Applying Constructionist Design Methodology to Agent-Based Simulation Systems

Kristinn R. Thórisson, Rögnvaldur J. Saemundsson,
Gudny R. Jonsdottir, Brynjar Reynisson, Claudio Pedica,
Pall Runar Thrainsson, Palmi Skowronski

RUTR-CS07003

Dec. 2007

Reykjavik University – Department of Computer Science

Technical Report

Applying Constructionist Design Methodology To Agent-Based Simulation Systems

Kristinn R. Thórisson,[1] Rögnvaldur J. Saemundsson,[3] Gudny R. Jonsdottir,[1]
Brynjar Reynisson,[2] Claudio Pedica,[2] Pall Runar Thrainsson,[2] Palmi Skowronski[2]

[1]Center for Analysis & Design of Intelligent Agents
[3]Department of Computer Science
Reykjavik University
Kringlan 1, 103 Reykjavik, Iceland

[2]School of Business
Ofanleiti 2, 103 Reykjavik, Iceland

{thorisson, rjs}@ru.is

Abstract. One of the many benefits of agent-based modeling is the ability to develop modules in parallel, with teams focusing on isolated modules with well-defined interfaces. This also presents a challenge, however: Integrating a system with a large number of modules with complex interactions, developed by many people, is a significant challenge. *Constructionist Design Methodology* (CDM) is an approach for building highly modular systems of many interacting components. Originally proposed for use in artificial intelligence research, CDM's strength lies in simplifying the modeling of complex, multi-functional systems that require architectural evolution of tangled data flow and control hierarchies. We have adapted CDM for the creation of agent-based simulations, resulting in a new version called CDM-S, and used it in the development of a family of agent-based market simulations where selected elements of an economy, including employees, companies, banks and consumers, are modeled at multiple levels of abstraction, from specific knowledge of single individuals to monolithic consumer groups. The systems have been built by a total of 15 Master's students over a period of 10 weeks in two consecutive periods. Here we describe the CDM-S and present data on the application of the CDM-S in the construction process, detail the amount of time spent on selected tasks, and discuss future prospects for the approach. The results support prior conclusions that the approach is a powerful high-level design approach for developing systems with a large number of modules with complex interactions. Future improvements need to address the tendency of (inexperienced) teams to often fall back on known development practices, ignoring the principles of the methodology, and thus reducing the benefits of the approach.

Introduction

The creation of agent-based simulation systems requires integration of a large number of functionalities that must be carefully coordinated to achieve coherent and desired runtime behavior. This work is made more difficult by the fact that they are typically built by numerous people over long periods, months, years, sometimes even decades. We are developing a family of agent-based simulation models with the goal of studying the generation, organization, development and evolution of the knowledge embedded in an industry at a high level of detail. In addition to providing a useful design methodology specifically aimed at multi-agent systems, the aim of our work is that it can lead to better comparisons between alternative methodologies for building such systems. We have used a version of the Constructionist Design Methodology (CDM) described by Thórisson et al. (2004). Here we present an adaptation of the CDM to multi-agent simulation systems, CDM-S, and present selected results from its usage in a relatively large simulation model of knowledge development in a market economy. We also give a description of our knowledge development model, to provide the context of the evaluation.

Our original plan, dating back to 2005, was to create models of knowledge evolution in a market economy that would go beyond the current state of the art in terms of detail and predictive power. Prior efforts that had focused on knowledge development had treated knowledge as a constant or a pre-determined factor at best – examples include the Java Enterprise Simulator described in Terna (2003) and other models of knowledge development that focus on the industry level of analysis, such as that developed by Cowan et al. (2003). While basing our work on this prior research, we wanted to build a platform that would enable the investigation of knowledge development dynamics, evolution and change in society (c.f. Nissen and Levitt 2004). The relative static nature of prior efforts would not have been sufficient for this purpose – we needed a knowledge model that went beyond these in terms of dynamism and flexibility. At the very outset we decided that the methodology would be key in ensuring that we would end up with a modular, modifiable and – perhaps most importantly – a maintainable and expandable model.

The models we have built so far are relatively large, composed of many types of agents each implementing multiple decision-making policies; the running simulations count up to 100 modules for a single system and thus provide a non-trivial test-case for the methodology, both in terms of hardware and software. The system has had many developers and evolved over the course of its lifetime. One of the challenges in our work is thus the management of software created by many students at different points in time. The architectural foundation of the approach is based on the concept of a network of interacting agents, communicating via messages through blackboards (Thórisson et al. 2004). This approach has emerged from the development of interactive artificial intelligence systems, stemming from the fact that many of the problems inherent in the development of agent-based systems are shared by those developing large robot systems (cf. Bischoff & Graeffe 1999), as these also require the integration and coordination of a large number of heterogeneous modules. In combination with the OpenAIR[1] message and routing protocol, our methodology makes explicit the functionality of each agent as well as their interfaces.

The resulting simulation models consist of agents at multiple levels of abstraction, i.e. individual, firms and other organization, and industry, each containing a number of decision-making policies. In most cases such policies are relatively simple; in every case we tried to mirror their natural counterpart to a first approximation. Individuals (simulated employees) represent the agent type that is least abstract agent in the system – a single Individual software agent corresponds to a single individual agent in the real world; the Market agent is the most abstract in that a single Market agent represents thousands of consumer agents in the real world. The Firms lie there in between, being partly represented by Individuals (the employees) and partly by monolithic rule sets that determine the firms' policies.

The paper is organized as follows: First we give an overview of our methodology, CDM-S, and present the resulting simulation framework. Then we describe the application CDM-S to the development process, presenting information on development process and detail selected parts of the process and draw conclusions from the data.

CDM-S: CDM for Simulation

As already mentioned, our methodology builds on the original Constructionist Design Methodology as described in Thórisson et al. (2004), which was originally designed for use in the creation of interactive artificial intelligence systems and is based on the concept of multiple interacting components; modularity in the approach is made explicit by using one or more blackboards (cf. Adler 1992) for message-passing. It has been pointed out by Thórisson et al. (2005), among others, that the benefits of message-based, publish-subscribe blackboard architectures are numerous; among the most obvious ones is that for modules whose input and output is fully defined by messages in the system, the messages embody an explicit representation of the modules' contextual behavior. In some cases the messages become a key driving force around which the modules are defined and organized. This is because messages in such systems mirror directly the abstraction level of the full system. This makes the blackboard architecture a powerful tool for incremental building and de-bugging of complex, modular systems.

Our present adaptation of this approach specializes the CDM for agent-based simulations; we call the resulting methodology *CDM for Simulation*, or CDM-S. The original CDM has 9 defined principles; CDM-S simplifies some of these and includes two new additional steps, for a total of 11. The full set of steps of CDM-S are:

[1] http://www.mindmakers.org/openair

1. *Define high-level goals*. Specify the primary motivation and goals behind the system to be developed.
2. *Define the scope of the system*. Specify at a high level what the intelligent system is intended to do. Ask yourself *What is the set of questions that my system supposed to answer?* – this is the most important question you will ever ask, when building a simulation! Then follow up with these four questions: {a} What is the data? {b} Where is the data? {c} How is it shared? {d} How is it processed/changed? Use of narratives and storylines are encouraged, as a template of expected behavior of the simulation. Start with an initial write up of a few high-level examples of system behavior. From this, a more detailed specification of the abilities of the system to answer questions can be built, using information from step 1. This will guide the selection of which modules to include and what their roles should be. It may be useful to think also in this step how the system may be expected to be modified and evolve.
3. *Modularization*. Define the functional areas that the system must serve, and divide this into modules with roughly defined roles. The modules can then be recursively sub-divided using the same approach (principle of divisible modularity). This step is best done in a group meeting where all developers can contribute to the effort. Try to match information exchange (messages) in your model with information exchange in the system(s) to be modeled.
    a. *Modules*. This step operationalizes the role of each module and defines their interfaces. Modules with highly different functional roles should typically not need access to the same set of data – if they do it means they may have close functional dependencies; consider coupling them or putting them into the same executable with shared access to the same data.
    b. *Blackboards*. The system is built using modules that communicate via a pub-sub mechanism and/or blackboard(s). The most obvious reason for modules to share a blackboard: They need to communicate directly. Consider using two or more blackboards if {a} there is a wide range of information types in the total set of messages in the system, e.g. co-existing complex symbolic plan structures and simple boolean switches; {b} there is a wide range of real-time requirements for the information flow, e.g. high-frequency vision processing and low-frequency plan announcements; {c} the system needs to run on multiple computers to achieve acceptable performance. It is natural that modules with messages containing similar content share a blackboard. Some modules are a natural bridge between blackboards, as for example when a module producing decisions to act requires perceptual data to make those decisions. Blackboards serve both as engineering support and system optimization.
    c. *Unit tests*. Design and implement simple tests that use monitor modules to test for "normal" and "obvious" behavior of the system. These tests return known results and span short periods of time. They are typically run for sub-parts of the system, i.e. subsets of modules and their isolated interactions, but can also be used to track behavior holistically.
4. *Test system against scenarios*.
    a. *Expected communication (blackboard) throughput.* Network speed and computing power puts natural constraints on the maximum throughput of the blackboard. Make sure the architecture and the hardware setup meets performance expectations.
    b. *Efficient information flow.* Static or semi-static information that is frequently needed in more than one place in the system may be a hint that the processes using that information should share an executable. (This is not a good idea when the processes sharing the information are of very different nature.)
    c. *Convenience with regard to programming languages and executables*. If two modules are written in the same language it may be convenient to put them together into one executable. This is especially sensible if {a} the modules use the same set of data, {b} are serially dependent on each other, {c} have similar update frequency requirements, {d} need to be tightly synchronized, or {e} are part of the same conceptual system.
5. *Iterate through 1-4 as often as necessary.*
6. *Assign modules to team members*. The natural way to assign responsibilities is along the lines of the modules, following people's strengths and primary interest areas. Every module gets one Lead that is responsible for that module working, for defining the messages it receives and the messages it posts. A good way to assign tasks, especially if the number of team members is small, is to borrow from extreme programming and assign them to pairs: One

Lead, chosen for their primary strength, and one Support, chosen by their interest and/or secondary strength. One individual can serve both roles, in different teams.

7. *Write module "shells" in a breadth-first approach*. This is a key method for achieving a good overview of message flow and interdependencies in the system: Start with "empty boxes" before creating full system that do very little but enough to start running the system in a few limited example scenarios. Use case scenarios from step 2 to help implement partial modules.
8. *Integration Test - test all modules in cooperation*. Holistic test (can use unit tests together) to verify gross system behavior. This step *always* takes longer than expected! It is also one of the most overlooked steps, yet *it cannot be avoided* – ignoring it at the outset will only force it to happen later in the process, delaying the ability to accurately estimate the project's full scope, and making it more likely that deadlines are missed and planned functionality has to be cancelled.
9. *Build modules to specification*. Build all modules to their next function specification. This step is naturally done in frequent alteration with the prior step. Write modules with resilience (graceful degradation): A distributed system can be very fragile; write modules to be resistant to downtime. Give modules with temporal knowledge: If the modules are unaware of the passing of time, they are less likely to represent the behavior of the system they are trying to simulate. Use the benefits being able to freely mutate modules (split and merge) as the design and implementation process unravels.
10. *Tune the system with all modules operating*. This step can be arbitrarily long, depending on the complexity of the interaction between modules and the complexity of the message content being transmitted between them. Computational intensity of some modules may require them to be put on separate computers.
11. *Return to 7* until all modules have reached full specification or the system is abandoned.

Application of the CDM-S

We will now describe the application of these design principles to the development of a family of simulation models family of agent-based simulation models meant to study the generation, organization, development and evolution of the knowledge embedded in an industry. We will focus in particular on the market mechanism in this model, as it had to be re-engineered from Phase 1 to Phase 2. The time spent by the teams working on version 1 and 2 are provided and compared. We begin with a short discussion on the software and programming language chosen.

Software

The software we chose for constructing our models is Java™ and Psyclone (CMLabs 2007); portability factored heavily in the choice of both. Python scripts were used for turnkey setup, making it easy for team members to start the simulation through a single command. Like related middleware such as Swarm,[2] Psyclone supports architectural re-configuration very well. This ability makes it easier to build systems where the exact final model is not known before construction, as is very often the case with complex simulations. Psyclone's architectural mutability is the result of flexible module APIs and ease of moving processes between computers. In Psyclone an implemented architecture can be radically changed relatively quickly; changes involving re-routing messages, temporal dependencies, and re-organizing distribution across machines can be achieved with relative ease. Semantic interfaces used for specifying data flow provide great flexibility in changing layout after the initial system is built. All parameters in our models are centralized and we are therefore able to modify them quickly, allowing for many variations and comparative runs of the models.

We chose XML to specify all aspects of the architecture that undergo frequent changes or require significant tuning. At present this includes subscriptions to message types and parameter settings. The use of Psyclone's whiteboards (a type of blackboard, Thórisson et al. 2005) supported flexible and easy-to-use real-time monitoring and statistics generation.

2 http://www.swarm.org/wiki/

System Modeling

The design occurred in two phases, both involving a group of Master's students who had not used CDM-S or similar methodology before. The first phase was implemented by a group of 10 Master's students, the second by a group of 3 Master's students and one undergraduate. In Phase 1 the students started from scratch; in Phase 2 they started from the platform designed and implemented by the first group. The framework enables us to ask detailed questions of how market forces influence knowledge development within the company as well as at the industry level, and how this affects the competitiveness of firms.

To provide a context for these evolutionary processes our models include agents at multiple levels of abstraction, i.e. individual, firms and other organization, and industry. Similarly, our model can represent how resource constraints, such as the lack of talented employees or the lack of funding, influence the knowledge buildup in an industry and eventual knowledge substitution.

To give the readers a sense of the complexity involved in the models, a short overview of the framework is in order. Further details are provided in Saemundsson et al. (2006).

Individuals. The first source of complexity stems from our new morphogenic[3] model of knowledge with highly dynamic properties. Individuals are atomic agents in the system, containing mechanisms

Task	Hours	%
Design architecture	15	**8**
Design bank	2	**1**
Design monitors	2	**1**
Design target consumer groups	10	**6**
Programming bank	20	**11**
Programming monitors	12	**7**
Programming target consumer group	80	**45**
Integration sessions	30	**17**
Defining message types	8	**4**
TOTAL	**179**	100

Table 1. Total amount of hours spent by the Market team on various tasks in the development of the market mechanism in Phase 1 of the project.

for knowledge acquisition, retention and application, salary negotiation and employment search. They can be initialized to have different knowledge (**K**s) in the beginning of a simulation. A learning rate describes how quickly/slowly they learn, that is, acquire new **K**s and increase the level, *l*, of their already-acquired **K**s. Individuals evaluate their salaries in connection with the rest of the world (average salaries, employment ads) and decide if they think they are more valuable than their salaries state, using a heuristic comparing their own salary with that of the market for each **K**. This mechanism is realized by the Job Center module (see below).

Firms. Individuals are employed by Firms that offer services on the market, based on the knowledge endowments of their employees. Firms compete with other firms on the service market, as well as in factor markets (employees and funding). Decision-making in the firm is based on policies local to each firm, including a training policy, an alertness policy and development policy. A Firm can add products to its suite of products, but must produce a specific amount before it can start selling it, simulating that it both takes time and resources to develop a new product. Product development is done by Individuals hired by the Firms through the Job Market.

University. The University controls general knowledge development within all knowledge fields, i.e. the state-of-the-art. It also controls the availability of new knowledge fields, i.e. new inventions. In the University each **K** has an associated "difficulty" rate that determines how intrinsically complex that knowledge is. This interacts with the learning rate of Individuals, which varies. The University holds l_{max} for all **K**s, i.e. the maximum possible state-of-the-art knowledge of a particular field or topic.

[3] Morphogen: An agent that controls the growth and shape of something, e.g. biological tissue.

	AVERAGE	TOTAL	%
Refactoring	19	76	22
Project	2	6	2
Studying	7	14	4
Messages	12	36	11
Testing	21	83	24
Docum.	25	75	22
Groupwork	11	33	10
Other	6	18	5
TOTAL	**103**	**341**	100

Table 2. Time spent by the groups on various tasks in Phase 2. (Average, total time and percentage spent by each developer.)

Job Center. A single agent called Job Center manages all hiring of employees. Companies advertise for individuals with specific levels of **K**s. Unemployed individuals can answer these advertisements and the Job Center finds the most suitable employee through a method that takes into account the salary demands of the individuals. The Job Center also has a role in calculating average salaries which individuals use when evaluating their own salaries.

Bank. The Bank is a simple loaning institution that the companies can apply for a loan from. All Firms start with some amount of cash – they will apply for loans if they run out of cash. The reasons why firms take loan in the simulation are mainly two: Either the firms need more money to be able to pay their operational costs or the firms need money to be able to invest in more production capacity or knowledge. Also, we wanted to see what influence limited or unlimited resources, money in this case, will have on the behavior of Firms and the ability for new Firms to enter the market. The Bank advertises its current interest rates every day, so the Firm can evaluate whether it is favorable or not to take a loan. We have global rates that changes every day and vary from circa 4% to 20 %. This variation simulates whether it is inefficient to take loan or not. As most of the other modules the Bank sends information, every day, about its capital and liquid resources that triggers the monitor that updates its plots.

Market. The market is composed of target groups; each group is initialized with a particular set of values determining its behavior with regard to the products offered by the firms. The Market includes various shifting policies and can be set to gradually change its size or shift for higher/lower values of learning rate *l.* At runtime, customer groups receive product advertisements from companies and make a decision of how much they will buy, according to how well the product meets their needs and the price of the product. A dialogue between the Market and the Firms tells the Firms how much they should produce of the advertised product.

Setup of Simulations & System Evolution

Our initial model contained 40 Individuals, 5 Firms, one Market, one University, a Job Center and a Bank (Figure 1). This turned out to be impossible to run on a single computer, even our fastest one, so we ended up distributing the system onto 12 computers. This, however, turned out to create bottlenecks in the networking. Using CDM-S steps 3a and 3b repeatedly on the design then helped us work out a new scheme for combining information from many messages into single messages so much fewer messages were being sent. Subsequently, this model ran fairly well on 12 machines.

The amount of time spent by one of the Market Design Team, which produced the design depicted in Figure 1 for the initial version of the model. Table 1 provides a breakdown for the various tasks that this team spent time on. This particular initial Market mechanism turned out to be a bottleneck for the system at runtime, as well as somewhat to simplified for the purposes of our simulation.

As we increased the complexity to the Individuals and Firm modules significantly they turned out to overload the CPUs on the machines we were running the system. The solution turned out to be the creation of a factory for the Individuals and Firms: The factories can be requested to "manufacture" Individuals and Firms with specified parameter settings – agents thus created live in the same execution space and their execution is thus more efficient (they still communicate via whiteboards and thus their interactions can be explicitly tracked). As the main constraint on resources has been computing power, factories have enabled significant speedups of the system runtime. We have tested

runs with over 100 modules; the tests indicate that the number of modules should be extended even further, to several hundred.

In spite of these advances, however, we found that the market mechanism still presented a bottleneck, and also that the Target group design in Phase 2 was not sufficient to ensure realistic distribution of sales between competing firms. To improve on the design one team in Phase 2 focused exclusively on re-engineering the way this mechanism worked, as well as making it more sophisticated. The results are shown in Figure 2. With the introduction of the Auctioneer, target groups no longer listen to product advertisements from the firms and decide for themselves what to buy. At the start of each day, they send Auctioneer messages requesting products and the Auctioneer returns what they bought. The introduction of the Auctioneer module also required minor changes to the policy implementation that is used to simulate how people's preferences change over time. Another addition to target groups is the functionality of updating or adding new target groups on the fly.

To guaranty upward scaling of target groups, a new server architecture was introduced. Instead of each target group being an individual executable, a target group server is created that has many target group instances. In the new design the target group server oversees interrupts and sending and receiving of messages on behalf of the target groups. This reduces the amount of messages that have to be transmitted and decreases the load on the network.

The amount of time spent on this modification is shown in Table 3. Typically, in a system with the number and complexity of interactions between modules, one would expect a significant amount of time spent on reworking the interaction and interconnection between the modules, rather than the module mechanisms themselves or other parts of the system. However, the results show that most of the time is actually spent on the modules themselves, and then on running the resulting system. In fact, most of the time is spent testing the running of the whole system. This indicates that at least one the goals of the CDM-S was achieved: To help balance the time spent on those aspects that typically fall by the wayside, such as running and tweaking the system.

Results: Experience From Using CDM

Overall, the use of CDM-S turned out to be highly beneficial, as unique module and message names enabled the groups to communicate efficiently about their interactions. We used the architecture that

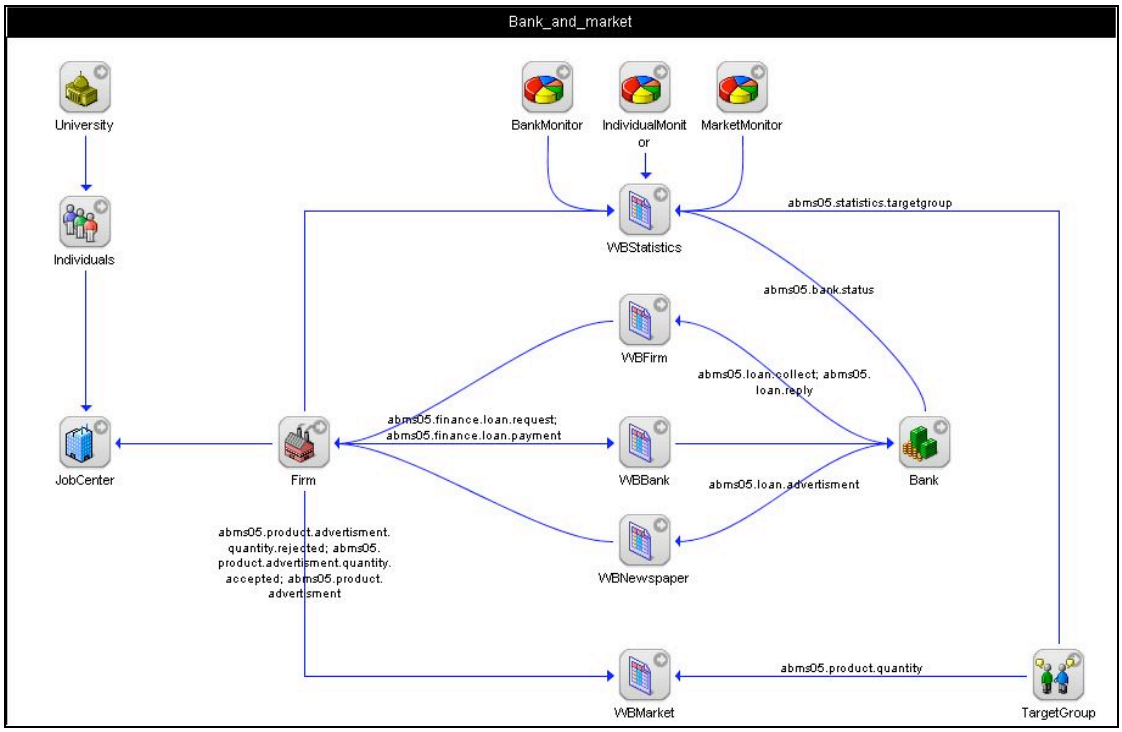

Figure 1. *Design of major system components in Phase 1 of the project. Compare implementation of market mechanism to that of Phase 2 (Figure 2). All messages follow the OpenAIR specification for message type names.*[1]

the team leaders defined at the outset and it proved to be fairly good – although of course several adjustments to the initial architecture and module design were needed throughout the project. Our original global design from Phase 1 proved solid and stayed relatively unchanged, with small adjustments to single classes. The Bank agent was redesigned once from scratch.

As the CDM-S predicted, in both Phase 1 and Phase 2 (although less so in the latter), we found that we had to spend more time than expected integrating the modules together in the integration session. One reasons was in many cases that the values in the messages that were being sent between the agents were not in the same scale. That affected the execution of the solution because many wrong decisions were taken in the modules when they got values in a different scale than was expected. This shows the importance of running the whole system together and observing its behavior. As the CDM-S warns (and most of the students turned a blind eye towards), the interesting information was not gotten from the system until after considerable parameter tweaking (and debugging).

We feel that CDM-S represents an improvement over CDM when applied to agent-based simulations. The resulting model for representing knowledge and innovation in markets is new, one that we have implemented relatively efficiently in a relatively short period of time. Our plans to take a closer look at knowledge substitution, radical innovation and non-contiguous jumps in product and service evolution

TASK	HOURS	%
Design / redesign	14	7
Market module	49	23
Auctioneer module	29	14
Bank module	3	1
Monitor	10	5
Code cleanup	7	3
Turnkey launch	12	6
Testing / bug fixes	17	8
Runs and tweaks	70	33
TOTAL	**211**	100

Table 3. Total hours spent by a 3-person team on various tasks in the reimplementation of the market mechanism Phase 2 by the introduction of an Auctioneer module. While this would typically be considered a major change to a complex model like this one, more time is actually spent on running and tweaking the model (33%) than on general Design (7%), Auctioneer module (14%), Bank module (1%) or Market module (23%) alone. However, overall, the total amount of time spent on redesign of the modules involved is 44%.

seem to be well supported by the present platform: In our runs so far we have shown the framework to produce predictable baselines, given sensible starting points. The resulting model has a clear potential to address several classes of questions regarding the generation, organization and evolution of knowledge in economic settings. The current distribution of modules in the model along an axis of abstractness can be fairly easily changed through the mutability functionality of the CDM-S. Coupled with the model's scalability we envision being able for example to substitute the Market module with several more fine-grain modules that approximates a large group of consumers more closely.

Thórisson et al. (2004) had reported a total estimated development time of 2 mind-months for a relatively complex interactive multimedia system, which they claimed strongly supported the conclusion that the Constructionist Methodology sped up system creation. Our results here also support this conclusion, perhaps not quite as strongly. We have reason to believe that the time spent on the relatively complex model, and the results achieved, indicate that CDM-S is a good design methodology, well worthy of further study and refinement. However, until other design methodologies provide comparable evaluations, we will not know how much or how little the CDM-S increases productivity in the creation of complex dynamic systems, over and above alternative approaches.

Although the CDM is intended to help coordinate the work of teams with experts widely varying expertise, our teams were highly uniform in nature, consisting mostly of CS Masters' students. This aspect of the methodology was therefore not put to the test. Another aspect that the CDM is meant to help with but not tested in our work is the subject of team collaboration: teams in remote locations with minimal interaction. In our case the communication between development teams throughout the project was rather frequent. Several meetings where held when critical decisions concerning whole of the

project needed to be made. The students agreed that the election of a leader for each group proved to be an excellent idea and sped up decision processes considerably although all major decisions where posed to the whole group. Internal communication between group members in the teams was also good. We feel therefore that it is difficult to make any comments about these aspects of the CDM based on our experience in this project.

Perhaps because of this there was a clear and strong tendency of the teams to fall back on standard software development practices. This included, for example, client-server models, blocking, strong coupling and interdependencies between components, strong assumptions about temporal synchronization, and other such issues which translate badly to highly distributed agent-based, realtime, and interactive systems. To be specific, principles 3c and all substeps of principle 4 were not being taken to heart by the team. Of course, to be of any help whatsoever, a methodology must to be put to use; we found that in many design sessions principles of the CDM were not applied where and when they should have, partly because of the students' in experience with it and partly because the instructors were not forceful enough to promote and ensure its continuous and rigorous application.

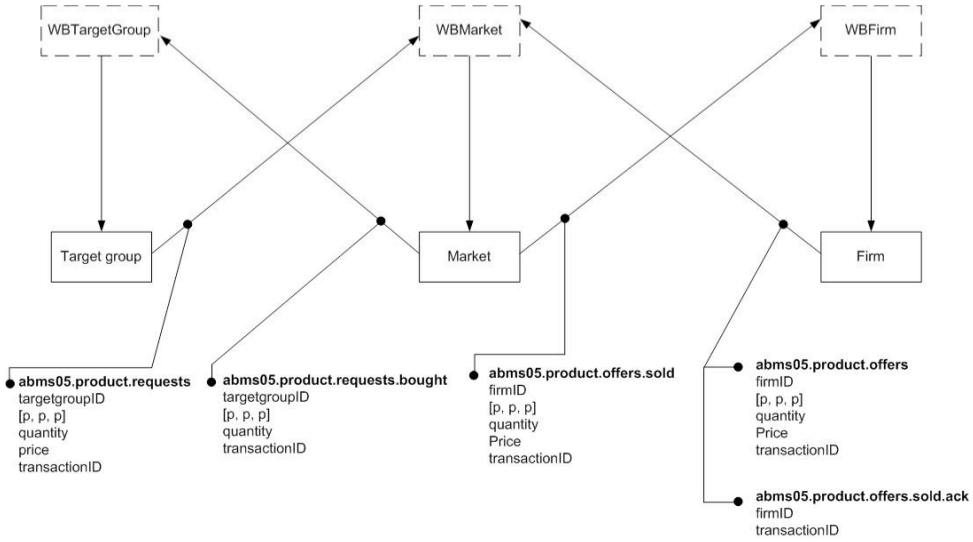

Figure 2. Diagram showing partial view of message interaction between a subset of the modules in one of the models developed.

These principles happen to be extremely important factors in building large, distributed systems. Many students did not realize until very late in the process of building the system – some even not until the very end – how criticial the principles proposed by the CDM are for helping build such a large and complicated simulation. This "dodging of selected principles" is something that needs to be improved but we have as of yet no well-formed ideas about what form that improvement could take, other than training. One possible way might be to supply detailed guidelines on the application of CDM principles to various practical problems, and provide examples and clear starting points for the various aspects of the design process. And one obvious solution is to be more adamant in the application of the methodology.

Conclusions

We have presented some of the results from the application of the Constructionist Design Methodology for simulation, CDM-S, to the creation of a complex, multi-scale model of a knowledge and market economy. The whole project had architectural challenges that were all solved through the use of the methodology; things that were assumed to be trivial, such as time synchronization and related third party software, which originally were not considered to be among the chief concerns, turned out to cause the biggest problems.

The model we built with the CDM-S represents everything from the various kinds of knowledge an individual can hold, and the dynamics of acquiring and maintaining such knowledge, to the size of firms with dozens of employees, all the way up to markets, composed of thousands of consumers. The mutability of our framework makes it possible to build several models with different assumptions and test the outcomes against each other. It also allows us to maintain easy control of all parameters that we

are interested in exploring, even at runtime. Explicit representation of module interaction through messages, which can be inspected both on paper and at run-time, increased the system's transparency, and helped all team members' understanding of how the various parts of the system work. The model's high degree of extensibility, and the resolution it provides for simulating the phenomena modeled, speaks well of CDM-S as a methodology for agent-based systems.

Inherent flexibility and scalability in the present framework, stemming from the use of the Constructionist Design Methodology, continues to enable continuous extensions to the model, increasing its complexity without disturbing the overall architecture. Agent types can easily be expanded without affecting the design and implementation of other agents.

Among the main issues to attend to in the use of the CDM-S for building large agent-based systems are methods for ensuring that the teams actually adhere to the principles of the methodology – a methodology is only useful if its principles are actually applied in a process – in our experience there is a strong tendency to ignore its principles in favor of already learned basic principles of software development.

Acknowledgments

This research was supported by a Marie Curie European Reintegration Grants within the 6th European Community Framework Programme. The work was in part also made possible by research grants from The Icelandic Centre for Research (Rannís). Psyclone is a trademark of Communicative Machines Ltd., Scotland. The authors would like to thank the team from Phase 1 who built the initial version of the system and contributed the some of the data and architectural details provided here; they are Maria Arinbjarnar, Hilmar Finnsson, Hafthor Gudnason, Vignir Hafsteinsson, Grétar Hannesson, Jónheidur Isleifsdottir, Arsaell Th. Johannsson, Gunnar Kristjansson, Sverrir Sigmundarson

References

Adler, R. (1992). Blackboard Systems. In S. C. Shapiro (ed.), *The Encyclopedia of Artificial Intelligence*, 2nd ed., 110-116. New York, NY: Wiley Interscience.

Bischoff, R. & V. Graefe (1999). Integrating Vision, Touch and Natural Language in the Control of a Situation-Oriented Behavior-Based Humanoid Robot. *IEEE International Conference on Systems, Man, and Cybernetics,* pp. II-999 - II-1004. Tokyo, October.

CMLabs (2007). *The Psyclone Manual*. Communicative Machines Inc.

Cowan, R., Jonard, N., & Özman, M. (2004). Knowledge Dynamics in a Network Industry. *Technological Forecasting and Social Change, **71***(5):469-484.

Nissen, M.E., & Levitt, R.E. (2004). Agent-based modeling of knowledge flows: illustration from the domain of information systems design. *Proceedings of the 37th Hawaii International Conference on System Sciences.*

Saviotti, P. P., & Krafft, J. (2004). *Towards a generalised definition of competition.* Presented at the *DRUID Summer Conference 2004 on Industrial Dynamics, Innovation and Development*, Elsinore, Denmark, June 14-16.

Saemundsson, R., K. R. Thórisson, G. R. Jónsdóttir, M. Arinbjarnar, H. Finnsson, H. Gudnason, V. Hafsteinsson, G. Hannesson, J. Isleifsdottir, Á. Jóhannsson, G. Kristjansson, S. Sigmundarson (2006). Modular Simulation of Knowledge Development in Industry: A Multi-Level Framework. *Proceedings of the First International Conference on Economic Sciences with Heterogeneous Interacting Agents*, Italy, June 15-17.

Terna, P. (2003). *The quest for the enterprise: jES, a Java Enterprise Simulator*. Manuscript. Torino University.

Thórisson, K. R., T. List, C. Pennock, J. DiPirro (2005). Whiteboards: Scheduling Blackboards for Semantic Routing of Messages & Streams. In K. R. Thórisson, H. Vilhjalmsson, S. Marsella (eds.), *AAAI-05 Workshop on Modular Construction of Human-Like Intelligence*, Pittsburgh, Pennsylvania, July 10, 8-15. Menlo Park, CA: American Association for Artificial Intelligence.

Thórisson, K. R., Benko, H., Arnold, A., Abramov, D., Maskey, S., Vaseekaran, A. (2004). Constructionist Design Methodology for Interactive Intelligences. *A.I. Magazine*, **25**(4):77-90. Menlo Park, CA: American Association for Artificial Intelligence.