View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Scholarship at UWindsor

University of Windsor

Scholarship at UWindsor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2009

A Communication Choreography for Discrete Step MultiAgent Social Simulations

Muhammad Naushin Hasan
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation

Hasan, Muhammad Naushin, "A Communication Choreography for Discrete Step MultiAgent Social Simulations" (2009). *Electronic Theses and Dissertations*. 8035.
https://scholar.uwindsor.ca/etd/8035

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

A Communication Choreography for Discrete Step Multi-Agent Social Simulations

By

Muhammad Naushin Hasan

A Thesis
Submitted to the Faculty of Graduate Studies
through Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor.

Windsor, Ontario, Canada

2009

© 2009 Muhammad Naushin Hasan

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-57602-1
Our file Notre référence
ISBN: 978-0-494-57602-1

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

A Communication Choreography for Discrete Step Multi-Agent Social Simulations

by

Muhammad Naushin Hasan

APPROVED BY:

Dr. Kemal Tepe
School of Electrical and Computer Engineering

Dr. Dan Wu
School of Computer Science

Dr. Ziad Kobti, Advisor
School of Computer Science

Dr. Jianguo Lu, Chair of Defence
School of Computer Science

29 May, 2009

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Considerable research has been done on agent communications, yet in discrete step social agent simulations there is no standardized work done to facilitate reactive agent-to-agent communication. We propose an agent-to-agent interaction framework that preserves the integrity of the communication process in an artificial society in a 'time-stepped' discrete event simulator. We introduce the modeling language called Agent Choreography Description Language (ACDL) in order to model the communication. It serves in describing the common and collaborative observable behaviour of multiple agents that need to interact in a peer to peer manner to achieve some goal. ACDL further adopts the parallel and interaction activities to model proper communication in an artificial society. The ACDL communication framework is implemented and tested in REPAST. It employs a communication manager to generate and execute ACDL specification according to agent's communication needs.

DEDICATION

To my loving parents who have been a constant source of inspiration since the beginning of my education.

And to my loving and caring wife who has always patiently supported me.

Thank you for taking care of everything during the program.

This thesis would not be possible without you.

Also to my brothers and friends who gave me the necessary support and help to get me this far in my life.

ACKNOWLEDGEMENT

I would like to give my sincere gratitude to my supervisor Dr. Ziad Kobti, for his excellent supervision, guidance, and encouragement thought this research work. Without his help and his great patience to me, this work presented here would not have been possible.

I would also like to thank my internal reader, Dr. Dan Wu and my external reader, Dr. Kemal Tepe for participation as my thesis committee and spending their precious time to review this thesis and putting down their comments, suggestions on the thesis work. Their confidence in my abilities has been unwavering, and has helped to make this thesis a solid work.

I would like to thank my wife Mirza T Nasreen for her patient, love, support and encouragement during the research and thesis writing.

Finally, I am very grateful to all my family and friends for being supportive and always there for me.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Artificial societies are the agent-based computational models of social processes (Epstein and Axtell 1996; Gilbert and Conte 1995) or social phenomena mainly used for social analysis. Epstein and Axtell (1996) built an entire artificial society called 'Sugarscape' from the bottom up by modeling society's agents and their interactions. 'Sugarscape' was the first computational study of entire artificial societies (North and Macal 2007). According to (Epstein and Axtell 1996), Cellular Automata (CA) + Agents = Sugarscape. The underlying space of the modeling is the Sugarscape, which is a CA, containing random or concentrated distribution of sugar. Agents live and metabolize on that Sugarscape by gathering and eating sugar. If sugar is depleted and agents starve, they die.

The general structure of an artificial society model consists of three components: (a) a population of autonomous agents, (b) a separate environment, and (c) agent behavioral rules governing the interaction of agents with one another, the interaction of agents with their environment, and the interaction of environmental sites with one another (Epstein and Axtell 1996; Lawson and Park 2000). The environment in the model is typically a two-dimensional grid of cells, often forming a toroid. It contains heterogeneous distribution of one or more resource of interest to the agents. The society population consists of agents that move over the landscape and interact with the environment and with other agents.

The simulation technique used to model artificial societies is discrete event simulation (DES). Generally, at each tick of the simulation clock as the simulation progresses, agents move in the environment and interact with each other and the environment and gradually the whole society begins to evolve. Though, the simulation technique is called discrete event simulation, it is essentially a time-based simulation approach, where the system states are changed in each time-steps and the corresponding agent actions or behaviors to take place in each time steps are called 'events'.

One of the major aspects of the social simulation is the ability of the agents to communicate with each other or with the environment. But unfortunately, most works on social simulation and artificial social systems have the least focus on agent communications (Malsch et. al 2007). Typically, in social simulations agents involve in an indirect mode of interactions, where messages are transmitted from sender to receiver on a one-to-one basis (Malsch et. al 2007). No works have been found in this paradigm at the time of writing this thesis, where it deals in details of agent communication: how to implement agent communication framework with the simulation model, with even the effort to preserve the temporal and contextual accuracy of the agent communications.

In most Cellular Automata (CA) based artificial society models the discrete event simulation engine logic uses "time-step" scheduling where multiple agent behaviors occur at each time step (Lawson and Park 2000). At a given time-step, all the agents in the agent-list are processed sequentially. Therefore agent to agent communication can easily lose the temporal and contextual accuracy of the messages being passed. For example, in a non-trivial social simulation, there are two agents A_1 and A_2 communicating some time sensitive information. At any time step t_1, A_1 is being processed from the agent list. At t_1, A_1 asks for (query) anything to A_2 related to A_1's current time t_1. In this case, A_2 may not live on the same time step as A_1's. A_2's current time t_2 either could be $t_2 = t_1$ (Both A_1 and A_2 are processed) or $t_2<t_1$ (A_1 is processed before A_2) or $t_2>t_1$ (If the agent list is shuffled at each time step and A_2 is processed before A_1). Clearly this simulation technique loses the temporal and contextual accuracy of the information interchanged.

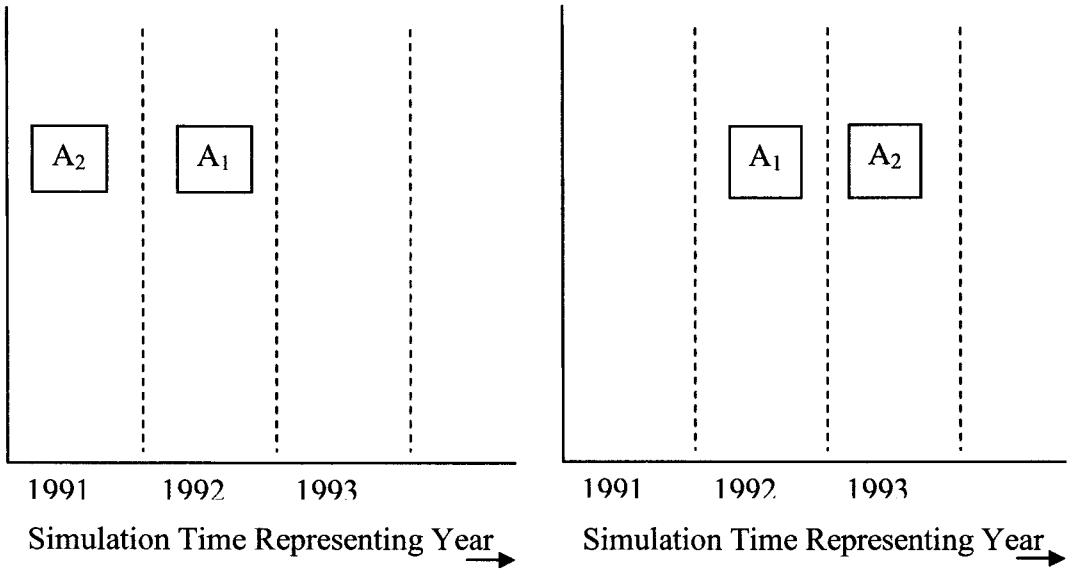

Figure 1-1 (a) Problem Scenario 1 **(b)** Problem Scenario 2

If the underlying simulation model increases its time-step by a year and if we let A_1's current year, t_1 = 1992. Now A_1 asks A_2 "what is the unemployment rate for the current year?". As explained earlier, A_2's current year could be either 1992 (Figure 2) or 1991 (Figure 1-a) or 1993 (Figure 1-b). If A_2 lives in year 1991 or 1993, A_1 gets back wrong information from A_2. Surely this scenario introduces the problem of wrong or outdated information.

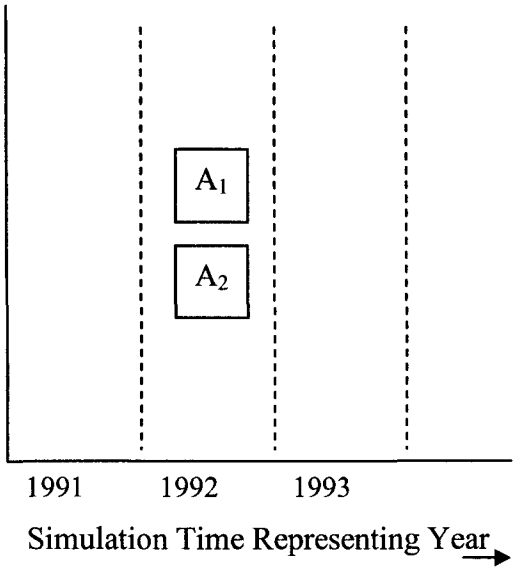

Figure 1-2 Correct Scenario

Only when A_2 is processed before A1 and they both live on the year 1992, A_1 gets the accurate information. Therefore, with the existing simulation technique agents are able to communicate but there are failing scenarios where the communication could lose its temporal and contextual integrity. Usually, like other behaviors agent communication is also modeled as a behavioral subroutine. As mentioned earlier, simulation mechanism is strictly controlled by time steps and in those time steps agent communication should take place. Therefore, the problem in agent communication found in this paradigm is due to the specific simulation technique or the simulation algorithm used to model the agent behaviors and the society.

1.1 Motivation

In most of the artificial society simulations reactive agents are being used (Malsch et. al 2007) and therefore the simulation models lack agent to agent communication capabilities in temporal context. Those reactive agents only interact with the environment and there is no direct agent to agent communication. In order to model human societies realistically it is necessary to have the communication capabilities with proper temporal

and contextual accuracy into the model along with the use of behavioral agents so that the simulated models can represent human behavior as closely as possible. In this, thesis we only consider the communication part in temporal context in an artificial society model.

1.2 Thesis Contribution

This thesis is mainly concerned about creating an agent communication language called ACDL (Agent Choreography Description Language) based on WS-CDL (Web Service Choreography Description Language) and using ACDL build a framework for agent to agent communication in the context of social simulation. As seen earlier in this chapter, is that modeling agent to agent communication by using direct method calls could lose the temporal and contextual accuracy of the messages and therefore producing erroneous results. Our contribution provides a way to preserve that temporal and contextual accuracy of the messages intended for communication in social simulation and producing the accurate result.

1.3 Thesis Layout

This thesis is organized as follows:

Chapter 2 represents all the basic concepts and definitions and Chapter 3 represents related works in our problem domain. Chapter 4 contains the proposed framework and the design and it is followed by Chapter 5 where we detail our implementation and experimentations. In the end Chapter 6 focuses on conclusion and future works.

2. BACKGROUND

2.1 Artificial Society

The term 'artificial society' first originated in the works of (Builder and Bankes 1991) and later popularized by (Epstein and Axtell 1996). An 'artificial society' is a generic class of agent based simulation model (Lawson and Park 2000) which is used to simulate various social phenomena or processes. The simulation of agents and their interactions is known as agent-based modeling (Axelrod 1997). It is also called bottom-up modeling and artificial social system. The goal of agent-based modeling is to study and understand the properties of complex social systems. Social processes e.g. populations dynamics, group formation, environmental and economic impacts, propagation of disease, cultural influences, combat, etc. are usually complex (Epstein and Axtell 1996). It is impossible to decompose those complex processes clearly into simpler sub-processes. Therefore the isolated analysis of the simpler processes cannot be aggregated to yield the analysis of the complex social process as a whole. In social context, these models help to understand how macroscopic social behavior can emerge from various microscopic social phenomena. With growing popularity agent based society models are used in applications from the social sciences, in military applications, biology, chemistry, ecology, engineering, geography and marine biology etc.

2.2 Simulation Techniques used in Artificial Society Model

Being a well established modeling tool (Law 2007) Discrete-event simulation (DES) is used to model artificial societies (Lawson and Park 2000). A system is modeled in terms of its state at each point in time in DES (Banks et al. 2005). In Agent Based Modeling Systems (ABMS) or modeling an artificial society there are two types of scheduling algorithms that are associated with DES (North and Macal 2007):

- Time-step scheduling
- Discrete-event scheduling

2.2.1 Time-step scheduling

The time-step scheduling algorithm for discrete event simulation to evolve an artificial society is (Lawson and Park 2000):

```
initialize society landscape;
initialize agent population;
t = 0; // time counter, (0 <= t <= T) (T = max. simulated time)
while(t<= T){
      perform agent actions;
      update society landscape;
      update the agent list;
      randomize the agent list; // To minimize artifacts in the
simulation result
      t++;
      }
generate statistical report;.
```

Figure 2-1 (Lawson and Park 2000) Time-step scheduling Algorithm

Time-step scheduling algorithm involves fixed-increment integer time-counter to track the flow of time. All events of interest and agent behavioral actions must occur at one of these integer-time steps. In this algorithm, first, the society landscape and the agent population initialization are done. Now if the maximum simulated time for the model is T, the artificial society model evolves synchronously according to Figure 2-1. Most multi-agent based simulation environments are time-driven DES tools (Sansores and Pavon 2005).

2.2.2 Discrete-event scheduling

In discrete event scheduling algorithm an event list is maintained. Event list is the list of all future events, ordered by time of occurrence. This algorithm repeatedly determines the most imminent possible event in the list and advances the simulation clock to this event's scheduled time of occurrence. Also in each execution step the algorithm generates future events (if any) and places them in the event list with proper ordering. The pseudo code for the discrete event scheduling algorithm for discrete event simulation to evolve an artificial society would be (Lawson and Park 2000; Banks et al. 2005):

```
initialize society landscape;
initialize agent population;
e =dequeEvent();
while(e.time<= T){
      perform event actions;
      update society landscape;
      update/schedule the event list;
      e =dequeEvent();
      }
generate statistical report;.
```

Figure 2-2 (Lawson and Park 2000; Banks et al. 2005) Discrete-event scheduling Algorithm

2.3 Agent Communication

Communication is one of the key components in an artificial society. The agents need to be able to communicate or interact with the environment or with each other if they need to cooperate, collaborate, and negotiate and so on. Generally in a multi-agent system, agents interact with each other by using some special communication languages, called agent communication languages (ACL) which are based on speech act theory (Searle 1969) and that provide a separation between the communicative acts and the content language. The two most-widely used ACLs in practice are KQML and FIPA-ACL. But neither has yet been considered as standards. In regards to multi-agent simulation they are considered heavy weight languages.

2.3.1 KQML

KQML (Knowledge Query and Manipulation Language) was the first ACL with a broad uptake and was developed in the early 1990s as part of the US government's DARPA Knowledge Sharing Effort (KSE). It is a language and protocol for exchanging information and knowledge that defines a number of performative verbs and allows message content to be represented in a first-order logic-like language called KIF (Knowledge Interchange Format) which is another deliverable of KSE. (Genesereth and Fikes 1992). KQML is the outer language format for the agent communication as it defines the envelope format for the messages and KIF is concerned with the message content.

Each KQML message has a performative (the action indicating verb) and a number of parametres (attribute-value pairs). An example KQML message is shown in Figure 2-3.

```
(ask-one
      :content     (PRICE IBM ?price)
      :receiver    stock-server
      :language    LPROLPG
      :ontology    NYSE-TICKS
)
```

Figure 2-3 (Woolridge 2001) Sample KQML message

Among few of the constraints for KQML are: building different implementations of KQML were not tightly constrained and therefore those implementations failed to interoperate with each other. The semantics of KQML were never rigorously defined. KQML performatives do not have commissives by which agents can make commitments to each other so that they can coordinate their activities to achieve a common goal (Woolridge 2001).

2.3.2 FIPA ACL

Currently the most used and studied agent communication language is the FIPA ACL (Foundation for Intelligent Physical Agents Agent Communication Language), which incorporates many aspects of KQML (Labrou *et al.*, 1999) and syntactically it is the same as KQML. The primary features of FIPA ACL are the possibility of using different content languages as it does not mandate any specific language for the message content (Woolridge 2001) and the management of conversations through predefined interaction protocols. It has a richer set of performative (the action indicating verbs) than KQML. An example FIPA ACL message is shown in Figure 2-4.

```
(inform
      :sender      agent1
      :receiver    agent2
      :content     (price food2
150)
      :language     sl
      :ontology    hpl-auciton
)
```

Figure 2-4 (Woolridge 2001) Sample FIPA ACL message

FIPA ACL excels over KQML by incorporating semantics with the help of a formal language called SL. The semantics of the FIPA ACL map each ACL message to a formula of SL, which defines a constraint that the sender of the message must satisfy if it is to be considered as conforming to the FIPA ACL standard (Woolridge 2001).

2.4 WS-CDL

WS-CDL (Barreto et al. 2005) is a XML-based language that is used to specify the peer-to-peer collaboration of participants from a global or public point of view. It is a multi-participant contract of the interactions among the participants on which each of them have agreed upon. WS-CDL specifies the ordering of messages that the participants

exchange and the operations they offer across the domains of interactions. This is a description language not an executable language. A WS-CDL document conceptually has two parts (Fredlund, L. 2006; Mendling and Hafner 2006): a *static* part, i.e., the invariant part that describes e.g., variable, token and channel definitions and a *dynamic* part that states the interaction among the partners. (Barreto et al. 2005; Ross et al. 2005; Barros et al. 2005) provides in-depth details.

2.4.1 Static Part

This part defines all the specifications needed to define the collaborating parties. According to (Barreto et al. 2005; Ross et al. 2005; Barros et al. 2005; Fredlund, L. 2006; Mendling and Hafner 2006) Table 2-1 introduces the WS-CDL elements that belong to the static part.

Table 2-1 Summary of the WS-CDL elements that constitute the static part

WS-CDL Entity	**Description**
roleType	The interactions are taken place among various roleTypes
relationType	Identifies the mutual relationship between two roleTypes
participantType	Set of all the roleTypes that belong to the same physical entity
informationType	Describes the types for many of the variables that might be used in a choreography
variable	Expresses information about commonly observable objects in a collaboration
token	Express parts or alias of a variable for reference purpose
channelType	Specifies where and how the interaction should take place between multiple participantTypes
package	The root of every choreography definition and contains both the static and the dynamic parts of a choreography

2.4.2 Dynamic Part

This part defines all the peer-to-peer interactions among the parties involved. *Dynamic* part constitutes the core of the *choreography*. The root element for the *dynamic* part is the *choreography* element.

choreography. A *choreography* specifies where and how the interaction should take place between multiple *participantTypes*. It is basically the container for a collection of WS-CDL *activities* that may be performed by one or more of the participants. According to (Barreto et al. 2005; Ross et al. 2005; Barros et al. 2005) the three types of WS-CDL activities are summarized on Table 2-2.

Table 2-2 Summary of the WS-CDL *activities*

WS-CDL activities	**Description**
An ordering structure	Specifies the orders in which the interactions should take place. It could be sequential, parallel or conditional and they are represented by the *sequence*, *parallel* and *choice* elements respectively. They can be used in a nested manner in the choreography
A WorkUnit-Notation	It is represented by *workunit* element and is used to guard and/or provide a means of repetition of those activities enclosed within the *workunit*.
A basic activity	It is used to describe the lowest level actions performed within choreography.

A basic activity can be either:

An interaction activity: It is represented by *interaction* element and is the basic building block of communication among the participating entities in choreography. *interaction* is regarded as the base atom of the choreography composition. The exchange of information between the collaborating partners occurs inside this element. It specifies the *relationTypes* that are involved in the information interchange and the direction of the

message flow by using the attributes *fromRole* and *toRole*. The *operation* attribute inside *interaction* captures the name of the operation associated with the *interaction*. The *exchange* element constitutes the real message that is to be passed between the communicating parties.

A perform activity:. It specifies a separately defined choreography to be performed.

An assign activity: It specifies assignments of variables within a *roleType*.

A silentAction activity: It specifies participant specific non-observable operational details to be performed.

A noAction activity: It specifies participant specific points where the participants do not perform any action.

A finalize activity: It specifies finalizer block for the choreography.

3. LITERATURE REVIEW

This chapter reviews some of the works done related to our thesis area. Here we focus on the works done on communication in artificial society, communication in agent based modeling and time management in distributed simulation.

3.1 Communication in Artificial Society

Artikis and Pitt (2001) suggested a model of artificial society that facilitates communication. The requirements for their open agent society model are: 1) a need to make the organizational and legal elements of a multi-agent system externally visible, 2) open societies should be neutral with respect to the internal architecture of their members, and 3) communication and conformance of behaviour are at least as important as intelligence. An agent society based on this model consists of the following entities: 1) a set of agents, 2) a set of constraints on the society (norms and rules), 3) a communication language, 4) a set of roles that agents can play, 5) a set of states the society may be in, and 6) a set of owners of the agents. Davidsson (2001) and Davidsson and Johansson (2006) extended that model by introducing a stakeholder or owner of the society.

Buzing *et al* (2003) built a discrete time stepped artificial society called VUscape (a two-dimensional grid based spatial model with resource distributed in the cells and agents consuming the resource in the cells) which is based on the popular Sugarscape model (Epstein and Axtell 1996). The authors introduced a framework for modeling communication and cooperation in an artificial society where communication and cooperation behaviour are evolved in the society by means of an environmental pressure. Agents in this model are given 'talk' and 'listen' capabilities. By listening an agent can get information from other agents' resources and locations. By talking an agent can 'broadcast' its resource and location information to other agents. Authors ran experiments on both with communicating and non-communicating agents in their society. The

communicating agents tend to have larger surviving population than non-communicating agents. Also with communication, the behaviour of the system seems more stable than without communication.

In their continuation of work (Buzing et al 2004; Eiben et al. 2005) tried two different approaches to communication among the agents in the society - centralized approach referred to as 'Multicast Model' and distributed approach referred to as 'Newscast model'. Multicast Model is a spatial communication scheme where multicasted messages travel along agents axes. Multicast communication is implemented by a centralized message board where agents can post messages by their talk capability and can read the messages from it by their listen capability. The newscast model is a fully distributed information propagation protocol for large-scale peer-to-peer computing. In newscast communication messages are transferred directly between the agents without a third party like a message board as in the centralized approach. Each agent contains a cache where it holds messages received from other agents along with their IDs and addresses. The senders of messages are the friends of an agent and agents are only allowed to communicate among the friends. According to the experimental results, newscast communication was found less effective than multicast communication as agents tend to move less and die out faster. According to the authors the difficulty with the distributed approach is not being able to remove the outdated information from the agent's cache and that leads to false information to other agents. (Buzing et al. 2005) also studied the evolution of communication and cooperation in VUscape with the multicast model.

3.2 Communication in agent based modeling

Among the very few works done specifically on agent to agent communication on multi-agent simulation (Gokturk and Polat 2003) proposed a three-layered approach to the agent communication in the context of distributed multi-agent simulation. The highest layer is the content layer where the actual message is expressed as Knowledge

interchange Format (KIF) or Semantic Language (SL). The next layer is called the communication layer, where the message contents form the upper layer and is encapsulated using an agent communication language such as KQML or FIPA-ACL. The bottom layer is the transport layer to take care of the specific transfer related issues e.g. converting the ACL messages so that they can be transferred over actual connection. The authors implemented the second layer using KQML and the third layer using HLA (High Level Architecture) which is a distributed simulation standard (IEEE 1516) that aims at interoperable and component-based reusable simulations. In their implementation they have difficulties in establishing point-to-point links (how the agents can refer to each other) among the agents and also to maintain the order of the sent messages (known as send-order reception problem) in the receiving end. Both of these difficulties arose because of the limitations present in HLA architecture. The authors solved the point-to-point links problem with implementing a publish-subscribe model of messages and the send-order reception using a time-stamping scheme.

3.3 Time Management in Agent-based Simulation

Most of the time management and synchronization works are focused on distributed event-driven DES. In distributed simulation as the agents are executed in physically separated locations there is a need to synchronize the event changes that occur at different computational nodes. Two main approaches to ensure correct time stamp order are: conservative and optimistic synchronization (Pawlaszczyk and Timm 2007). By means of lookahead, Conservative algorithm strictly ensures that all the events occurring at different computational nodes are always synchronized so that there does not appear any causality error (the message ordering problem). Optimistic algorithm on the other hand allows the causality error to occur and then fix it by means of rollback to a nodes last correct state. Figure 3-1 illustrates how rollback is done on out-of-order (straggler) message. (Pawlaszczyk and Timm 2007) proposed a hybrid time management approach by combining optimistic synchronization approach and domain-specific knowledge based on FIPA request interaction protocol. In this approach rather than doing expensive

rollbacks out-of-order messages are delayed for execution. Figure 3-2 the delayed execution of the straggler message.

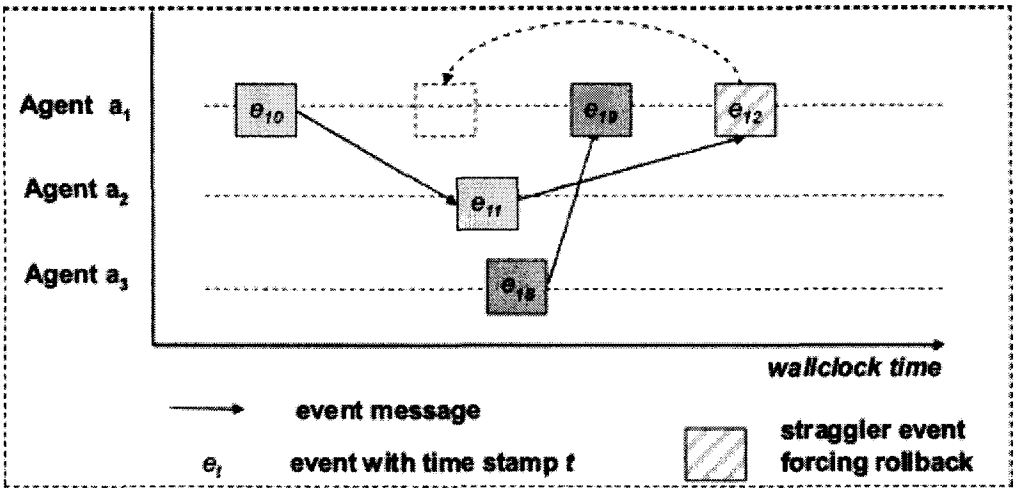

Figure 3-1 (Pawlaszczyk and Timm 2007) Scenario in distributed simulation causing violation of event ordering (causality error). When event e_{12} arrives, e_{19} has already been processed by agent a_1 (straggler event).

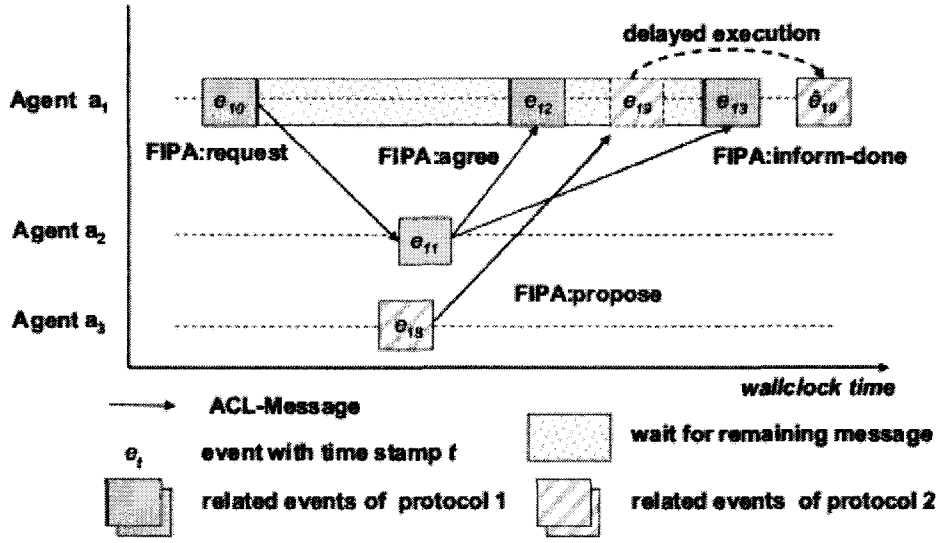

Figure 3-2 (Pawlaszczyk and Timm 2007) Delayed event execution based on protocol information. Agent a_1 receives a proposal from Agent a_3, while he is waiting for an inform-done message of Agent a_2. Instead of immediately processing the incoming message, the execution is delayed. Thus, event order is preserved and still valid.

Braubach et al. (2004) presented a standard-compliant middleware called time service component to enable the simulation of process-flows in distributed MAS. The time service component allows the timed synchronization among the distributed participants and controls the progress of the over-all process flow. The authors have implemented the time service as a FIPA-compliant agent, and that can be used to couple heterogeneous subsystems implemented on different agent platforms.

Huang et al. (2005a; 2005b) built a special agent interface called Smart Time Management (STM) on top of HLA. STM can take over event's time-stamp tagging work, maintain a lookahead value and unify different time management approaches (conservative and optimistic) provided by the HLA. STM presents a unified and scalable middle layer to allow the user to construct an HLA federation with an unanimous Time Management interface when solving the synchronization issue. For optimistic approach STM also extends the interfaces with the smart rollback, state-saving, and fossil collection mechanisms.

Helleboogh et al. (2005a) proposed semantic duration models to capture timing requirements using the technique of duration modeling that reflect the semantics of MAS activities in an explicit model. And the authors also built a time management infrastructure based on the semantic duration model description to integrate all time management functionality into a MAS transparently. The idea of duration modeling is to maintain a logical clock for each agent and advance that clock for each primitive that is executed by the agent. The duration of a primitive performed by an agent is the (logical) time period it takes until the effects of that primitive are noticeable. The developer has to describe all timing characteristics by means of assigning logical durations to each of the primitives. Advancing the logical clock in a way that is independent of computer loads and processor speeds, enables repeatable simulation results. There are two possible levels of duration model low level and high level. Low level models are directly tied to programming language implementation. The authors took the high level approach that ties to the semantics of the MAS model. The duration model is usually used for agent

deliberation model but the authors extended this model to accommodate other agent activities e.g. the activities that agents perform on the environment.

Helleboogh et al. (2005b) introduced time management adaptability in MASs. Time management adaptability allows a MAS to be adaptive with respect to its execution platform, where arbitrary and varying timing delays can produce error in the simulation. It also allows customizing the execution policy of a MAS to suit the needs of a particular application. The authors employed time models as a means to explicitly capture the execution policy derived from the application's execution requirements. They classified and evaluated time management mechanisms which can be used to enforce time models and also introduced a MAS execution control platform which combines both previous parts to offer high-level execution control. These three constituent parts of time management adaptability are shown in Figure 3-3.

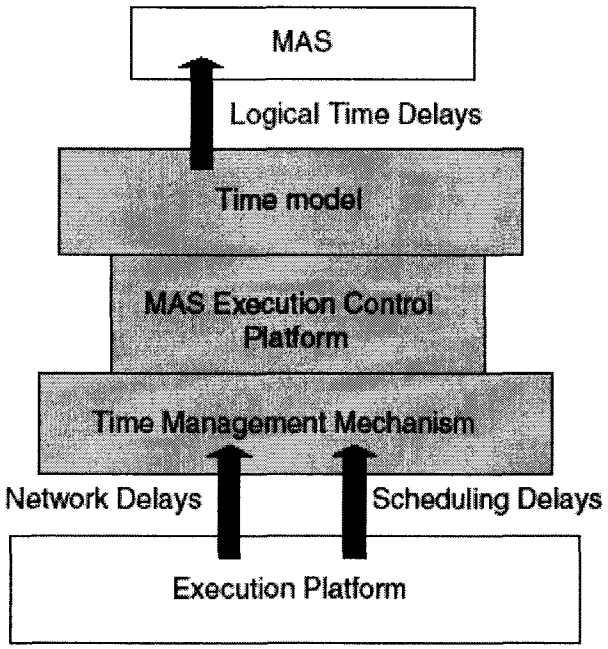

Figure 3-3 (Helleboogh et al. 2005b) Time management adaptability.

4. METHODOLOGY AND FRAMEWORK

In this chapter we present an overview of our approach to solve the inconsistency problem that we have presented in the Introduction. There are two major parts to the solutions approach: one is to use the new proposed language ACDL and another is to use the communication framework in the simulation engine that uses the ACDL to accomplish the communication task.

4.1 ACDL

We have first noted that, WS-CDL can be applied to model varieties types of interactions (peer to peer collaboration) among participating web services. Then we thought that we could introduce the same concept to model the communication between the agents in an artificial society. Therefore, we proposed ACDL to incorporate into the artificial society simulation so that all the interactions among the agents are properly preserved. But for social simulation perspective we need to have a very shortened and simple set of language constructs that specifies the contracts between the communicating agents. In that contract we simply want to state the rules of sequential message flows between the communicating parties. The following section gives the detailed overview of the ACDL model.

In chapter 2, we have seen that the *package* element is the root of the choreography in WS-CDL and therefore it is the container for both the static and dynamic parts. The syntax of WS-CDL *package* construct is given in Figure 4-1.

```
<package
   name="NCName"
   author="xsd:string"?
   version="xsd:string"?
   targetNamespace="uri"
   xmlns="http://www.w3.org/2005/10/cdl">

   <informationType/>*
   <token/>*
   <tokenLocator/>*
   <roleType/>*
   <relationshipType/>*
   <participantType/>*
   <channelType/>*

   Choreography-Notation*
</package>
```

Figure 4-1 (Barreto et al. 2005) WS-CDL *package* syntax

Other than the *Choreography-Notation* (used to define a choreography) construct the rest of the elements constitute the static part. As we are interested in the dynamic part as it contains the actual interaction, in ACDL we have made *choreography* element the root element. Now looking at the choreography syntax (Figure 4-2), we can see that it also has some static parts e.g. *variableDefinitions* as well as some housekeeping elements like *exceptionBlock, finalizerBlock* etc. The *Activity-Notation* constructs are used to define various types of activities in choreography. As we are only interested in the core interactions we just kept the *Activity-Notation* construct in our choreography syntax definition. From the major three type of WS-CDL activity (Table 2-2) we kept the *ordering structure* construct as using this we can specify in our choreography which activities to run in *parallel.* As discussed in the problem statement that the timing problem of the communicated messages are introduced due to the sequential nature of the simulation engine. Therefore, using this ACDL specification we can specify which interactions to be executed in parallel. The syntax of *parallel* activity is given in Figure 4-3. A *parallel* activity can contain 1 or more other activities. Also from Table 2-2, we kept the *basic activity* construct as using this we can specify an *interaction* activity (the basic building block of a choreography) in our choreography.

```
<choreography  name="NCName"
      complete="xsd:boolean XPath-expression"?
      isolation="true"|"false"?
      root="true"|"false"?
      coordination="true"|"false"? >

   <relationship  type="QName" />+

   variableDefinitions?

   Choreography-Notation*

      Activity-Notation

   <exceptionBlock  name="NCName">
      WorkUnit-Notation+
   </exceptionBlock>?

   <finalizerBlock  name="NCName">
      Activity-Notation
   </finalizerBlock>*
</choreography>
```

Figure 4-2 (Barreto et al. 2005) WS-CDL *choreography* syntax

```
<parallel>
Activity-Notation+
</parallel>
```

Figure 4-3 (Barreto et al. 2005) WS-CDL *parallel* activity syntax

From the WS-CDL *interaction* activity syntax we have kept only the mandatory elements (interaction and *participate*) for ACDL *interaction* activity syntax as they are the minimal constructs to specify any interaction.

```
<interaction  name="NCName"
              channelVariable="QName"
              operation="NCName"
              align="true"|"false"?
              initiate="true"|"false"? >

   <participate  relationshipType="QName"
                 fromRoleTypeRef="QName"
toRoleTypeRef="QName" />

   <exchange  name="NCName"
              faultName="QName"?
              informationType="QName"?|channelType="QName"?
              action="request"|"respond" >
     <send      variable="XPath-expression"?
                recordReference="list of NCName"?
                causeException="QName"? />
     <receive   variable="XPath-expression"?
                recordReference="list of NCName"?
                causeException="QName"? />
   </exchange>*

   <timeout  time-to-complete="XPath-expression"
             fromRoleTypeRecordRef="list of NCName"?
             toRoleTypeRecordRef="list of NCName"? />?

   <record  name="NCName"
            when="before"|"after"|"timeout"
            causeException="QName"? >
     <source  variable="XPath-expression"? |
expression="XPath-expression"? />
     <target  variable="XPath-expression" />
   </record>*
</interaction>
```

Figure 4-4 (Barreto et al. 2005) WS-CDL *interaction* activity syntax

And after all the derivation from WS-CDL we have the ACDL language definition presented in Figure 4-5.

```
choreography ::=
<choreography>
      Activity-Notation*
</choreography>
Activity-Notation ::=
<parallel>
            <interaction name="NCName" operation="NCName">
                  <participate fromRoleType="QName"
toRoleTypeRef="QName" />
             </interaction>*
</parallel>
```

Figure 4-5 ACDL definition

In each *interaction* in ACDL we have a *name* attribute which identifies an *interaction* element uniquely in the document and also an *operation* attribute which mainly contains the method name to invoke. The *participate* element contains *fromRoleType* and *toRoleType* attributes which specifies the entities involved in the interaction and specifies the direction of operation.

4.1.1 ACDL Example

Consider the following interaction between two agents: Agent A requests some amount of food from agent B and agent B sends back some food as per the request for food from agent A. In the UML sequence diagram this scenario is illustrated in Figure 4-6.

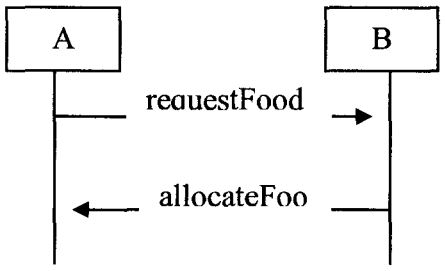

Figure 4-6 UML Sequence Diagram for Two Agents' Interaction

According to our simple ACDL definition this interaction could be modeled as Figure 4-7.

```
<?xml version="1.0" encoding="UTF-8"?>
<choreography>
    <parallel>
        <interaction name="Agent Interaction1" operation=" requestFood">
            <participate fromRole="A" toRole="B"/>
        </interaction>
        <interaction name="Agent Interaction2" operation="
allocateFood">
            <participate fromRole="B" toRole="A"/>
        </interaction>
    </parallel>
</choreography>
```

Figure 4-7 ACDL representation of Figure 4-6.

4.2 Communication Framework

We have introduced a *communication manager* in the society model. This manager is responsible for generation and then execution of an ACDL file for a particular simulation clock tick. In each simulation clock tick the agents inform the communication manager about their communication needs and the manager registers the communicating agents to build an ACDL file that captures all the interaction needs for that simulation clock. And at the beginning of the next simulation clock tick it executes the ACDL file where all the communicating agents are executed in parallel manner, which means all the agents requiring communication are executed in their requested time frame in parallel. The rest of the agents from the agent list which do not engage in any communication activities are then executed sequentially by the simulation engine. The proposed scheduling algorithm is shown in Figure 4-8.

```
initialize society landscape;
initialize agent population;
t = 0; // time counter, (0 <= t <= T) (T = max. simulated time)
while(t<= T){
      perform agent communication and agent action with the help of
communication manager for the communicating agents;
      perform agent actions for non-communicating agents;
      update society landscape;
update the agent list;
t++;
create choreography for next tick; // Communication Manager Generates
// the ACDL file to be executed in next tick.
}
generate statistical report;
```

Figure 4-8 Scheduling Algorithm with the Communication Manager.

In this manner we can avoid the sequential execution cycle of the underlying simulation engine which was the cause of the problem e.g. if agent A_2 needs to communicate with agent A_1 at some simulation time t = 7, the parallel execution mechanism guarantees that both A_1 and A_2 are in t = 7. This two phase proposed architecture is given in Figure 4-9 and Figure 4-10.

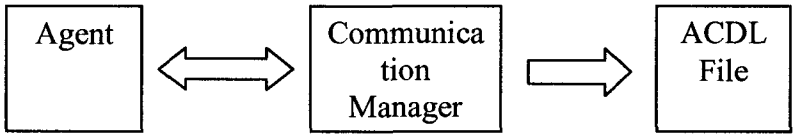

Figure 4-9 Phase 1: Generation of ACDL file

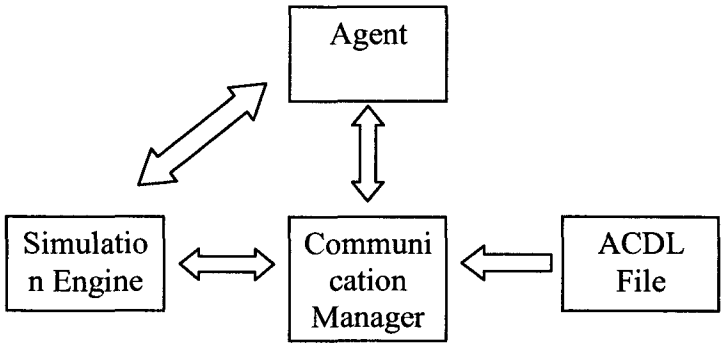

Figure 4-10 Phase 2: Execution of ACDL file.

While executing the agents concurrently we needed to have finer grain control on the agent execution codes by synchronizing the agents. For example, if agent A_2 requests agent A_1 for allocating some resources, each of the request interaction and respond interaction are captured in ACDL and while executing these interactions both A_2 and A_1's execution codes are run in parallel. By means of synchronization technique on the agent execution code we made sure that the requested resource is not consumed in A_1 before the A_2's request has been fulfilled. The pseudocode for agent execution is shown in Figure 4-11. The synchronized method for resource consumption is shown in Figure 4-12 where it locks the responding agent thread and the method for receiving resource is shown in Figure 4-13 where the requesting agent thread releases the lock from the responding agent thread.

```
1. Consume resource; // this is synchronized
2. Consider communication needs and register with the
   communication manager if needed;
3. Perform all other agent actions if needed;
4. update the agent local time;
```

Figure 4-11 Agent execution pseudocode.

```
While( ! requestProcessed){
    Wait();
}
Consume the resource for this clock tick;
```

Figure 4-12 Synchronized resource consumption pseudocode – locking the responder

```
if( ! resourceReceived){
    notify();
}
Add the received resource to the inventory;
```

Figure 4-13 Synchronized resource consumption pseudocode – releasing the responder

5. IMPLIMENTATION AND EXPERIMENTS

5.1 Experimental Setup

The Implementation of ACDL and the communication framework and the experiments were performed using the popular multi-agent simulation toolkit REPAST under Java 6 SDK in Windows XP environment running on Toshiba Satellite Intel® Celeron® 1.73 GHz, 1.99 GB of RAM.

5.1.1 Repast – The Simulation Environment

The Recursive Porous Agent Simulation Toolkit (Repast) is one of the leading free, open-source large-scale agent modeling toolkits available in pure Java (North and Macal 2007). (Tobias and Hofmann 2003) performed a survey on 16 agent modeling toolkits and mentioned that, "we can conclude with great certainty that according to the available information, Repast is at the moment the most suitable simulation framework for the applied modeling of social interventions based on theories and data" (Tobias and Hofmann 2003). It also supports .Net (Repast.Net), Python scripting (Repast Py) and point and click modeling for non-programmers in their latest release on 2005 called Repast Symphony. In our modeling and experimentation we will be using the Java version of Repast, called Repast J. It is an integrated simulation development framework that provides almost all the necessary constructs (Java API) for easy and rapid development, maintenance and execution of simulations.

Repast was created at the University of Chicago. Subsequently, it has been maintained by organizations such as Argonne National Laboratory. Repast is now managed by the nonprofit volunteer Repast Organization for Architecture and Design (ROAD). ROAD is lead by a board of directors that includes members from a wide range of government, academic, and industrial organizations. The Repast system, including the source code, is available directly from the web (ROAD 2005). Repast seeks to support

the development of extremely flexible models of living social agents, but is not limited to modeling living social entities alone.

From (ROAD 2005): "Our goal with Repast is to move beyond the representation of agents as discrete, self-contained entities in favor of a view of social actors as permeable, interleaved, and mutually defining; with cascading and recombinant motives. We intend to support the modeling of belief systems, agents, organizations, and institutions as recursive social constructions" (ROAD 2005).

5.1.2. Repast – Fundamental Components

Repast has four fundamental components, namely *simulation engine*, the *input/output* (I/O) system, the *user interface*, and the support *libraries* (Figure 5-1). These components are implemented in the core layer and using the external layer they are accessed by the user. Out of those four layers the most important layer is the *simulation engine* which is responsible for executing simulations. The *simulation engine* has four main parts, namely the *model*, the *controller*, the *agent* and the *scheduler*.

Repast *model* holds all the detailed specification and the definition of the simulation to be run by the *scheduler*. Those detailed specifications usually include the list of agents to be executed, the simulation initialization instructions, and the user interface specification. The *controller* works as a bridge between the *model* and the *scheduler*. It activates the *model* to be run and manages the interactions between the user or batch execution system and the *model*.

Agents are the key entities in agent based simulation and in Repast there could be various types of *agents* to model e.g. geographically situated agents, network-aware agents, etc. They are created by users from components and template classes within Repast. *Agents* receive data from the Repast *I/O* and also provide results to it. As the

scheduling is closely related to the problem and solution of this thesis, a detailed discussion on it is given in next sub-section.

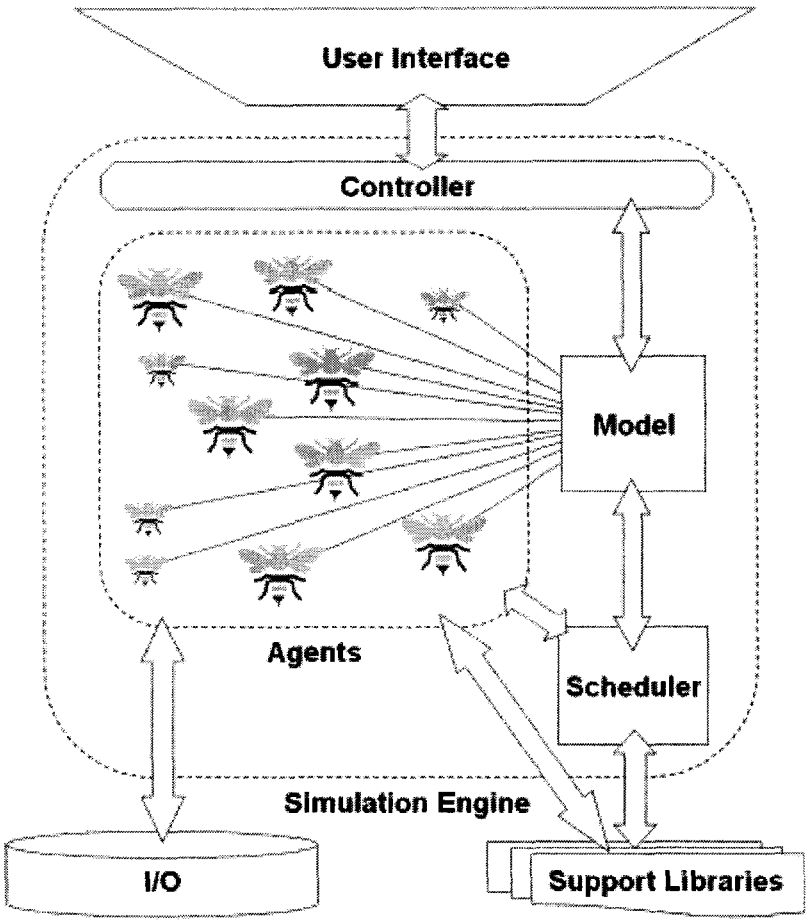

Figure 5-1 (North and Macal 2005) Repast – Fundamental Components.

5.1.3. Repast - Scheduling Mechanism

Repast operates like a discrete event simulator whose quantum unit of time is known as a *tick*. In each *tick* events are executed in an orderly manner. For example, if event A is scheduled for tick 1, event B for tick 2, and event C for tick 3, then event A will execute first, then event B and C at last. Repast is more like a discrete time simulator though it appears to users as a discrete event simulator.

By Repast scheduling mechanism at each of the simulation *tick* a set of agent behaviors which is called an *action* gets executed. RePast scheduling consists of three phases of behavior, a preparatory, an execution, and a post- or cleanup phase. RePast then schedules these in the appropriate order to occur every tick.

The agents' *actions* are eventually method calls on the agent objects. RePast represents these method calls separately from the objects themselves through the *BasicAction* class. A *BasicAction* consists of a single abstract public void execute() method. Any classes that sub-class a *BasicAction* must implement this method, and it is in this method that the actual method call or calls to be scheduled should occur. So, for example, if the agent behavior is encapsulated by a *step* method, then the *BasicAction's* execute method would iterate through all the agents and call this *step* method on each one. This *BasicAction* gets scheduled for execution at some specific tick.

BasicAction-s can be created in two ways, either by the modeler or implicitly by a Schedule object. In the first, the modeler will sub-class a *BasicAction*, implementing the execute method accordingly. This sub-class is usually created as inner class (anonymous or otherwise). In the second, the modeler provides an object reference and the name of the method she wishes to execute as arguments to a Schedule object's schedule method. The schedule object will then dynamically create and load the byte-code for a *BasicAction* class whose execute method calls the named method on the specified object. For example, suppose a model class contains a method named "run" in which all the agents are iterated through calling a method named "step" on each agent. To schedule this run method, the modeler passes the name of the method, that is, "run," and a reference to the model to the Schedule object. No sub-classing is necessary; the Schedule object does all the work. Furthermore, because the byte-code for the *BasicAction* is dynamically created, there is no performance penalty, as there might be with a solution that relied on reflection.

5.2 Implementation

This section presents how the ACDL file is generated and executed in our implementation in detail.

5.2.1. Communication Manager

The Communication Manager is implemented as a separate class called *CommunicationManager*. It has a temporary data structure where the communicating agents registers themselves in a particular *tick*. At the end of the tick the *CommunicationManager* generates the ACDL file from that temporary data structure with the help of Java DOM parser. And the beginning of next *tick* it executes the ACDL file generated from last *tick*. Basically when the SAX parser parses the ACDL file it creates the required instances from the ACDL file as per the Java class mappings described in the next sub-section and using Java Reflection API the instances are executed. Using Java thread and synchronization we have made sure that the activities run and parallel and proper synchronization i.e. when the requester and the responder agents are run in parallel, the responder agent should not consume its resources before it serves to request.

5.2.2. ACDL to Java Mapping

Following (Pu et al. 2007) for each ACDL element we have implemented a corresponding JAVA class. We already know that both the *parallel* activity and the *interaction* activity are ACDL activities. Therefore we modeled an abstract class called *ACDLActivity* as the base class of all *activity* classes and then extended it to create *ParallelActivity* and *InteractionActivity* class to create the ACDL instances *parallel* and *interaction* respectively. *ACDLActivity* class contains attributes name and *ActivityList* which is an *ArrayList<ActivityList>* that lists all the *activities* contained in current *activity* and also an abstract method Run to be implemented in its children classes. All the

actions in an activity are performed by the *Run* method. The corresponding class diagram is provided in Figure 5-2.

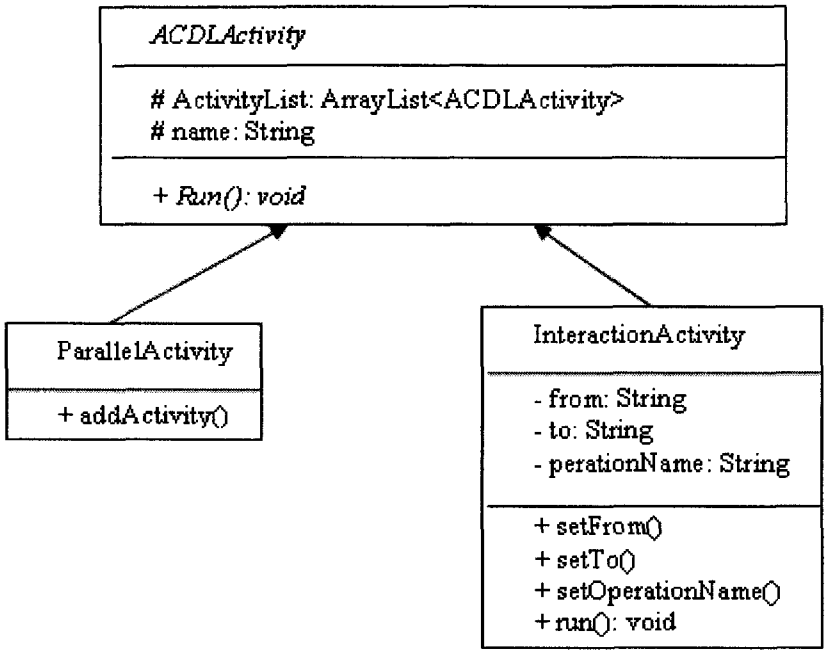

Figure 5-2 Class diagram for ACDL elements.

All the *activities* contained in an instance of Java class *parallelActivity* are executed concurrently. Therefore, every element in the *ArrayList<ActivityList>* will be fetched and called in its method Run in parallel by using JAVA threads. As *interaction* is the base atom of the choreography composition, it is the most atomic instruction to be executed. With the help of JAVA synchronization we have made sure that for a request – response interaction scenario, request is always processed first before the response while executing the parallel Run methods of the *interactionActivity* instance.

5.2.3. Agent Control

In our implementation of the framework and in all the case studies the agents have the control structure shown in Figure 5-3. This control is encapsulated in agent's *step* function. At each *tick* of the simulation this control is get executed.

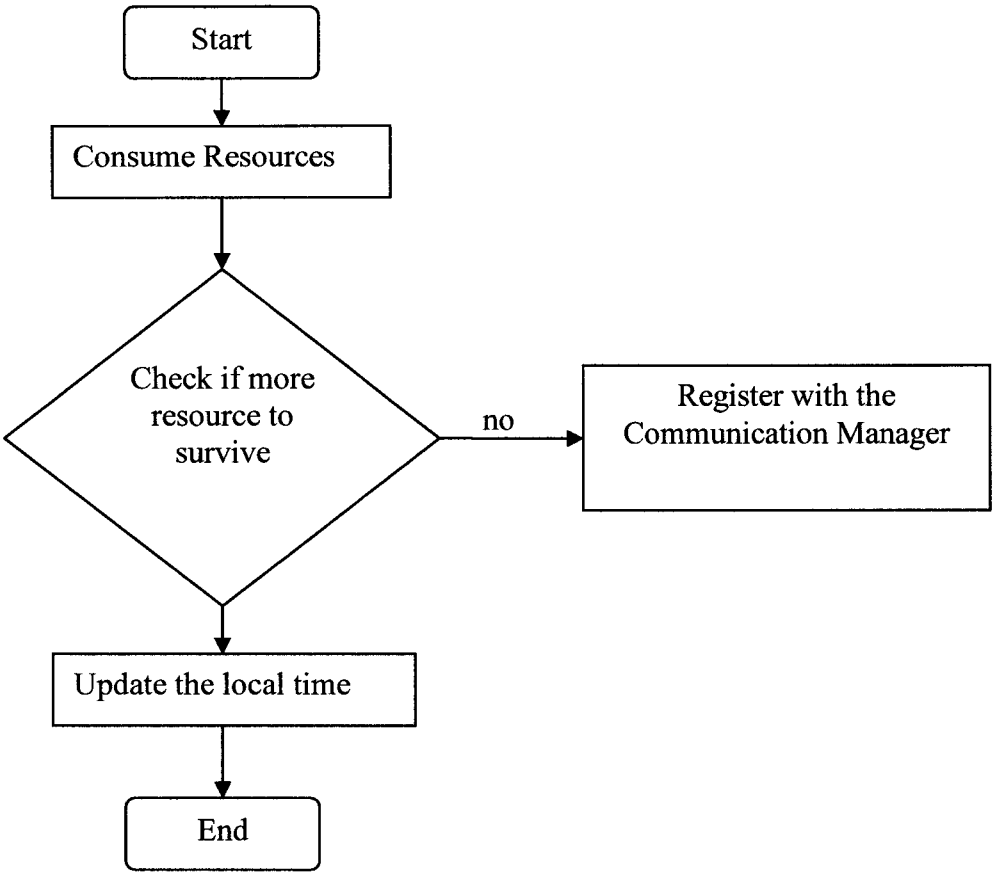

Figure 5-3 Agent's control flow – the *step* function

The Agent first consumes its resources for the current tick and then checks if it has enough resource to survive for another *tick* and if not it registers with the communication manager to take care of the communication by which the current agent requests resources from another agent who has enough resource. As a last step the agent update the local time to the time of the current simulation *tick*.

5.2.4. Simulation Control

The simulation control is encapsulated in the *execute* method inside the scheduler section of the model. This is the main control section of the simulation from where the simulator

makes calls to perform the agent controls as described in the previous section and also the calls to the communication manager to take care of all the communication issues. The core steps are:

1. Process choreography : The scheduler calls the *processChor* method of the *CommunicationManager* to process the generated ACDL file. In this step the *CommunicationManager* executes all the communication needs and also executes communicating agents in parallel for the current tick. The agents are executed by simply calling its *step* function as describes in the previous sub-section.

2. Process non-communicating agents: The scheduler just iterates through the agent list and executes all the non-communicating agents. The agents are executed by simply calling its *step* function as describes in the previous sub-section.

3. The scheduler updates the global time.

4. Create Choreography: While executing the communicating or non-communicating agents in steps 1 and 2, there might be more communication needs and agents might have registered their communication needs with the *CommunicationManager's* temporary data structure. In this step the scheduler call the *CommunicaitonManager's createChor* method to generate the ACDL file from that temporary data structure to make it ready to be executed in the next simulation clock *tick*.

5.3 Experimentation

This section presents the experiments and result analysis of our implemented framework performed with the help of few case studies. As our implementation is a new one, we have validated the experimental results against manually computed validation table. We have built our table based on simple scenario and less number of iteration for each of the experimental cases. All the agent interactions are resource based i.e. in each simulation ticks agents either consume resource or gathers resource by communicating requests to

other agents if resource level goes below a certain threshold value. And we also validate our results by comparing the resource levels in the validation table and the experimentation table. For each of the case studies we have the simulation run for two sub cases – one with the communication framework enabled and another and another is without the communication framework, just using the direct method calls as a means of communication / interaction. We also compared (based on the agent resource and the number of dead agents) how simulation results can vary in those two setups.

5.3.1 Case Study 1

With our implementation of the framework we have built a simple society of two agents A1 and A2.

5.3.1.1. Initialization. At the initialization of the simulation (when global and local times are all 0) Al has $100 (resource in simulation) and A2 has $0 of money.

5.3.1.2. Society (Simulation) Rules. At each simulation step each agent consumes $50. If any time during the simulation the resource goes below $0 the agent dies. We have set the communication criteria based on 'need for resources'. If any agent has less than $50 then it requests (initiates communication) the other agent to give it another $50 for survival. And if the requested agent has more than $50 it sends back the requester agent $50.

Table 5-1 shows the simulation run result for the test case with the implemented framework. Figure 5-4. shows the generated ACDL for the two simulation clock ticks when the global simulation time, T = 0 and T = 1. Note here that, at the end of time T = 0, both the request and corresponding response are generated in the ACDL. At time T = 1, there is no ACDL entry as there is no agent who has greater than $50 to serve another's request. Therefore at next simulation clock, T = 2 both the

agents die. We also have the simulation run results without the framework, shown in Table 5-2.

In the generated ACDL in Figure 5-4, the root element is *chorepgraphy*. It contains number of parallel elements which again holds number of interactions to be executed in parallel. Inside an *interaction* we have a *name* attribute which is uniquely generated and operation attribute which is actually a method name. A *participate* element is contained inside an interaction with two attributes *fromRole* and *toRole*. *fromRole* represents from which object the interaction is to initiate and *toRole* represents the destination object where the initiated interaction should direct to. Therefore the semantics of the interaction element is that, "call the method in operation attribute on the *toRole* object from the *fromRole* object". In the generated ACDL we represented the agent instances by their IDs. Therefore the semantics of the first interaction, "Agent Interaction1" is to call the '*processRequest*' method on Agent-1 from Agent-2.

```
<?xml version="1.0" encoding="UTF-8"?>
<choreography>
    <parallel>
        <interaction name="Agent Interaction1"
operation="processRequest">
            <participate fromRole="2" toRole="1"/>
        </interaction>
        <interaction name="Agent Interaction2" operation="addMoney">
            <participate fromRole="1" toRole="2"/>
        </interaction>
    </parallel>
</choreography>
```

Figure 5-4 Class Generated ACDL for the case study.

Table 5-1 Values received from the simulation run with the implemented framework.

Global Time(T)	Agent Specific Info.	A_1	A_2
0	Local Time	0	0
	Money($)	100	0
1	Local Time	1	1
	Money($)	0	0

Table 5-2 Values received from the simulation run without the implemented framework.

Global Time	Agent Specific Info.	A_1	A_2
0	Local Time	0	0
	Money($)	100	0
1	Local Time	1	0
	Money($)	50	0
	Local Time	1	1
	Money($)	50	Dead(-50)

It can be noted here that at the end of the simulation time, t = 1, with the framework implemented both A_1 and A_2 has $0 but without the framework implementation A_1 has $50 and A_2 dies. Table 5-3. shows how different values are generated from those two different approaches for the simulation. Results of Table 5-1 are the exact same as the results of our manually computed validation table.

Table 5-3 Values received from the simulation run without the implemented framework.

Methodology	Simulation Time	A_1 (resource)	A_2 (resource)	# of Dead Agents
Without Framework	0	100	0	0
	1	50	(-50)	1
With Framework	0	100	0	0
	1	0	0	0

5.3.2 Case Study 2

We have 5 agents' ($A_{1,\ldots,5}$) society for the second case study and run it for 3 iterations (*ticks*) so that the simulation results can be easily verified with manually computed validation table.

5.3.2.1. Initialization. At the initialization of the simulation (when global and local times are all 0) the agents have the values following Table 5-4.

Table 5-4 Initial values for the agents in case study-2.

A_1 (resource)	A_2 (resource)	A (resource)	A_4 (resource)	A_5 (resource)
300	150	101	50	102

5.3.2.2. Society (Simulaiton) Rules. At each simulation step each agent consumes \$50. If any time during the simulation the resource goes below \$0 the agent dies. We have set the communication criteria based on 'need for resources'. If any agent has less than \$50 then it requests (initiates communication) the other agents to give it another \$50 for survival. When an agent needs to communicate, it starts searching from the beginning of the agent list to search for the non-communicating agent that has enough resource to share. This scheme is simulation specific. We have kept this simple structure concentrating more on the core communication and timing issue. For example in any other simulation scenario an agent can communicate with the closest neighbors along its axes where it is situated in the simulation space. And if the requested agent has more than \$50 it sends back the requester agent \$50.

Table 5-5 Values received from the simulation run without the implemented framework.

Time (tick)	A_1 (resource)	A_2 (resource)	A_3 (resource)	A_4 (resource)	A_5 (resource)
0	300	150	101	50	102
1	250	100	51	0	52
2	150	50	1	0	2
3	50	0	1	-50(Dead)	-48 (Dead)

Table 5-6 Values received from the simulation run with the implemented framework.

Time (tick)	A_1 (resource)	A_2 (resource)	A_3 (resource)	A_4 (resource)	A_5 (resource)
0	300	150	101	50	102
1	250	100	51	0	52
2	150	50	1	0	2
3	0	0	1	0	-48 (Dead)

Table 5-5 represents the simulation values with the implemented framework and Table 5-6 represents the values with implementation and Table 5-7 shows the different results that we have got from those two runs based on number of dead agents. Table 5-6 produces the same values as our verification table.

Table 5-7 Values received from the simulation run without the implemented framework.

Methodology	**# of Dead Agents**
Without Framework	2
With Framework	1

6. CONCLUSION and FUTURE WORK

This thesis proposes an agent communication language called ACDL, which is a subset of WS-CDL and a communication framework to model proper communication among the agents in an artificial society. With the trivial simulation case studies our experimental results show that the communication framework is able to produce the expected simulation results and therefore the correctness of the messages intended to communicate is preserved in the temporal context. Also from the experiments we have found that our implementation of the framework is able to solve the timing problem mentioned in the introduction. Also the implemented framework produces more accurate result than the regular simulation scheme where the communications are modeled by direct method calls.

Possible future work for this would be to build the framework as a package into the Repast agent simulation toolkit and test it with large scale simulation scenarios like 'sugarscape' (Epstein and Axtell 1996) and also to extend the language vocabulary by adopting more language constructs from WS-CDL to provide the agents with more interaction scenarios. Also, the regular agent communication languages constructs from KQML or FIPA ACL could be incorporated with ACDL language constructs and in that way more generic XML parsers can be used to process KQML or FIPA ACL performatives (actions).

7. REFERENCES

1. Banks, J., Carson, J., Nelson, B. and Nicol, D. (2005). Discrete-event system simulation - fourth edition. *Pearson.*

2. Epstein, J.M., and Axtell, R.L. (1996). Growing artificial societies. Social sciences from the bottom up. *Cambridge, MA: MIT Press.*

3. Gilbert, N. and Conte, R. (1995). Artificial Societies: the computer simulation of social life. *UCL Press, London.*

4. Schelling, T.C. (1969). Models of Segregation. American Economic Review. Papers and Processdings 59 (2): 488-93.

5. Schelling, T.C. (1971a). Dynamic Models of Segregation. Journal of Mathematical Sociology 1: 143-86.

6. Schelling, T.C. (1971b). On the ecology of Micromotives. The Public Interest 25 (Fall): 61-98.

7. Schelling, T.C. (1978). Micromotives and Macrobehavior. Norton.

8. North, M. and Macal, C. (2007). Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation Oxford University Press: New York, NY, 2007

9. Sawyer, R. K. (2003). Artificial Societies: Multiagent Systems and the Micro-Macro Link in Sociological Theory. Sociological Methods Research 2003; 31; 325.

10. Malsch, T., Schlieder, C., Kiefer, P., Lübcke, M., Perschke, R., Schmitt, M. and Stein, K. (2007). Communication Between Process and Structure: Modelling and Simulating Message Reference Networks with COM/TE. *Journal of Artificial Societies and Social Simulation vol. 10, no. 1.*

11. Timm, I., J., Scholz, T., Herzog, O., Krempels, K. and Spaniol, O. (2006). *From Agents to Multiagent Systems. Multiagent Engineering; International Handbooks on Information Systems. Springer Berlin Heidelberg.*

12. Sansores, C. and Pavón, J. (2005)Agent Based Simulation for Social Systems: From Modeling to Implementation. *Lecture notes in computer science ISSN 0302-9743. Springer, Berlin, ALLEMAGNE.*

13. Drogoul, A., D. Vanbergue, T. Meurisse.(2002) *Multi-agent Based Simulation: Where Are the Agents?* In: J.S. Sichman, et al. (Eds.): *Multi-Agent-Based Simulation II: Third International Workshop, MABS 2002*. Lecture Notes in Computer Science, Vol. 2581. Springer. Bologna, Italy (2002), 1-15.

14. Wang, F., Turner, S. and Wang, L. (2005). Agent Communication in Distributed Simulations. *Multi-Agent and Multi-Agent-Based Simulation; Agent Communication in Distributed Simulations*. Springer Berlin / Heidelberg.

15. Buzing, P., Eiben, A., and Schut, M. (2005). Emerging communication and cooperation in evolving agent societies. *Journal of Artificial Societies and Social Simulation*, 8(1).

16. Eiben, A. E., Schut, M. C., and Toma, T. (2005). Comparing multicast and newscast communication in evolving agent societies. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation* (Washington DC, USA, June 25 - 29, 2005). H. Beyer, Ed. GECCO '05. ACM, New York, NY, 75-81.

17. Buzing, P.C., Eiben, A.E., Schut, M.C., Toma, T. (2004). Cooperation and communication in evolving artificial societies. *Evolutionary Computation, 2004.* CEC2004. Congress on Volume 2, 19-23 June 2004 Page(s):2030 - 2037 Vol.2.

18. Buzing, P.C., Eiben, A.E. and Schut, M.C. (2003). Evolving agent societies with VUscape. In W. Banzhaf, T. Christaller, P. Dittrich, J.T. Kim, and J. Ziegler,

editors, *Proceedings of the Seventh European Conference on Artificial Life*, volume 2801, pages 434-441. Springer Verlag, 2003.

19. Silberschatz, A., Galvin, P. and Gagne, G.(2002). Operating System Concepts Seventh Edition. *John Wiley & Sons, Inc.*

20. Lawson, B. and Park, S. (2000). Asynchronous Time Evolution in an Artificial Society Model. *Journal of Artificial Societies and Social Simulation* vol. 3, no. 1.

21. Tobias R and Hofmann C (2003) Evaluation of free Java-libraries for social scientific agent based simulation. *Journal of Artificial Societies and Social Simulation*

22. ROAD: Repast 3.1 (2005), available at http://repast.sourceforge.net/repast_3/index.html.

23. Collier, N. (2003). RePast: An Extensible Framework for Agent Simulation. http://www.econ.iastate.edu/tesfatsi/RepastTutorial.Collier.pdf

24. Axelrod, R. (1997). The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration. *Princeton, N.J.: Princeton University Press.*

25. Filippo Castiglione (2006). Scholarpedia Article, available at: http://www.scholarpedia.org/article/Agent_based_modeling 1(10):1562.

26. Builder, C. H. and Bankes, S. C. (1991). Artificial Societies: A concept for basic research on the societal impacts of information technology. RAND Report p-7740.

27. Genesereth, M. R. and Fikes, R.E. (1992). Knowledge Interchange Format, Version 3.0 Reference Manual . Technical report logic-92-1, Computer Science Department, Standford University.

28. Woolridge, M. and Wooldridge, M. J. 2001 *Introduction to Multiagent Systems*. John Wiley & Sons, Inc.

29. Artikis, A. and Pitt, J. (2001). A formal model of open agent societies. In Mueller, J., Andre, E., Sen, S., and Frasson, C., editors, Proceedings of the Fifth International Conference on Autonomous Agents, pages 192-193, Montreal, Canada. ACM Press.

30. Davidsson, P. (2001). Categories of artificial societies. Lecture Notes in Computer Science, 2203.

31. Davidsson, P. and Johansson, S. (2006). On the Potential of Norm-Governed Behavior in Different Categories of Artificial Societies. *Computational and Mathematical Organization Theory*, Vol. 12(2-3), pages 169-180, Springer, 2006.

32. Gokturk, E. and Polat, F. (2003). Implementing agent communication for a multi-agent simulation infrastructure on HLA. In Proceedings of the *18th International Symposium on Computer and Information Sciences*, LNCS. Springer-Verlag.

33. Pawlaszczyk, D.; Timm, I. J. (2007). A Hybrid Time Management Approach to Agent-based Simulation. In: Proc. *29th Annual German Conference on Artificial Intelligence (KI 2006), LNCS*, Vol. 4314, pp. 374-388. Springer, 2007.

34. Braubach, L., Pokahr, A., Lamersdorf, W., Krempels, K. H. ans Welk, P. O. (2004): A Generic Simulation Service for Distributed Multi-AgentSystems. In: Trappl, R. (eds.): *Cybernetics and Systems* (Vol. 2), Vienna, Austria, (2004) 576-581.

35. Huang, j., Tung, M., Wang, K. and Lee., M. (2005a). Smart Time Management—the unified time synchronization interface for the distributed simulation.

Computer Standards & Interfaces Volume 27, Issue 2, January 2005, Pages 149-161,

36. Huang, j., Tung, M., Hui, L. and Lee., M. (2005b) An Approach for the Unified Time Management Mechanism for HLA. *SIMULATION*.2005; 81: 45-56.

37. Helleboogh, A., Holvoet, T., Weyns, D. and Berbers, Y. (2005a). Extending time management support for multi-agent systems. In *Multiagent and Multiagent-based Simulation*, New York, USA, Lecture Notes in Computer Science, Vol. 3415, 2005.

38. Helleboogh, A., Holvoet, T., Weyns, D. and Berbers, Y. (2005b). Towards Time Management Adaptability in Multi-agent Systems. *Adaptive Agents and Multi-Agent Systems III* (2005), pp. 88-105.

39. Barreto, C., Burdett, D., Fletcher, T., Kavantzas, N., Lafon, Y., & Ritzinger, G. (Eds.) (2005): Web Services Choreography Description Language version 1.0. In *W3C Candidate Recommendation.* Retrieved November 9, 2005 from http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109.

40. Fredlund., L. (2006). Implementing WS-CDL. *In Proceedings of the second Spanish workshop on Web Technologies (JSWEB 2006).* Universidade de Santiago de Compostela, November 2006.

41. Mendling, J. and Hafner, M. (2006) From WS-CDL Choreography to BPEL Process Orchestration. *Technical Report JM-2006-07-24.* Vienna University of Economics and Business Administration.

42. Pu, G., Shi, J., Wang, Z., Jin, L., Liu, J., and He, J. (2007). The Validation and Verification of WSCDL. *In Proceedings of the 14th Asia-Pacific Software Engineering Conference* (December 04 - 07, 2007). APSEC. IEEE Computer

Society, Washington, DC, 81-88. DOI= http://dx.doi.org/10.1109/APSEC.2007.93

43. North, M. J. and Macal, C. M. (2005). Escaping the accidents of history: An overview of artificial life modeling with Repast. In *Artificial Life Models in Software*, *A. Adamatzky and M. Komosinski, Eds. Springer, Heidelberg*, Germany. 115--141.

VITA AUCTORIS

Muhammad Naushin Hasan was born in Chittagong, Bangladesh. He received his bachelor degree in Computer Information Systems at the University of Windsor. Currently he is completing his masters' degree in computer science from the University of Windsor and expects to graduate in May 2009.