulations.
2. We discuss what distinguishes multi-agent simulation from other types of discrete-event simulation.
3. Subsequently we present our process model and class structure of trading agents.
4. We introduce the role of culture in international trade; study of the role of culture is the motivation for our simulation work in Smalltalk.
5. We introduce CORMAS, a multi-agent simulation environment developed in France.
6. We explain and demonstrate the application we developed with CORMAS, and discuss some results.
7. We conclude the presentation with a summary of the advantages and disadvantages of Smalltalk that we experienced during the project.



OO simulation: not a novelty

- SIMULA 67
  - discrete event simulation language
  - the first OO standard
- Concepts implemented in SIMULA were at the basis of Smalltalk ,for instance:
  - the CLASS concept (defining **data structure** and **behavior** of objects)
  - class **hierarchy**
  - automatic **garbage collection**

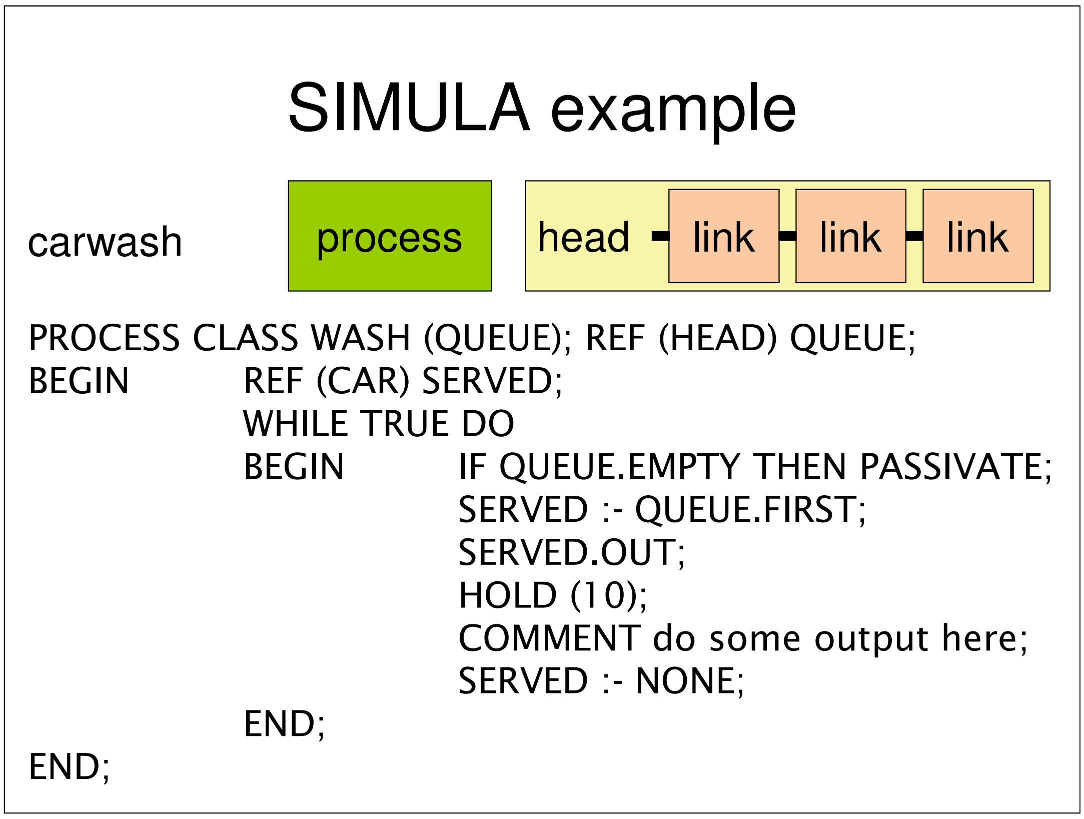
Object-oriented (OO) technology is well-suited for simulation of discrete systems. Discrete systems are systems that contain discrete entities. Discrete simulation entails the building of models of such systems including the entities, and studying the behavior of the entities and the system as a whole in the course of time, given some initial situation and externally generated events. By their nature, software objects are well-suited to represent entities in discrete simulations. In fact, they are invented for that purpose.

The foundations of OO technology are laid in the sixties of the previous century, in SIMULA 67 [1]. The objective of the designers was to create a language for discrete simulation, but they actually created a generic programming language, that used to be deployed for a range of other applications. Many information system represent collections of discrete entities. This explains why the object-oriented paradigm has been so successful in many area’s, and has become the most widely applied approach for software engineering.

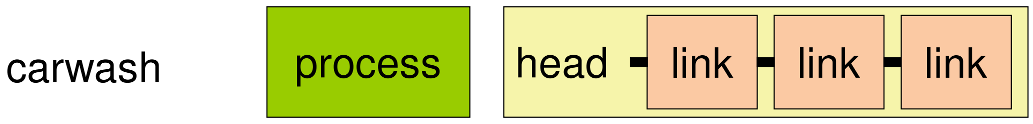
SIMULA 67 includes many concepts and features that are at the basis of modern OO technology. Some of these are the class as a data structure, of which instances can be created dynamically during program execution, the class as a process, linking the behavior of instances to their individual data structures, the hierarchy of classes and subclasses, enabling the inheritance of data structure and behavior, automatic garbage collection, and interactive debugging. Smalltalk and later OO languages, that are not to be named in a Smalltalk conference, borrowed these concepts from SIMULA. To summarize: object-orientation is invented for simulation.

---

[1] Stein Krogdahl (2003) The Birth of Simula. In: Janis A. Burbenko jr. et al. (Ed.) Proceedings of the HiNC 1 Conference, Trondheim, June 2003 , IFIP WG 9.7, in coop. with IFIP TC 3. <http://heim.ifi.uio.no/~steinkr/papers/HiNC1-webversion-simula.pdf>



SIMULA example



```
PROCESS CLASS WASH (QUEUE); REF (HEAD) QUEUE;
BEGIN        REF (CAR) SERVED;
             WHILE TRUE DO
             BEGIN        IF QUEUE.EMPTY THEN PASSIVATE;
                          SERVED :- QUEUE.FIRST;
                          SERVED.OUT;
                          HOLD (10);
                          COMMENT do some output here;
                          SERVED :- NONE;
             END;
END;
```

The SIMULA language has features that ease a simulation programmer's life, like a built-in time scale and random generators for a range of distributions. Furthermore, it has built-in classes PROCESS, LINK, and HEAD. HEAD objects serve as the head of a queue, LINK objects are entities that can display queuing behavior. PROCESS objects act as servers. The use of these will be illustrated in the example of a carwash, adapted from the DECSYSTEM-10 SIMULA Language Handbook [2].

The class WASH is defined as a subclass of PROCESS, having an argument QUEUE. In SIMULA, the value of an argument of a class has to be specified at object creation. In this case the argument refers to an object of type HEAD (SIMULA is a typed language!). The body of a class is activated at object creation, so it serves as the generator method. In the example a local variable SERVED is declared. It refers to a CAR object. This example class has no other behaviors than it's initial (and in this case ever lasting) one. If the queue is empty, the WASH passivates itself, which means that it will wait until it is activated. It knows what PASSIVATE means, because it is a subclass of PROCESS, for which this behavior is defined.

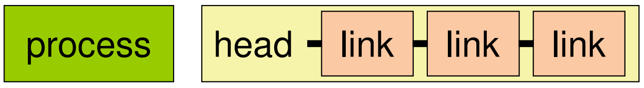
If the queue is not empty, WASH takes the first car from the queue and tells the car to remove itself from the queue (SERVED.OUT). As will be shown later on, a CAR object knows how to remove itself from a queue by the OUT procedure, because it is defined as a subclass of LINK, from which it inherited this behavior. Service is assumed to take 10 time units in this example: HOLD(10). Then it may do some data recording, let the served car fall prey to the garbage collector (SERVED :- NONE), and inspect the queue again.

---

[2] Graham Birtwistle, Lars Enderin, Mats Ohlin, Jacob Palme (1976) DECSYSTEM-10 SIMULA Language Handbook, Part I. Swedish National Defense Research Institute, C8398.

SIMULA example (cntnd.)

```
REF (WASH) CARWASH;
...
LINK CLASS CAR (NUMBER); INTEGER NUMBER;
BEGIN       INTEGER INAT;
            INAT := TIME;
            INTO (CARWASH.QUEUE);
            IF CARWASH.IDLE THEN ACTIVATE CARWASH;
END;
...
CARWASH :- NEW WASH (NEW HEAD);
FOR I := 1 UNTIL N DO
BEGIN       NEW CAR (I);
            HOLD (POISSON(12, U));
END;
...
```



By ‘REF (WASH) CARWASH’ a variable referring to an object of type wash is declared. In the example it is a global variable. It is used in the body of the class CAR, where a car enters the queue of a carwash. By defining CAR as a subclass of LINK, CAR inherits the queuing behavior of that class, so it knows what INTO means. After entering into the queue, the car inspects the carwash. If it is idle, it is activated (cf. the driver blows the horn).

In the main program, not completely shown here, a new object of type WASH is created, with a new queue as its argument, and CARWASH is made to refer to it. Subsequently, the program generates new cars to be washed, with a random time gap, average 12 time units.

The example presented here is a simple simulation of a single server. The SIMULA manual describes how the classes WASH and CAR can be applied to support the decision of the owner of a carwash whether or not to invest in a second carwash unit. The entrepreneur has observed with pain in his heart, and in his wallet, that sometimes cars turn back if their drivers have seen the length of the queue. However, by means of a simulation based on actually observed arrival data he learns that the cost of a second carwash unit will not be compensated by the extra revenues.

The example, almost half a century old now, shows the power of an object oriented programming language to formulate simulations concisely and efficiently. SIMULA is not a popular language nowadays, but the feasibility of object-orientation for simulation remains. In Smalltalk, classes can be just as efficiently defined to represent entities and their behaviors in simulations.

Simulation in Smalltalk

- Some examples in ecological simulation:
  - Baveco c.s. (1992, 1994, 1997)
  - Practical application: ecoduct design

In the nineties, Smalltalk was used for ecological modeling of populations of animals. In this type of simulations, individual animals move on a grid representing a landscape. The simulated individual animals can browse, meet, eat each other, mate, breed, fight, die etc. An example of Smalltalk simulation in ecology is the EcoTalk software of Hans Baveco c.s. [3, 4, 5].

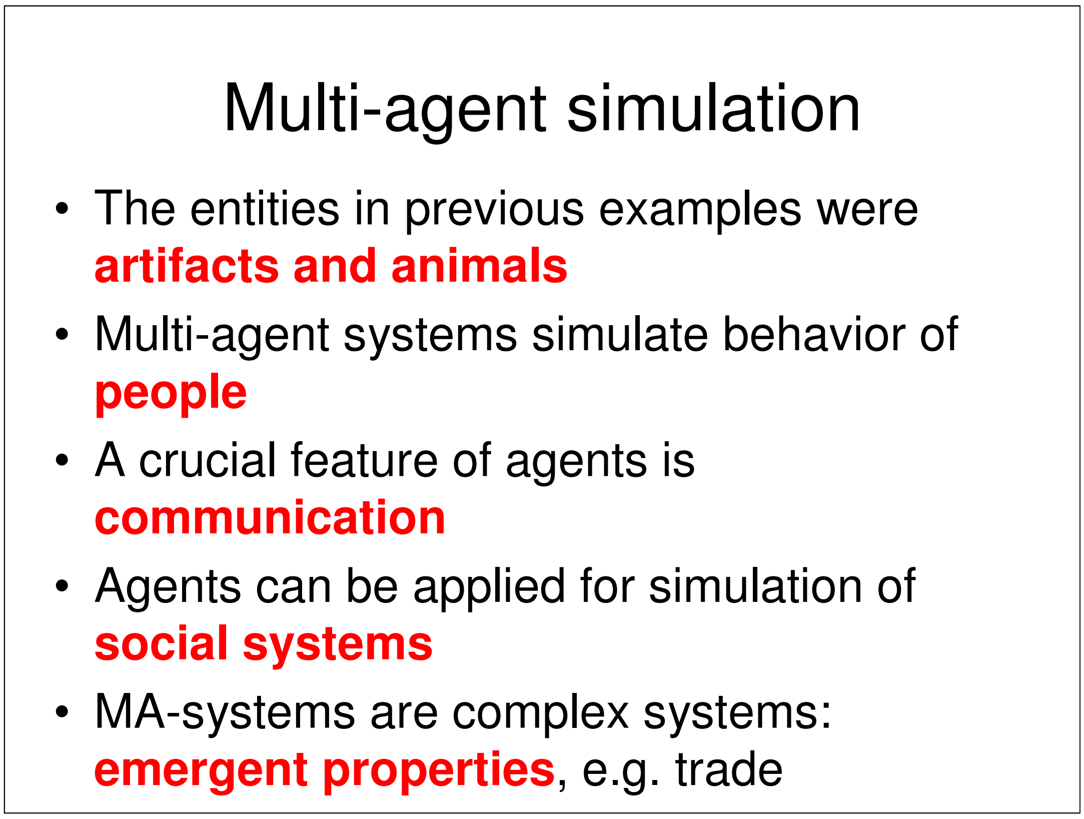
The advantage of simulating individual organisms is that the resulting output does not only represent population size and dynamics, but also represents the geographical dispersion of species, and resulting patterns of mating. It can for instance give an indication if populations are sufficiently mixed to prevent inbreeding. A practical application was the dimensioning of an ecoduct. Before investing in building these game passages over highways, their effectiveness for mixing the populations of particular species on both sides of the highway could be simulated.

---

[3] J.M. Baveco and R. Lingeman (1992) An object-oriented tool for individual-oriented simulation: host-parasitoid system application, Ecological Modelling 61: 267-286.

[4] J.M. Baveco and A.M.W. Smeulders (1994) Objects for Simulation: Smalltalk and Ecology, Simulation 62(1): 42-56.

[5] Hans Baveco (1997) Population dynamics in object-oriented and individual-based models, PhD Thesis, Wageningen, IBN-DLO.



Multi-agent simulation

- The entities in previous examples were **artifacts and animals**
- Multi-agent systems simulate behavior of **people**
- A crucial feature of agents is **communication**
- Agents can be applied for simulation of **social systems**
- MA-systems are complex systems: **emergent properties**, e.g. trade

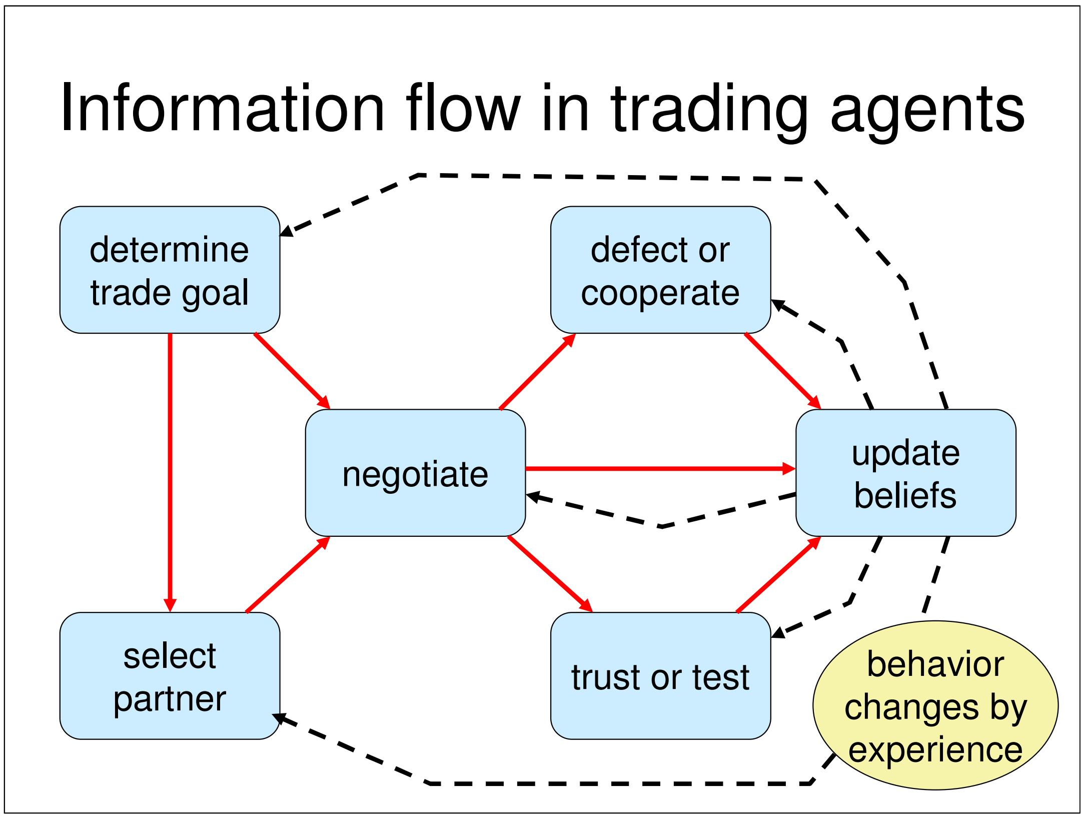
Simulation examples presented so far, concerned the behavior of artifacts like cars and a carwash, and animals in ecosystems. In multi-agent systems the simulated entities in the system include people, or organizations of people. According to Nigel Gilbert [6], a crucial feature that distinguishes agents in a multi-agent simulation from other simulated entities is their ability to exchange messages, and to apply the information gathered from messages - combined with own observations and knowledge - to plan their behavior.

Multi-agent systems can be used to simulate social systems, in which people interact with each other and with their environment. The behavior of such systems is complex in the sense that the behavior of the system as a whole is not a simple aggregate of the decisions of the individuals making it up. For instance, if two individuals decide to buy an apple and to eat it, and only one apple is available, the result at system level will not be the eating of two apples.

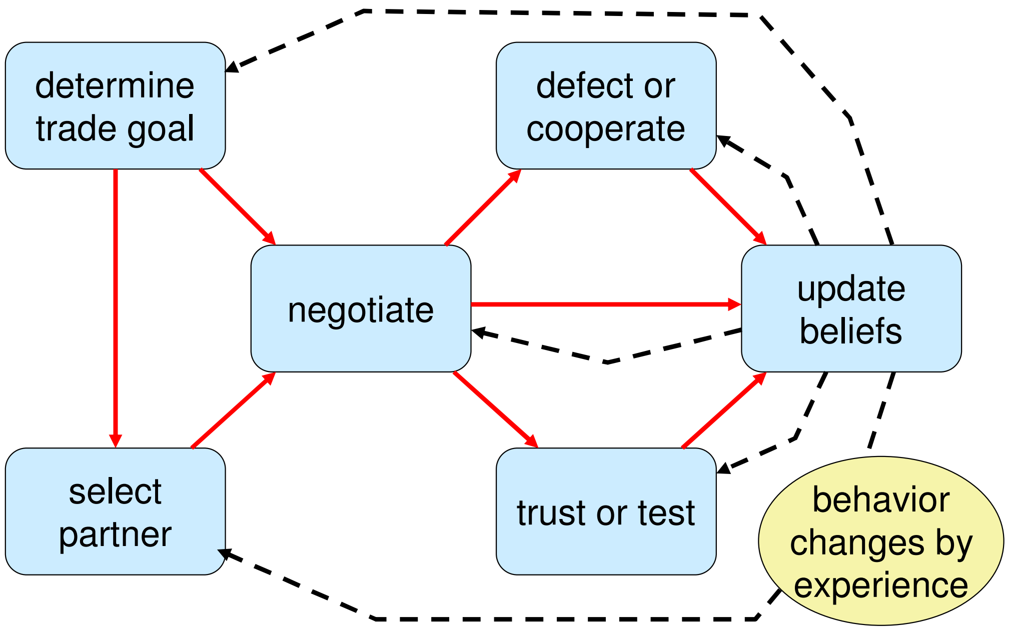
Complex systems have properties that are called emergent. These properties emerge from the behavior of the entities that make the system up, but are not properties of the entities. For instance, trade is an emergent social phenomenon because it is the consequence of individual decisions, but it cannot be a property of a single individual.

---

[6] Nigel Gilbert (2008) Agent-based models. Thousand oaks: SAGE Publications.



Information flow in trading agents



The simulation presented here models trade. The agents represent traders (or trading organizations, that are represented by a single agent). The processes modeled in the artificial agents are the following.

Trade goal determination: the trade goal can for instance be “buy high quality organically grown tomatoes”. It can externally be given, or follow from strategic or tactic deliberations of the agent.

Trade partner selection: given set of the trade goals and beliefs about other agents, possibly based on experience or messages from other agents, an agent can select a trade partner and propose to negotiate.

Negotiation: agents exchange business proposals, including price and other conditions, like information or certificates to be provided with the goods and other conditions like refunds in case of defection.

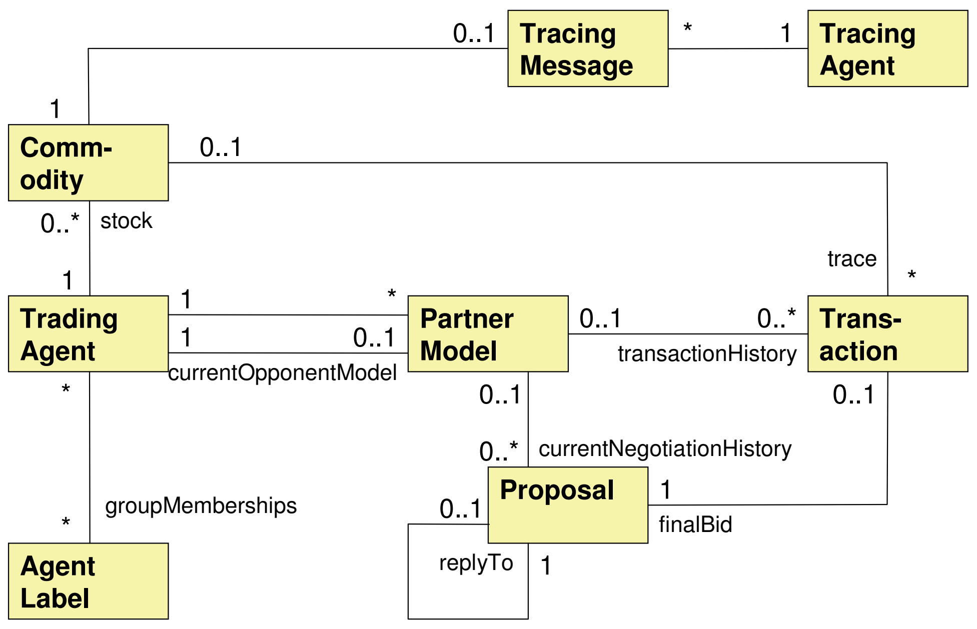
Delivery: when a contract has been closed, it comes to delivery. An agent can either comply with the contract (cooperate) or defect, e.g. deliver traditionally grown in stead of organically grown tomatoes.

Trust or test: an agent can trust its partner to cooperate, or put the deliveries to the test and enforce the punishing of the defector, e.g. take the matter to court. In our simulation an authority called the tracing agent punishes defectors by a fine in case of revealed deceit.

Beliefs update: agents maintain beliefs about market prices, their stock, their financial situation, trustworthiness and benevolence of partners, and the risk of being punished. While trading they may gain experience and receive information from other agents and (failure of) punishment, etc. These updated beliefs have their effects on the other processes.

Conceptually these processes are simultaneous, but the implementation environment of the agents may require an iteration, possibly coordinated by a central simulation clock.

Class structure

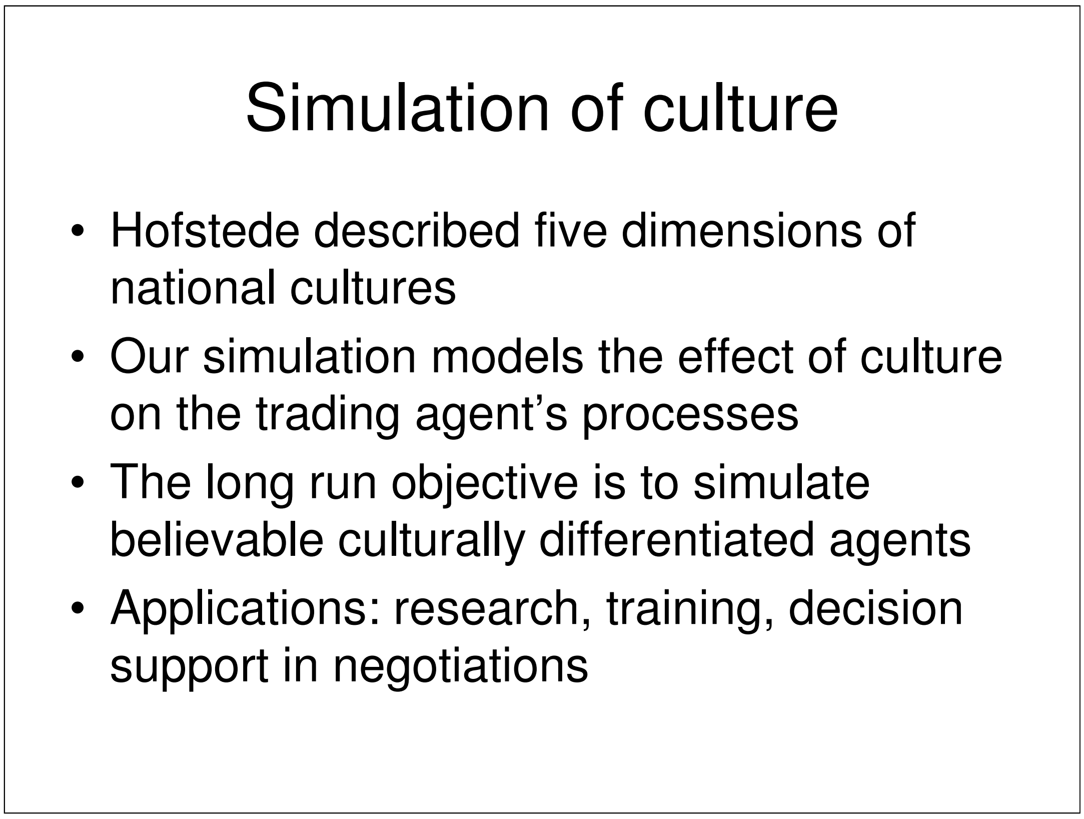


As the model is to be simulated in an object-oriented environment, we need a class model.

Trading agents are represented by objects that maintain partner models of each individual partner. In the partner models they maintain beliefs about partners, for instance about trustworthiness. Furthermore, the history of transactions is recorded in the partner model, as is the history of proposals in the most recent negotiation. The proposals are the actual messages sent to and received from (potential) partners

The agents keep stock of the commodities they possess. Commodities can be delivered to other agents by sending them as the content of a message. If agents send commodities for inspection to the tracing agent, they attach them to the tracing message. The tracing agent records its findings in the messages before sending them to the involved agents.

The trading agents are modeled to have a set of observable labels. They cannot directly inspect each other's traits, such as the actual honesty, but they can assess the value of each other's labels. One of the purposes of the labels is to model culturally relevant agent properties, for instance to indicate group membership and status, so that agents can estimate their group distance and status difference with other agents.



Simulation of culture

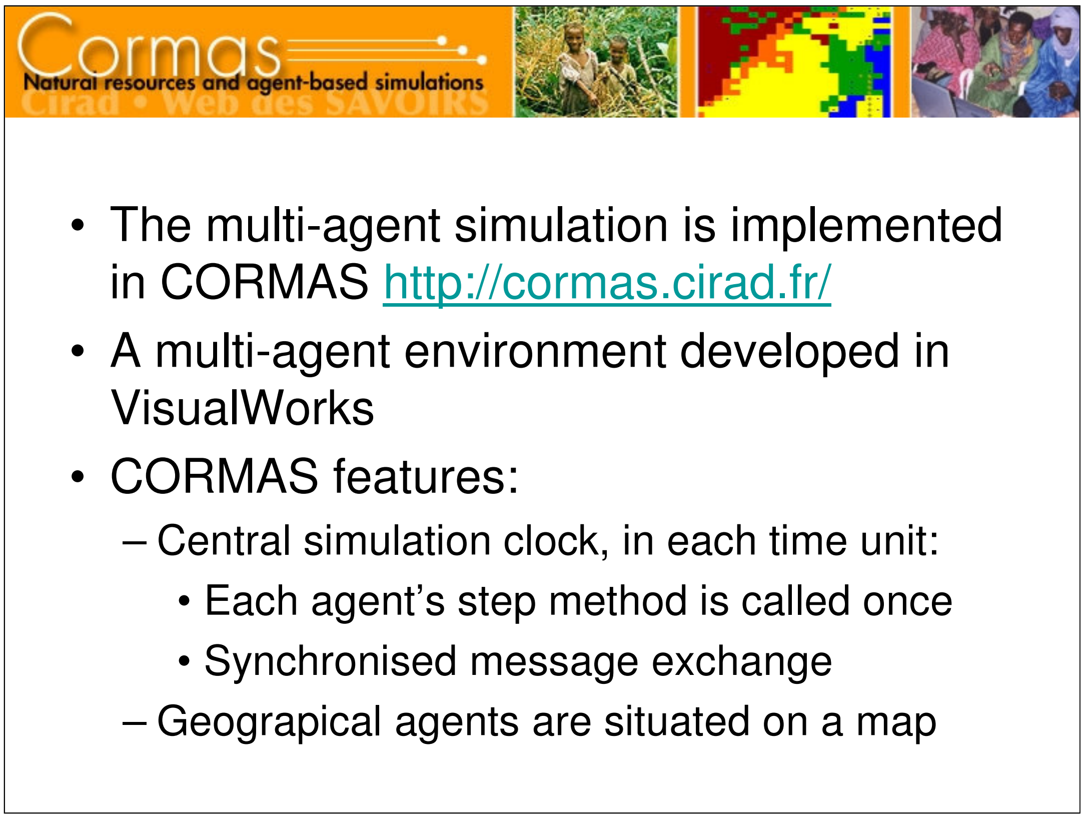
- Hofstede described five dimensions of national cultures
- Our simulation models the effect of culture on the trading agent’s processes
- The long run objective is to simulate believable culturally differentiated agents
- Applications: research, training, decision support in negotiations

Hofstede described five dimensions of national cultures [7]. One of these is the dimension of individualism versus collectivism. Does a person primarily feel to be an individual or a member of a group? Is a person responsible for its own well-being or are group members responsible for each other? Does a person have to keep self-respect, or does a person have to save face for its in-group? Does a trader have to treat all customers equally, or does a trader have to favor in-group members? The distinctions in behavior between individualistic and collectivistic cultures can be modeled in formal rules [8]. These rules can be implemented in agents in a multi-agent simulation, that perform the trading agent’s processes presented in a previous slide.

The log run objective of this work is to simulate believable culturally differentiated agents that can be used in systems for research (e.g., test the effect of different institutional arrangements in intercultural trade), training (e.g., in business schools and multinational companies), and decision support (e.g., personal coaching systems for negotiation support).

[7] G. Hofstede (2001) Culture’s consequences, second edition. Thousand Oaks: SAGE Publications.

[8] G.J. Hofstede, C. Jonker, T. Verwaart (2008) Individualism and collectivism in trade agents. In N.T. Nguyen et al. (eds.) New Frontiers in Applied Artificial Intelligence. Berlin, Heidelberg: Springer-Verlag, Lecture Notes in Artificial Intelligence 5027: 492-501.


Cormas
Natural resources and agent-based simulations
Cirad • Web des SAVOIRS

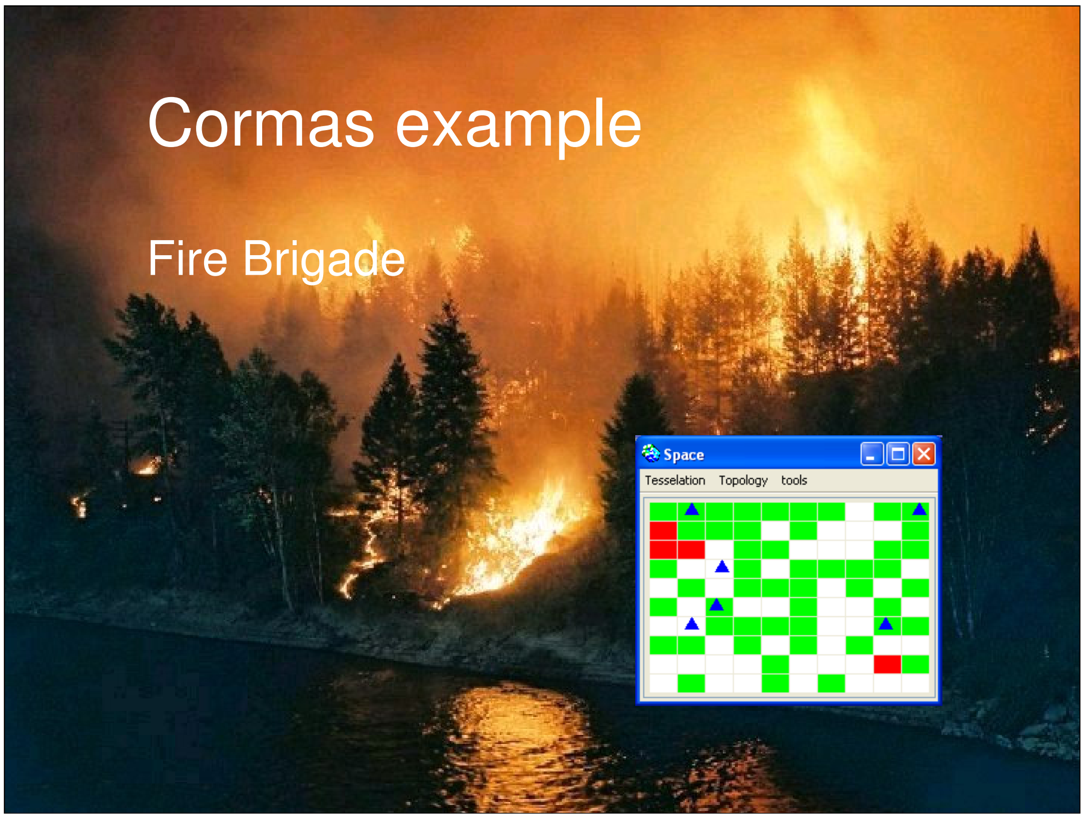
- The multi-agent simulation is implemented in CORMAS http://cormas.cirad.fr/
- A multi-agent environment developed in VisualWorks
- CORMAS features:
  - Central simulation clock, in each time unit:
    - Each agent’s step method is called once
    - Synchronised message exchange
  - Geograpical agents are situated on a map

The simulation is implemented in CORMAS, a multi-agent environment programmed in Smalltalk. It is based on the VisualWorks development system. It has been used many times in a range of applications; most of them are land use and common resource sharing applications. Many interesting examples can be found at the CORMAS site [9].

CORMAS agents are objects in the CORMAS environment. They can inherit many behaviors from CORMAS generic agent classes, such as geographical awareness and capabilities to communicate with other agents synchronized by the central clock. The central clock’s unit of measurement is a “time step”, which may represent any real time interval in the simulation. Every agent has a step method, that is called by the system exactly once in every time step. The synchronized communication system intercepts message sent in a time step an delivers them in the addressee’s mailboxes between two time steps, so that in each time step agents can collect the messages sent to them in the previous time step.

The CORMAS environment provides a range of methods to visualize the progress of the simulation and the current state of variables.

[9] CORMAS. <http://cormas.cirad.fr/ >



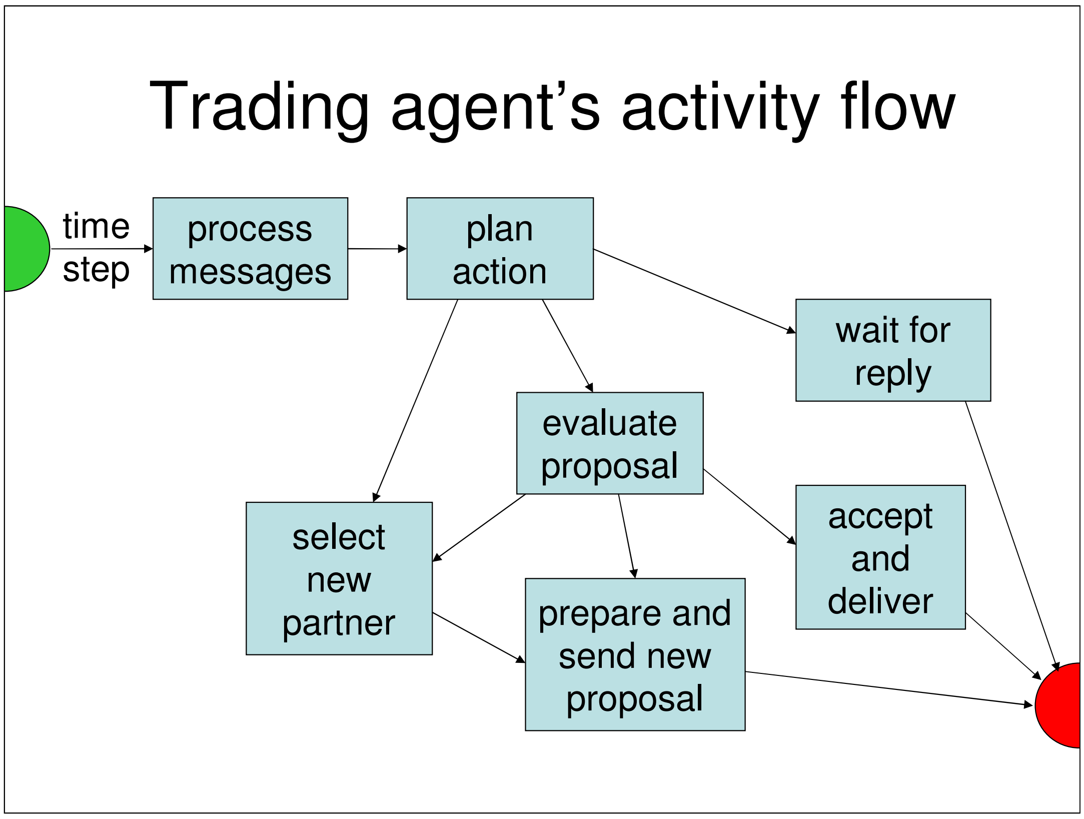
Cormas example

Fire Brigade

An example of CORMAS from the ‘CORMAS tutorial 1’ manual is a fire brigade model where the agents fight forest fires. The agents are located on a grid that represents the environment. In the fire brigade model, the fire fighters are initially located at random, and they move randomly until they spot a fire or receive a fire message. The user can start fires on the grid. As soon as a fire fighter spots more than a (from beforehand) specified amount of fires in its neighborhood, it sends a message with the coordinates to the other fire fighters to come and help. The fire fighter’s step method is as follows:

```
step
    self readMaill.
    self alarmCallingAggregate.
    self extinctFire
```

CORMAS has classes that define the space (grid) and the symbols that indicate the conditions of cells (empty, fire or tree) on the space and the locations of agents (fire fighters). Standard interfaces are provided to initialize the space and to generate events (in this case, start a fire). Visualization tools are included. This simulation uses the tools “probes”, “messages”, and “space”. The user can observe the simulation’s progress. The “space” view dynamically depicts the condition of the cells and the location of the firemen. In this view, the user can observe the fires and the movements of the firefighters.



Trading agent’s activity flow

As our simulation is implemented in CORMAS, an agent’s behavior is implemented in the step method.

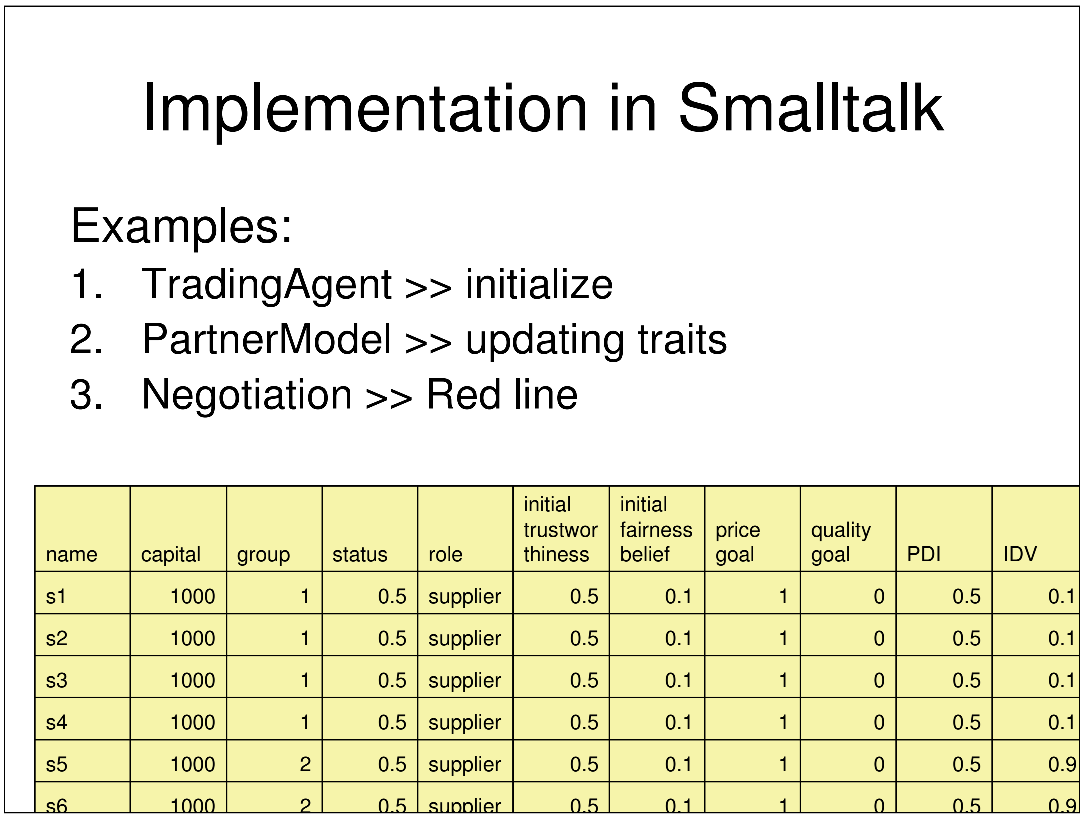
All messages sent by other agents in the previous time step have to be processed. For that purpose the agent first checks if it has received tracing reports. It uses these reports to update beliefs. In case the agent has bought the goods, it uses the tracing information to update its belief about partners trustworthiness and about the quality of goods in stock. In case the agent has sold the goods, it uses the tracing information to update the belief about the other’s benevolence and, in case of a fine, resolve to be honest in the future (but this honesty decays until renewed punishment occurs).

After processing the tracing reports, the agent processes deliveries of goods, and decides if it will trust or trace them. Finally, all received proposals are archived. This information can be used for future partner selection.

After processing all messages, the agent plans its actions. If it has no ongoing negotiations, it selects a new partner, using beliefs that result from previous negotiations. If a negotiation is going on but no reply was received, the agent will wait if the partner has not had sufficient time to reply; otherwise it will update its partner beliefs and select a new partner.

If a reply to an own proposal has been received, the agent will evaluate it, and decide to either accept the proposal, or to break-off, update partner beliefs, and select a new partner, or send a new proposal in reply and wait for the answer. When the agent has accepted a proposal, it updates its beliefs about the partner and either (if it is buying) send a confirmation or (if it is selling) decide if it will deliver truthfully or deceive, and act accordingly.

The decision taken by the agents and the belief updates are parameterized by culture. To the extent that agents are configured as members of a collectivistic culture, group membership and group distance between an agent and its partner play a role.



Implementation in Smalltalk

Examples:

1. TradingAgent >> initialize
2. PartnerModel >> updating traits
3. Negotiation >> Red line

name	capital	group	status	role	initial trustworthiness	initial fairness belief	price goal	quality goal	PDI	IDV
s1	1000	1	0.5	supplier	0.5	0.1	1	0	0.5	0.1
s2	1000	1	0.5	supplier	0.5	0.1	1	0	0.5	0.1
s3	1000	1	0.5	supplier	0.5	0.1	1	0	0.5	0.1
s4	1000	1	0.5	supplier	0.5	0.1	1	0	0.5	0.1
s5	1000	2	0.5	supplier	0.5	0.1	1	0	0.5	0.9
s6	1000	2	0.5	supplier	0.5	0.1	1	0	0.5	0.9

In the presentation, three code examples are explained.

1. Trading agents are initialized from two excel files, one containing agent parameters and the other defining every agent's initial stock.

2. An example of modeling of an agent's culture-dependant evaluation of a partner:

```
acceptabilityOf: aTradingAgent
    |pm|
    ^self idv * (pm := self partnerModels at: aTradingAgent name) fairness
        + ((1.0 - self idv) * (pm trustworthiness max: 1.0 - (pm groupDistanceWith: self)))
```

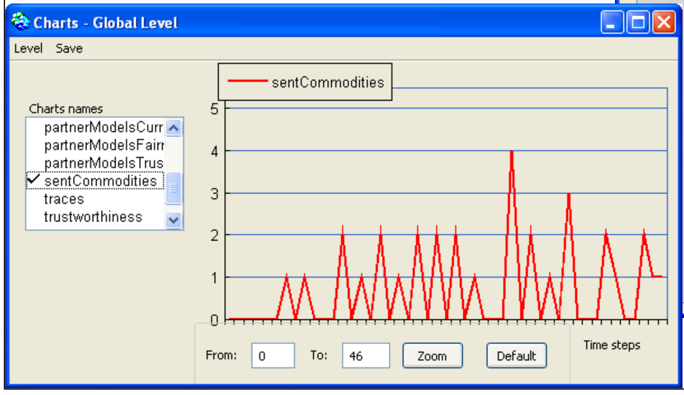
In this formula, *idv* represents the individualism index in the interval [0, 1]; for individualistic agents, trade partners are acceptable if they are believed to be fair traders; collectivistic agents accept partners that belong to the same group, or with whom they have developed a trust relationship.

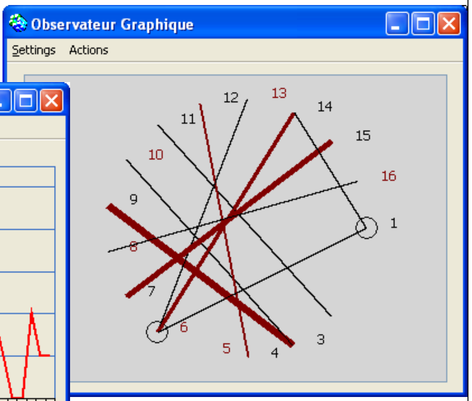
3. A part of code was added to the oneLevel:to:withGC: method at the FontionObs class to show a red line if negotiation is starting between two trading agents and becomes thicker as it progresses. After setting a boolean (*bool*) determining if a color change of the line drawn showing a message send is needed the next part was added/changed as well:

```
lw := aGC lineWidth.
bool     ifTrue: [aGC lineWidth: acom currentNegotiationHistory size].
aGC paint: (bool          ifTrue: [ColorValue perform: #darkRed]
                          ifFalse: [ColorValue perform: #black]).
```

Demonstration

- Initialize
- Graphs
- Message sends and negotiations
- Logs





From the CORMAS interface through the Simulation menu the Interface simulation menu item can be selected from which a simulation can be initialized and started. In the initialize screen started from that menu item the method can be selected which is to be used for initialization of the agents (init) and the method can be selected which is to be performed every step of a model run (step:). Probes which have to be recorded can be selected here to which can be viewed in graphs. Parameters can also be selected here and it's value which can made to change during the model run too. How many times a model run has to be run during a simulation can be set there to.

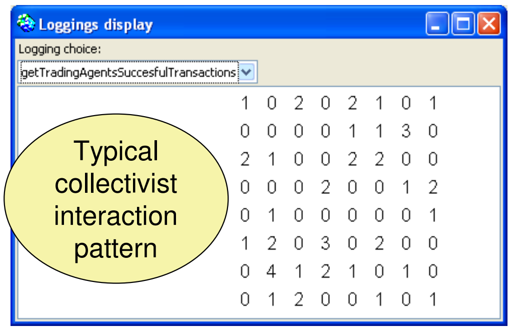
From the Visualization menu and then the Probes menu item it is possible to see the values of the selected probes in the initialization phase just by selecting one. That way it is noticeable when a trace was done for example.

From the Visualization menu and then the Messages menu item it is possible to see the messages send by the trading agents send to each other (select actions menu and then circle position menu item to get a propper view (and set all parameters to 0 from the Settings menu and then the Change parameters menu item). When two trading agents are truly negotiating the visible message line becomes red and gets thicker as negotiation goes on. That way it is visibly possible to see how long negotiations take.

From the Visualization menu and then the Logs menu item it is possible to see the different logs we can retrieve from the data of the trading agents and it's partner models. With the getTradingAgentTrustworthiness menu choice for example it is possible to see what trustworthiness value a trading agent has of the other trading agents.

Results

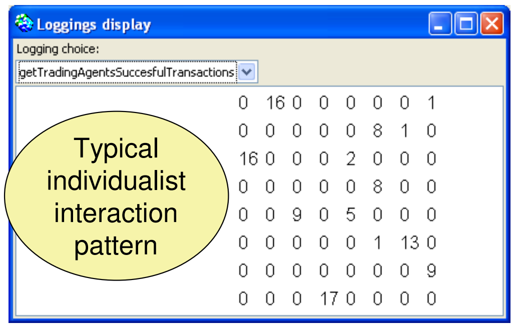
- Trading agent <-> partner models
- Partner model <-> Log files
- Trustworthiness changing
- Number of successful transactions



Loggings display

Logging choice: getTradingAgentsSuccesfulTransactions

```
1  0  2  0  2  1  0  1
0  0  0  0  1  1  3  0
2  1  0  0  2  2  0  0
0  0  0  2  0  0  1  2
0  1  0  0  0  0  0  1
1  2  0  3  0  2  0  0
0  4  1  2  1  0  1  0
0  1  2  0  0  1  0  1
```



Loggings display

Logging choice: getTradingAgentsSuccesfulTransactions

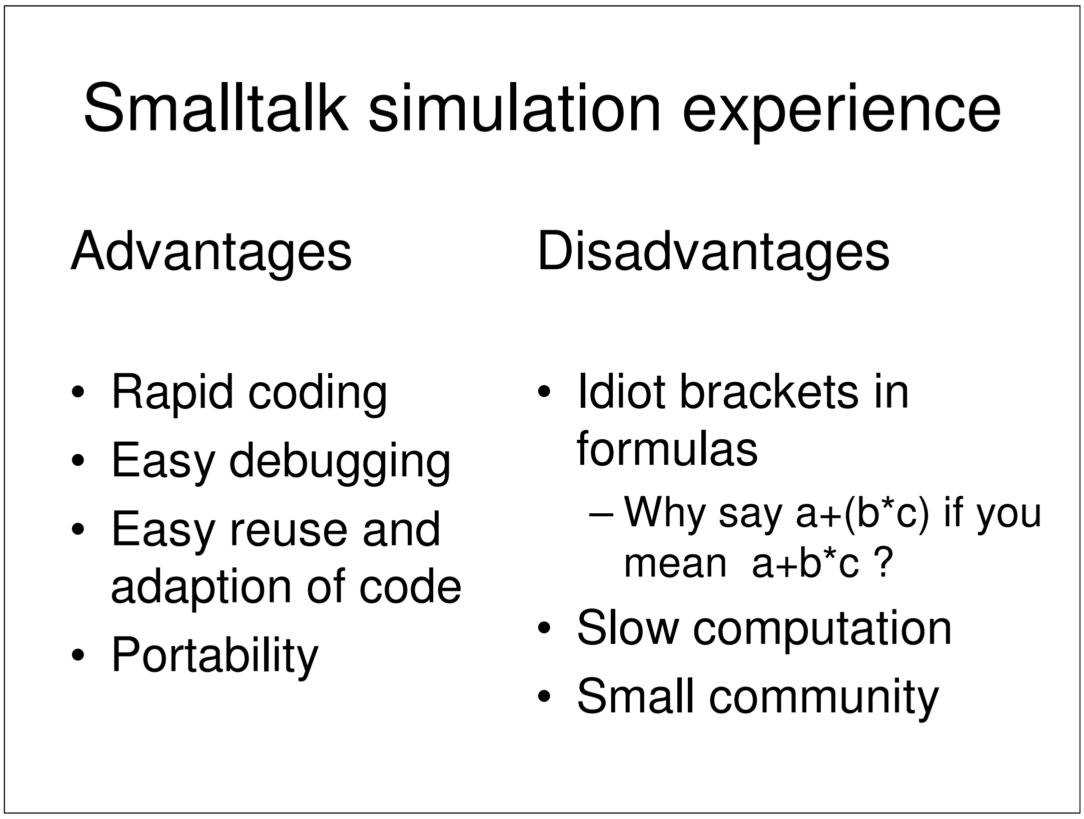
```
0  16 0  0  0  0  0  1
0  0  0  0  0  8  1  0
16 0  0  0  2  0  0  0
0  0  0  0  0  8  0  0
0  0  9  0  5  0  0  0
0  0  0  0  0  1  13 0
0  0  0  0  0  0  0  9
0  0  0  17 0  0  0  0
```

With the data found in the partner models the trading agents have, the results of a model run can be analyzed so it becomes clear which changes in the initialization causes which trait values to change an trading agent values over others.

With these partner models log files are generated so the data can be saved for later research. For example there are log files from the trustworthiness and fairness values a trading agent has over the other trading agents in it's partner models and also log files from the number of traces done and fines given in a model run.

From the trustworthiness logs it was very visible how these values changed during the model run and the trading agents starting to reevaluate that value in their partner models as the model run progressed.

Example of results: simulated transaction rate between in-group relations in societies of collectivistic and individualistic agents; collectivists trade with many in-group members, while individualists develop trusted partnership relations.



Smalltalk simulation experience

Advantages

- Rapid coding
- Easy debugging
- Easy reuse and adaption of code
- Portability

Disadvantages

- Idiot brackets in formulas
  - Why say a+(b*c) if you mean a+b*c ?
- Slow computation
- Small community

Using Smalltalk, time needed to code the trading agent model was not overspent. No major problems were experienced when implementing the design of our model, except with the priority of numerical operators.

According to standards of debuggers in current development environments, finding and resolving the bugs proved to be very easy with the VisualWorks standard debugger.

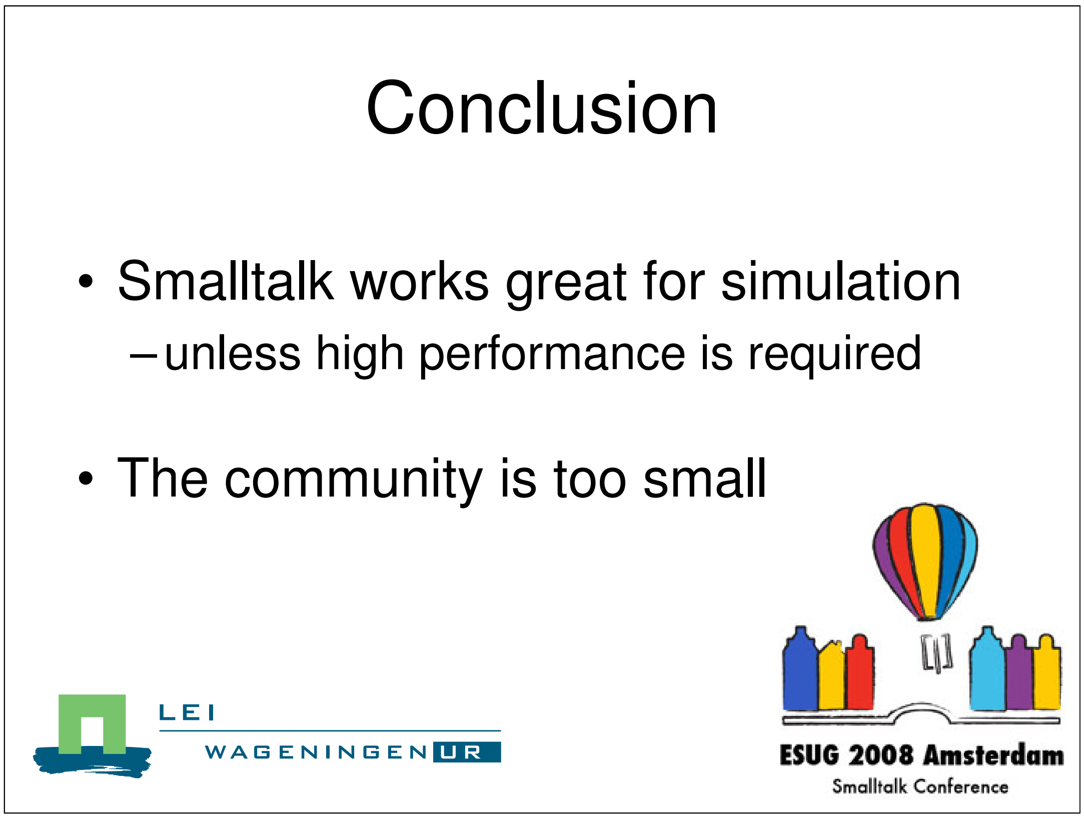
Changing and moving code proved to be very easy, also thanks to the refactoring browser. Overriding code (in overriding packages of course) in the CORMAS model and adding functionalities was no problem.

The portability was proven. The project was started on an apple G4 Powerbook and was later moved to a Thinkpad dual core notebook. Moving from the OSX platform to the Windows platform was done with great ease and no problems.

In Smalltalk the precedence of operation does not match the formula's as usually represented. This meant that implementation had to be carefully fitted with brackets and checked and rechecked continuously. This made it very accident prone to have big formula's implemented.

In the beginning running the model was very slow and time consuming. Moving it to a faster computer improved the speed, but the conclusion remains that for Smalltalk simulation you need to have a powerful notebook or desktop, and still simulations are time-consuming.

CORMAS is a tool that enables rapid model building, but only a small number of visualization tools is available. If the Smalltalk simulation community were larger, probably more sophisticated visualization classes would be available.



Conclusion

- Smalltalk works great for simulation
  - unless high performance is required
- The community is too small

LEI WAGENINGEN UR

ESUG 2008 Amsterdam
Smalltalk Conference

Our general conclusion is that Smalltalk works great for simulation, unless high performance is required, but that the community is too small to enable broad availability of reusable components.

More information on modeling of the influence of culture on trade

<http://www.verwaart.nl/culture>