Applying Constructionist Design Methodology to Agent-Based Simulation Systems

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

[1] Center for Analysis & Design of Intelligent Agents
[2] Department of Computer Science
[3] School of Business
Reykjavik University
Kringlan 1, 103 Reykjavik, Iceland
{thorisson,rjs}@ru.is

Abstract. Among the benefits of agent-based modeling is parallel development and implementation of components. Integrating large numbers of agents developed by many is, however, a significant challenge. Further, architectural changes can require significant redesign. We have developed *CDM-S*, the *Constructionist Design Methodology for Simulation*, an agent-oriented methodology for developing, implementing and evolving multi-agent systems. CDM-S's strength lies in simplifying modeling and construction of systems with architectural evolution of complex control hierarchies and data flow. We have applied CDM-S in the development of a family of market simulations where companies, employees, banks and consumers are modeled at multiple abstraction levels. These were designed and built by 14 students over a period of 10 weeks. Experience shows CDM-S to be a promising high-level methodology for constructing large multi-agent systems. Here we describe CDM-S and present data on its application in the development process.

Keywords: Design Methodology, Agent-based systems.

1 Introduction

The creation of multi-scale agent-based simulation systems requires integration of a large number of functionalities that must be carefully coordinated to achieve coherent and desired runtime behavior. Typically this work is done according to standard software practices. Development of multi-agent systems is more difficult using these methodologies than standard IT system construction as the characteristics and requirements of these systems are drastically different in many key aspects. Agent-based systems are often built by numerous people over long periods, months, years, sometimes even decades, evolving in the process. Few - if any - methodologies have addressed the many issues that complicate such work. Additionally, agent-based systems often assume or require concurrency throughout.

A. Håkansson et al. (Eds.): KES-AMSTA 2009, LNAI 5559, pp. 203–212, 2009.

We have adopted a methodology that was designed for artificial intelligence, the Constructionist Design Methodology (CDM) [1], to address the special issues encountered in agent-based simulation systems. The result, *Constructionist Design Methodology for Simulation* (CDM-S), has been used in the development of a family of agent-based simulation models. The methodology bears some relation to [2] and [3], although ours has the benefit of having been tested and honed in a broader range of projects [4,5,6]. The models built so far with CDM-S are fairly large, composed of many types of agents, each implementing multiple decision-making policies; the running simulations created to date count up to 100 agents for a single system. These provide a non-trivial test-case for the methodology.

In this paper we present CDM-S and its use in the construction of a family of simulation systems. First we give an overview of the methodology and present the resulting simulation framework at a relatively high level. Then we describe the application of CDM-S to the development process, presenting information on development process and detail selected parts of the process and draw conclusions from the data.

2 CDM-S: Constructionist Design Methodology for Simulation

The original CDM has 9 defined steps; CDM-S simplifies many of these and adapts to simulations. It also adds three new additional steps - 7, 11, 12. Step 7 is one of the keys to successful adaptation of CDM to simulation. The full set of steps of CDM-S is:

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 simulation 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 story lines 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 agents 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 agents with roughly defined roles. The agents 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. Agents communicate via a pub-sub mechanism and/or blackboard(s). Try to match information exchange (messages) in your model with information exchange in the system(s) to be modeled.

   (a) *Agents.* This step operationalizes the role of each agent and defines their interfaces. Agents 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. Define descriptive names for message types and draw flow charts of agent communication.
   (b) *Blackboards.* Blackboards serve both as engineering support and system optimization. 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 that form natural data groups (e.g. co-existing complex symbolic plan structures versus simple boolean switches) {b} there is a wide range of real-time requirements for the information flow, e.g. high-frequency micro-decisions versus low-frequency plan announcements; {c} the system needs to run on multiple computers to achieve acceptable performance. It is natural that agents with messages containing similar content share a blackboard.
   (c) *Unit tests.* Design simple case scenarios that return known results and span short periods of run-time. They are typically run for sub-parts of the system but can also be used to track behavior holistically. Define information snapshots to be published for statistical and monitoring purposes; use monitoring agents to test for "normal" and "obvious" behavior of the system.
4. *Test system against scenarios.*
   (a) *Expected communication (blackboard) throughput.* Network speed and computing power puts natural constraints on the maximum throughput of each 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 agents are written in the same language it may be convenient to put them together into one executable. This is especially sensible if {a} the agents 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 agent types to team members.* The natural way to assign responsibilities is along the lines of the agents, following people's strengths and primary interest areas. Every agent type gets one Lead that is responsible for that agent working, for defining the messages it post and receives. 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 agent "shells" in a breadth-first approach.* Partial agents are used to achieve a good overview of message flow and interdependencies in the system: Start with mock-ups, 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 mock-ups.
8. *Test all agents in cooperation.* Use the unit tests to verify sub-assemblies. This step always takes longer than expected! It is also one of the most overlooked steps, yet it *cannot* be avoided - doing so 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 canceled.
9. *Build agents to specification.* Build all agents to their *next* function specification. This step is naturally done in frequent alteration with the prior step. Write agents with resilience (graceful degradation): A distributed system can be very fragile; write agents to be resistant to downtime. Give agents temporal knowledge: If agents are unaware of the passing of time they are less likely to represent the behavior of the system they are supposed to simulate. Use the benefits of being able to freely mutate agents (split and merge) as the design and implementation process unravels.
10. *Tune the system with all agents (and agent shells) operating.* This step can be arbitrarily long, depending on the complexity of the interaction between agents and the complexity of the message content being transmitted between them. Computational intensity of some agents may require them to be put on separate computers.
11. *Return to 7 until all agents have reached full specification.*
12. *Expand / evolve the system.* Continue to add agents to the system, as well as modifying existing agents necessary, going back to step 2 or even 1, running through to 11, until the system is abandoned.

3 Application of CDM-S

We will now describe the application of these design principles to the development of a family of agent-based simulation models meant to study the generation, organization, development and evolution of the knowledge embedded in an industry. The original plan 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. While we have not yet fully achieved this goal we have made significant headway towards a model with endogenous support for knowledge evolution.

The software we chose for constructing our models is Java™ and Psyclone [7]; 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 using

a single command. Like related middleware such as Swarm[1], Psyclone supports architectural re-configuration very well. In Psyclone an implemented architecture can be radically changed with relative ease as to re-routing messages, temporal dependencies, and re-organizing distribution of agents across machines. Semantic interfaces used for specifying data flow provide great flexibility in changing layout after the initial system is built [5]. All parameters in our models allow centralized access, allowing for many variations to be made for comparative runs of models.

3.1 System Modeling

The market economy models consist of agents at multiple levels of abstraction, i.e. individual, firms and other organization, and industry: Multiple types of agents with a number of decision-making policies (for a detailed account see [4]). In most cases the policies are relatively simple; in every case we tried to mirror their natural counterpart to a first approximation. The individuals represent the agent type that is least abstract, in that a single Individual agent corresponds to a single individual in the real world; the Market agent is the most abstract, in that a single Market agent represents thousands of consuming 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 their policies.

The work proceeded in two phases, roughly 5 weeks in duration each. Each phase involved a group of Master's students in the course Agent-Based Modeling and Simulation, who had not used CDM-S or similar methodology before. A total of 14 Master's students worked in parallel on building separate parts of the system. 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, and provides for an interesting case study; the time spent by the teams working on version 1 and 2 are provided and compared. The first phase was implemented by a group of 11 Master's students, the second by a group of 3 Master's students and one undergraduate. In Phase 1 the students started from scratch and used CDM; in Phase 2 they started from the platform designed and implemented by the first group and used CDM-S, which had been revised from among other things lessons learned from Phase 1 (see Figure 1). Both teams were led by the same two instructors.

So, our models include agents at multiple levels of abstraction, i.e. individual, firms and other organization, and market. Similarly, our model can represent how resource constraints, such as the lack of talented individuals 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 given in [4].

Individuals. The first source of complexity stems from our new morphogenetic[2] model of knowledge with highly dynamic properties. Individuals are atomic

[1] http://www.swarm.org/wiki/

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

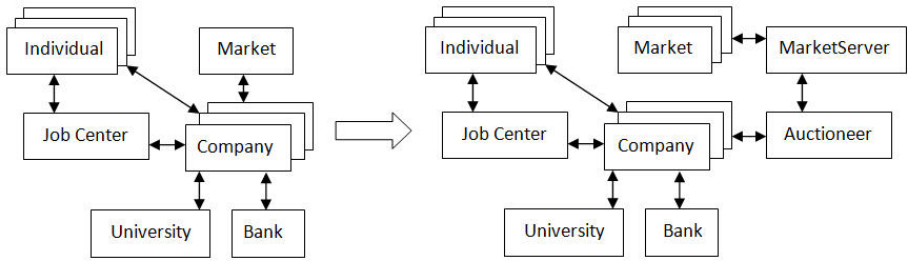

Fig. 1. Agents implemented in Phase 1 are shown on the left and modifications and refactoring in Phase 2 is shown on the right

agents in the system, containing mechanisms 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 (acquire new **K**s). They can learn new **K**s by going to special training at the University and their performance on a particular **K** improves over time as they use it to produce products that require that **K**. 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**.

Companies / 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 product/service markets, as well as in factor markets (employees and funding). Decision-making in the firm is based on local policies, including a training policy, an alertness policy and development policy. A firm can add products to its suite of products, but must invest both time and staff on product development before it can start selling.

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 at any point in time.

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 posts statistics such as average salaries.

Bank. The Bank is a simple loaning institution that the companies can apply for a loan from. All companies start with some amount of cash - they will apply

for loans if they run out of cash. The bank advertises its current interest rates every day, so the firm can evaluate whether it is favorable or not to take a loan. Global rates change every day and vary from 4% to 20%. As most of the other agents the bank publishes information about its capital and liquid resources.

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 its product sweet spot. 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.

4 Results

Our implemented model in Phase 1 contained 40 Individuals, 5 Firms, a Market, a University, a Job Center and a Bank. This turned out to be impossible to run on a single computer, even our fastest one, so we distributed the system onto 12 computers. This, however, turned out to create bottlenecks in the networking. We used CDM-S steps 3a and 3b repeatedly on the design and worked out a new scheme for combining information flow between companies and individuals into fewer messages, achieving a significant reduction in message traffic. The resulting model ran fairly well on 12 computers.

One of the main lessons learned in Phase 1 was that integrating the agents together in the integration session took more time than expected; frequently going back to fix agent behavior as a result of integration problems. 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 system as many wrong decisions were taken by the agents. This shows the importance of having a runnable version of the system with agent shells as early as possible in the design phase and frequently running the whole system together and observing its behavior.

The initial mechanism for the market turned out to be too simplified for the purposes of our simulation and did not 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. With the introduction of the auctioneer, target groups no longer listened to product advertisements from the firms and decided for themselves what to buy (see Figure 1). Following CDM-S's step 7, agent shells were created early on eliminating integration problems in Phase 2.

To guaranty upward scaling of target groups a new server architecture was introduced in Phase 2. 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 oversaw 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 by both market teams is shown in Table 1.

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. As suggested by CDM-S, explicit representation of agent 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. Typically, in a system with the number and complexity of interactions between agents as found here, one would expect a significant amount of time spent in Phase 2 on reworking the interactions and interconnections between the agents, rather than the agent mechanisms themselves or other parts of the system. However, the results show that most of the time is actually spent on the agents themselves, and then on running the resulting system. This indicates to us that at least one of our goals was achieved through the use of CDM-S: To help balance the time spent on those aspects that typically fall by the wayside, such as running the system under various conditions; CDM-S helped us manage complexity related to the network of agent interactions and allowed us to focus on agent functionality (see Table 1).

Thórisson et al. [1] had reported a total estimated development time of 2 mind-months for a fairly 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, but 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 promising design methodology worthy of further study and refinement. However, until other design methodologies provide comparable evaluations, we will not know how

Table 1. Total hours spent by two 3-person sub-teams. In Phase 1 the market and related functions were implemented and in Phase 2 the market was revised with added functionality and robustness. Implementing mock-ups of agent interactions beforehand in Phase 2 contributes directly to valuable time getting devoted to running and tweaking the model (44%).

	Phase 1		Phase 2	
TASK	Hours	%	Hours	%
Defining message types	8	**4%**	7	**3%**
Design / redesign	15	**8%**	14	**7%**
Market agent	90	**51%**	49	**25%**
Auctioneer agent			29	**15%**
Bank agent	22	**12%**	3	**1%**
Monitor	14	**8%**	10	**5%**
Runs and data gathering	30	**17%**	87	**44%**
TOTAL	179	**100%**	199	**100%**

much or how little the CDM-S increases productivity in the creation of complex dynamic systems, over and above alternative approaches.

4.1 Design Process and Teamwork

The iterative steps in CDM-S support teamwork and work distribution in a large group of developers, by among other things facilitating parallel development of agents. This was especially evident in Phase 1, where 11 students were working on the system simultaneously. The breadth-first approach emphasized in step 7 is another key to time-saving. It insures that communication between agents or sub-assemblies work before the details of each agent is implemented, allowing for a runnable version of the full system to be available from early in the implementation time. This also greatly minimized integration problems as the mock-ups serve as test interfaces for other's team's unit tests.

The agent development was part of the class homework so teams were working independently hours on their agents; integration sessions were held every other week. Starting from system goals originally provided by the teams' leaders, architectures were quickly built, requiring only minor adjustments throughout the project. Our original design from Phase 1 proved solid and stayed relatively unchanged, though minor adjustments to single classes where needed. The bank agent had to be redesigned once; thanks to CDM-S this could be done with only minor changes to the surrounding architectural components.

The communication between groups throughout the project was good. Several meetings where held when critical decisions concerning whole of the system needed to be made. The students agreed that the election of a leader for each group proved to be an excellent decision that sped up development processes considerably, although all major decisions where posed to the whole group. Internal communication between group members in our groups was good. The use of CDM-S turned out to be highly beneficial with the relatively large team of developers working in parallel; the agents/messages dual view it provided on the system enabled the collaborating groups to communicate efficiently about interactions between the system's agents, as well as the layout of the architecture, in part and in full.

5 Conclusions

We have presented 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 our CDM-S methodology, which resulted in a system with a high degree of extensibility. The relatively high simulation resolution the models provide is a direct result of using CDM-S. The current distribution of agents 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 to for example

substitute the Market with several more fine-grain agents that approximate a large group of consumers more closely.

The family of models developed has a clear potential to address several classes of questions regarding the generation, organization and evolution of knowledge in economic settings. The methodology had a large impact on the success of the work. In addition to providing a robust approach to multi-agent systems, the hope is that this work can also lead to more detailed comparisons and evaluations of alternative approaches for building such systems.

Acknowledgments. This research was supported by a Marie Curie European Reintegration Grants within the 6th European Community Framework Programme and 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.

References

1. Thórisson, K.R., Benko, H., Arnold, A., Abramov, D., Maskey, S., Vaseekaran, A.: Constructionist design methodology for interactive intelligences. A.I. Magazine 25(4), 77–90 (2004)
2. Giret, A.: A multi agent methodology for holonic manufacturing systems. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M. (eds.) AAMAS, Utrect, Netherlands, July 2005, p. 1375 (2005)
3. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) 12(3), 317–370 (2003)
4. Saemundsson, R.J., Thórisson, K.R., Jonsdottir, G.R., Arinbjarnar, M., Finnsson, H., Gudnason, H., Hafsteinsson, V., Hannesson, G., Isleifsdottir, J., Jóhannsson, Á., Kristjansson, G., Sigmundarson, S.: Modular simulation of knowledge development in industry: A multi-level framework. In: Proc. of the First Intl. Conf. on Economic Science with Heterogeneous Interacting Agents, Bologna, Italy (June 2006)
5. Thórisson, K.R., List, T., Pennock, C., DiPirro, J.: Whiteboards: Scheduling blackboards for semantic routing of messages and streams. In: Thórisson, K.R., Vilhjalmsson, H., Marsella, S. (eds.) AAAI 2005 Workshop on Modular Construction of Human-Like Intelligence, Pittsburg, Pennsylvania, July 2005, pp. 8–15 (2005)
6. Ng-Thow-Hing, V., List, T., Thórisson, K.R., Lim, J., Wormer, J.: Design and evaluation of communication middleware in a distributed humanoid robot architecture. In: Prassler, E., Nilsson, K., Shakhimardanov, A. (eds.) IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2007) Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware, San Diego, CA (2007)
7. CMLabs: Psyclone Manual (2007)