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

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

VIRTUAL COMMUNITIES AND TEAM FORMATION

by

YANRU ZHANG

A thesis

submitted to the School of Computer Science
and the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements

for the degree of
Master of Science, Information and System Science

School of Computer Science
Carleton University
Ottawa, Ontario
Canada

September 5, 2003

© Copyright 2003, Yanru Zhang

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitons et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88792-8

Our file Notre référence

ISBN: 0-612-88792-8

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

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

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

Canada

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis.

Virtual Communities and Team Formation

submitted by

YANRU ZHANG

B.Eng., Beijing Union University
China, 1995

in partial fulfillment of
the requirements for the degree of
Master of Science, Information and System Science

Michael Weiss, Ph.D.

(Thesis Supervisor)

Douglas Howe, Ph.D.

(Director, School of Computer Science)

Carleton University

September 5, 2003

Abstract

Virtual communities play an important role in bringing people together and establishing trust. A key research issue is to study the formation, evolution, and dissolution of teams in virtual game communities.

Based on the analysis of the requirements of game communities, we introduce four essential elements for team development in different game phases. A lobby service is a basic but important element to facilitate the team formation. A reputation mechanism is an integral part of a lobby service. Coordination models and protocols are needed for game collaboration, and cover team formation, dissolution and invitation of new players. We present a coordination model based on the Layered Reactive Tuple Spaces, protocols of joining and leaving a game, and a reputation model for game communities.

We chose a cooperative multi-player online game as our case study to illustrate the proposed models and protocols in team formation and dissolution.

Acknowledgements

I would like to take this opportunity to express my gratitude to my thesis supervisor, Dr. Michael Weiss, for his technical guidance, enthusiastic support and heuristic discussion. His earnest research attitude often inspires me with valuable ideas during my thesis research and will direct my future work.

I would also like to thank Dr. Son Vuong of the University of British Columbia, for his comments for this work.

Deep appreciation is expressed to my dearest parents, and my wonderful husband Wei Chen. Without their constant love, support and encouragement, this work could not be completed.

My fellow graduate students, Minying Huang and Ivan J. Araujo, had valuable discussions with me and gave me suggestions on doing research.

This thesis is a subproject of SCE (Scalable Collaborative Environment) Project, which is mainly supported by CITR (Canadian Institute for Telecommunications Research).

Yanru Zhang

Table of Contents

ABSTRACT	III
ACKNOWLEDGEMENTS.....	IV
TABLE OF CONTENTS.....	V
LIST OF TABLES.....	VIII
LIST OF FIGURES.....	IX
CHAPTER 1 INTRODUCTION	1
1.1 Introduction to Virtual Communities	1
1.2 Motivation	2
1.3 Goals.....	4
1.4 Thesis Outline.....	5
CHAPTER 2 BACKGROUND AND RELATED WORK.....	7
2.1 Virtual Communities	7
2.1.1 Definition of Virtual Communities	7
2.1.2 Applications.....	8
2.2 Issues of Virtual Communities	9
2.2.1 Awareness	9
2.2.2 Privacy.....	10
2.2.3 Trust.....	10
2.2.4 Knowledge Sharing	11
2.2.5 Collaboration and Coordination	11
2.3 Collaborative Virtual Game Communities.....	12
2.3.1 Introduction to Online Games	12
2.3.2 Requirements of Game Communities	14
2.3.3 Lobby Services in Virtual Game Communities	17
2.3.4 Reputation in Virtual Game Communities.....	18

2.3.5 Coordination in Virtual Game Communities	20
2.4 Agents and Multi-Agent Systems.....	22
2.4.1 Agents.....	22
2.4.2 Multi-Agent Systems.....	22
2.4.3 Team Formation in Agents.....	23
2.5 Coordination Models.....	24
2.5.1 Coordination Model Components	24
2.5.2 Coordination Model Types.....	26
2.5.3 Linda Model	30
2.5.4 Extensions to Linda Model.....	32
2.6 Chapter Summary	39
CHAPTER 3 MODELING VIRTUAL GAME COMMUNITIES.....	41
3.1 Lobby.....	41
3.2 Layered Reactive Tuple Spaces Coordination Model.....	43
3.2.1 Layered Model	44
3.2.2 Reactive Model.....	46
3.2.3 Primitives of LRTS Coordination Model	48
3.3 Reaction in LRTS Coordination Model	50
3.3.1 Fields, Tuples and General Tuples	50
3.3.2 Reaction Tuples and Reaction Class	52
3.3.3 Trigger a Reaction in LRTS Model.....	53
3.4 Protocols of Joining and Leaving the Game	54
3.4.1 Joining the Game.....	56
3.4.2 Leaving the Game	57
3.4.3 Coordination in Playing the Game	59
3.5 Reputation Management.....	60
3.5.1 Reputation Management Model in Game Communities	60
3.6 Chapter Summary.....	62

CHAPTER 4

CASE STUDY: MULTI-PLAYER ONLINE GAME AGENT WORLD 64

4.1 Introduction to a Game Agent World.....	64
4.2 Examples of Applying LRTS Coordination Model.....	66
4.2.1 Selecting and Joining a Team with Reputation Information	67
4.2.2 Beginning a Game	70
4.2.3 Inviting Other Players	72
4.2.4 Newly Joined Player.....	76
4.3 Reactions in Game Agent World.....	78
4.3.1 Checking a New Player's Identification.....	79
4.3.2 Leaving Game before Game is over.....	80
4.3.3 Message Communication	81
4.3.4 Moving Steps in Game Board	83
4.4 Chapter Summary.....	85

CHAPTER 5 CONCLUSIONS 86

5.1 Contributions.....	86
5.2 Future Work	88

CHAPTER 6 BIBLIOGRAPHY 89

List of Tables

Table 2.1 Types of coordination models	28
Table 3.1 Field and Tuple with null value and actual value	51
Table 3.2 Main reactions in LRTS coordination model	54
Table 4.1 Reactions in game Agent World.....	78

List of Figures

Figure 2.1 Pre – game element: Lobby service	15
Figure 2.2 Game elements: Coordination mechanism and Protocols.....	16
Figure 2.3 Post – game element: Reputation management	17
Figure 2.4 Illustration of JavaSpaces technology	33
Figure 2.5 Example class Car that implements Entry interface	34
Figure 2.6 Main methods declared in JavaSpace interface	34
Figure 2.7 MARS system interface	37
Figure 3.1 Visualization social relationships in a lobby	42
Figure 3.2 Class diagram of Lobby and its components	42
Figure 3.3 Layered Reactive Tuple Spaces Coordination Model.....	46
Figure 3.4 Tuple spaces in LRTS model	48
Figure 3.5 TupleSpace interface in LRST	49
Figure 3.6 Class diagram of TupleSpace interface and its implementation	50
Figure 3.7 Class diagram of Tuple and Field	50
Figure 3.8 Examples of the general tuple	51
Figure 3.9 Format and example of the reaction tuple.....	53
Figure 3.10 Abstract class Reaction	53
Figure 3.11 A player connects to the lobby.....	56
Figure 3.12 Protocol of joining a game	57
Figure 3.13 Protocol of inviting a new player to continue the game.....	59
Figure 3.14 Intra - game coordination (actions and messages)	60
Figure 3.15 Virtual game communities’ reputation model	61
Figure 4.1 Each agent represents a player who participates in the game	64
Figure 4.2 Lobby server starts game servers, each of which has a local tuple space	67
Figure 4.3 A game tuple is posted to the global tuple space	67
Figure 4.4 Protocol of joining Agent World game.....	68

Figure 4.5 Global tuple space reads all game servers' status	68
Figure 4.6 Lobby interface shows the status of each game	69
Figure 4.7 New player Alice's reputation (has not been rated yet)	70
Figure 4.8 Player Alice's reputation changed after playing once.....	70
Figure 4.9 Player Alice's reputation changed after urgently leaving a game.....	70
Figure 4.10 Players joining the game have access to the local tuple space.....	71
Figure 4.11 A join tuple is posted	71
Figure 4.12 Player blocks for begin tuple	71
Figure 4.13 Local tuple space unblocks waiting players: posting a begin tuple	72
Figure 4.14 Inviting a player in different game server: invitation from inviter	73
Figure 4.15 Inviting a player in different game server: decision from invitee.....	73
Figure 4.16 Inviting a player via the layered tuple spaces	74
Figure 4.17 Game Server (A) writes invitation tuple to the global tuple space	74
Figure 4.18 Local tuple space stores invitation tuple	75
Figure 4.19 Player reads invitation tuple from the local tuple space	75
Figure 4.20 Interface for Player B to make a decision (from player Alice)	75
Figure 4.21 New player X reads current move tuples from the local tuple space...	76
Figure 4.22 Local tuple space stores a new player tuple	77
Figure 4.23 New player picks up move tuples if they exist	78
Figure 4.24 Player tuple with player info is posted to the global tuple space.....	79
Figure 4.25 Installing Checking ID reaction in the global tuple space	79
Figure 4.26 Checking ID reaction object is retrieved and executed.....	80
Figure 4.27 Installing Leave reaction in the local tuple space	81
Figure 4.28 Leave reaction object is executed	81
Figure 4.29 Fragment code for leave reaction.....	81
Figure 4.30 Message tuple is posted to the local tuple space	82
Figure 4.31 Fragment code for message reaction.....	82
Figure 4.32 Reading message from local tuple space	82
Figure 4.33 Move tuple is posted to the local tuple space.....	83

Figure 4.34 Players coordinate steps via local tuple space.....	83
Figure 4.35 Installing Move reaction	84
Figure 4.36 Fragment code of Move reaction	85

Chapter 1 Introduction

1.1 Introduction to Virtual Communities

With the growth of global computer networks¹, virtual communities have become a new way for people to interact. People are beginning to realize that the Internet and other network technologies are not only affecting the way businesses operate, but also our everyday lives [22]. One of the simplest examples of our participation in a virtual community is an online chat. Through a chat application, one can have discussions on a large variety of subjects with a multitude of different people, many of whom were never met in person.

However virtual communities are in no way limited to online chat or bulletin boards. Some virtual communities are commercial ones, operating for a profit and are used by millions of people. Some virtual communities are for education purposes such as distance learning. Within online learning communities, students and teachers located in geographically different areas are able to communicate and interact with each other. Still other virtual communities are used to support local groups, such as helping people who need health services.

¹ Statistics from Nielsen//NetRatings, a global standard for Internet audience measurement and analysis, shows that “the number of people with access to the Internet via a home PC increased from 563 million people in Quarter 3, 2002 to 580 million in Quarter 4 2002.”

1.2 Motivation

Virtual communities play an important role in bringing people together, sharing knowledge, and establishing trust. Thanks to the development of Internet technology and networks, virtual communities have become a hot research topic and many issues related to virtual communities come to the stage including trust management [1], knowledge sharing [27] and privacy [15] etc. Much research has been done on already formed teams¹ or communities. Team development in virtual communities (formation, evolution, and dissolution) has received less attention. However, dynamic and quick team formation is one of the important phases of community development and maintenance. Community members do not only pay attention to the advanced technology (for example, rich three-dimensional graphics) to allow people in different locations to meet, but also call for social aspects of the team development.

When looking further into team formation and evolution, another interesting problem that attracted us was the design of support systems for coordination and collaboration between the participants of the communities. It is clear that collaboration and coordination is necessary between users from team formation to dissolution. The community's members need to coordinate in order to gain more knowledge about this community, and about their partners, and then to make decisions based on the information they obtained and further cooperate with each other. Coordination helps

¹ See Section 1.3 for our definition of the term “team”

community members find each other and/or potential teams. We should have a solution to the question: how do we support the collaboration and coordination in the earlier team development among the community participants?

Among many kinds of online communities, we are particularly interested in online multi-player game communities that become one of the most promising Internet applications. In game development, research on Artificial Intelligence (3-Dimension, path finding algorithms etc.) has been done, but less effort has put on coordination for the multiple players and team development. However the trend of game development is from single player games to team-based games, so it is compulsory for players to coordinate their ideas and activities to form a team and to reach their common goals. This thesis is part of the SCE (Scalable Collaborative Environment) Project [47]. The initial focus of this project is on the collaborative multi-player games [46].

Though coordination models have been presented for some application domains from e-commerce to network management, based on our analysis, few of them are really suitable for the domain of Internet games. Some models are built for peer-to-peer communications, some specifically for mobile agents. They are not flexible and do not scale to the needs of distributed Internet applications. By proposing our own framework and coordination model for online games we fill this gap. Our specific focus is on supporting the formation and dissolution of teams.

Besides the coordination issue, there are still other elements to be considered during team formation after initially analyzing the requirements of online game communities. Communities should provide some services to support linking people together, which is the basis of team formation. These services, intended to facilitate team formation, are called lobby services. Many current online game communities (such as Yahoo! Games [48]) are providing only limited lobby services.

1.3 Goals

The term “team” in this thesis refers to a certain number of community members forming a group for a specific game. Online game communities are composed of a number of different size, different game type teams. In order to form the team efficiently, we will analyze the domain requirements of online multi-player games. Based on the results of our study, we will design a lobby service, describe the coordination model for players, and present protocols for joining, and leaving the game, and inviting new players to a game. A reputation mechanism will be used in the lobby services, in invitation protocols, and in joining protocol. The Coordination mechanism plays a central role in the collaborative virtual game environment. We will implement a coordination model API (Application Programming Interface) and apply the models, protocols and lobby services together to a multi-player online game. Hence we could infer what role these issues play in the formation and dissolution of online game communities.

Many online games are competitive. Cooperation is still possible in those games, and often of decisive strategic importance. For example, consider a battle game, in which temporary alliances against a mutual opponent can benefit all members of the alliance. However, richer cooperation behavior can be studied more conveniently in the context of a cooperative game. As cooperative games require that all players work towards a common goal, the notion of a social goal is already embedded into the game by design. We will choose this type of game to be our case study.

The proposed lobby services of online communities, coordination models and protocols are general rules and are not restricted to online games only. Another similar domain such as distance learning is an interesting field as well and can be used as a suggested study area for future work.

This thesis intends to partially realized the SCE Project's aims: specifying, developing and demonstrating an open and scalable collaborative environment for the development and deployment of Internet-based distributed applications involving several simultaneous participants who are geographically dispersed.

1.4 Thesis Outline

The rest of the thesis is structured as follows:

- *Background Material.* Chapter 2 introduces the basic concept of virtual communities, their applications, and issues. It then describes the requirements for

virtual game communities. It elaborates on four important elements when considering team formation in game communities: lobby service, reputation mechanism, coordination model, and protocols for joining, and leaving a team, and inviting other players. Later on, in the coordination model concept, example systems and their applications are presented, since each of them has a different application domain. The Linda coordination model, which serves as the foundation of our proposed model, is described in detail, as well as its extensions.

- *Modeling virtual game communities.* Chapter 3 opens with a discussion of a lobby service for game communities. The proposed coordination model is introduced, and implementation details of the proposed model, are presented. This discussion is followed by the protocols of joining, and leaving a game, and inviting partners.
- *Case study.* In chapter 4, we apply the coordination model, protocols for joining or leaving teams, and the reputation model to a cooperative online multi-player game, Agent World. Typical coordination and reaction scenarios are elaborated.
- *Concluding remarks.* Finally chapter 5 summarizes the thesis contributions, and suggests topics for future work.

Chapter 2 Background and Related Work

2.1 Virtual Communities

2.1.1 Definition of Virtual Communities

Analyzing various definitions of community, [20] provides the following concise definition: “1) a group of people; 2) who share social interaction; 3) and some common ties between themselves and the other members of the group; 4) and who share an area for at least some of the time.” A community has also been defined as “a social grouping that exhibits to varying degrees: shared spatial relations, social conventions, a sense of membership and boundaries, and an ongoing rhythm of social interaction” [32].

Based on the meaning of the word "virtual" (i.e., computer-mediated), a virtual community can be understood as a social grouping that emerges from the webs of personal relationships embedded in computer networks. It describes the union between individuals or organizations who share a set of common interests, and use electronic media to communicate with each other. Their ability to communicate is not restricted by time or location [33]. The earlier virtual communities are based on the desire of participants who would like to communicate through long distance. MUD is a well-known socially motivated system supporting the virtual community.

2.1.2 Applications

The concept of virtual communities has been adopted in many different application domains such as e-commerce (online auctions), e-learning (virtual classrooms), and e-gaming (multi-player games) and many other fields.

➤ **Commercial Communities:**

Suppose you want to buy a product on eBay [43], an online auction site. The buyers and sellers interested in that product temporarily form a community created for the purpose of advertising the product, negotiating its price, and, perhaps, coordinating the transfer of ownership through an escrow service.

➤ **Educational Communities:**

If you have registered as a distance learner with an online university, you will meet with your professor and classmates in a virtual classroom. Virginia Tech Cyberschool [24] is a successful online university. Consumers become to know the meaning and understand the concept of e-learning, and are willing to pay.

➤ **Entertainment Communities:**

Another scenario is online gaming, where you could meet with friends online to play multi-player games, improve your skill, accumulate points, or increase your ratings. Everyday, tens of thousands of players play online such as at the Yahoo game site [48].

Other types of communities such as search communities, subscriber-based communities, email-based communities can be found in “Online Community Report” [44].

2.2 Issues of Virtual Communities

Virtual communities are intended to aggregate people, and to provide them with an engaging environment in which they can interact with others. People can find out who are community members, which community members have some special expertise, and what community members are doing. It allows them to coordinate their activities with those of others. In that online auction example, for instance, sellers have more opportunities to expand their markets, while buyers can extract more value from sellers. What then are the basic issues of virtual communities?

2.2.1 Awareness

Awareness is defined as the ability to maintain and constantly update a sense of our social and physical context. Awareness is an essential precondition for making contact with other members of a community, since we need the information of who is in the same virtual place, who shares common interests with us, and who is available for collaboration when we join a community [32]. Increased awareness encourages informal spontaneous communication between community members, and contributes to their ability to make informed decisions.

“Socialware” [21] is an important research result in supporting network communities and awareness using multi-agent systems. The socialware systems try to build a social infrastructure to help community activities. Two systems mentioned in [21] are

“community organizer” and “community board”, which bring people together to form communities and support the smooth communication respectively.

2.2.2 Privacy

Awareness needs to be balanced with privacy. If we consider common interests as the basis for the formation of a community, it is sometimes difficult to find users who share our interests on the network. The reasons for this are the large size of the network, people do not like to post too much of their personal information in a publicly accessible place, or do not update their profiles as their interests change. We need to address this issue by not only providing a means for safely and securely introducing users with similar interests to each other, but also by protecting their privacy. The *Yenta* system [15] addresses the issue by introducing the people to each other, while protecting their privacy when sharing their interest profiles.

2.2.3 Trust

The stability of a community depends on the right balance of trust and distrust [1]. We are faced with information overload, increased uncertainty, and risk-taking as prominent features of modern life. As members of a society, we cope with these complexities and uncertainties by relying on trust. Trust is the basis for all social interaction. As a survey on trust in e-commerce communities summarizes, “Trust is about social relationships, and about building networks that deliver what they promise, be it a product, a

collaboration, or simply reliable information” [17]. Research in [1] follows this definition for trust: “trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action (or independently of his capacity of ever to be able to monitor it) and in a context in which it affects [our] own action.” The trust model is based on reputation mechanism. Concepts of reputation and its model in virtual game communities especially will be discussed in Chapter 3 (section 3.5).

2.2.4 Knowledge Sharing

Differences in knowledge encourage people to communicate. Knowledge sharing facilitates the formation of virtual communities. It also plays an essential role in the maintenance of the communities so that people can benefit from participating in it. This will, in turn, attract more people to join, and thus contribute to the growth of virtual communities. The richer the knowledge accumulated in the communities, the more interesting the communities become for their members, and the more difficult for members to leave. Among the systems that support knowledge sharing, system “CoMeMo” [27] is one that supports the community knowledge evolution.

2.2.5 Collaboration and Coordination

Collaboration and coordination are required when forming a community, pursuing individual goals while meeting community's social goals, and changing the structure of a community. Community members have incomplete knowledge of their partners and their environment. This forces them to collaborate and coordinate with each other. In distributed Internet applications, participants can furthermore choose to collaborate and coordinate directly or indirectly. Each mode has different benefits according to the application domain, which is elaborated in this chapter (section 2.5).

2.3 Collaborative Virtual Game Communities

Recently the concept of collaborative virtual environments has attracted considerable attention. These have been defined as “computer-based, distributed, virtual spaces or sets of places” [6]. In such places, members of a community can meet and interact with each other, with software agents, or with virtual objects. Access to this kind of community “is not limited to desktop devices, but might well include mobile or wearable devices, public kiosks, etc.”[6]. In a collaborative virtual environment, community members will pursue both their own and social goals. With common social goals, members will adjust their individual goals, so that they contribute toward making the community more attractive and they collaborate and coordinate their activities to achieve their common goals.

2.3.1 Introduction to Online Games

The game industry has come of age. Game has become a sought-after content, which is appearing on a growing number of interactive platforms ranging from PC, Xbox, PlayStation2 to PDA and mobile phones. A game can be defined as a simulation that “works wholly or partly on the basis of a player's decisions” [26]. Players meet online and form virtual game communities for different kinds of games. Online multi-player games are considered one of the promising applications of collaborative virtual environments.

When we explore any game website such as [45] [48], we can learn that categories of games include the traditional, simpler games and some complex ones. In [26], it divided the multi-player games into two categories: one is **classical game** such as board game (Chess, etc.), and the other is **pure computer game**. In [23], pure computer games can have:

- *Action game* “is the most popular game genre on the market today”[14]. These games involve the human player controlling a virtual avatar in the virtual world. Tomb Raider (<http://www.tombraider.com>) is a popular action game. Recently team is the base for this kind of game that human players or AI partners could be team members.
- *Adventure game* emphasizes more on story, plot and puzzle solving than on fighting. Players interact with other characters to solve puzzles. Example game of this genre is Monkey Island (<http://www.lucasarts.com>).

- *Role-playing game* (RPG), in which a human can play different types of characters, such as warrior or a thief. Thousands of players interact in the same game world like Ultima Online (<http://www.uo.com>). “Interaction with NPCs (Non Player Characters) and an intricate plot are also important in this genre” [14].
- *Strategy game*, where a player makes decisions and controls many units (often with military origin) against one or more opponents. The core to this genre is to manage the construction of units. The Age of Empires (www.ensemblestudios.com) stands out as a good example.

There is no clear separation between classical and pure computer games since our case study Agent World game [38] is a classical game but it is also a strategy game.

2.3.2 Requirements of Game Communities

After entering game communities, players desire to obtain some utility from these communities. The utility may be achieved through personal enjoyment of online games, the challenges from the different level proficiencies, a need to share their experiences and feelings, and a desire to exchange the knowledge. They intend to find people with similar interests and skill levels or to find their potential teams, which is provided by Lobby Service, which is a pre-game element.

Figure 2.1 Pre – game element: Lobby service

From the beginning until the end of a specific game, players in the team coordinate their behaviors and actions so that they can reach their common goals smoothly. Coordination takes effect at team formation, dissolution, and game interaction. Everyone has an intuitive understanding of coordination, and can give real life examples. Coordination is what turns the players in the hockey game into a team, or individual instrumentalists into an orchestra. One definition is that coordination is about “managing dependencies between activities” [25]. This definition focuses on the interdependence between activities (the coordinated moves of hockey players, or the synchronized performance of complex scores). Other definitions consider coordination the very foundation of programming. Gelernter [18] goes as far as saying that “Programming = Computation + Coordination”. This means that separating computation from coordination allows not only the reuse of computational components, but also that of coordination logic. We can conclude that coordination involves active entities that interact with each other at certain times and in certain places following agreed-upon rules.

Game coordination should follow some rules such as how to join a potential team, how to leave the game gracefully, and how to invite team partners based on the information from lobby services. Game coordination protocols are necessary to govern players' actions. In technical terms, a **protocol** is a set of conventions governing the handling and especially the formatting of data in an electronic communications system. This is the third element needed in game communities.

Figure 2.2 Game elements: Coordination mechanism and Protocols

Trust and reputation management in game communities is the fourth element. People interact with independent parties and therefore reputation information is vital for them to negotiate. People meeting online have no knowledge of others because they haven't met before, so they will rely mostly on the reputation mechanism to make decisions whether to make a deal with other parties. Reputation is an expectation about an agent's behavior based on information about or observations of its past behavior [1]. It could come from either direct interaction or recommendation from a reliable resource or both.

Figure 2.3 Post – game element: Reputation management

2.3.3 Lobby Services in Virtual Game Communities

Lobby is a meeting place for community members to find out about games, potential partners or potential teams. Players in game communities will select their partners based on their preferences via lobby services. Lobby services are used to provide player useful information and to facilitate interaction and team formation. Similar interest and skill level players make the future smooth coordination possible.

Usually the players can find the game rules, other player's personal information, and the types of games from the lobby. For a specific game, players' rankings are provided for other players to make a decision to create a team such as in [48]. In [45], one can find a number of players with a certain rating or to find out a specific player's rating. However, all current lobby services are in the style of phone directory. A player looks at each directory then tries to find a suitable team or partners. For the new players, it is not so easy to find the team partners as they expected. Lobby services should provide more

information or assistance for players to make decisions to form the team quickly and efficiently.

2.3.4 Reputation in Virtual Game Communities

2.3.4.1 Reputation in Virtual Game Communities

Reputation plays an important role in e-gaming from formation, evolution to dissolution. Reputation of each player is updated at the end of game and then could be one of the factors in lobby services.

- Reputation mechanism promotes a team formation.

Different players from distributed locations meet in a virtual place. Some of them may meet before physically. They know each other well and can decide whether to join together this time for a specific game since they have different skill levels and experience. However, most players online don't know others well. With the help of direct interaction and/or reputation of their partners, players could make a decision with whom to collaborate.

- Reputation mechanism is a form of both individual control and social control.

Reputation is one of the efficient ways to control players' behavior so that they should abide by both game rules and social rules. Game rules are the law for players to follow in order to continue the game smoothly and win the game. Whether to win the game will mostly depend on player's skill and experience. However, social rules include each player's social task and his social attitude towards the community. For example, he

should select his corresponding level to join game according to his real skill level. Otherwise, he cannot continue the game proficiently, which will affect the total scores of all team members and may get negative evaluation from other players, leading to decrease his current rating. If every player follows the basic etiquette, game communities could evolve healthily and attract more participants to join.

➤ Reputation mechanism is an incentive for players to make contributions.

With credit or rating as reputation, a player with higher reputation has more chances to be selected as a team member when a team is formed dynamically. Suppose in a strategy game, a player leaves game urgently. The team asks for a similar level or higher-level player to join the team. If player A, with higher level, is invited and he agreed to join the game to replace the leaving player, A will definitely get positive feedback at the end of the game because of his contribution to the game and to team formation. The more positive feedback A got, the more reputation he will own, which brings him more opportunities to get high scores.

2.3.4.2 Issues in Reputation Management

The reputation management models or mechanisms especially in e-commerce community have been put forward, which usually are divided into two main categories: one is non-computational ones, the other is computational ones, part of which are simple and easy to use but others are more complex. EBay [43] is an important practical example of reputation management of computation method. The seller will get positive

(+1), neutral (0) or negative (-1) feedback from buyers who have dealt with this seller after each auction. Seller's reputation is calculated as the sum of this rating over last six months. New comers in eBay start with zero feedback.

Some trust models or reputation mechanisms still have some problems as [41] described:

- Adopt a new or change identity easily

After getting negative feedback from other partners, a user may have lower reputation than the beginners. Thus he would have the incentive to discard original identity and register as a beginner with relatively higher reputation.

- Performing fake transaction for high reputation

In some systems (e.g. OnSale Exchange), the overhead of performing fake transaction is low, which leads to two friends can make fake transaction then increase the reputation value for each other. Though some mechanisms allow each user to rate another only once, the user can still create a new identity to give his friend fake but high reputation value.

2.3.5 Coordination in Virtual Game Communities

How can we apply the concept of coordination to virtual game communities? If the game players pursue their own goals, based on their individual interests and skills, why

do they still need to coordinate their activities with other players? The answer lies in considering how games have evolved from single-player games to multi-player games with cooperative goals. There are three motivations for using coordination in multi-player games:

- In a single-player game, successful completion mainly depends on the player's skill. However, with the popularity of networks, modern game play is moving away from single player to team based game with cooperative goals. Game developers are “using teams as fundamental parts of their design”[30]. In online games, each player has incomplete information about the state of the game and other players in the game. Players need to help each other to complete the game, including exchanging their partial views of the game, suggesting moves to each other, and reassigning the workload among them.
- Coordination can make games more appealing. In the early development of games, game developers focused mainly on visual effects. However, today players expect more than exciting graphics and surprising plots. They are paying more attention to being able to find other players with similar personalities and interests, to changing a game dynamically to meet their own needs (e.g., SimCity: <http://simcity.ea.com>), and to how they are perceived by other players (i.e., their reputation as a team player).
- If coordination is not explicitly represented in the game, this does not mean that coordination is not needed. For example, the actions of non-player characters (NPCs) need to be coordinated with the moves of human players. Making

coordination explicit, and separating it from computational components can avoid duplication of effort in developing non-player characters from scratch each time. Similarly, the rules of a game, if encoded as coordination rules, can be adapted more easily to create new games from existing ones.

2.4 Agents and Multi-Agent Systems

2.4.1 Agents

As a new paradigm for developing software applications, agent technology has been widely adopted in the artificial intelligence and computer supported communities. “They are being used in an increasing variety of applications, ranging from relatively small systems such as assistants to large, open, mission-critical systems like electronic marketplaces”[37]. Although there is no universally accepted definition of agents, [39] has characterized agents by four properties: autonomy, social ability, reactivity and proactiveness. Agents are autonomous computational entities (autonomy), which interact with their environment (reactivity) and other agents (social ability) in order to achieve their own goals (proactiveness). In [37], the author also mentioned the domain characteristics of adopting agent technology: an inherent distribution of data, control, knowledge, or resources; the system can be naturally regarded as a society of autonomous collaborating entities; and legacy components must be made to interoperate with new applications.

2.4.2 Multi-Agent Systems

In a virtual environment, there will be more than one user in the community to communicate and coordinate with each other, then multi-agent system (MAS) are more and more becoming an ubiquitous paradigm for the design and implementation of complex software applications [7]. MAS are applications that deal with the interaction of groups of intelligent agents attempting to cooperate to solve problems. From a simple view, MAS are considered as a collection of collaborating autonomous agents, each representing an independent locus of control [37].

Two views of MAS have been pointed out by [7], one is the compositional view, and the other is holistic (non-compositional). According to the compositional view, the MAS is the mere sum of agents, which act to achieve their tasks. The interaction between the agents adds nothing but composes agents into an ensemble to work together. This view disregards the social aspects such as social rules. Instead, the holistic view of multi-agent systems introduces notions such as socialware [21]. “These approaches assume that the full understanding of multi-agent systems calls for a comprehensive theoretical setting for agent societies, defining what is the world where agents live and socialize (the agent’s space), which kinds of individuals populate the world, and how the world is ruled.” They recognize that social rules are a part of source of intelligence in multi-agent systems, which goes beyond the one provided by single agents and represents a natural place where global system properties can be embodied.

2.4.3 Team Formation in Agents

Team formation involves a lot of coordination, which is one of the key concepts in the multi-agent systems. Therefore the process of team formation can be studied in the multi-agent systems in the distributed environment. Interactions including information flow and goal realizing are needed inside a team. Researchers of [2] applied a standard technique of token passing to deal with the communication and consistent decision-making inside a team. This approach is relied on the fact that team is expanding from a single agent. Two algorithms for team formation, which are based on simple self-interested agents and sophisticated self-interested agents, are presented in this research. Another approach for team formation by multi-agent systems is to set up the architecture for agents to discuss the team formation and subsequently work as team members until the collective goal has been fulfilled using structured dialogues [11].

2.5 Coordination Models

2.5.1 Coordination Model Components

Based on the discussion in section 2.3.2, it is concluded that a coordination model has three important components:

- **Coordination entities:** are "actively computing entities" [3], often programmed in different languages. Agents have their individual goals, as well as social goals. For individual goals, agents can choose their own way to achieve them.

However they need to use a coordination mechanism to communicate or negotiate with other agents, if they share the same activity and common goals.

- **Coordination media:** Entities coordinate themselves directly or indirectly. Coordinating directly means that entities may transmit message over some forms of channels [3]. For this to work, they need to know others' identities, location and available time to reach. Peer to peer communication such as the message passing paradigm underlying sockets is one of the examples using the direct method to realize coordination. In contrast, indirect coordination media uses a shared data space. Its main design goal is to provide a persistent data repository that uses an associative addressing mechanism to realize anonymous communication among the coordination entities. Agents communicate by sending data to the shared data space while interested partners can receive data by registering with the space. There could be a single data space, multiple data spaces, or nested data spaces depending on the requirements of the application.
- **Coordination rules:** are the laws used to govern the relationship between entities and coordination media. One way to embed coordination rules is to embed them inside the agents, which refers to "subjective coordination" for inter-agent communication [34]. There are some disadvantages since agent designers need to anticipate all future situations that an agent may meet. However it is not easy to foresee every coming event. Another way is to embed coordination logic in the coordination media to make it programmable [4][28],

which separates the concerns of coordination and computation. We will propose our coordination model for online games based on this concept.

2.5.2 Coordination Model Types

The selection of the agent coordination model has an impact on the multi-agent system design. What criteria are used to justify the coordination model or how to select the coordination model according to the application domain? Researchers of [10] point out a number of forces that impact many coordination models. Forces, a term used by software patterns, “identify the different types of criteria engineers use to justify their designs and implementations.” Among them, we pay more attention to the force of temporal and spatial coupling for our Internet application research. In [4], spatial and temporal coupling are considered “two main characteristics” to “distinguish different coordination models” as well.

Spatially coupled models require that agents share a common name space so that agents can communicate by explicitly naming the receiving agents. A naming service should be provided to support this ability since the sending agents don’t need to explicitly refer to the receiving agent’s location. If the MAS are spatially coupled systems, they allow the agents to communicate in a peer-to-peer manner. This model is more suitable for statically located agents than for mobile agents. In the former case, “the protocols, locations and coordination times can be agreed upon a priori and network involvement is minimized” [10]. However, in the latter case, mobile agents

often move and rely heavily on the stability of the network, its locating and direct messaging are expensive operations.

Spatially uncoupled models allow sending agents to interact with others without having to name receiving agents so naming services are not required in this kind of system, resulting in fewer system components to manage. Different type agents can serve the same purpose or the same task. “With these models, the information an agent has to share and its type are more important than its name.” This model enforces anonymous interactions.

Temporally coupled models require some form of synchronization when agents communicate. Temporal coupling is important for spatially uncoupled models, which does not focus on direct agent-to-agent communication since the environment is used as the information exchange medium. Agents still need to agree on what to share, when to share it, and how to share it. In this model, agents share a common knowledge representation and are aware of schedules and positions when exchanging information.

Temporally uncoupled models need no requirement for synchronization. When agents communicate, they don’t need to know others’ schedule then to meet and exchange information with them at predetermined time. This model realizes the asynchronous interactions. However, it relies more on knowledge representation and on how to use the environment to transfer knowledge.

Based on the above analysis, four categories of coordination models are presented in both spatial and temporal view (Table 2.1) [4].

Spatial \ Temporal	Coupled	Uncoupled
	Direct	Blackboard-Based
Coupled	Meeting-Oriented	Linda-Like
Uncoupled		

Table 2.1 Types of coordination models

- **Direct Coordination**

In **direct** coordination models, the communication and coordination between the agents are either by peer-to-peer or by client-server way. For Internet applications, general adoption of direct coordination models is not suitable. Initiating agents cannot know how many other agents or what kind of agents will appear later because of the dynamic creation of agents. Example model *Agent Tcl* [19] provides direct communication between two agents, based on message passing.

- **Meeting – Oriented Coordination**

In **meeting-oriented** coordination, an active entity must assume the role of initiator to open a meeting point. Usually, only local agents can participate in the locally constrained meeting that takes place at a given execution environment. This model still needs to enforce a strict synchronization between agents to join the meeting together.

However the interaction in the meeting is missing because the schedule and the position of agents cannot be predicted in some applications. One implementation example is *Ara* [29]: one agent assumes the role of meeting server, which announces a meeting point at one hosting environment so that incoming agents can enter the meeting to coordinate each other.

- **Blackboard – Based Model**

In the **blackboard-based** coordination model, agents interact by using a shared data space, which is used as common repository to store and retrieve messages. This model is spatially coupled since agents must agree on a common message identifier to communicate and exchange data via blackboard. In [10], it is pointed out that there is a supervisor who decides which agent may make a modification to the blackboard. The supervisor acts as a scheduler for the agents. It embeds the coordination mechanisms into the blackboard. This model allows that messages can be left on blackboards without needing to know neither the location of the corresponding receivers nor the time they will read these messages. Receivers could read messages without needing to know neither the location of the message sender nor the time messages were left. In addition, any interaction between agents is forced to perform via a blackboard so that hosting environments can easily control all interactions. The **blackboard-based** coordination model has been proposed and implemented for some mobile agent applications. Agent system *ffMAIN* [12] defines mobile agents that interact – both with other agents and with the local resources of the hosting execution environment – via an information

space accessed through the HTTP protocol, where data can be stored, read and extracted.

2.5.3 Linda Model

The **Linda-like** coordination model is similar to the blackboard-based coordination model but the access to the data space is based on associative mechanisms [5]. The shared data space is called **tuple spaces** through which information is organized in **tuples** and is retrieved in an associative way via a pattern-matching mechanism. This means that the receiver of the data specifies a template tuple in which it is interested; the matching tuples will be read or taken from the space. In an open, wide and dynamic environment, it is hard for an agent to have complete and updated knowledge of its execution environments and of other agents. “It is worthwhile to integrate these pattern-matching mechanisms in a coordination model to simplify agent programming and to reduce application complexity”[4]. The concept of tuple space has been implemented in the *Jada* system [8]. The ObjectSpace can be used to store and associatively retrieve object references and, can be created by agents for private interaction. *Jada* has been developed in the context of PageSpace [9], which is to enhance the WWW middleware with coordination technology for building cooperative Internet applications such as online games.

The Linda model has the following advantages:

Spatial uncoupling: The producer of a tuple does not need to know who consumes it, and the consumer does not need to know who produces it.

Associative addressing: Tuple spaces use an associative addressing scheme based on pattern matching. Tuple spaces provide a globally shared data space to all processes or agents regardless of machine or platform boundaries, as long as they have access to the space.

Temporal uncoupling: Tuples have their own life span, independent of the processes that produced them, or any processes that may consume them. This enables temporally disjoint processes to communicate in a seamless manner.

These advantages make it suitable for use in the open distributed systems, which lead to our coordination model for multi-player games building on it.

Linda model provides a set of coordination operations (coordination primitives) over the tuple space:

Out: stores a tuple to the tuple space. For examples in our case study (Chapter 4), *out* a move tuple (“*move*”, *Alice*, *source*, *destination*) means player Alice makes a move in the game board from *source* to *destination*.

Two primitives concerning retrieving the tuples from the tuple space are **in** and **rd**. The tuples are retrieved in an associative way: there is a template tuple as an argument and a returned tuple matching the template. The matching mechanism is that both the returned tuple and the template tuple have the same tag, same number of fields and each of both tuples matches. Both operations are blocking, which means the thread will be blocked if

no matching tuples are found. However, the selection of the matched tuples is random if multiple tuples match the template. The difference between **in** and **rd** is that the former extract the matching tuple from the space while the latter still leave the matching tuple in the space. The details and implementation of tuples, fields and matching mechanism can be found in Chapter 3 (section 3.3.1).

The last two operations are **inp** and **rdp**, the non-blocking version of **in** and **rd**. The tuple will be returned if it is matching the template or otherwise return null. We have corresponding operations in our proposed coordination model but with some refinement in order to meet the needs of the online multi-player games.

2.5.4 Extensions to Linda Model

There exist some limitations in the Linda model. One is the “multiple read” problem. For example, if there are two tuples in the tuple space matching the template, we cannot guarantee each will be read exactly once or get all the matching tuples at one time. The original Linda model only has one global tuple space, which could be a central point of failure. All communications between the agents will stop if there is something wrong with this data space. As long as the agents have the access to the global tuple space, they can modify the content of tuple space using basic operations, which leads to the security problem. One tuple space does not have scalability for Internet applications. The tuple space in Linda model is mere a data repository without control, which makes it inflexible to deal with changeable Internet environment.

Due to the limitations of Linda model, basic model has been extended in various ways “from high speed parallel computations to Internet-based multi-agent architectures” [3]. Usually the extensions to the original Linda model have several categories; we restrict our research on two of them in this thesis for introducing our proposed coordination model for online games.

➤ **Adding Primitives:**

The original Linda model introduces some limitations such as “multiple read problem”. Therefore in this extension, new primitives are introduced to “address specific problems or to enrich the expressiveness of the coordination languages” [3]. Examples of this kind of model include JavaSpaces [16] and T Spaces [40].

- **JavaSpaces**

Figure 2.4 Illustration of JavaSpaces technology [16]

The JavaSpaces technology is developed by SUN Microsystems. JavaSpaces act as virtual spaces between providers and requesters of network resources or objects in a distributed application. It is a new realization of “tuple space” and aims at providing “distributed repositories of information based on Linda tuple space” [3]. Space is the collection of tuples that are Java objects. Tuples are instances of classes implementing the Entry interface.

```
public class Car implements Entry {
    public String name;
    public String model;
    // other fields
}
```

Figure 2.5 Example class Car that implements Entry interface

Tuple in JavaSpaces is serializable and each field of a tuple object is serializable as well. Spaces, which are Java tuple spaces, are shared, persistent and associative. Access to tuple space is defined as Java methods, which are declared in the JavaSpace interface as shown in Figure 2.6.

```
public interface JavaSpace {
    public final long NO_WAIT = 0; // don't wait at all
    //Lease write(Entry e, Transaction txn, long lease);
    //Entry read(Entry tmpl, Transaction txn, long timeout);
    //Entry readIfExists(Entry tmpl, Transaction txn, long timeout);
    //Entry take(Entry tmpl, Transaction txn, long timeout);
    //Entry takeIfExists(Entry tmpl, Transaction txn, long timeout);
    //EventRegistration notify(Entry tmpl, Transaction txn,
        RemoteEventListener listener, long lease,
        MarshalledObject handback);
    //Entry snapshot (Entry e) throws RemoteException;
}
```

Figure 2.6 Main methods declared in JavaSpace interface

JavaSpace interface has basic operations as defined in Linda model but with different method names. It enhances the Linda model in the following ways:

Lease: the tuples in the space have lifetime, that is lease time. When the time is over, the tuple will be removed from the space.

Notification: If an external agent would like to be notified when a specific tuple is written to the space, it first should register as a listener. So if each tuple, written to the space, matches the specific template, then the registered listener will be notified.

Transaction: rely on the external transaction services for the correctness of the performed operations.

Type: classes can be used as tuple types and field types, enhancing the matching mechanism by considering the subtype in matching policy.

One application of **JavaSpaces** is to build an interactive messenger service, a collaborative application that allows user to keep track of a group of online friends and communicate with them.

- **T Spaces**

T Spaces, a product from IBM Research Division, adds more advanced operators and extends the power of tuple spaces with database features traditionally found in large enterprise database systems so that it may support more complex applications. It enables communication between applications and devices in a network of heterogeneous computers and operating systems. It possesses network ubiquity through platform

independence since it is implemented in Java. The T Spaces system is appropriate for any application that has distribution or data storage requirements. It can perform many of the duties of a relational database system without imposing an overly restrictive (and primitive) type system, a rigid schema, a clumsy user interface or a severe runtime memory requirement.

The salient features of the T Spaces system are:

More operations: Besides the standard set of Linda model tuple space, T Spaces implement set-oriented operators such as *scan* and *consumingscan*.

Dynamic operations: T Spaces API allows programmer to define new operations dynamically, whose code can be downloaded into the T Spaces server and used immediately.

Database indexing: T Spaces data manager indexes all tagged data for highly efficient retrieval.

Event notification: Applications can register to be notified of events as they happen in the T Spaces server.

T Spaces is also useful as a synchronization mechanism and for central control of real-time events, such as chat rooms or multi-player game controllers and distributed application such as shared whiteboard as well.

➤ Adding Programmability

In Linda model, the tuple space is mere data repository without flexibility and any

control over the coming tuples. Adding programmability to the space enables the space to react with specific actions to the accesses made by agents. “Reactions could access the tuple space, change its content and influence the semantics of the agents’ accesses” [4].

Two typical examples adopting programmability are MARS [4] and TuCSoN [28] system.

- **MARS System**

The MARS (Mobile Agent Reactive Spaces) system, developed at University of Modena, is a coordination architecture that implements concept of programmable reactive tuple space for mobile/static agent applications. MARS assumes that each execution environment has one tuple space, which is independent of the other site’s space. The space is the only way for coordination between agents. Agents access the space by invoking the methods of object that implements MARS interface (Figure 2.7), which is the subclass of JavaSpace.

```
public interface MARS extends JavaSpace {
    // method interface inherited from JavaSpace
    // void write(Entry tup, Transaction txn, Identity who);
    // Entry read(Entry req, Transaction txn, Identity who);
    // Entry take(Entry req, Transaction txn, Identity who);

    // methods added by MARS
    Vector readAll(Entry req, Transaction txn, Identity who);
    Vector takeAll(Entry req, Transaction txn, Identity who);
}
```

Figure 2.7 MARS system interface

Two important properties of MARS are programmability and reaction. Therefore this model can embody the computational abilities within the tuple space and assume the specific reactions to the access events. Reaction is coded as meta-level tuples and stored in the meta-level tuple space. **MARS** system has been used in WWW information retrieval application, online auction, and network management area.

- **TuCSoN**

“**TuCSoN** (**T**uple **C**enters **S**pread over **N**etworks) is a coordination model for Internet applications based on network-aware and mobile agents”[28]. It borrows *tuple center* idea from **LuCe** (**L**ogic **T**uple **C**enter) model, a system for constructing a multi-agent system with autonomous and different kind of agents. Two key features of **TuCSoN** system are coordination space and coordination media.

- Coordination space: **TuCSoN** provides an interaction space over the collection of network nodes, which is associated to a tuple center denoted by a locally unique identifier. So each tuple center could be identified by means of either absolute name, unique over the network or relative local name, unique within a single space.
- Coordination media: “**TuCSoN** exploits tuple centers as its coordination media, where a tuple center is a tuple space enhanced with notion of behavior of specification” [28]. Therefore tuple center is not only the standard tuple space, but also whose behavior in response to coming events can be defined to embed coordination rules. The behavior of each tuple center can be defined separately and independently. Defining new behavior for a tuple center is achieved by enabling the

definition of reactions to events. Reaction in TuCSoN is a set of non-blocking operations and can access the tuple center and modify the information. The tuple center contains the ordinary tuples that adopt Prolog syntax and specification tuples that are the Prolog unitary clauses.

TuCSoN system has case study in e-business field. Today's B2B scenarios call for the integration of heterogeneous resources, services and processes. Virtual Enterprises (VE) and inter-organizational Workflow Management Systems (WfMS) is a TuCSoN system application.

Our proposed coordination model for online games will deploy the idea of “adding programmability” to the tuple space. The implementation of the model API is based on the tuple space definition from [36], adding more operators to the basic Linda operations and incorporating “reactive” features. It could be adapted to other tuple space platforms. Implementation detail of tuple space will be discussed in Chapter 3 (section 3.3).

2.6 Chapter Summary

This chapter provided the background knowledge to our research on virtual game communities. We have presented the definitions and applications of virtual communities. Then the basic issues and example supporting systems help us better understand the concepts in general. It is followed by the requirements analysis for online game communities.

We put forward four elements related to the team development in different game phases: lobby services in pre-game, a coordination model and protocols in game, and the reputation mechanism in post-game. The current approaches and their limitations of these elements were presented.

Coordination model is a necessary element in game communities to support the participants' collaboration. Different coordination models and examples were discussed, from which we can find basis for our proposed model suitable for online games. Team formation in agents was also covered in this chapter.

Chapter 3 Modeling Virtual Game Communities

Four elements of team development will be explained in the following sections.

3.1 Lobby

Awareness is one of the key features of virtual communities and is an attribute to facilitate contact with other members. It involves knowing who is around, and what activities are under way. So one lobby service of the community is to provide an approach for people to find partners via awareness.

A lobby can provide “thin” information in the form of a directory of where to find desired partners, such as through chat rooms or forums for players to communicate. “Richer” information the lobby could provide, for example, is to visualize the social relationship between the players as “socialware” [21] system put forward. Players could be represented in different color, and size in visualization for their skills and interest etc. Distance is to represent closeness of the relationship as displayed in Figure 3.1. New player has shorter distance between Game A than that of Game B, indicating that he has more interest in game type A. The color of each player shows the different skill. Based on the matching result with other similar players, the game types and previous interaction, one can make a decision to select which member as a partner or to join which team.

Figure 3.1 Visualization social relationships in a lobby

Figure 3.2 shows the basic class diagram of our lobby and its components.

Figure 3.2 Class diagram of Lobby and its components

A lobby is the collection of players and games. A player can provide some personal information to the lobby via *updatePlayer()* method or *updateInfo()* method. For instance, he can pass some skill tests to prove that he has some level skills to join specific teams. Games can be added to or removed from the lobby in terms of the number of games. One type game is composed of many teams. Lobby is able to match the players to the suitable teams or partners.

3.2 Layered Reactive Tuple Spaces Coordination Model

The coordination model in game communities has the following components:

- **Coordination entities:** In virtual game communities, coordination entities include human players and software agents representing human players or leaders. In various kinds of games, agents exchange their actions, strategies, knowledge, and locations to coordinate with each other for reaching social goals. For the online game Agent World (section 4.1), coordination entities are human players.
- **Coordination media:** In online games, direct coordination means that players need to know others' identities, locations and available time. However, this is not the usual case for every player in Internet environment. A player could not figure out who will join the same game and when. A team sometimes forms dynamically. Therefore, our coordination model adopts indirect coordination media that uses a shared data space. Player agents could communicate anonymously using an associative addressing mechanism in this persistent data repository. Provided player

agents had access to this shared data space, they can send data (message or strategy etc.) to the space while others can retrieve data.

- **Coordination rules:** Coordination rules are needed to govern the relationship between player agents and shared data space. One choice is to embed coordination rules inside the player agents. The disadvantage is that agent designers need to anticipate all future situations an agent may meet in this dynamic and quick changing environment. From a reusability and scalability point of view, this strategy is not a preferred one. Each time the social rules are changed, or the communication protocols are modified, regardless of the change complexity, each agent involved in the game needs to be rewritten to reflect the change. We will embed the rules in data space in our game coordination model to separate the concerns of coordination and computation.

3.2.1 Layered Model

In the Linda model there is only one tuple space [18], which is insufficient for current multi-player games. The more agents use tuple space, the lower efficiency of agents' communication, for the case they all need the access to only one space at the same time. Different games have different rules and players care about security and privacy. So we need **multiple** tuple spaces, which means new tuple spaces can be created, depending mainly on the number of players, or the number of games. For example, on Yahoo card games [48], each table in different rooms would need one tuple space to coordinate 2 or

4 players. So our game coordination model allows **multiple** shared data spaces for players' coordination.

In a game community, there is at least one **local tuple space** for the player agents of each game. All player agents who have access to this space will coordinate their activities via this shared repository. We need one leader agent for teams in this kind of game, who has access to local data space and have control about when to begin the game, when the game is over. The lobby can create the leader statically or dynamically depending on the game community's requirement.

In order to support coordination between peer communities' leaders, we propose an extension to the basic tuple space, which introduces multiple layers of tuple spaces. For example, in addition for the tuple space for individual games, there can be a global data space for game type. This global tuple space is used to allow leader agents to communicate on issues of team formation, team member's departure, and rating players. Leader agents have access to both local spaces and global tuple spaces, while player agents can only access the local tuple spaces for their respective games as shown in Figure 3.3. This structure is scalable in terms of the number of players, the number of games and game types.

Figure 3.3 Layered Reactive Tuple Spaces Coordination Model

3.2.2 Reactive Model

The original Linda tuple space has no "control" over the operations on the space. It is simply a persistent data repository. One disadvantage by adopting this way in agent-to-agent coordination is that if the coordination policy (social rule) is not directly supported by the tuple space, it has to be charged upon agents. It is likely to increase the programming and complexity of agents. In our model, we add some "life"(reactions) to the tuple space to make it reactive.

Reactive tuple space can provide following advantages:

- Some specific policies for the interactions between the agents and its environment can be implemented as reactions in order to achieve better control.
- Reactions of the space can simplify programming task of agents. Embedding

coordination rules in the tuple space allows the agents to focus only on their tasks, not on social rules. If social rules happen to change, it is not necessary for agents to be reprogrammed.

- Furthermore, the reactive tuple space achieves a clear separation of concerns between algorithmic and coordination issues.

We create a **reactive tuple space** by allowing reactions to be associated with specific operations on some tuples. A reaction is a piece of code that is fired when the associated operation is invoked.

A tuple space is a collection of different kinds of tuples. Therefore the original tuple space will act as both *general tuple space* and *reactive tuple space* in our model (Figure 3.4). As a general tuple space, it will store the general tuples (section 3.3.1) such as *player tuples*, or *begin tuples*. As a reactive tuple space, it will store the *reaction tuples* (section 3.3.2), which have a field of “reaction object” that can be executed on this general tuple space.

We use these reaction tuples to embed coordination rules in spaces so the behavior of the spaces can be programmed. Reactions can ease the player agents programming and allow the coordination entities to be reusable. Examples of reactions in a game could be found in Chapter 4 section 4.3.

Figure 3.4 Tuple spaces in LRTS model

Therefore the extension model, which has a layered structure and the spaces with reactions to the coming operations on specific tuples, is based on **Layered Reactive Tuple Spaces** (LRTS). It supports coordination between community leaders, between players and between players and leaders. This coordination model, which is scalable and reusable, meets the needs of Internet game environment.

3.2.3 Primitives of LRTS Coordination Model

Our coordination model is developed in Java, an object-oriented language. Java is an interpreting language and its program is executed on Java Virtual Machines (JVM), which makes it portable and platform independent. Besides the features of reliability, simplicity and ubiquity of Java, its potential is realized for security, built-in networking including multi-level network supporting and multi-threading [35]. Because of these benefits, Java is an attractive development language for Internet applications.

Our model adds more primitives to the original Linda model (implemented in [36]) for the needs of the game communities (Figure 3.5).

```

public interface TupleSpace {
    // Extract tuple matching template from tuple space
    // Block if none available
    public Tuple in(Tuple template)
        throws InterruptedException;

    // Extract matching tuple, return null if none available
    public Tuple inNoBlock(Tuple template);

    // Deposit a tuple in tuple space, without tail position
    public void out(Tuple tuple);

    // Read tuple matching template, block if none available.
    public Tuple read(Tuple template)
        throws InterruptedException;

    // Read tuple matching template, return null if none available.
    public Tuple readNoBlock(Tuple template);

    // Return all tuples matching template
    public ArrayList readAll(Tuple template)
        throws InterruptedException;

    // Return the number of tuples matching template
    public int readNumber (Tuple template);

    // Deposit a tuple in tuple space with tail position.
    public void write(Tuple tuple);

    //Deposit a tuple with known position
    public void write (Tuple template, int position);

    // Extract tuple matching template from tuple space
    // decrease this tuple's channel position by 1
    public Tuple take(Tuple template) throws InterruptedException;

    // Similar to 'take', no block
    public Tuple takeNoBlock(Tuple template);
}

```

Figure 3.5 TupleSpace interface in LRST

The player agents exchange the tuples via this layered reactive tuple spaces by means of primitives. Each tuple space will implement the TupleSpace interface (Figure 3.6).

Figure 3.6 Class diagram of `TupleSpace` interface and its implementation

3.3 Reaction in LRTS Coordination Model

3.3.1 Fields, Tuples and General Tuples

A tuple space is a collection of tuples and a tuple is a collection of named fields as the

Figure 3.7 shows the class diagram of these two basic classes in the model [36].

Figure 3.7 Class diagram of `Tuple` and `Field`

Each field is a Java object (Figure 3.7), composed of three instance variables: name, type and value. Name refers to the field name, used to differentiate fields, type refers to a Java object type and value has the difference between null value (formal field) and defined value (actual field). Each field can represent the primitive data type. A `Field` uses `match()` method to decide if `field1` is matching `field2`: `field1` should have the same

name, same type and same value (if field1 is not a formal field) as field2. For instance, two fields, one is (“player”, “Marry”) doesn’t match another (“player”, “John”) even they have the same name (player) and same type (String) but with different value. However if the former one changes to (“player”, String. class), it will match the latter since it contains the same name, same type and it is a formal value. Matching fields is used to decide if two tuples are matched.

The LRTS tuples are Java objects, instance of tuple classes. A tag is “a literal string to distinguish between tuples representing different classes of data ” [36]. For example, tag with “player” in the game refers a general *player tuple* as examples shown in Figure 3.8.

```

Examples of general tuple:
("player", Field ("name", "John"))
// a player tuple with player name "John" (value is a String object)

("tail", Field ("position", 3))
// a tail tuple with position at 3 (value is a primitive type)

("leave", Field ("name", String.class))
// a template tuple with name of null value

```

Figure 3.8 Examples of the general tuple

A template tuple contains at least one formal value as summarized in Table 3.1.

Value	Field	Tuple
Null value	Formal field	Template tuple
Actual value	Actual field	Actual tuple

Table 3.1 Field and Tuple with null value and actual value

Match() in Tuple class is used to match another tuple, which is an essential process of tuple space. In Agent World game (Chapter 4), every step made by each player will be posted to the tuple space and others can read or take the step tuple from the space using this pattern matching mechanism. Tuple1 matches tuple2 means that both have the same tag, the same number of fields and each field matches. For sample, player Simpson took an action “put” in the game board from grid (5,4) to grid (5,5). The move tuple, (“move”, Field (“player”, “Simpson”), Field (“action”, “put”), Field (“from”, “(5,4)”), Field (“to”, “(5,5)”), will be posted into the space. Other players should construct a template tuple with same tag “move” and four template fields to match this *move tuple* then update the step on their local game boards (see section 4.3.4).

3.3.2 Reaction Tuples and Reaction Class

Tuple with “reaction” tag refers to a *reaction tuple*. When define a reaction tuple, we will consider three components associating reactions to access events: Tuple Type (T), Operation Type (O) and Reaction (R). So a *reaction tuple* has the form of (T, O, R): the reaction R is executed when a player agent invokes the operation O (trigger of operation) on a tuple type T. A reaction tuple (Figure 3.9) is based on these components: tuple type, operation on this tuple and real reaction.

```
Format of reaction tuple:
("reaction", Field ("template", TupleType),
               Field ("operation", OperationType),
               Field ("reactionObj", Reaction));

Example of reaction tuple:
("reaction", Field ("template", "player"),
               Field ("operation", "out"),
```

```

        Field ("reactionObj", reactionObj));
// reaction tuple: reaction object "reactionObj" will be executed when
//the operation "out", performs on this player tuple to the space

```

Figure 3.9 Format and example of the reaction tuple

All reactionObjs in the reaction tuple are Java objects that are derived from abstract class *Reaction* shown in Figure 3.10.

```

public abstract class Reaction {
    public abstract boolean executeReaction(TupleSpaceImpl space,
                                           Tuple template);

    public abstract Tuple executeReaction(TupleSpaceImpl space);
}

```

Figure 3.10 Abstract class *Reaction*

Abstract class *Reaction* has two overloaded abstract methods called *executeReaction()* with different parameters and returned type. Based on the operations to be performed on the tuple space and results, different reaction objects are defined as subclasses of the *Reaction* class and one abstract method is defined to specify the reactive behavior. The execution of reaction will either affect the contents and operation of original template tuple or return the player agent a general tuple.

3.3.3 Trigger a Reaction in LRTS Model

Before any operations on the tuples, we should install the *reaction tuples* with reaction object in the tuple spaces. Servers, either game servers or lobby server in game Agent World, are responsible for this installation. Servers could install different reaction tuples according to the requirements of the application and coordination rules. Player agents have no idea about what tuple spaces will response to the coming tuples. Old reactions

could be removed and new ones could be added when the coordination rules are changed.

When operation O performs on a general tuple T in the tuple space, tuple space will check if there is a corresponding reaction tuple for this operation by using pattern matching mechanism in the reactive tuple space. The matching mechanisms are same in both the general tuple space and reactive one. If reaction tuple exists, reaction object R in the reaction tuple will be obtained and executed. As defined in the *executeReaction()* method (Figure 3.10), the reaction object has access to the tuple space and can perform any operations on it. All reactions in our model are post-reaction, which means operation is the trigger of a reaction. It may change the template tuple (example: section 4.3.3 Message Communication) or change the tuple space content (example: section 4.3.2 Leaving) or return a different tuple (example: section 4.3.4 Reading Move) as displayed in Table 3.2.

Tuple Type	Operation (trigger of reaction)	Reaction Results
Written Tuple	out	May remove or modify the original one and rewrite the tuple to the space with position
Matching Tuple	read	May read the original tuple in order and return the matching tuple

Table 3.2 Main reactions in LRTS coordination model

3.4 Protocols of Joining and Leaving the Game

In a virtual game community there is, conceptually, one large community consisting of all game players, which is, in turn, composed of many sub-communities or teams for different kinds of games and different skill levels' players. Players differ in their experience, domain knowledge, skill level, interest, and reputation. For example, some players are beginners, while others play at an advanced level; some prefer action games, while others want to play only strategy games; some have received high ratings from their peers, others may not have been rated at all.

Each game community has a community leader or a lobby leader, whose task is to control access to the community, to maintain the reputations of its members, and to provide lobby services. Sub communities (teams) for the same kind of game have their own leaders. Every team leader can interact with leaders of the same level peer communities to share knowledge, make recommendations for new members, and coordinate the re-formation of the team, should a member leave.

The player has the right to join the community, make contributions to the evolution of the community, and leave the community at any time. However, virtual game communities, like physical communities, still need social rules to govern the behavior of players. For instance, a player is free to leave a game prematurely, which may be frowned upon by the community. In [13], session management is discussed in collaborative environment and two approaches of session management model are presented: initiator-based and joiner-based. In the first case, initiator sends invitations to

other users who they want to invite and in the second case user who wants to join the session browses a list of sessions, and selects one to join. Besides the direct invitation, the process could be done with the help of team leaders since they have more information about players. Following protocols are based on the basic strategy games such as board games.

3.4.1 Joining the Game

Upon joining a virtual game community, each player will be represented by a **player agent**. They first enter the lobby where they can check how many kinds of games can currently be played and who has joined games (Figure 3.11).

Figure 3.11 A player connects to the lobby

Newcomers could also learn about the different types of games (e.g., the game rules), and obtain information for joining or leaving a game from the lobby. Based on the information they received, description of the other team members (interest, reputation, etc.), players select a team to join. This process is demonstrated in Figure 3.12.

Figure 3.12 Protocol of joining a game

3.4.2 Leaving the Game

Leaving a game can be classified either as normal leaving, or urgently leaving. **Normal leaving**, after the game is over, will not affect the progress of the game. A player sends a leave request to the team and leader agent update this player's rating.

In the case of **urgently leaving**, a player leaves the game unfinished. This player's departure results in a decrease of his rating for future attempts to join the community. Cooperation and coordination exist not only between players in one team, but also between teams. A candidate can be found in one of several ways:

- At the time a player leaves, a new player joins the team and replaces the former player to continue the game. This new player will continue the state left by the leaving player. (Reader could refer to section 4.2.4)
- The remaining players could send an invitation to players who are on waiting list. This is a high-level invitation protocol that the invited players are selected based on their reputation such as experience and skill level. The leader agent coordinates with invitee in other teams or other leader agents during the invitation. If the right candidate has been found, he will be asked to join the team and assigned a new job according to the dynamic team formation. Figure 3.13 illustrates that initiator and invited players belong to a same leader. If two players are in different leaders' control, then the layered structure will be used (section 4.2.3).
- If no right candidate found, or upon the request of other players, leader agent will create a software agent to replace the leaving player. Later on, a new human player can join the game and replace the software agent.

By these approaches, the game will not be disrupted or maliciously destroyed by the decision of a player to leave the game prematurely. The reputation system helps constrain player behavior to follow both game rules and social rules. Suppose John is a beginner for the game Dominoes, but he selects to join a team of advanced players instead. As a result, he could not complete the game adequately, or collaborate with others harmoniously. His final score will be affected, as well as his rating.

Figure 3.13 Protocol of inviting a new player to continue the game
(the initiator player A1 and invitee B1 are in the same leader control)

3.4.3 Coordination in Playing the Game

Coordination between all players happens not only at the beginning of the game to form a team or at the end of the game to re-form a team dynamically, but also in the middle of game. The players communicate with each other about each step they made, as shown in Figure 3.14.

Figure 3.14 Intra - game coordination (actions and messages)

In the above figure, each player has his turn to play the game. When getting the turn, he will take steps in the game and notify other players to update the current game status. Players can also communicate by chat messages.

3.5 Reputation Management

3.5.1 Reputation Management Model in Game Communities

In a virtual game community, each player agent will know others by interacting with them directly or by looking the credit history of the players. Credit history is kept and

updated after game is over. Online game reputation model will base on the player's personal skill, game attitude towards the team and final contribution to the team.

Figure 3.15 Virtual game communities' reputation model

The model (Figure 3.15) includes both the objective criteria and subjective criteria. Objective criteria rely on the player's personal skill and experience such as number of played games and scores he has won. We can decide several factors to be the objective reputation criteria:

- Number of games played: totally how many games this player played before
- Wins: the number of winning games
- Losses: the number of lost games
- Unfinished: some players may leave the game
- Score: the sum of score
- Awarded points: more game played, more points accumulated to the score

Subjective reputation criteria include the following factors:

- Player skill level: this could be level of "beginner", "intermediate" and "advanced" level proved by the test result when a new player registers for a game

- Feedback: Evaluation is given by other players in a team when the game is finished, then an overall feedback will be calculated as this time and kept for future reference.

We will implement the partial reputation mechanism on objective factors in case study (section 4.2.1). Reputation management is an integral part of requirement for a lobby. The above model, in somewhat degree, can avoid the easily changed identity since the new player will have zero score while an existing player has more scores and awarded points with more game played. This model could be studied further but is not the focus of this thesis.

3.6 Chapter Summary

In this chapter, four elements related to team development in different game phases were explained in detail.

- Pre-game: Lobby services to facilitate the team formation. It provides not only the “thin” information, but also the richer one for players to make decisions to select team or team members.
- Game: Coordination between the players for team formation, dissolution, and for playing the game. The coordination model based on LRTS (Layered Reactive Tuple Space) for games was put forward with illustration of tuples, reactions and examples.

Protocols of joining, leaving the game and inviting other players to the collaborative environment were presented.

- Post-game: Reputation management in game communities. The Reputation mechanism for game communities, which keeps player's rating history, is used as one of the lobby services and used in the protocols.

Chapter 4

Case Study: Multi-player Online Game Agent World

4.1 Introduction to a Game Agent World

We will apply our coordination model and reputation model to a game Agent World [38]. A game board of Agent World is a grid filled with pieces (squares) of different colors, destinations (circles), and representations of the players, as shown in Figure 4.1. In the basic scenario, the whole game board is visible to each player. An advanced version of the game can also be played in which only a portion of the game board is visible to a player. Thus we can model the situation that players only have partial information about the game environment.

Figure 4.1 Each agent represents a player who participates in the game

The objective of the game is to move all pieces to their correct destination (which is the destination of the same color as the piece) in a minimal number of moves. Only one player at a time can make a move to a field next to him. Making a move means that a player can step to a free field, pick up a piece, or put down a piece. The common goal for the players then becomes to coordinate their moves with those of others to minimize the total number of moves expended. There are various possible strategies players can follow, for example, a simple "relay" strategy in which agents move pieces along a chain (passing them from one to another), instead of each agent executing the required moves individually. When we apply our coordination model to this game, we will use the following types of tuples:

begin tuple: notification to all waiting players that the game begins since it has the required number of players

game tuple: contains game status information

game over tuple: indicates that game is over

invitation tuple: posted by player A to invite player B to begin a game or to replace an urgently leaving player

join tuple: posted by a player when joining a game

leave tuple: posted by a player after leaving a game before the game is over, it contains this player's identification and his playing order/position in the game

move tuple: indicates a move on the game board (e.g. pick, skip, undo etc.)

message tuple: contains the message source, destination and content

player tuple: contains a player's unique identification, password and reputation information

reaction tuple: a special tuple that instructs tuple space to take reactions according to coming tuple and the operation over this tuple

record tuple: a tuple with player's record (here we simply use parameter "score")

4.2 Examples of Applying LRTS Coordination Model

In the game Agent World, there is one lobby server and a certain number of game servers as shown in Figure 4.2. The game servers are leaders at the same level as illustrated in Layered Reactive Tuple Space model (Figure 3.3). When the lobby server starts, it will start these game servers. Each game server will host a copy of the Agent World game, but with a different configuration of game parameters such as the number of players, the number of pieces to move, the view size, as well as how many players are currently waiting, and their identities. In this case study, one game server only manages one game and uses one local tuple space. Some simple examples of applying the LRTS coordination model could be found in [42]. In this section, we have more to illustrate.

Figure 4.2 Lobby server starts game servers, each of which has a local tuple space

4.2.1 Selecting and Joining a Team with Reputation Information

In this example, we also partially implemented a reputation model. Joining protocol (Figure 4.3) corresponds to Figure 3.12. On start-up, each game server will write a *game tuple* to the global tuple space:

```

Tuple gameTuple = new Tuple ("game");

// configure other game information
gameTuple.add (new Field("name", name));
gameTuple.add (new Field("numberOfPlayers", numberOfPlayers));
gameTuple.add (new Field("worldSize", worldSize));
gameTuple.add (new Field("view", view));
gameTuple.add (new Field("numberOfPackets", numberOfPackets));
gameTuple.add (new Field("state", state));
gameTuple.add (new Field("list", playerList));

globalTupleSpace.write (gameTuple);
  
```

Figure 4.3 A game tuple is posted to the global tuple space

Figure 4.4 Protocol of joining Agent World game

A player who wants to join the game must connect to the lobby server, from which he could get lobby services. As soon as the lobby server receives a connection request from a player, it will read all current game tuples from the global tuple space:

```

Tuple template = new Tuple ("game");
template.add (new Field ("name", String.class));
template.add (new Field ("numberOfPlayers", Integer.class));
template.add (new Field ("worldSize", Integer.class));
template.add (new Field ("view", Integer.class));
template.add (new Field ("numberOfPackets", Integer.class));
template.add (new Field ("state", Integer.class));
template.add (new Field ("list", String.class));

return globalTupleSpace.readAll(template);

```

Figure 4.5 Global tuple space reads all game servers' status

Then the lobby server presents a lobby interface showing the status of each game in progress (Figure 4.6).

Figure 4.6 Lobby interface shows the status of each game

From the information the player received, he will decide which game to enter. A new player's information (e.g. reputation) is shown in Figure 4.7. Here we deploy a simple mechanism that displays the information: how many times a player played a game, how many games he left urgently, and total score he won. Each time when a player finished a game successfully, reputation will be changed at the lobby server side as shown in Figure 4.8. If this player behaves uncooperatively, for example by leaving a game urgently, his reputation is changed as well and score will not be accumulated as shown in Figure 4.9. Based on the above reputation information, other players could decide if

this player is worth playing or collaborating with. We could deploy more sophisticated reputation mechanism according to different game requirements.

Figure 4.7 New player Alice's reputation (has not been rated yet)

Figure 4.8 Player Alice's reputation changed after playing once

Figure 4.9 Player Alice's reputation changed after urgently leaving a game

4.2.2 Beginning a Game

Once a player joins a game, he has access to the local tuple space (Figure 4.10). Each joining request from a player will cause the game server to post a *join tuple* to the local tuple space (Figure 4.11).

Figure 4.10 Players joining the game have access to the local tuple space

```

Tuple template = new Tuple ("join", new Field ("name", playerName));
localTupleSpace.write (template);
  
```

Figure 4.11 A join tuple is posted

Then a player will be blocked until a *begin tuple* appears in the local tuple space.

```

Tuple begin = new Tuple ("begin");
//block here waiting for begin tuple
localTupleSpace.read (begin);
  
```

Figure 4.12 Player blocks for begin tuple

Local tuple space will check how many players have already joined the game (including the player who has just joined), and post a *begin tuple*, if appropriate to notify all waiting players:

```

state = localTupleSpace.readNumber (new Tuple ("join"));
// update game server status change to all login players
updateGameInfo ();
if (state == numberOfPlayers) {
    Tuple begin = new Tuple ("begin");
    // wake up waiting handler threads
    localTupleSpace.out(begin);
}

```

Figure 4.13 Local tuple space unblocks waiting players: posting a begin tuple

The *begin tuple* lets all blocked players begin the game and game board will display for each player.

4.2.3 Inviting Other Players

In section 3.4.2, we know that after one player leaves the community, the remaining players could invite others to re-form the team based on the player's reputation. Invitation in Figure 3.13 happens in the control of one team leader. In this example, invitation happens between different team leaders. We have corresponding sequence diagram for inviting players in the game Agent World (Figure 4.14 and Figure 4.15).

Figure 4.14 Inviting a player in different game server: invitation from inviter

Figure 4.15 Inviting a player in different game server: decision from invitee

This process involves the use of layered tuple spaces (Figure 4.16).

Figure 4.16 Inviting a player via the layered tuple spaces

Player A1 leaves the game before it is finished and player A2 has had the experience of playing with player B before or knows that player B is of a good reputation. Therefore player A2 will send the invitation request to his local tuple space, which can be read by the game server A. Server A writes the *invitation tuple* to the global tuple space, from which it can be read by other game servers (e.g., game server B).

Game Server A writes the *invitation tuple* to the global tuple space:

```

/**
 * construct invitation tuple with important parameters:
 * inviter's and invitee's name and game server the inviter belongs to
 */
Tuple invitationTuple = new Tuple ("invitation",
                                   new Field ("inviter", playerName),
                                   new Field ("invitee", inviteeName),
                                   new Field ("gameserver", this.i));
globalTupleSpace.write(invitation);

// lobby server notifies the corresponding Game Server (e.g. B)
lobbyServer.notifyInvitation();

```

Figure 4.17 Game Server (A) writes invitation tuple to the global tuple space

Game server B then forwards this invitation to player B by taking out the *invitation tuple* from the global tuple space and writing it to the correct local tuple space.

```
Tuple template = new Tuple ("invitation",
    new Field("inviter", String.class),
    new Field("invitee", String.class),
    new Field("gameserver", Integer.class));

//remove invitation tuple from the global tuple space
Tuple invitationTuple = globalTupleSpace.takeNoBlock (template);

// ...
localTupleSpace.write (invitationTuple);
//notify players
```

Figure 4.18 Local tuple space stores invitation tuple

Player B will get the invitation from the inviter via local tuple space.

```
Tuple template = new Tuple ("invitation",
    new Field("inviter", String.class),
    new Field("invitee", player),
    new Field("gameserver", Integer.class));

Tuple invitationTuple = localTupleSpace.takeNoBlock(template);

// get information in invitation tuple
```

Figure 4.19 Player reads invitation tuple from the local tuple space

The interface for player B is illustrated in Figure 4.20. Player B then decides whether to accept the invitation.

Figure 4.20 Interface for Player B to make a decision (from player Alice)

Invitation protocol used is not limited to the team re-formation; it also can be used in the first time team formation.

4.2.4 Newly Joined Player

Three advantages of Linda model: *associative addressing*, *spatial uncoupling* and *temporal uncoupling* (section 2.5.3) are fully exploited in the game Agent World. One typical example is that a new player replaces an urgently leaving player in the middle of a game. The new player could have current game status by picking up all *move tuples* in time order as shown in Figure 4.21.

Figure 4.21 New player X reads current move tuples from the local tuple space

- **Spatial uncoupling:** communication is anonymous. The producer of move tuples does not need to know who will pick them up later and the consumer does not know who produces these *move tuples*. Player X only needs to pick up all *move tuples* for him to get game board status.

- **Associative addressing:** Tuple spaces use an associative addressing scheme based on pattern matching. As long as players have access to the space, they can perform read or write operation to the shared data space.
- **Temporal uncoupling:** Move tuples have their own life span, independent of the players who produced them, or players who consume them. Even if player 1 leaves the game, his *move tuples* still exist in the local tuple space. So it is possible for new player X to pick up and continue the game smoothly.

A player leaves urgently and he posts a *leave tuple* to the local tuple space (see section 4.3.2 for leaving). Later player X picks up this *leave tuple* and knows which position he should be in the game.

```

Tuple template=new Tuple("join", new Field("name", playerName));

// whether there is an urgent leaving player
Tuple leave = new Tuple ("leave", new Field ("name", String.class),
                        new Field("order", String.class));
Tuple hasLeave = localTupleSpace.inNoBlock(leave);

if (hasLeave == null){ // no one leave urgently before
    localTupleSpace.write(template);
    //...
} else { //replace the urgent leaving player
    // get the position
    // state = ...
    localTupleSpace.write(template, state);
    //...
}

```

Figure 4.22 Local tuple space stores a new player tuple

The new player then picks up all *move tuples* in the local tuple space to get current game status.

```

// construct a template move tuple
Tuple moveTuple = new Tuple ("move", new Field ("action",
String.class), new Field ("source", String.class),
new Field("from", String.class), new Field("to", String.class));

int moveNumber= localTupleSpace.readNumber(moveTuple);

if (moveNumber!=0){ // this player joins to replace another player
    ArrayList moves=localTupleSpace.readAll(moveTuple);
    // update each move step
}

```

Figure 4.23 New player picks up move tuples if they exist

4.3 Reactions in Game Agent World

The *Reaction* class and the algorithm of triggering a reaction were presented in Chapter 3 (section 3.3). In our case study, both global and local tuple spaces could be installed different reactions, therefore player agents only care about their own goals: to keep the total number of moves to a minimum, not the social rules or coordination rules. Here are four typical example reactions as shown in Table 4.1. When “out” or “read” operates on these tuples, the reaction object will be executed.

Tuple Type	Operation (trigger of reaction)	Reaction
Player Tuple	out	Checking if this player’s identification is unique, if so write the player tuple to space and remove the original tuple
Leave Tuple	out	Removing the leaving player’s join tuple and pause the game until the new one replaces an urgent leaving player
Message Tuple	out	Checking message destination and creating new message tuples for players
Move Tuple	read	Reading the same move tuple for three times (since three players) and then increasing the tail position to read next move tuple

Table 4.1 Reactions in game Agent World

4.3.1 Checking a New Player's Identification

Each player who joins the game should login to the lobby server using a unique identification. The new player writes a *player tuple* with user identification, password and other personal information such as reputation to the global tuple space (reputation will change as games played).

```
Tuple playerTuple=new Tuple ("player", new Field ("name", playerName),
    new Field("password", password), new Field("totalNumber",0),
    new Field("unfinished",0), new Field("score",0));

// only out this player tuple to the space
globalTupleSpace.out (playerTuple);
```

Figure 4.24 Player tuple with player info is posted to the global tuple space

Global tuple space has this reaction to check a new player's identification and it should have this reaction installed.

```
/** lobby server installs checking unique id reaction in global space
 */
// construct a reaction object, which is the subclass of Reaction
Reaction reactionObj = new CheckUniqueName();
reactionTuple = new Tuple("reaction", new Field("template", "player"),
    new Field("operation", "out"), new Field("reactionObj",
    reactionObj));

globalTupleSpace.out (reactionTuple);
```

Figure 4.25 Installing Checking ID reaction in the global tuple space

When the player executes “out” a *player tuple*, the global tuple space checks reactions installed. If this space has a reaction for this operation (*out*) over this kind of tuple (*player tuple*), it performs the reaction by executing *reaction object*.

```

Template = new Tuple ("reaction", new Field ("template", "player"),
    new Field("operation", "out"),
    new Field("reactionObj", CheckUniqueName.class));

reactionResult = globalTupleSpace.readNoBlock(template);
// global space has reaction for this operation
if (reactionResult!=null) {
    //get reaction object ...
    // execute reaction
    success = reaction.executeReaction(globalTupleSpace, playerTuple);
    // return player info ...
}

```

Figure 4.26 Checking ID reaction object is retrieved and executed

Player tuple with unique user identification and password will be written to the global tuple space and will last forever after the global space executes reaction. Therefore next time the same user could use this identification to login to the lobby server as an existing user. Otherwise, this player should login again using a new identification. After login successfully, player will get a lobby interface as shown in Figure 4.6.

4.3.2 Leaving Game before Game is over

A player can leave the game at any time, but it is not a good behavior to leave before game is over since it may affect his reputation. However if this is the case, the player's departure will lead a *leave tuple* posted to the local tuple space. Reaction in local tuple space for *leave tuple* will remove this player's *join tuple* and stop the game until a new player joins the game. The new player removes the *leave tuple* and replaces the leaving player as discussed in section 4.2.4.

```

/**
 * game server installs reaction for leave tuple in local tuple space
 */
Reaction reactionObj = new LeaveReaction();
reactionTuple = new Tuple("reaction", new Field("template", "leave"),
    new Field("operation", "out"), new Field("reactionObj",
    reactionObj));
localTupleSpace.out(reactionTuple);

```

Figure 4.27 Installing Leave reaction in the local tuple space

When the leave tuple is “out” to the local space, the LeaveReaction object will be executed, leading to the removal of corresponding *join tuple*.

```

// construct template tuple for: the operation on the coming tuple
// local tuple space has the reaction
Tuple leaveTuple = new Tuple("leave", new Field("name", playerName), new
    Field("order", agent));
//check if leaveReaction exist
if (reactionResult!=null){
    // get the real leaveReaction
    leaveReaction.executeReaction(localTupleSpace, leaveTuple){

    // notify other players
}

```

Figure 4.28 Leave reaction object is executed

```

/**
 * executeReaction() in LeaveReaction class
 */
// get the leaving player's name and his playing order
// find corresponding join tuple and remove it
Tuple join = new Tuple ("join", new Field("name", playerName),
    new Field ("position", position));
if (localTupleSpace.takeNoBlock(join)!=null){
    return true;
} else
    return false;

```

Figure 4.29 Fragment code for leave reaction

4.3.3 Message Communication

For all players in one game server, there is one local tuple space for them to coordinate not only for taking their actions (steps) but also for exchanging their messages. Messages could be game strategies, suggestion to others or help information. Player can cooperate by talking to each other so that they can better understand the strategy and other players' intention to finish the task successfully. They can send messages to the tuple space with source player, destination players and message content.

```
Tuple messageTuple = new Tuple("message", new Field("from", from),
                                new Field("to", to), new Field("content", content));

localTupleSpace.out(messageTuple);
```

Figure 4.30 Message tuple is posted to the local tuple space

Reaction to message tuple is done by local tuple space by checking the destination of the message and writing corresponding *message tuples* to local space for destination players.

```
// get the message source and destination
// write message tuple for each destination
for (each destination){
    messageTuple = new Tuple("message", new Field("from", from),
                            new Field("to", destination),
                            new Field("content", content));
    localTupleSpace.write(messageTuple);
}
// broadcast message
```

Figure 4.31 Fragment code for message reaction

Each player will read message tuple for him after getting notification.

```
Tuple template=new Tuple("message", new Field("from", String.class),
    new Field("to",player),  new Field("content", String.class),
    new Field("position",String.class));

Tuple result=localTupleSpace.takeNoBlock(template);
```

Figure 4.32 Reading message from local tuple space

4.3.4 Moving Steps in Game Board

Every step will have a *move tuple* written to the local tuple space.

```
/**
 * construct the move tuple then write to local space
 */
Tuple moveTuple = new Tuple("move", new Field("action", action),
                             new Field("source", source), new Field("from",
                             from), new Field("to", to));

// write move tuple to local space with Field "position"
localTupleSpace.write (moveTuple);
```

Figure 4.33 Move tuple is posted to the local tuple space

The field “action” in a move tuple means picking a packet, putting a packet, stepping to neighbor field and skipping. The fields “source”, “from” and “to” refer to the “source” player who takes this action, “from” one field “to” another field respectively. Each *move tuple* has time order for players to read. The game server then notifies other players to read the *move tuple* and this step is updated on all players’ game boards simultaneously. This process is illustrated in Figure 4.34.

Figure 4.34 Players coordinate steps via local tuple space

One problem for other players to read the *move tuples* is coordinating who will be responsible for counting order of *move tuples*. If this were the job of a player agent, a channel policy of *move tuples* would be deployed for all players to read the tuples correctly. Then part of the player agent's task is devoted to his own goal: to read the *move tuple* then update this move on the game board and part of the task is devoted to implementing the coordination rule of channel policy.

As we have discussed in the LRTS coordination model (Chapter 3, section 3.2), the reactive tuple space is to “promote the separation between the individual perception of coordination and the global coordination issue, enabling the modeling and shaping of the interaction space independently of the interaction entities.”[31]. Local tuple space has a reaction to read existing *move tuples*, which always allows each player to read the next *move tuple* and allows the same *move tuple* to be read three time (three remaining players). Therefore player's behavior is fully focused on his individual goal. If this coordination rule changes, it is not necessary to change the implementation of each player agent, but to install another reaction in the local tuple space. Local tuple space should install the reaction first. Reading the move tuples by the players will trigger this reaction.

```
Reaction reactionObj = new MoveReaction();
reactionTuple = new Tuple("reaction", new Field("template", "move"),
    new Field("operation", "read"), new Field("reactionObj",
    reactionObj));
localTupleSpace.out(reactionTuple);
```

Figure 4.35 Installing Move reaction

When each player reads new *move tuple*, MoveReaction object is executed.

```
// tail refers to the order of move tuple
static protected int tail=1;

// number means there are 3 players remaining
static protected int number=3;

// executeReaction()
Tuple result=null;
Tuple move = new Tuple ("move", new Field ("action", String.class),
    new Field("source", String.class),
    new Field("from",String.class), new Field("to",String.class) );
move.add(new Field("position", new Integer(tail)));

if (number>0){
    // read the same move tuple three times by three players
    result=localTupleSpace.readNoBlock(move);
    number--;
    if (number==0){
        // ready for reading next move tuple
        number=3;
        tail++;
    }
}
// return result;
```

Figure 4.36 Fragment code of Move reaction

4.4 Chapter Summary

In this chapter we studied a cooperative online multi-player game Agent World. We applied lobby service, the proposed coordination model LRTS and a reputation model to this game. Examples mainly have two parts. The first part has scenarios using layered tuple space for joining the team, beginning the game and inviting a new player to the game. The last part includes four typical examples of reactions in the tuple space: checking player's identification, leaving the game, reading messages and reading game board steps of each player. Four elements of game communities are fully investigated, which could be used in other games or other similar collaborative applications.

Chapter 5 Conclusions

Virtual game communities are beginning to play an important role in our daily life, and have become a hot area for academic research. Our research on virtual communities is part of the SCE (Scalable Collaborative Environment) project [47]. The goal of this project is to develop an open and scalable collaborative environment for Internet-based distributed applications involving geographically dispersed participants. A key component of collaboration is coordination. As the initial focus of this project has been on collaborative multi-player games, we studied coordination mechanisms in game communities. Our particular focus was on team formation, evolution and dissolution. Initial results of coordination in game communities are published by ACM Crossroads. Readers could refer to [42] for some basic applications.

Below, we summarize our contributions and give an outlook on further research.

5.1 Contributions

We can point out following contributions from this work:

- This thesis provided a study on the requirements of game communities in order to better understand the role of coordination in the team development from formation to dissolution. We highlighted four elements related to the team in different game phases: lobby services in pre-game, coordination mechanism and protocols in game

and reputation management in post-game. Each element has different but indispensable responsibilities in the whole process of the game.

- This work improved our understanding of coordination models. We had a summary of current research on coordination models. We learned those models' categories and their features for a specific application domain. This part of thesis serves as basis for our proposed coordination model.
- We defined a coordination model for online games. Coordination is the core to team development in game communities. We proposed a coordination model based on the Linda model. Moreover, this model has layered structure to meet the needs of game communities. Each tuple space in the model has the reaction ability as needed by the application. This model, based on LRTS (Layered Reactive Tuple Space), is scalable, reusable and suitable for the open, heterogeneous Internet environment.
- We defined coordination model API and implemented this architecture in a cooperative online multi-player game - Agent World. We described implementation issues and implementation results of this model. In this real game, typical scenarios of layered tuple spaces and reactions are presented.
- We presented lobby services a virtual community could provide. Lobby service can have most of the information, not only the "thinner" but also a "richer" one. It is an approach for one player to aware other players. In order to make decisions to join a team or to initially form a team, players can choose their favorite games based on

their interests, or players' reputation provided by the lobby services of virtual communities.

- We described a set of coordination protocols for joining, leaving a game, and inviting a player to re-form the team based on the LRTS model. We implemented the protocols in our case study.

5.2 Future Work

We believe that our coordination model is not limited to applications in multi-player games. Coordination mechanisms play a central role in any type of collaborative virtual environment. One application domain of great interest for our future work is e-learning. Similar problems arise from managing groups of users, matching up online classes with learners, and supervising the progress of a class. A direct application of the results from our study of the e-gaming domain is even possible, since games are a popular way of imparting learning material. We also can deploy software agent technology to represent a human player in e-gaming domain or to represent student and teacher in e-learning domain. Reputation mechanism, protocols and lobby services could have more possibilities for exploration in different domains.

Chapter 6 Bibliography

1. A. Abdul-Rahman, and S. Hailes, "Supporting Trust in Virtual Community", the 33rd Hawaii International Conference on System Sciences (HICSS), IEEE, 2000.
2. S. Ambroszkiewicz, et al, "Team Formation by Self-Interested Mobile Agents", In Chengqi Zhang and Dickson Lukose (Eds.) Multi-Agent Systems, Proc. 4th Australian DAI Workshop, Brisbane, 13h July, 1998, published in Springer LNAI 1544
3. N. Busi, et al, "Coordination Models: A Guided Tour", in A. Omicini, F. Zambonelli, et al (eds.), Coordination of Internet Agents: Models, Technologies, and Applications, pp.6-24, Springer, 2001.
4. G. Cabri, et al, "Reactive Tuple Spaces for Mobile Agent Coordination", Proceedings of the 2nd International Workshop on Mobile Agents, Lecture Notes in Computer Science No. 1477, Springer, September 1998
5. N. Carriero, D. Gelernter, "Linda in Context", Communication of the ACM, Vol. 32, No. 4, pp. 444-458, 1989
6. E. Churchill (ed.), Collaborative Virtual Environments, Springer, 2001
7. P. Ciancarini, et al, "Multiagent System Engineering: The Coordination Viewpoint", Agent Theories, Architectures, and Languages, 2000
8. P. Ciancarini, et al, "Jada - Coordination and Communication for Java Agents", Lecture Notes in Computer Science No. 1222, pp. 213-226, Springer 1997

9. P. Ciancarini, et al, "Towards an interactive Web", <http://flp.cs.tu-berlin.de/pagespc/ieeeip/ciancarini.html>, last accessed August 2003
10. D. Deugo, M. Weiss, and E. Kendall, "Reusable Patterns for Agent Coordination", in A. Omicini, F. Zambonelli, et al (eds.), *Coordination of Internet Agents: Models, Technologies, and Applications*, pp.347-368, Springer, 2001
11. F. Dignum, et al, "Agent Architecture and Theory for Team Formation by Dialogue", ATAL'00 (Agent Theories, Architectures, and Languages 2000)
12. P. Domel, et al, "Mobile Agent Interaction in Heterogeneous Environment", *Proceedings of the International Workshop on Mobile Agents, Lecture Notes in Computer Science No. 1219*, pp. 136-148, Springer, April 1997
13. W. Keith Edwards, "Session Management for Collaborative Applications", *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, 1994
14. C. Fairclough, et al, "Research Directions for AI in Computer Games", *Proceedings of the 12th Irish Conference on AI and Cognitive Science*, 2001
15. L. Foner, "Community Formation via a Distributed, Privacy-Protecting Matchmaking System", in T. Ishida (ed.) *Community Computing and Support Systems*, *Lecture Notes in Computer Science No.1519*, pp. 359-376, Springer 1998
16. E. Freeman, et al, *JavaSpaces: Principles, Patterns, and Practice*, Addison-Wesley, 1999.
17. F. Fukuyama, "E-Commerce and the Challenge of Trust",

<http://www.ml.com/woml/forum/ecommerce1.htm>, last accessed August 2003

18. D. Gelernter, and N. Carriero, "Coordination Languages and their Significance",
Communication of the ACM, 35(2), pp.97-107, February 1992
19. R. Gray, "Agent Tcl: A flexible and secure mobile-agent system", Proceedings of
the fourth Annual Tcl/Tk Workshop (TCL '96), Monterey, California, July 1996
20. R. Hamman, "Computer Networks Linking Network Communities", Chapter 4 in C.
Werry, M. Mowbray (Eds.), "Online Communities", Prentice-Hall, Inc., 2001
21. F. Hattori, et al, "Socialware: Multiagent Systems for Supporting Network
Communities", in T. Ishida (ed.) Community Computing and Support Systems,
Lecture Notes in Computer Science No. 1519, pp. 330-341, Springer 1998
22. T. Ishida, "Towards Computation over Communities", in T. Ishida (ed.) Community
Computing and Support Systems, Lecture Notes in Computer Science No.1519,
pp.1-9, Springer, 1998
23. J. E. Laird, and M. van Lent, "Human-level AI's Killer Application: Interactive
Computer Games", Proceedings of the National Conference on Artificial
Intelligence American Association of Artificial Intelligence (AAAI), pp.1171-1178,
2000
24. T. Luke, "Building a Virtual University: Working Realities from the Virginia Tech
Ctberschool ", Chapter 7 in C. Werry, M. Mowbray (eds.), "Online Communities",
Prentice-Hall, Inc., 2001
25. T. Malone et al, "The Interdisciplinary Study of Coordination", ACM Computing
Surveys, 26(1), pp.87-119, March 1994

26. C. Niederberger, and M. Gross, "Towards a Game Agent", CS Technical Report 377, Swiss Federal Institute of Technology Zurich, 2002
27. T. Nishida, et al, "CoMeMo-Community: A System for Supporting Community Knowledge Evolution", in T. Ishida (ed.) Community Computing and Support Systems, Lecture Notes in Computer Science No. 1519, pp. 183-200, Springer 1998
28. A. Omicini, and F.Zambonelli, "Tuple Centers for the Coordination of Internet Agents", ACM Symposium on Applied Computing, ACM, 1999
29. H. Peine, T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents", Proceedings of the 1st International Workshop on Mobile Agents", Lecture Notes in Computer Science No. 1219, pp. 50-61, Springer, April 1997
30. J. Reynolds, "Tactical Team AI Using a Command Hierarchy", in S. Rabin, (ed.), AI Game Programming Wisdom, pp.260-271, Charles River Media, 2002
31. A. Ricci, et al, "Objective vs. Subjective Coordination in Agent-Based Systems: A Case Study", COORDINATION 2002, Lecture Notes in Computer Science No. 2315, pp.291-299, Springer, 2002
32. J. Schlichter, et al, "Awareness: The Common Link Between Groupware and Community Support Systems", in T. Ishida (ed.) Community Computing and Support Systems, Lecture Notes in Computer Science No.1519, pp.77-93, Springer, 1998
33. P. Schubert, "The Pivotal Role of Community Building in Electronic Commerce", the 33rd Hawaii International Conference on System Science (HICSS), IEEE, 2000
34. M. Schumacher, Objective Coordination in Multi-Agent System Engineering,

Lecture Notes in Artificial Intelligence No.2039, Springer, 2001

35. M. Weiss, course notes for COMP 4104 “Distributed Programming in Java”,
<http://www.scs.carleton.ca/~weiss/courses/4104/handouts/01-introduction-2.pdf>,
 last accessed August 2003
36. M., Weiss, course notes for COMP 4104 “Distributed Programming in Java”,
<http://www.scs.carleton.ca/~weiss/courses/4104/handouts/06-flow-3.pdf>, last
 accessed August 2003
37. M. Weiss, “Patterns for e-Commerce Agent Architectures: Using Agent as
 Delegates ”, 8th Conference on Pattern Languages of Programs, 2001
38. D. Weyns, Home Page of the Project “Sociality in Multi-Agent Systems”,
<http://www.cs.kuleuven.ac.be/~danny/SocialityInMAS.html>, last accessed August
 2003
39. M. Wooldridge, and N. Jennings, Intelligent Agents: Theory and Practice, The
 Knowledge Engineering Review, 10(2): 115-152, 1995
40. P. Wyckoff, et al, “T Spaces”, IBM Systems Journal, 37(3), pp.454-474, August
 1998
41. G. Zacharia, “Collaborative Reputation Mechanisms in Electronic Marketplaces”,
 Proceedings of the 32nd Hawaii International Conference on System Sciences, 1999
42. Y. Zhang, and M. Weiss, “Virtual Communities and Team Formation”, ACM
 Crossroads, Fall 2003
43. eBay, <http://www.ebay.ca>
44. Online Community Report, <http://www.onlinecommunityreport.com/features/10>,

last accessed August 2003

- 45. MSN Games, <http://zone.msn.com/default.aspx>
- 46. Magnetar Company, <http://www.magnetargames.com/Company/Partners.htm>
- 47. Scalable Collaborative Environment Project: <http://www.cs.ubc.ca/~sce/>
- 48. Yahoo Games, <http://games.yahoo.com/>