Programmer Cognition in Explicit Coordination Modeling: Understanding the Design of Complex Human Interaction and Coordination

Sirong Lin

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Computer Science and Applications

Deborah G. Tatar
Stephen H. Edwards
Manuel A. Pérez-Quiñones
Eli Tilevich
Christopher J. DiGiano

November 29, 2011

Blacksburg, Virginia, USA

Keywords: parallel thinking, explicit coordination modeling, TupleSpace, ubiquitous parallelism

Copyright © 2011 by Sirong Lin

Programmer Cognition in Explicit Coordination Modeling: Understanding the Design of Complex Human Interaction and Coordination

Sirong Lin

Abstract

Parallel thinking is a mindset that enables computer scientists to think about and implement systems that allow activities to happen concurrently. This mindset is needed in designing and implementing a wide range of computer systems involving coordinated components (e.g., parallel, distributed, and multi-user systems). No matter what the coordinated component is, whether human or computer, the underlying issue is to imagine coordination between these components and manage the distribution and reintegration of coordinated work. The rapid development of multi-core technologies has attracted people's attention back to parallelism. Ubiquitous and pervasive computing further brings parallelism into the everyday experiences of non-computer scientists. Designing and developing for ubiquitous parallelism become an essential and heavy responsibility for every software designer and developer. This situation creates a new standard for every one working in the computing field; simply understanding the techniques and algorithm in parallel-distributed computing to support parallel computing resources is not enough; the ability to create support for parallel human activities is also needed. Therefore, the need to train CS students to have a "parallel thinking" mindset is more urgent than ever.

This doctoral work approaches the pedagogy of parallel thinking by teaching CS students to model coordination for parallel human activities explicitly. Although most participants started with an undeveloped imagination for human coordination, they were able to improve by focusing on coordination issues in the context of a class. The research method was to study a semester-long experimental class in the Department of Computer Science at Virginia Tech through a qualitative design-based research approach. Multiple types of data were collected using methodological triangulation to maximize validity. The data analysis process was guided by Grounded Theory (GT) through a systematic set of procedures. The outcomes provide a rich, thick, and detailed description about how CS students conceptualize and approach parallel thinking. The research contributes to CS education, programmer cognition literature, and computer supported collaborative system design and development by elaborating and analyzing various challenges in coordinated system creation, and making suggestions about pedagogical solutions, and software infrastructure and tools design.

This dissertation is dedicated to:

My parents: XueChang Lin （林学昌） and BaoLian Zhang（张宝连）

My husband: Ling Chen（陈岭） and my son: Jialin Chen（陈家霖）

Acknowledgments

The writing of this dissertation has been one of the most significant academic challenges I have ever had to face. I would never have been able to finish my dissertation without the guidance and patience of my academic advisor and committee members, endless support from my family, and generous help from my friends.

I would like to express my deepest gratitude to my advisor, Dr. Deborah Tatar, for her excellent guidance, caring, patience, and encouragement through these years. Most importantly, I have learned from her to look at Computer Science from a different perspective, from the outside of the computer scientists' object-world. I also wish to thank my committee members: Dr. Manuel Pérez-Quiñones, Dr. Eli Tilevich, Dr. Chris DiGiano, Dr. Stephen H. Edwards, and Dr. Wuchun Feng (for the first couple of years), for their advice and suggestions in my research work. Special thank goes to Professor Steven Harrison for sharing his insight in HCI in the lab meetings and giving suggestions for my research. I also like to thank Dr. Ribbens and the department of Computer Science, for offering me funding for several years.

My great gratitude goes to the colleagues in the POET research lab and h.Lab for their valuable feedbacks and suggestions to my research work and collegial support — Kibum Kim, Jamika Burge, Meg Kurdziolek, Laurian Vega, Stacy Branham, Joon-Suk Lee, Tejinder Judge, Michael Stewart, Bobby Beaton, Nouf Alaloula, Promita Chakraborty, and Michelle McLeese. In addition, this work could not have been completed without the aid of study participants in the CS 4984 class in Fall 2005. Thanks also go to the researchers I met in SRI International (Dr. Jermey Roschelle, Dr. Charles Patton, Dr. Patti Schank, and Mr. John Brecht) for their collaboration and

feedbacks on TupleSpace project. I also thank NSF for their funding support (NSF ITR/IERI Grant #REC 0427783).

I would like to thank my parents, sister, parents-in-law, sisters- and brothers-in-law, and many other family members for supporting and encouraging me with their best wishes. I would also like to acknowledge my American parents, Mrs. Phyllis A. Randall and Dr. Cliff W. Randall, for treating me like their own daughter and supporting me with whatever (e.g., food, housing, baby-sitting) I needed at the moment. I also thank many of my good friends such as Shuping Dong, Bin Li, Fang Meng, Min Lan, Xiumei Song, and many others for all of their moral and academic support and friendship through these years.

Most of all, I would like to thank my husband, Ling Chen, for his endless supports and encouragements. He has been always there cheering me up and standing by me through the good times and bad. And I specially thank my dearest son, Jialin Chen, for being a good boy in the past 5 years while I have been challenged as a mom and a doctoral candidate.

Table of Contents

Abstract ii

Acknowledgments iv

Table of Contents vi

List of Figures xi

List of Tables xii

1. Introduction 1
   1.1 Statement of the Problem 2
      1.1.1 Example One: CSCL Software Development 3
      1.1.2 Example Two: CSCW and Social Software Development 4
   1.2 Purpose of the Research 5
   1.3 Research Question 7
   1.4 Research Approach 9
   1.5 The Researcher and the Research 10
   1.6 Contributions 10
   1.7 Dissertation Organization 11

2. Literature Review 15
   2.1 Programmer Cognition 16
      2.1.1 Programming and Program Design 16
      2.1.2 Research in Programmer Cognition and Related Areas 18
   2.2 Example Systems Designed for Collaborative Learning 20

2.2.1 Traditional Classroom Environment vs. Collaborative Classroom Environment 20
2.2.2 Example 1 — Classroom Response Systems 22
2.2.3 Example 2 — Match-My-Graph 22
2.3 Languages and Models for Coordination and Concurrency 25
2.3.1 TupleSpace 26
2.3.2 Other Models and Languages 33
2.3.3 Other Alternatives in Distributed Computing Programming 34
2.4 Teaching Parallelism and Coordination in Computer Science 37
2.4.1 Teaching Parallelism in CS Education 37
2.4.2 Coordination in CS Education 39

3. Research Methodology 40
3.1 Rationale for Qualitative Research Strategy 42
3.2 Participants 43
3.2.1 Participants Selection: Students Participants vs. Professional Participants 43
3.2.2 Participants Recruitment 44
3.2.3 Participants Overall Information 44
3.2.4 Participants' Previous Experiences 45
3.3 Research Design Overview 48
3.3.1 Study Overview 48
3.3.2 Pilot Study 50
3.4 Data Collection Methods 52
3.4.1 Method 1 (M1): Semi-structured Face-to-face Open-ended Interviewing 54
3.4.2 Method 2 (M2): Close Observation 55
3.4.3 Method 3 (M3): The Collection of the Participants' Email-messages, Class Website Board Discussions and Questions 56
3.4.4 Method 4 (M4): Frequent Programming Code Check-in and Final Turn-in Products 56
3.4.5 Method 5 (M5): Questionnaires 56
3.5 Data Analysis and Synthesis 57
3.5.1 Qualitative Data Analysis Strategy 57
3.5.2 Data Analysis Procedure 58
3.5.3 Data Organization 59

3.5.4 Analytic Coding Processes 64
3.5.5 Conclusion Drawing 66
3.6 Ethical Considerations 66
3.7 Issues of Trustworthiness 69
3.8 Limitations of the Study 70
3.9 Chapter Summary 71

4. Experimental Class 72
4.1 Class Schedule 72
4.2 Teaching Materials 73
4.3 Teaching TupleSpace Programming 73
4.4 Coping with Students' Learning Barriers 76
4.4.1 Teaching Non-Java Programmers 76
4.4.2 Teaching Template Matching 77
4.4.3 Teaching Coordination 77
4.5 Class Assignments 78
4.5.1 Design "Tuple Learning Activity" 81
4.5.2 Clicker 82
4.5.3 Team Project 87
4.6 Chapter Summary 95

5. Findings: Students' Learning Experiences and Barriers 96
5.1 Student Programmers Faced Sociological Challenges to Understand Complex Human Coordination and Interaction 97
5.1.1 Participants Took Utilization of the Computer's Ability as a Design Brief 97
5.1.2 Participants Brought Strong Assumptions about User Experience into the Design Task 101
5.1.3 Participants Had Difficulty Envisioning User Coordination through Computers 102
5.1.4 Participants Had Difficulty Envisioning Coordination and Individual's Experience 104
5.1.5 Summary 105
5.2 Student Programmers' Conceptualization of Using TupleSpace to Form Coordination Model 106
5.2.1 Participants' Feedback about TupleSpace 106

5.2.2 Explicit Modeling Coordination and Interaction through TupleSpace 108
5.2.3 Undeveloped Competence in Modeling of Coordinated Activities 111
5.2.4 Technical Roadblocks in TSpaces Programming 113
5.3 Student Programmers Faced Technological Challenges in Forming Computational Model through TupleSpace 117
5.3.1 GUI Design in Java 117
5.3.2 Programming Tools 118
5.4 Chapter Summary 124

6. Discussion 126
6.1 Engineering Thinking and Machine-centered Mindset Hinder User-centered and Value-based System Design 128
6.1.1 Going-in Assumptions in Coordinated System Design 128
6.1.2 Participants' View of Computing: Machine-centered 130
6.1.3 "Rigid" Engineering Thinking Constrains "Soft" Design Skills 132
6.1.4 Undeveloped Social and Human Science Knowledge Limited Participants' View about Computer Technology 134
6.1.5 Other Possible Factors 135
6.1.6 Implications to User-centered and Value-based System Design 136
6.2 Underdeveloped Imagination of Coordination Further Prevented System Level Thinking 138
6.2.1 Summary of Underdeveloped Imagination of Coordination 138
6.2.2 Understanding the Incompetence 139
6.2.3 Implications: Making Coordination and Interaction More Visible 140
6.3 Encouraging Parallel Thinking through Explicitly Modeling Coordination 141
6.3.1 Benefits of Explicit Coordination Modeling with TupleSpace 142
6.3.2 Implication for CS Education in Multi-core and Parallelism 146
6.4 Fundamental CS Principles Need More Attentions 147
6.4.1 Event-driven Programming Paradigm 148
6.4.2 Model-View-Controller Design Pattern 148
6.4.3 Summary 149
6.5 Chapter Summary 149

7. Conclusion 151

7.1 Conclusion 152
7.1.1 Cognitive Challenges in Designing Coordinated Systems 152
7.1.2 Conceptualization in Explicit Human Coordination Modeling 153
7.1.3 Technological Challenges in Building Computational Model in Coordinated System Design 155
7.2 Suggestions and Future Work 156
7.2.1 Suggestions for CS Education 156
7.2.2 Use and Develop Software Infrastructure and Tools 157
7.2.3 Future Work 157
7.3 Final Reflection 158

References 160

Appendix A. Informed Consent Form 169

Appendix B. Demographic Information Sheet 174

Appendix C. Semi-structured Face-to-Face Interview Protocols 177
C1. Semi-structured Face-to-face Interview Protocols (Pre-Class Interview) 177
C2. Semi-structured Face-to-face Interview Protocols (Post-Class Interview) 179

Appendix D. Programming Reflection Questionnaires 181

Appendix E. Tuple Class Major Events and Teaching Materials 185
E1. Tuple Class Major Events and Teaching Materials 185
E2. Detailed Class Timeline and Events 188
E3. Template Matching Exercises 239
E31. Tuple Template Matching Exercise 1 239
E32. Tuple Template Matching Exercise 2 242

Appendix F. Participant Demographic Information and Self-reported Experiences 243

Appendix G. Tuple Design 247

Appendix H. Statistical Results from Programming Reflection Questionnaire 255

Appendix J. Related Publications 257

List of Figures

Figure 1. Traditional classroom and networked classroom response system diagram..... 21

Figure 2. Match-My-Graph, an example of a powerful fine-grained coordinative activity. 24

Figure 3. Basic operations and resource dissemination in TupleSpace..... 29

Figure 4. Qualitative research procedure in this doctoral work. 41

Figure 5. Qualitative data analysis procedure (brief view). 60

Figure 6. Qualitative data analysis procedure (detailed view) 63

Figure 7. The screenshot of the Clicker program interface 85

Figure 8. Apples-to-Apples screen shot with responding Tuples..... 110

Figure 9. The screenshot of Space Monitor..... 123

List of Tables

Table 1. Three Principal Storylines Arising from The Data 14
Table 2. TupleSpace Features and the Potential Benefits 32
Table 3. Participants' Previous Experiences (Participants n =14) 46
Table 4. Data Collection Methods and Outcomes 53
Table 5. Intermediate Data Description 61
Table 6. Analysis Result Description 62
Table 7. Data Processing Flow 67
Table 8. Research Question in Relation to Research Methods and Data Analysis 68
Table 9. Summary of the Class Assignments and Teaching Goal 79
Table 10. Comparison of TupleSpace and Database 83
Table 11. Team Project Summary 89
Table 12. Interesting Design Issues and Choices Arose from the Design Act-Out 92
Table 13. Survey Results Regarding Learning and Using TupleSpace / TSpaces 107
Table 14. Survey Results Regarding the Use of Eclipse (Question 2) 120
Table 15. Experimental Class Major Events and Teaching Materials 185
Table 16. Participant Demographic Information and Self-reported Prior Experiences 243
Table 17. Tuple Design for Algorithm Organ 247
Table 18. Tuple Design for Apples-to-Apples 248
Table 19. Tuple Design for Crossword Puzzle 249
Table 20. Tuple Design for Krypto 250
Table 21. Tuple Design for Hangman 252
Table 22. Tuple Design for Math Bingo 253
Table 23. Tuple Design for Pictionary 254

Chapter 1

Introduction

Multi-core technologies challenge almost every aspect of our assumptions about how computers should operate and how they can operate more efficiently. Parallelism, though not a new concept, is, appropriately, attracting attention in the computing field again. Furthermore, ubiquitous and pervasive computing brings parallelism into the everyday experiences of non-computer scientists. Designing connectivity for virtually everyone on the planet is an essential and heavy responsibility for software designers and developers. The ability to imagine and manage the distribution and reintegration of the coordinated work between parallel-distributed machines and users becomes crucial in designing successful computer systems in computer science. Yet the current CS curriculum is not designed to train CS students to engage in *parallel thinking*, a mindset in which they can easily think about and implement systems that allow activities to happen concurrently. This research work believes that it is crucial for every computer scientist to have such a mindset to design for multiple core, parallel, distributed machines, and connected users.

The focus of this research work is to understand CS students' cognitions and beliefs and trace their actions in the course of learning a specialized coordination language to design and implement parallel-distributed, multi-user computer systems to augment human face-to-face activities. The findings from this research highlight the sociological and technological cognitive challenges that CS students faced in learning, designing, and developing the target systems. The challenges partly elaborate the *object*

world of the Computer Scientist, in the sense that Bucciarelli described the object world of the engineer:

> It is the object as they see and work with it that patterns their thought and practice, not just when they must engage the physics of the device but throughout the entire design process, permeating all exchange and discourse within the subculture of the firm. This way of thinking is so prevalent within contemporary design that I have given it a label — "object-world" thinking. (Bucciarelli 1994), p. 4

The "object-world" thinking that CS students started with both failed to account for parallel thinking and acted as an obstacle, preventing CS student developers from perceiving the issues. Explicit modeling of human coordination and interaction through a specialized coordination language, such as TupleSpace, is one way to help them focus on coordination and interaction issues, further to approach parallel thinking.

This chapter begins with an overview of the research background and the problem that the research wants to investigate. Following this is the research purpose, accompanying research questions, and research contributions. The chapter further describes the overall research methodology, and the organization of the dissertation.

1.1 Statement of the Problem

Parallelism is the natural way to solve complex problems in human problem-solving history: when one problem becomes too complex to solve by one person, the problem is divided into small subtasks, and each subtask can be conquered separately. In computing, parallelism becomes a natural topic when computer scientists pursue a way to run programs faster, no matter whether the goal is to keep multiple cores in a processor to calculate more productively, or perform a complicated search over a distributed database to get a quicker result. Despite many differences between parallel computing and distributed computing, one common aspect is the need to manage the physically independent processes, either processors or computers (over networks), to collaboratively achieve a common goal or accomplish a large task. From this perspective, both parallel

programs and distributed programs are coordinated programs (Carriero and Gelernter 1990). Coordinating these individual entities through allocating subtasks (the individual entity will finish the task asynchronously) and facilitating the communication to synchronize individual subtask is the central issue in the coordinated activities.

In early 1990s, Carriero and Gelernter (Carriero and Gelernter 1990), the authors of TupleSpace, anticipated "parallelism will become, in the not too distant future, an essential part of every programmer's repertoire." Twenty years later, their claim has become true and been accepted by more and more people in the field. However, their claim of "coordination — a general phenomenon of which parallelism is one example — will become a basic and widespread phenomenon in computer science" has not been yet given enough attention. This research work focuses on the coordination issues raised in parallel-distributed multi-user systems. We believes that if the complex coordination between computer users can be understood well, software designers and developers should not have much difficulties imagining the coordination between machines and processors.

Coordination plays an important role in two types of multi-user coordinated systems: computer supported collaborative learning (CSCL) systems and computer supported cooperative work (CSCW) systems. Although the two types of systems have different perspectives in design and development, the core is to enable multiple users to conduct their tasks (either to achieve learning goals together, or to accomplish job duties collaboratively) in parallel. How software designers and developers perceive the system goal largely affects the success of a system. The following two sections discuss the potential issues in design and development in these two systems respectively.

1.1.1 Example One: CSCL Software Development

As computers become more ubiquitous and pervasive, more and more computers (e.g., desktops, laptops, tablets, and handhelds) enter classroom and become one of the most important tools to support learning. Research (Apple Computer Inc. 1995; Marshall 2002) shows that technology can enhance students' learning if it is used in appropriate ways. CSCL systems have especially shown great potential to improve learning inside a

classroom by encouraging students' coordination in various ways (Tatar, Roschelle et al. 2003; Vahey, Tatar et al. 2006; Roschelle, Tatar et al. 2007). Using these systems means that students can learn in a much more interactive way than that a traditional lecture-based classroom can provide — learners take roles, contribute ideas, solve aspects of a larger problem, and work together as a team. This basically means students learn through active participation and coordination, which may be fun.

However, many teachers are still uncertain and reluctant to use new technologies. One of the reasons for their reluctance may be lack of familiarity with the state-of-the-art technologies; another possible reason is that they do not have enough confidence that these computer technologies will enhance students' learning in classroom (Marshall 2002). Teachers do not want to spend extra valuable class time solving technical problems or administrating these computer systems either.

To build more robust (technically) and powerful (educationally) systems, people from multidisciplinary teams, such as teachers, pedagogy designers, and software developers and programmers, are often brought together to build CSCL systems. This ensures that not only technical requirements but also educational and social requirements can be designed as key components in such a software system. For example, to build good systems supporting educational values that teachers or pedagogy designers bear in mind, software developers need to understand that learning gains are often based on using technology in a way that teaches contents but *also* respects the fact that learning is in part a social process. Increasing coordination between students in a proper way can therefore enhance learning. However, it is quite challenging for software programmers to acquire these ideas. Mal-adapted or non-optimal systems can still be built quite often even if software developers and programmers listen carefully to teachers or pedagogy designers.

1.1.2 Example Two: CSCW and Social Software Development

Computer supported cooperative work (CSCW) systems (i.e., groupware) are designed to facilitate a group of people to collaborate in work environment. Social systems have become so popular that millions of users interact with each other virtually

by connecting to the online systems every day. Grudin addressed eight challenges (Grudin 1994) for groupware developers in design and evaluation. Many of the challenges were not due to technical issues *per se*, but resulted from not understanding the unique demands that groupware imposes on developers and users. Sixteen years have passed yet many of these challenges still remain. Furthermore, Shirky identified a deep need to design not only for raw connectivity, but for *human-to-human* communication, for group success (Shirky 2003). Shirky's call suggests that not only do Grudin's issues remain, but also that they have spread into the wider world. More recently, social computing software has been widely discussed because of the wide usage and acceptance of "Web 2.0" (O'Reilly) applications such as Wikipedia, Facebook, Blogs, Flickr, Amazon Reviews, and Google Maps. All of these systems rely on the central ideas in Web 2.0 of encouraging more participation and supporting better collaboration to enhance what we might call "Web 1.0" information and resources. The new opportunities make software development more complex compared with the bare-bones simplicity in Web 1.0. As Garrett, who coined the term Ajax, says:

> The biggest challenges in creating Ajax applications are not technical. […] Instead, the challenges are for the designers of these applications: to forget what we think we know about the limitations of the Web, and begin to imagine a wider, richer range of possibilities. (Garrett 2005)

However, to forget what we think we know is not always easy. It is critical to understand these true challenges and examine the obstacles that keep software designers and developers from seeing these values and conditions when designing and developing the coordinated systems.

1.2 Purpose of the Research

To identify the process of learning that prevents software designers and developers from more spontaneously and successfully approaching the design and development of parallel-distributed coordinated systems to support complex human interaction and coordination, an experimental class was taught to identify the cognitive

challenges that programmers encounter in designing and programming parallel-distributed coordinated activities. The pedagogical goal was to examine the ways to help student programmers better learn collaborative system design and development.

To be more specific, the purpose of the current research is to gather a rich experience of how CS students conceptualize and approach the design and implementation of parallel-distributed, multi-user, coordinated systems to support complex human interaction and coordination using a specialized coordination language. By examining the phenomenon, the research aims (1) to understand the cognitive challenges in parallel-distributed and multi-user interactive system design and development, (2) to understand the challenges in terms of system-level thinking, and (3) to examine a pedagogical approach to support parallel computational thinking in computer science.

First, although the design of parallel-distributed computing systems is frequently taught in CS, programmer-cognition-pertinent issues in this complex area have not previously been explored deeply. Multi-user interactive systems such as CSCL, CSCW, and social networking systems use underlying parallel-distributed structures, but in many senses are more complicated than either parallel programming or distributed programming systems. Although nearly every college student in CS major (and every other college student) uses networked, distributed, multi-user software every day, e.g., instance messaging (IM), most current CS students are initially trained or exposed to programming single-threaded systems. They have little or no exposure to multi-threaded or multi-user systems, at least in the first 2 to 3 years of their undergraduate study, only limited to structured and hierarchical interactions on the Web (such as creating web pages). Arguably, tying together the envisionment of multi-threaded and multi-user systems can ultimately lead to a deeper understanding of the possibilities inherent in both.

Second, driven by the motivation to improve programming performance, much past and current research in the area of programmer cognition or programming psychology (Curtis 1984) focuses on how programmers use programming language features and programming tools. There is little research on teaching systems-level thinking or on the design and development processes in CS. Software engineering

researchers investigate how to structure the design and development of programs, but primarily with an eye towards increasing efficiency and reducing complexity, rather than pedagogy. System-level thinking, design, and development enter into this study because the movement of technology into the world means that underlying architectural assumptions have value-laden consequences. Designing systems that reflect end users' needs and system values is essential.

Third, the rapid development of multi-core technologies challenges our assumptions about how computers can coordinate faster and more efficiently. Parallelism goes beyond the computer scientists' world and becomes ubiquitous and pervasive as well. Designing successful connectivity in computer science not only means for parallel machines but also means for parallel human actions. Parallelism has had a broader meaning than that in parallel computing. The need to integrate parallelism early and broadly across the CS curriculum has been more urgent than ever. SIGCSE has called for pedagogical solutions for the past several years, e.g., "re-frame parallelism using simpler concepts, preferably ones that students can think about and understand in a natural way" (Adams, Ernst et al. 2010). This research believes that concentrating on the fundamental aspect of parallelism, coordination, can solve the pedagogical challenge.

Ultimately, although the need for exploration and development in the area of supporting radically distributed control, development of emergent meaning and process, and appropriation of coordination in socio-technical systems arose in the context of supporting face-to-face interaction, the concepts pertain to support for any pervasive and ubiquitous computing application that affects the coordination between people. To better understand the complex human needs, programmers should have the ability to imagine wider and richer possibilities to support complex human coordination and interactions through technology.

1.3 Research Question

This research answers the following central research question (RQ) and it is further divided into three sub questions.

"*What can we learn from the experience about how CS students conceptualize and approach parallel thinking?*"

The following three aspects (RQ1-RQ3) are examined and addressed:

RQ1: What are the challenges to students in designing parallel-distributed multi-user systems to support complex human interaction and coordination?

This research first asks how student learners (as developers) conceptualize the target system that they are going to design: (1) how they conceptualize the value of the target system, and whether they understand the rationale and spirit behind the system features they are about to build; (2) how they perceive the end user involvement and the ways that end users will interact with one another, both through and in relationship to the system. If they have difficulty, what can we do to help them to better understand these issues, and how we can accomplish this goal in the context of teaching at the university level?

RQ2: How do CS students conceptualize the coordination model through a specialized coordination language?

This question asks how learners master the specialized coordination language, TupleSpace, and use it to build the coordination model in a coordinative system: (1) what are the opportunities that TupleSpace model can provide to collaborative system design and development; and (2) what challenges programmers face envisioning the relationship between the coordination model and computational model, and how these challenges affect their imagination of building complex collaborative systems; (3) what is easy and what is hard in learning.

RQ3: What are the technological challenges to students in building computational model through explicit coordination modeling?

This question asks what technical challenges to CS students have in learning and developing target systems. This needs to be studied because coordination cannot be totally isolated from sequentiality because a system needs to be designed as a whole. Many challenges are general ones in coordinated system design even though the specialized coordination language is not used.

1.4 Research Approach

The approach to investigating programmer cognition in the area of parallel-distributed socio-technical system design and development was to study a semester-long experimental class, CS4984-Designing Distributed, Networked Activities for Learning. With the approval of the university's Institutional Review Board (IRB), the class was offered in the Department of Computer Science at Virginia Tech in Fall, 2005. The same class was taught again in 2006 as a normal pedagogical experiment, but not a part of this study. Data were collected primarily during the first class offering in 2005 with some supplemental data collected in 2006.

Fourteen students, who were 6 senior undergraduates and 8 first-year graduates, from the experimental class participated in the research. Participants received and signed informed consent letters. The students in the class were encouraged to participate in the research, but a separation was maintained between their grade in the class and their participation in the research. They were asked to participate in all phases of the design and implementation of a multi-user, co-located, and fine-grained coordinated interactive system.

The experimental class was a design-based research study (Hoadley 2005) in which multiple data collection methods were used to maximize methodological triangulation (Guion 2002) and ensure the noting of potentially important phenomenon. The researchers engaged in each class session to observe students' learning experiences and in-class design activities, and also traced their after-class progress. Research data include: observation notes and videos of every class session, semi-structured face-to-face interviews with open-ended questions conducted at the beginning and the end of the class, in-class exercise and assignment programming source code, documentation, and reports, email messages and class website board discussions, and questionnaires and surveys. To address the subjectivity and strengthen the credibility of the research, the researcher took various procedural safeguards such as triangulation of data sources, triangulation of methods, and inter-rater reliability checks with professional colleagues (see Sections 3.3, 3.7, and 3.8 for more details).

1.5 The Researcher and the Research

The primary researchers in this study were the author of this dissertation (who is a doctoral student in the Department of CS, at Virginia Tech) and her academic advisor (who is a professor in the same department and also has been a Human Computer Interaction (HCI) researcher and cognitive scientist in the field for many years). Another graduate student helped teach the class, but did not engage in gathering and analyzing the research data. In this dissertation, "researcher" or "I" (when the singular term is used) refers to the author of this dissertation, and "researchers" (when the plural term is used) or "we" mainly refers to the fellow researchers, mainly the author and her academic advisor. The plural term also refers to the researchers in SRI International, who were the research partners in the larger Tuple research project, according to the context.

The researchers gathered data both inside and outside the class throughout the semester. The professor was the instructor of the experimental class. The author acted as a graduate teaching assistant (GTA) to prepare the teaching materials and taught some class sessions during the semester as well.

The class was taught at a particular historical moment, in the context of a particular department and program. The findings speak to what we believe are grounding themes, but as in much design-based research and all qualitative research, should not be understood as claims that generalize across all conditions without careful thought.

1.6 Contributions

This research first contributes to understanding in HCI, Computer Supported Collaborative Work (CSCW), and Computer Supported Collaborative Learning (CSCL) by creating a better and deeper understanding of how computer programmers think about complex human interaction and coordination. As a result, the elements of system design that come from conviction can be disambiguated from those that arise from technical necessity. It also points out that the conceptualization of the complex human

coordination itself should be supported by software infrastructure in collaborative system design and development.

Second, this research work adds valuable facts and results to the research literature in programmer cognition and programming psychology. In the past thirty years, most research work in this area has focused on studying programming language and programming tools. There has been little study of programmers' conceptualization and software design. This research work especially studies programmers' capability of envisioning human coordination and interaction and how human interact with one another both through and in relationship to a collaborative system. In general, parallel-distributed computation is much harder to envision than single-threaded computation. When the nodes in a parallel-distributed network are people, in which the behavior at each node is indeterminate, the coordination must be understood to an even higher level than when the nodes in the network are processes and/or computers. Better understanding programmers' conceptualization in complex software design and development can help programmers design better systems.

Third, this research work contributes to computer education by promoting pedagogy of parallel thinking, which is an essential mindset for every computer scientist. It provides more insight into what it means to teach ubiquitous parallelism and proposes that *coordination,* one fundamental aspect of parallelism, needs to be taught as a focus. The research also draws attentions to several fundamental CS principles and knowledge that related in coordinated system development.

1.7 Dissertation Organization

This dissertation consists of seven chapters and appendices. This chapter, Introduction, introduces the problem statement and project background, purpose of study, research questions, general approach, and contributions.

Chapter 2, Literature Review, presents a detailed look at previous work that has influenced this research and provides background for the current research. Areas covered include a historical overview of programmer cognition, two example systems designed

for collaborative learning, coordination languages and the alternative technologies that support parallel-distributed computing, and teaching parallelism and coordination in CS.

Chapter 3, Research Methodology, describes the research design and the detailed procedures used to conduct this research. First, it introduces the qualitative research strategy that guides the study. Second, it describes the participants' recruitment, their background, and previous experiences. Third, it describes the research design overview. This chapter further addresses the detailed qualitative data collection procedure and analysis strategy and processes. Finally, it discusses the ethical considerations, issues of trustworthiness, and limitation of the research.

Chapter 4, Experimental Class and Students' Learning Experiences, gives an overall picture of what and how participants learned in the experimental class. It also presents the important events happened in the class through the semester. It mainly reviews the class from a pedagogical perspective.

Chapter 5, Findings, presents the major findings of this research, which mainly describes participants' experiences from a learning perspective. Three major findings emerge from this study. The first finding addresses the challenges to students in the design stage. Participants faced challenges in understanding end-user involvement and envisioning complex human coordination and interaction. Programmers' perception about human coordination and technology affect their design of useful and creative computational features. The second finding concerns the explicit coordination modeling process and the coordination language TupleSpace. It demonstrates the opportunity that explicit modeling coordination could bring into the design and development of coordinated systems. It also discusses the challenges associated with understanding the relationship between the coordination model and computational model in a coordinated system. It further discusses the challenges that learners encountered in learning TupleSpace. The third finding relates to the technological challenges associated in coordinated system development.

Chapter 6, Discussion, analyzes, interprets, and synthesizes the findings reported in the previous chapter. It takes into consideration of the literature on various research areas to understand programmer challenges and discusses the implications to multi-user

collaborative system design, user-centered value-based HCI, and computer science education. The challenges identified in Chapter 5 and analyzed in Chapter 6 follow three separate storylines, as illustrated in Table 1.

Chapter 7, Conclusion, draws the conclusion from this study and gives the researcher's suggestions for CS education, software infrastructure and programming tools design. It also includes the researcher's final reflection on this study.

Appendices include interview questions, demographic question forms, survey questions, and other related tables showing experimental class details and elaborating research findings, discussions, and implications.

Table 1. Three Principal Storylines Arising from The Data

Story line	Findings	Discussion	Implications and suggestions
Machine-centered and rigid engineering thinking constrain user-centered, value-based system design.	Section 5.1	Section 6.1.1 to 6.1.5	Section 6.1.6
Encourage parallel-thinking by Explicit Modeling of human coordination.	Section 5.1.3, 5.1.4, and 5.2	Section 6.2.1 and 6.2.2	Section 6.2.3 Section 6.3.2
Several fundamental concepts and programming paradigm in CS need more attention.	Section 5.3	Section 6.4.1 and 6.4.2	Section 6.4.3

Chapter 2

Literature Review

The purpose of this doctoral research was to explore the learning process of designing and developing parallel-distributed coordinative systems to support multi-user coordination with a specialized coordination language and further to encourage parallel thinking. The researcher sought to understand the cognitive challenges in the learning process. This chapter is a review of the current literature related to the research goal. The review was conducted throughout the data collection, analysis, synthesis, and conclusion phases. Four major areas of literature are discussed in Sections 2.1 through 2.4: (1) programmer cognition, (2) languages and models for coordination and parallel-distributed computing, (3) two example systems in collaborative learning, and (4) parallelism and coordination education in computer science,

First, the review of programmer cognition provides a historical overview of what has previously been studied in this area in the past 30 to 40 years. It also discusses the research methods used in this type of research work, which highlights the reason for choosing qualitative research methodology in this research. Second, the review of example systems in collaborative learning further discusses the target systems that this research attempts to design. Third, the review of various languages and models for coordination and concurrency highlights the benefits of using a coordination language to explicitly model coordination. Fourth, the review of current situation of teaching parallelism and coordination in computer science provides more background information about the urgent need for this and similar research work.

2.1 Programmer Cognition

Programming is a challenging human activity that usually involves the design of machine behavior that can assist human intellectual tasks. Program design plays a central role in the programming process, and is thought to be a very knowledge-intensive subtask of programming, with utilizing various kinds of knowledge (Hoc, Green et al. 1991). Programmer cognition has been defined as the mental activities of an individual programmer designing, building, testing, or comprehending code (Rosson 1996). The study of programming is in general useful to the development of the basic knowledge in cognitive filed. Programming, program design, and the research in programmer cognition and related areas are reviewed in the following sections.

2.1.1 Programming and Program Design

2.1.1.1 Definition of Programming

As technology rapidly changes, the definition of programming also changes accordingly. Three definitions are listed below from 1950s to 2000s.

> "The process of preparing a calculation for a machine can be broken down into two parts, 'programming' and 'coding'. Programming is the process of drawing up the schedule of the sequence of individual operations required to carry out the calculation." (Hartree 1950)
>
> "Programming is a human activity that is a great challenge, involving the design of machine behavior that can assist, and at times replace, humans in intellectual tasks." (Hoc, Green et al. 1991)
>
> "From a computer point of view, syntactically, a program is a text constructed according to a well-defined set of grammatical rules. Semantically, a program expresses a calculation or, to be more precise, a set of calculations. From a psychological point of view, we need to distinguish between the surface structure of a program and its deep structure, and its execution by the machine." (Detienne 2001)

From the above definitions, we can see a clear move from seeing programming as about the machine to seeing it as psychological activities. Hartree's 1950 definition offered clear definitions of programming, but without necessarily intending them as deep analyses of human activity. Hoc and Green's definition has started treating programming as a human activity. Detienne further clearly defines a program from the psychological perspective. This move also reflects the movement from low-level machine-oriented languages to high-level programmer-oriented languages. Regardless of how to define the meaning of programming, from the above definitions, we can tell programming is a very complex human activity. It usually involves the *high-level plan*, the *middle-level concepts*, and the *low-level details*. A programmer has to coordinate these levels, and make them in harmony with each other (Shneiderman 1986).

2.1.1.2 Program Design

Like other design activities (e.g., architectural design), program design has the special characteristic of being ill-defined, in the sense that some of the specifications of the problem are missing and part of solving the problem is to introduce new constraints. Another issue associated with program design is that there may be several acceptable solutions for the same problem, which cannot be easily evaluated according to any single criterion.

Programming activity usually involves designers' knowledge from at least two domains, the application (or problem) domain and the computing domain, between which designers establish a mapping. Programmers construct at least two types of mental models: a model of the problem and its solution in terms of application domain, and a model in terms of computing domain. Part of designers' work consists of passing from one model to the other. Depending on the features of design situations, the distance between these two models varies, which means in some cases building the bridge between two domains is quite complex. Successful programmers must master the syntax of a specific programming language, the semantics of computer programming, and the semantics of the task domain and reconcile the three (Shneiderman 1986).

2.1.2 Research in Programmer Cognition and Related Areas

In general, the purpose of studying computer programming in computer science is to design more appropriate programming languages and environments as well as more efficient training curricula (Hoc, Green et al. 1991).

The research in this area started in the 1970s (e.g., (Weinberg 1971), and see more examples in (Curtis, Soloway et al. 1986)) when computer scientists realized that the substantial costs and reliability of new developed programming tools were crucial to software development. It is one of the longest standing research concentrations within HCI. Research work in this area has explored various aspects of programming. It has made practical contributions to tool design and issues of practice for the whole range of programmers (from professional software engineers to casual programmers). It has also made theoretical contributions to the understanding of cognition in complex environments. Programming and software engineering include complex cognitive skills and serve as fertile ground for fundamental investigations of human cognition, contributing to the advancement of HCI in general (Irvin, Marian et al. 2001).

The early studies particularly tried to test whether certain language features could make programming easier, faster, or less prone to error. Sheil gave good examples of this kind of testing, such as programming notation (e.g., GOTO, data types), and programming practices (e.g., flowcharting, variable name) (Sheil 1981). These studies were generally conducted by computer scientists. The dominant research technique was to adapt the hypothesis testing methods in experimental psychology. Studies usually involved observation of small groups of subjects performing the same constrained, well-defined tasks. Statistical analyses tested the effect of one or more factors. For example, performance might be measured by some indicators such as "the time required completing a simple task" or "accuracy of response" from the experimental subjects.

The early studies were severely criticized later on (Brooks 1980; Sheil 1981; Curtis 1984). General criticisms emphasized that most of the studies lack a theoretical framework (Detienne 2001) and offer no explanatory model in terms of cognitive processes for how an effect works. In no case did they explain how and why a variable

has an effect on the performance. For example, Sheil argued that many of effects found in the studies of programming notation disappeared with practice or more experiences (Sheil 1981). Hoc also emphasized that most of studies were missing a psychological analysis and used only a superficial analysis of the task from a purely programming point of view (Hoc, Green et al. 1991). In addition, the theories that borrowed from psychology were badly used by some computer scientists. One remarkable example (Detienne 2001) is the inappropriate application of the limitations of short-term memory, which is restricted to 7 ± 2 isolated elements (Miller 1956), as a metric of cognitive load on programmers. This criteria was used to develop complexity metrics (McCabe 1976; Halstead 1977) to predict program comprehensibility. However, the metrics did not take "chunking" into account. Chunking is a process that extends the capacity of short-term memory by conceptually regrouping elementary items into single larger items of a different sort. Thus, a phone number may consist of ten numbers and therefore be difficult for most people to remember. However, if the area code is well known, such as 212 for New York, it will be remembered as a single unit rather than as three independent units. Experienced programmers, compared with novice programmers, tend to build larger chunks based on solution patterns.

As a result, in early 1980s, researchers such as Curtis, Soloway, Brooks and Sheil called for more theoretical methodology guided by the paradigm of cognitive science (Brooks 1980; Sheil 1981; Curtis 1984; Curtis, Soloway et al. 1986). Later research was conducted mainly by psychologists and ergonomists or by multidisciplinary teams including psychologists and computer scientists. Also, in the first workshop on Empirical Studies of Programmers in 1986, Curtis (Curtis 1986) and Soloway (Soloway 1986) raised questions over the generality and applicability of adopting the early research results to real programming because of the following facts: too many studies of tasks close to the activity of programming in "coding" phase, and too many studies of "novice student" programmers.

Since then, researchers have broadened their focus, and the research in this area has undergone a major change. First, the attention is no longer limited to novice programmers. Experienced programmers, including professional programmers, have

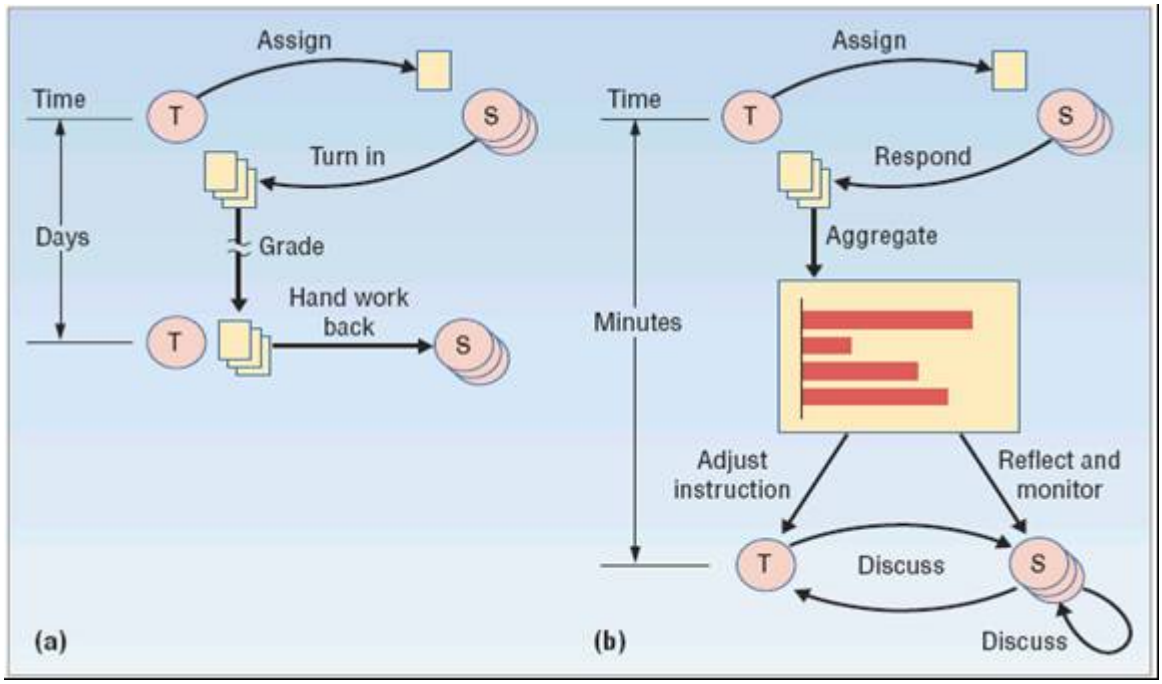
been studied. As a result, new themes like "the learning of new programming languages by experienced programmer" have been explored, addressing the mechanisms of knowledge transfer between languages and even between paradigms (Scholtz and Wiedenbeck 1990). Second, the attention is not limited to traditional programming languages. New languages and paradigms (e.g., spreadsheets (Karen, Curtis et al. 2000), declarative programming, object-oriented programming (Curtis 1986; Curtis 1995; Davies, Gilmore et al. 1995; Detienne 1995), new tools (e.g., visualization, tracers, editors), and new contexts (e.g., collaborative environments, multi-media, the Web) were also covered. Furthermore, interests moved from activities involving the manipulation of code (code producing, debugging, and the understanding of programs) to the stages before the production of code, such as, specification and design (Detienne 2001). Such research work allows the researchers in the field to address the reasoning mechanisms used in solving design programs. As a result, the theme of software reuse emerged. Another recent tread is to study programming as a collective activity, such as Extreme Programming (XP) (Beck 2000), instead of an individual activity.

2.2 Example Systems Designed for Collaborative Learning

To have a better understanding of the target systems that this research work concentrated on, two types of computer systems, i.e., classroom response systems and NetCalc (that supports networked collaborative activities) are demonstrated in the following sections. Their design goals and system limitations are also discussed.

2.2.1 Traditional Classroom Environment vs. Collaborative Classroom Environment

The following is a typical sequence in a traditional lecture-based classroom (see Figure 1a):



(a) Traditional Class Room (b) Networked Classroom Response System

Figure 1. Traditional classroom and networked classroom response system diagram. (used with permission of (Roschelle, Tatar et al. 2007))

(1) A teacher gives lecture while students listen.

(2) The teacher assigns homework.

(3) The students finish their assignments at home and then turn them in.

(4) The teacher grades the students' homework, and then gives it back to the students (usually in days).

(5) The students review the grade and ask questions, if needed.

In contrast, modern educational activities encourage learners to participate and engage in learning, and this approach is considered fundamental to modern Science, Technology, Engineering and Math (STEM) pedagogy (Roschelle, Schank et al. 2005). In a collaborative setting, groups of students and their teachers work together in more complex configurations than in traditional lecture-based classes. For example, students take different roles, contribute ideas, solve part of a larger problem, and work together as a team. Consequently they can learn more effectively from each other through coordination.

2.2.2 Example 1 — Classroom Response Systems

A classroom response system (Abrahamson 1999), also called Clicker, (see Figure 1b) is a set of hardware and software that can rapidly aggregate all students inputs anonymously for a multiple-choice question posed by their teacher. The system then presents a histogram that reveals students' understandings in minutes. The teacher can make instructional choices in response to the students' choices. This kind of systems becomes more and more popular in K-12 schools and universities across many STEM subjects because of its simplicity. However, this kind of technology is limited to a very narrow range of representational types and information-sharing topologies, that is, many-students-to-one-teacher. Therefore, they can only provide very limited coordination.

2.2.3 Example 2 — Match-My-Graph

In the past, different types of distributed, parallel, multi-user systems have been built to augment learning successfully. NetCalc is one such system. It runs on the Palm

OS. It extends the original desktop version, SimCalc (www.simcalc.umassd.edu) (Tatar, Roschelle et al. 2003; Vahey, Tatar et al. 2004; Vahey, Tatar et al. 2006). SimCalc is a powerful and flexible educational application that supports learning the mathematics of change and variation.

One coordinated activity in NetCalc is Match-My-Graph, shown in Figure 2. In this activity, students use a stylus to draw lines (called "functions") in a distance-time graph on the NetCalc screen to represent a car's motion. In the simplest form of Match-My-Graph, one student, the Grapher ("Green"), first uses a stylus to draw a line on the screen of his handheld device. The second student, the Matcher ("Purple"), then has the job of guessing what that line looks like. To do this, the Matcher draws a line on her handheld device screen, and then sends it to the Grapher via infrared beaming. The Grapher now has two lines displayed on his graph and the ability to animate two cars to enact the behaviors represented on the graph modeled on his screen. The Grapher must give a verbal hint to explain how the Matcher can make her line more like his; then the Matcher implements a change and sends the new guess to the Grapher. Back and forth, the Grapher and Matcher communicate in such an interactive way until a match between two lines is achieved.

Match-My-Graph is a good example that supports multi-user, parallel, and distributed interactions. This seemingly simple activity enables students' intensive focus on the subject and learning through interactive communication. Over 90% of their communications were on topic. Tremendous learning gains were associated with this and similar activities when used with 8th grade algebra learners (Vahey, Tatar et al. 2004). The benefits included not only pre-test or post-test gains on items associated with direct instruction, but also on qualitative problems from the Advanced Placement Calculus exam, which is a stand-in in the United States for college Calculus. Successful social interaction is an integrated bi-product of a central engagement in mathematical thought. This is very different from rewarding students with points when a desirable event occurs (Tatar, Lin et al. 2008).

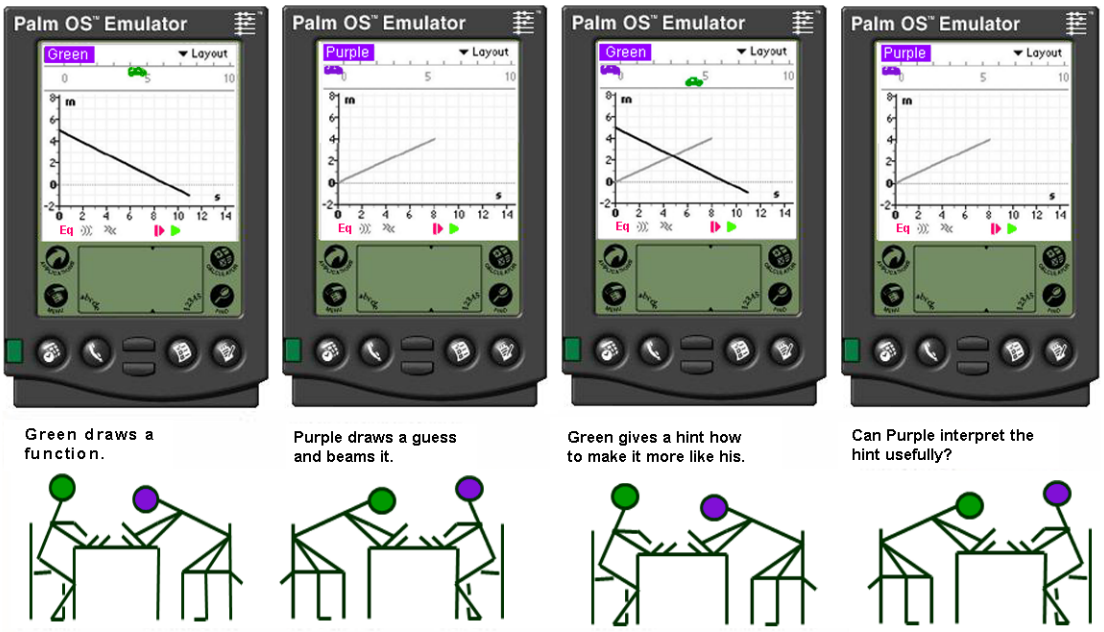


Figure 2. Match-My-Graph, an example of a powerful fine-grained coordinative activity. (used with permission of (Tatar, Roschelle et al. 2003))

Apparently, compared with a response classroom system, NetCalc provides simple yet richer interaction to enhance students' learning. While some other promising collaborative classroom systems (Roschelle, Schank et al. 2005) have good architecture from the educational perspective, they are hard to build from the technical perspective. Researchers in CSCL are calling for more powerful classroom network technology that can provide richer representations and interaction topologies (Abrahamson 1999; Abrahamson, Davidian et al. 2000).

2.3 Languages and Models for Coordination and Concurrency

Various specialized languages and models have been designed to help programmers model and utilize the important characteristics of parallel-distributed systems. The following sections describe the benefits of using explicit coordination modeling and the coordination language used in this research work. Other languages and models of coordination and concurrency are also reviewed from a high level. The purpose is not to compare or contrast these languages and models, but to point to some of affordances that can be provided by other approaches.

A coordination language is a linguistic embodiment of a coordination model, which facilitates the communication, synchronization, creation, and termination of different computational activities. It enables coordination to be modeled explicitly. In other words, programmers can use a coordination language to separate the coordinated activities from other computational ones to create two separate models: a coordination model and a computational model. A computational model contains the individual computational activities; and a coordination model contains the part responsible for communication and cooperation between those individual activities. The coordination model is the glue that binds separate computational activities into an ensemble (Gelernter and Carriero 1992). It can be viewed as a triple (E, L, M), where E represents the entities being coordinated, L represents the media used to coordinate the entities, and M represents the coordination rules. And the separation enables the *orthogonality* and

generality of the language, which contrasts the classical view of treating coordination as extensions of the sequential programming paradigm.

To exploit the full potential of massive parallel systems and provide a framework to enhance modularity and reuse of existing components, various coordinate languages have been designed. These languages differ in how they define the notion of coordination, what exactly is being coordinated (E), how coordination is achieved (L), and what the relevant rules for coordination (M) (Papadopoulos and Arbab 1998). Most of such systems roughly fall into two categories. The first category is data-oriented languages, which feature the notion of a shared memory space. In this category, a coordinator or coordinated process directly handles and examines data values through a shared space with primitives that provided by the language. A representative example is TupleSpace (originally called Linda) and its extensions; this is the kind of system we chose to use for our class. The second category is process-oriented (control-driven) languages or models, which function by means of observing state changes in processes and broadcasting events. Processes are treated as black boxes. They communicate with each other through connections, often called interfaces (input or output ports) or channels. The channels or interfaces are first-class objects in the programming environment. Communicating sequential processes (CSP) and actor-model-like systems are discussed as examples of this approach.

2.3.1 TupleSpace

2.3.1.1 Overview

The TupleSpace concept originated from the classic "Linda" model in parallel computing (Carriero and Gelernter 1989; Gelernter and Carriero 1992). It is a means of coordinating the computations of various processes by reading and writing structured data from and to a common space (Lehman, Cozzi et al. 2001; Roschelle, Schank et al. 2005). The common space is referred to as "TupleSpace" or "space" in this dissertation (also called "tuple space," "TupleSpace," "tuple-space," etc., in other literatures). The space is a globally shared, associatively addressed memory space. The basic element of a TupleSpace system is a tuple, which is a vector of typed values, or fields (structured data).

A TupleSpace is an unstructured collection of tuples. Tuples inside a space are associatively addressed via matching with other tuples, i.e. content matching. A tuple is created by a process and placed in the space via the *out()* primitive (from the perspective of the client, the tuple is pushed out). Tuples are read or removed by *read()* and *in()* primitives, which use a tuple template to match tuples in the space and return the first matching one. (Note that, because the space is unstructured, the choice among multiple matching tuples is arbitrary and implementation-dependent.) The "in" operation allows processes to coordinate such that one process uniquely works on each task.

TupleSpace provides a simple yet powerful mechanism for inter-process communication and synchronization — the communication between different processes takes place through the insertion and removal of tuples to/from the space (Bollella, Graham et al. 1999; Lehman, McLaughry et al. 1999). In other words, coordination happens by exchanging the resources of different processes through the space. This mechanism provides loosely-coupled properties that are different from other interaction mechanisms (Gregory 1982) such as shared variables, message passing, or remote operations.

- *Destination Uncoupling*: Unlike the message passing mechanism, in which the communication is usually partially anonymous — the sender always has to specify the receiver, the creator of a tuple requires no knowledge about the future use of that tuple or its destination.
- *Space Uncoupling* (a.k.a., Spatial Decoupling)*:* Unlike the shared variable mechanism, tuples are retrieved using an associative addressing scheme (i.e., contents matching). And multiple address-space-disjoint processes can access tuples in the same way.
- *Time Uncoupling* (a.k.a., Temporal Decoupling): Tuples have their own lifespan, which is independent of the processes that generate them (called Tuple Persistence (Gelernter and Bernstein 1985)), or any other processes that may read them. This enables time-disjoint processes to communicate seamlessly. In contrast, the tight temporal coupling in remote operation mechanism makes

processes or applications strongly interdependent — if the remote process hangs in a call, the calling process will also hang.

The creators of TupleSpace, Gelernter and Carriero, claimed that TupleSpace is specifically good at solving coordination problems (Gelernter and Carriero 1992). In recent years, the TupleSpace work expanded from pure parallel applications into the more diverse area of distributed computing. Figure 3 shows an example of three basic operations and the resource dissemination in TupleSpace. In this example, portable devices including pocket PCs, tablets, and laptops communicate with each other. The information transmitted between these devices is referred to as resources, which are defined as Tuples at different portable devices and represented by the yellow circle, blue triangle and red star in Figure 3.

2.3.1.2 Why TupleSpace?

TupleSpace was chosen to design collaborative systems in this research work in preference to other alternatives for several reasons. First, we wanted a language specialized for the explicit modeling of coordination. From the perspective of this research, another advantage of using such a coordination language resides in that it can help programmers focus on the problem or opportunity at hand without distraction from too many other details. Part of being able to master complex problems is to find a tractable form of problem representation. A dedicated language, like TupleSpace, is a candidate for such a representation. We choose TupleSpace in particular because it had been heavily used over a long time and represented a meaningful class of approaches in a simple tractable form. Linda was the first coordination language, whose purpose is solely for the coordination and communication requirements of an application among active agents (Gelernter and Carriero 1992). Gelernter and Carriero argued that generality enables the ability to cover the entire spectrum of concurrent activities: from multi-threaded applications executing on a single processor, through tightly-coupled, fine-grained parallel processing applications, to loosely-coupled, coarse-grained distributed applications.

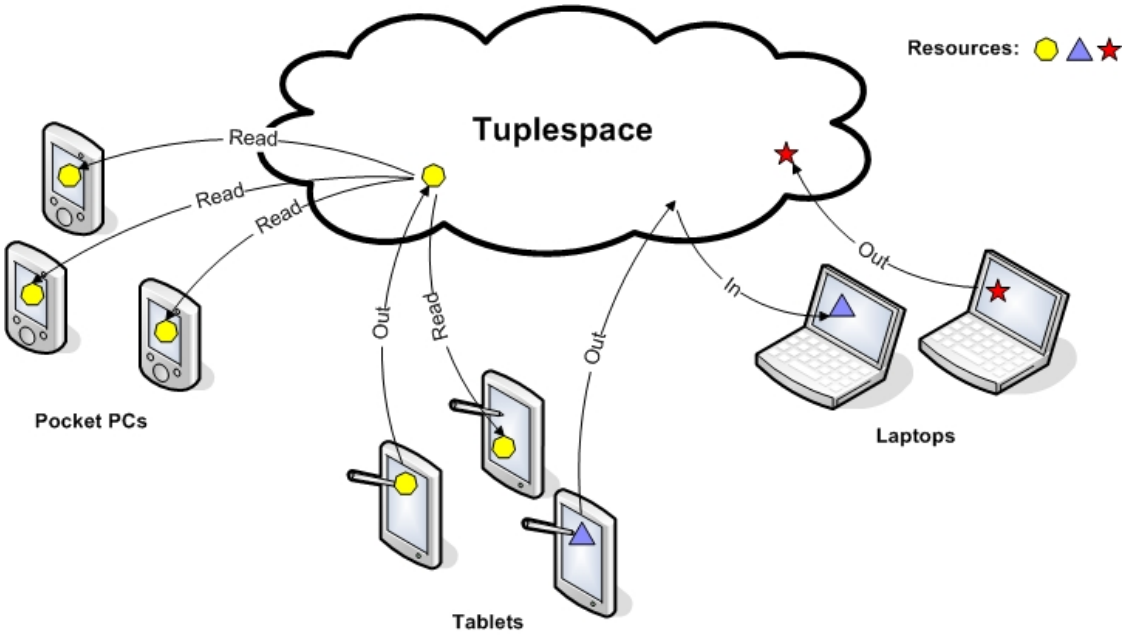


Figure 3. Basic operations and resource dissemination in TupleSpace

(1) A tablet writes a yellow circle in the form of tuple to the space via *out()*. Another tablet and three pocket PCs read it via *read()*. After these *read()* operations, the resource is still inside the space;
(2) A tablet writes a blue triangle to the space *via out()*, and then one laptop read and remove it via *in()*. The resource is not available in the space any more after the this operation;
(3) A laptop writes a red star to the space via *out()*. Because there is no other machine accesses it, the resource remains in the space.

Second, TupleSpace has been used to solve similar problems. We were particularly interested in the exchange of resource. As a coordination language, TupleSpace has been considered as a robust solution for coordinating the communication and synchronization between multiple processors or networked computers in the past two decades (Carriero and Gelernter 1988; Whiteside and Leichter 1988). In recent years, numerous research projects have been conducted in the distributed computing area using TupleSpace. In these projects, for example, TupleSpace was used to coordinate the communication between different devices, such as the devices showed in Figure 3 and a universal print solution (Lehman, Cozzi et al. 2001). Other applications used TupleSpace to coordinate the activities of mobile computational agents (Omicini and Zambonelli 1998; Murphy, Picco et al. 2003), such as mobile agents for Internet applications (Cabri, Leonardi et al. 2000). In addition, the use of TupleSpace was extended to a coordination infrastructure, Interactive Workspace, to support group work and solve problems collaboratively (Johanson and Fox 2004), which was a stab at utilizing TupleSpace in the HCI area.

Third, it is important to note that many other technologies provide mechanisms for coordination but either (1) are not languages for user-driven action or (2) do not support coordination as multi-directional endeavor. Although there are many other technologies available in distributed computing area, few of them are appropriate to support collaborative work in socio-technical systems, not to mention face-to-face interaction. For example, Web services technology is designed to support the generation of web pages on the client side and to respond requests and provide services on the server side. Remote Procedure Call (RPC) or similar implementations like Java Remote Method Invocation (RMI) are inappropriate and very complicated technologies for programmers to use. These low-level implications are discussed in detail in Section 2.3.4.

Fourth, TupleSpace can provide potential benefits for designing collaborative learning activities. The ultimate research goal of designing CSCL systems is to design and create socio-technical systems that go beyond planning to enable situated action (Tatar, Lin et al. 2008). The researchers want to develop systems that support embodied, focused, and social learning activities. TupleSpace has inherent features that give the

opportunity to achieve this. Table 2 shows six TupleSpace features which can easily support some of the special requirements of collaborative learning activities.

After a comprehensive review of TupleSpace candidate implementations, we chose TSpaces (Lehman, Cozzi et al. 2001), developed at IBM Almaden, as the final testing environment in this research work. TSpaces was selected for the following reasons. First, it is freely distributed, and has many peer users. Numerous commercial applications have been released both inside and outside IBM (Wyckoff, McLaughry et al. 1998; Lehman, McLaughry et al. 1999; Lehman, Cozzi et al. 2001). Second, it is relatively simpler than the competitive product from Sun, JavaSpaces (Sun Microsystems Inc. ; Freeman, Hupfer et al. 1999; Bishop and Warren 2002). The basic methods provided in TSpaces are as simple as those in the classical Linda (In TSpaces, *out* operation defined as *write* and *in* operation as *take*), which is good for novice programmers to start with. Third, TSpaces provides powerful functions and features that offer a great potential for further extension (e.g., notification function in TSpaces mentioned in Table 2).

Table 2. TupleSpace Features and the Potential Benefits

TupleSpace Features	Benefits in Design
Content based matching − enables Space Uncoupling	• Makes it easy to aggregate students' contributions (Tatar, Roschelle et al. 2003), and categorize information. • Makes it easy to form and reform team (classroom groups need to be reformed very often). • Makes it easy to support "look" function (Kim 2007).
Time Uncoupling (because of Generative Communications (Gelernter and Bernstein 1985) and Tuple Persistence)	Enables asynchronous interactions. • latecomer problem • disconnected mode
Destination Uncoupling	Enables anonymity in the system (senders and receivers can exchange data without the mutual knowledge of each other's location).
Space Uncoupling	• Enables distribution of unique information. • Enables cooperative work on shared data.
Robust and reliable system implementation	Makes teachers feel confident of using new technologies. With limited class time, teachers cannot afford any error or delay caused by technical failures and instabilities.
Notification (a special feature in TSpaces)	Enables synchronous interactions, e.g., supports instant notification between teacher-students, students-students.
Platform independence (TSpaces)	Supports across multiple platforms, e.g., desktop, laptop, tablet, and pocket PC. Reduces efforts to develop applications for multiple platforms.

2.3.2 Other Models and Languages

2.3.2.1 Processes-oriented Coordination Languages and Models

There are other languages that allow the explicit modeling of coordination. In addition to the data-oriented approach in which coordinators directly manipulate data through the space, instantiations of the process-oriented category treat processes (coordinators) as black boxes. Data is hidden inside the process. The communication between processes happens through explicit message passing, between a sender and a receiver. The coordination medium is stream- or channel-like connections (often referred to as input and output ports) between two processes. Messaging passing can be either synchronous two-way anonymous communication (i.e., rendezvous), as in communicating sequential processes (CSP) model (Hoare 1978); or asynchronous one-way communication, as in the actor model (Hewitt, Bishop et al. 1973). In general, the connection is point-to-point, but some models also support limited broadcasting, e.g., MANIFOLD supports multiple port-to-port stream connections. In these models, streams are first class citizens and able to hold data within themselves while connections may be dropped and get reconnected between different coordinators. Stream connections can be implemented using data bus rather than point-to-point connections between ports. In addition to using ports, processes can send out control messages or events to notify their current state or state changes to other interested processes. There are many other languages and models in this category in one way or another built the above CSP, such as PCL (Sommerville and Dean 1996) and pi-calculus (Milner, Parrow et al. 1992). These models emphasize the visualization or imagination of the connections but the channels are abstract. The programmer's attention is drawn to communication as a network rather than operations for a purpose. Resource exchange or use is not of direct concern to the programmer. The imagery here is similar to the tangle of connections on an old-fashioned telephone exchange.

2.3.2.2 Petri Nets

Petri nets are graphical and mathematical modeling tools for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic (MURATA 1989).

The analysis of a Petri net can reveal the information about the events and stage of the modeled system. A Petri net consists of two kinds of nodes, places and transitions, where arcs are either from a place to a transition or vice versa. In modeling, places represent conditions and transitions represent events. A transition (an event) has a certain number of input and output places representing the pre-conditions and post-conditions of the event. The presence of a token in a place indicates a configuration of the net. The value of a token (a discrete number of marks) on a place indicates a number of items or resources are available at that place. Two transitions are concurrent if they are causally independent, i.e., one transition may fire before or after or in parallel with the other. A Petri net focuses on representing the temporal sequence of the concurrent events in a system, but not explicitly represents the coordination of data or processes in the system.

2.3.2.3 Dataflow Concurrency

The dataflow programming model (Agerwala and Arvind 1982) promotes the value of data to become the main concept behind any program. All program data, whose input values have been previously computed, can be used simultaneously. The operations between the data are considered to be of secondary importance. The Petri net model focuses on a series of operations, emphasizing the behavior of the operations themselves. Dataflow leaves this synchronization implicit. Except for data dependencies in the program, dataflow instructions are constructed to avoid constraints on sequencing. Dataflow languages model coordination in the sense of prioritizing the importance of displaying dependencies in the data. However, they do not allow the programmer to model events or states. From this perspective, they can be considered not as models of coordination but as models of coordinative effects. Indeed, they are often called "concurrency" languages rather than coordination languages.

2.3.3 Other Alternatives in Distributed Computing Programming

Part of the argument for a coordination language is similar to the general argument for high-level languages: that the clear representation of a problem is a key

component in solving it. Several available technologies focusing on low-level implementation in distributed computing are also elaborated.

2.3.3.1 Socket Programming

A socket is one end-point of a two-way communication link between two programs running on a network. Socket programming has two major drawbacks from the perspective of the programmer interested in producing high-level experimentation. First, it allows control only over very low level network communication. Although the way to use sockets differs in each programming language, usually programmers have to take care of every detail, including the management of creating a socket, using it, and closing it. Unless absolutely necessary, programmers will avoid using sockets directly. Instead, alternative higher-level network programming technologies are preferred when available. Second, in fact, sockets are used constantly by processes for different purposes, all described as "machine communication"; however these complex interactions are not represented explicitly. Basic socket programming (client/server structure) usually implements a message-passing mechanism because it involves sending a message through a channel built between two sockets (one for client and the other for server). This kind of communication is tightly coupled, both temporally and spatially. It is possible to implement more complex interaction schemes using sockets.

2.3.3.2 Web-based Programming

Web-based programming generally involves writing source code that generates web pages on a client side, as well as providing services and appropriate responses on a web-server side. Although web-based programming can support distributed communication at a certain level through the Web, it is not fundamentally designed to support the fine-grained coordination in which the unit of exchange is much smaller than a document. Further, the benefits of web-based programming are less salient in this context than they are in many others. A high-level programming language is required to implement coordinated features such as aggregation of data across users. Simple languages such as pure HTML or web page scripts (e.g., JavaScript, or AJAX) do not suffice because these languages do not directly support communication and coordination between people. Additionally, constant Internet connection of all computers is not

always realistic in a classroom setting, nor can we expect students and teachers to have a web browser open all the time during a class. As a matter of fact, many teachers and school systems would object to the potential distraction of using a web browser in class.

2.3.3.3 Remote Procedure Call (RPC) and Remote Method Invocation (RMI) Programming

RPC lets a computer program to execute a subroutine or procedure in another address space (commonly on another computer on a shared network) without a programmer explicitly coding the details for this remote interaction. That is, a programmer would write essentially the same code whether the subroutine is local to the executing program, or remote. In the context of Object-Oriented Programming (OOP), RPC may be referred to as RMI, e.g., Java RMI, CORBA, and Microsoft DCOM. For example, Java RMI enables programmers to create distributed Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts (Sun Microsystems Inc.). RPC or RMI technology has some disadvantages compared with TupleSpace.

First, compared with the loosely-coupled property of TupleSpace, most of the RPC/RMI implementations are fundamentally coupled both temporally and spatially. This is because RPC or RMI mimics a function call and all communication is transient and synchronous. The calling application is blocked until the result is returned, and the calls cannot work until an appropriate remote target is found. This tight coupling suggests that RPC/RMI applications are strongly interdependent — if the remote application hangs in a call, the corresponding calling application will also hang. For some CORBA implementations, applications do not have to wait for each other's acknowledgement (Eugster, Felber et al. 2003). Second, in RPC/RMI applications, the system cannot perform most routing types unless client applications implement the logic by themselves (e.g., broadcast, multicast). RPC/RMI, therefore, is relatively non-expressive and complicated (Johanson and Fox 2004). Third, by using messages via a coordination component like TupleSpace, rather than using a direct remote procedure call, there is an opportunity for adding translators to the communication stream that fix protocol, data format, and data schema disparities (Lehman, Cozzi et al. 2001).

2.3.3.4 Publish/Subscribe Based Programming

Publish/Subscribe is an asynchronous messaging paradigm in which subscribers (receivers) express interest in certain messages without the knowledge of publishers (senders), whereas publishers send their messages to whomever has interest and a subscription, but not specific receivers (Eugster, Felber et al. 2003). This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology. However, this kind of systems has some inherent limitations (Johanson and Fox 2004). First, these systems provide no time uncoupling. An application must be running and subscribed to at the time of message generation to receive a copy of the message. Second, Publish/Subscribe is designed primarily for broadcast and multicast, but not for other routing patterns. It is not possible to implement Anycast in Publish/Subscribe-based systems because there is no way to squelch a message once one of the receivers acknowledges receipt. In contrast, TupleSpace can support Anycast very easily by performing a "in/take" operation that will prevent others from receiving the same tuple.

In summary, to envision the coordination between end users and between end users and machines is quite complicated, especially for programmers who have little experience in this area. The alternatives discussed in this section are not solely designed to support user-driven action or multi-directional coordination. Therefore programmers need to first master how to simulate these communication, which obviously increases programming difficulty.

2.4 Teaching Parallelism and Coordination in Computer Science

2.4.1 Teaching Parallelism in CS Education

Parallelism, though not a new concept, has attracted wide attention in the field as computing world is moving toward multi-core technologies fast. In the 1960s and 1970s, extensively research work had been done as a means to high-performance computing, and it had been neglected for a quarter century (Denning and Dennis 2010).

In general, programming for parallelism is more difficult than sequential programming because programmers need to manage concurrency issues. Parallel and distributed computing have been advanced and elective topics in CS, which are usually taught after sequential programming has been well established in learning. Most pedagogical attempts to improve teaching of parallel programming (Lin and Tatar 2011) were to create new curriculum to make parallel programming techniques and algorithms easier to understand and learn, or to develop new tools (McDonald 1992; Robbins 2003; Ben-Ari 2004) to reduce difficulties in programming. For example, Robbins (Robbins 2003) reported using remote logging to teach concurrency to help students visualize, understand, and debug concurrent programs.

The commonsense computing project across multiple universities studied the growth of intuition in concurrency (Lewandowski, Bouvier et al. 2010) among 66 CS college students. The study found that all of their "Students began their first computing course with essentially the same level of intuition as they began their first course involving concurrency." This suggests that prior CS curricula have not helped the students advance in gaining related concepts or prepare for the advanced level courses.

Most recently, the pedagogical trend has been to integrate parallelism into the entire CS curriculum (Graham 2007; Ernst and Stevenson 2008; Ernst, Wittman et al. 2009), from the entry-level courses. The goal is to increase the exposure of parallelism to CS students and enable them to think parallelism more naturally. However, the main problems were "what essentials in parallelism should be taught?" and "how to integrate these essentials into the CS curriculum, early and broadly?" Ernst and Stevenson (Ernst and Stevenson 2008) attempted to increase students' exposure to concurrency early and often in their curriculum through multi-threading programming. Ernst and Wittman (Ernst, Wittman et al. (2009) also reported integration of concurrency into freshmen courses through threaded programming. Their challenge was how to bootstrap concurrent programming with students having early, weak programming knowledge and skills. Last year, a SIGCSE'10's (an international conference of ACM Special Interest Group on Computer Science Education in 2010) panel discussion called for ideas and

solutions to "re-frame parallelism using simpler concepts, preferably ones that students can think about and understand in a natural way" (Adams, Ernst et al. 2010).

2.4.2 Coordination in CS Education

Denning (Denning 2004) in his famous talk "Great Principles in Computing Curricula" identified coordination as one of the five "windows" (i.e., computation, communication, coordination, automation, and recollection) of computing mechanics into the realm of computing. He referred computing mechanics as those deal with the structure and operation of computations. He thought the mechanics is "with the universally occurring phenomena that appear in computational processes and hardware and software components." Every core technology area (e.g., algorithm, data structures, human computer interaction, databases) expresses all five windows in its own way. For example, he thought "pervasive and mobile computing is a form of distributed computing, which is an aspect of both computation and coordination." He defined coordination in computing term as "multiple entities cooperating toward a single result." In this context, *communication,* a relevant concept to coordination, was defined as "sending messages from one point to another."

The coordination issues in CS are largely involved in computer-computer (e.g., synchronizations, races, deadlock, serializability), human-to-computer (e.g., input/output interface), and human-to-human (e.g., workflows as supported by communicating computers). Coordination is central to many computing courses, such as operating systems and parallel programming. However, current CS curricula do not teach coordination explicitly and address coordinative issues directly.

Chapter 3

Research Methodology

The purpose of this research work was to investigate how CS students learn the design and development of parallel-distributed socio-technical systems through a specialized coordination language and further approach parallel thinking. The research went through five major steps, as shown in Figure 4: (1) study design, (2) data collection and organization, (3) data analysis, (4) data synthesis and interpretation, and (5) conclusion drawing. A qualitative design-based research (DBR) (Brown 1992; Hoadley 2005) method was used to design the experimental class. The study followed a pedagogical sequence in the spirit of Bransford's notion of contrasting cases (Bransford, Franks et al. 1989), that starts by raising questions that it does not fully answer, leaving a conceptual gap for later instruction to fill in. Typical qualitative data gathering methods were used: participation in the setting, direct observation, in depth interviews, and analysis of documents and materials. The data analysis process was guided by Grounded Theory (GT), an inductive qualitative research method. Rather than starting with a hypothesis and trying to prove it, the researcher begins by collecting data in the field and lets the theory emerge or emanate from the data (Dunican 2005); the theory is *grounded* in the data. The theory developed from the data can then be tested by further research. This method was used as a popular approach for people exploring a new area of research (Salinger, Plonka et al. 2007).

This chapter describes the discussions of these areas: rational for qualitative research approach, description of the research participants, overview of research design, methods of data collection, analysis and synthesis of data, ethical considerations, issues of trustworthiness, and limitations of the study. The chapter ends with a brief summary.

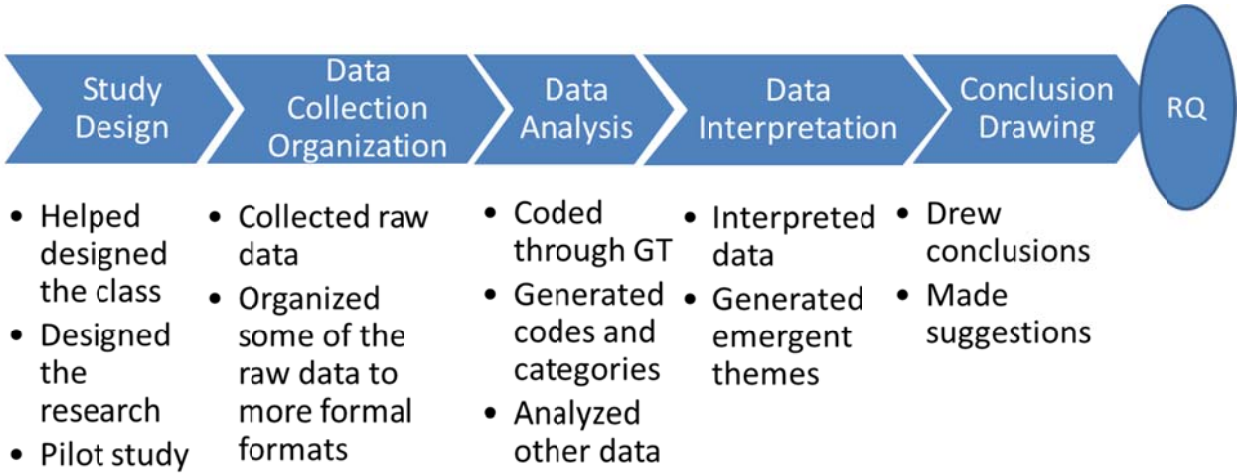


Figure 4. Qualitative research procedure in this doctoral work.

3.1 Rationale for Qualitative Research Strategy

Qualitative research is a field of inquiry that "cultivates the most useful of all human capacities: the capacity to learn" (Patton 2002). It involves in-depth understanding of human behavior and the reasons that govern human behavior. It is conducted in a natural settings rather than a controlled one (in contrast with quantitative research). It assumes that humans use what they see, hear, and feel to make meaning of social phenomena, and it relies on a variety of data-gathering techniques (Rallis, Rallis et al. 2003). In contrast, quantitative research is "a formal, objective, systematic process in which numerical data are utilized to obtain information about the world" (Burns, Grove et al. 2004). The qualitative method was chosen rather than the quantitative method in this research for these major reasons:

First, human behavior may be significantly influenced by the setting in which it occurs, and when this is the case, one must study that behavior in situations. In our case, the physical and social setting, including the classroom environment, the structure of undergraduate education, the internalized notions of norms, traditions, roles, and values that students and professors bring with them, are all crucial contextual variables. Research must be conducted in a setting where all the contextual variables are operating (Marshall and Rossman 2006). What we are trying to study, i.e., the conceptualization of software design and implementation, is a highly complex and long-term learning task and process. This process happened both inside and outside classroom during a semester.

Second, controlled experiments, standardized surveys, and close-ended questionnaires are common quantitative research methods. As suggested in the literature (Lewis 1982), the precision and rigidity of controlled experimentation may be inappropriate when a researcher is exploring new domains in which the independent and dependent variables are unclear. Standardized surveys and close-ended questionnaires are likewise inappropriate and insufficient for the purpose of this study. Observational studies, which are a typical qualitative research method, are applicable when the tasks are so complex and varied that setting precise goals for subjects would invalidate the experiment (Shneiderman 1986).

Third, the researchers were not, in this particular case, only interested in the average or general experience. We wanted to hear how computer science students describe their

experiences in their own voices. Generally speaking, qualitative methods typically produce a wealth of detailed information about a much smaller number of people and cases. This increases the depth of understanding of the cases and situations. An in-depth analysis of each individual would give us a deeper and better understanding of the variety of computer programmers' experiences.

Fourth, in general, experimental and quasi-experimental methods in quantitative research require comparison or control groups. In the experimental class offered in Fall 2005, we did not have meaningful control. We would not have tried to teach so much in a single semester without a tool, such as TupleSpace, which made the material tractable. Correlational or survey methods do not require comparison or control groups, but do require that people are already engaging in the practice to be studied.

In summary, qualitative methods increase the depth of understanding of the cases and situations studied (Patton 2002). They provide more opportunities to explore in this specific research area and gain rich and thick information that are used to answer the research questions.

3.2 Participants

3.2.1 Participants Selection: Students Participants vs. Professional Participants

This research chose student participants in preference to professional participants for these reasons:

First, the researchers thought that TupleSpace had great potential for developing collaborative learning systems (and potentially any collaborative systems). Its properties can support special educational and social requirements (discussed in Section 2.3.2), as well as provide a relatively easy way to learn and use. If senior undergraduate students in the computer science major, who are considered as intermediate to advanced programmers, are able to use TupleSpace, professional programmers would not be likely to have difficulty using it.

Second, the systems we were hoping to create in the experimental class, i.e., the distributed, fine-grained, coordinated, networked, multi-user systems, were not common in the industrial practice. College students were thought to be more appropriate for this purpose.

3.2.2 Participants Recruitment

The class was advertised in the normal way that an experimental class is announced in the Department of Computer Science, including a short online summary, the posting of flyers in key campus locations, and the creation of a class web site.

Our original plan was to create cross-disciplinary teams with pedagogy designers drawn from the School of Education because we also wanted to study how the education students (as pedagogy designers) and computer science students (as project developers and programmers) worked together to design collaborative learning systems. However, because of coordination issues, this class was not announced in the School of Education until it was too late to register, so no education students enrolled.

3.2.3 Participants Overall Information

Six senior undergraduate and eight first-year graduate students in the computer science program enrolled in this class and agreed to participate in our research. They received and signed informed-consent forms. A participant ID, e.g., P01-P14, was assigned to each participant randomly to protect their confidentiality. During the semester, they were encouraged to participate in research both inside and outside the classroom. However, a separation was maintained between their grades in the class and their participation in the research.

Participants included 13 male and 1 female students[1]. Among them, there were four Whites, one African-American, eight Asians (including non-resident aliens), and one Asian-American, aged from 21 to 33. Table 16 (Participant Demographic Information and Self-reported Prior Experiences) in Appendix F shows detailed information about each participant. The following section summarizes participants' previous experiences.

[1] In 2005 Computer Science Department Enrollment, the gender ratio was about 9:1 and the ethnical ratio was 56.71% Whites, 3.96% African-Americans, 9.91% Asians, 7.62% non-reported, and 19.36% non-resident aliens.

3.2.4 Participants' Previous Experiences

Participants' previous experiences were gathered at the beginning of the class through a demographic survey, questionnaires, and pre-class interviews.

3.2.4.1 General Programming Experience

All participants had more than three years programming experience at the college level; five participants started programming in high school or earlier. The average length of their programming experience was 6.5 years. Eight participants reported that they had held an internship or permanent job in the past; most jobs were programming-related in industry or in a university. Most of them were C++ programmers with more than three years' programming experience. They had different levels of proficiency in Java (See Table 3). Most of the non-Java programmers could read simple Java code because of the similarity between C++ and Java and their previous knowledge in Object-Oriented Programming (OOP). Other acquired programming languages reported by participants included C, C#, PHP, Visual Basic/Basic, HTML, JavaScript, and Scheme.

3.2.4.2 Network and Multi-user Programming Experience

Eight participants had taken network-related classes in the past, either at the undergraduate or graduate level, e.g., Computer Network Architecture and Programming. In those classes, participants had learned basic network architecture and principles such as the OSI protocol model, TCP/IP, and UDP. Some participants had had opportunities to use those basic concepts to build simple applications such as implementing a FTP client and server or a chat program using socket programming in C or Java. Among 14 participants, three claimed that they did not have networking programming experience either inside or outside class. However, it is believed that almost every participant had written a static web page in the past. Four participants had web programming experience generating dynamic web pages in PHP/JSP through a class or a job (a summer internship or a permanent job). Some participants reported they had experience with accessing databases through a network.

Table 3. Participants' Previous Experiences (Participants n =14)

Previous experience	Participants	Percentage
Java programming experiences		
3–5 years coding	2	14.29%
1–2 years coding	4	28.57%
No coding or only read simple code	8	57.14%
Networking class (e.g., Computer Network Architecture)	8	57.14%
HCI class (e.g., Intro to HCI)	10	71.43%
Sociology class (e.g., Intro to Sociology)	3	21.42%
Psychology class (e.g., Cognitive class)	5	35.71%
Education class (e.g., Tools for Online Education)	2	14.29%

3.2.4.3 HCI and GUI Design Experience

As shown in Table 3, ten participants had taken at least one HCI-related class in the past (e.g., Introduction to Human Computer Interaction), which meant most of them had entry-level human computer interaction design knowledge (e.g., scenario based design) or programming experience in GUI design. From the pre-class interviews, we knew that some participants programmed GUI in the Windows® operating system using Visual C++, Visual Basic, or Java. However, only the two experienced Java programmers had actually implemented user interfaces in Java using Swing. Five participants could read Swing source code; others were not familiar with Java Swing.

3.2.4.4 Prior Teamwork Experience

All participants had at least one experience working in a group with other people. Most people had positive teamwork experiences and they thought teamwork was really important for a successful group project. P08 said,

> I think teamwork is really important especially we have a team that everybody cooperates and everybody is giving ideas…. I always enjoyed doing teamwork…. In teams, your team member might think of ideas that you didn't think of or let's say you did come up with one but they may be able to elaborate even more in it. (P08, pre-class interview, September 01, 2005)

Several participants mentioned that if they had a good personal relationship with their partner(s), they would feel more comfortable working together as a team and usually produced better outcomes. Not surprisingly, some participants, such as P01, talked about their negative teamwork experiences: "In my other previous classes, I had many troubles working within groups. […] I couldn't really express myself. And I felt like lots of my ideas were like ignored kind of." (P01, pre-class interview, August 26, 2005)

3.2.4.5 Other Related Experiences

Some participants took Sociology, Psychology, and Education classes in the past (See the number and ratio in Table 3). As for teaching experience, two participants taught programming languages at high school or college as a full-time or part-time job. Others had only casual

experiences as tutors or as teaching assistants. Therefore, most participants did not have much direct experience in education.

Four participants had heard of TupleSpace in the past: one from an entry-level CS class, two from one of the researchers in this research work, and one from an operating system class. Therefore, their knowledge of TupleSpace was only limited to the term itself or the definition.

3.2.4.6 Summary of Participants' Experiences

According to their experiences described above, participants were considered as at least intermediate, if not experienced programmers, but novice interaction designers and programmers. First, participants had been or are being trained through a formal CS undergraduate or graduate curriculum and thus had a few years programming experience as CS major students. Furthermore, some of them had work experience solving real-world problems in software development through summer or permanent jobs either in industry or at a university. Therefore, they were intermediate to advanced software developers. Second, however, except for web programming, almost none of them had chances to design or implement a networked, multi-user application for a face-to-face environment before the class. Participants, who created static web pages only, did not have any opportunity to think about interactions among users. From this point of view, participants should be considered as novice designers and programmers. We saw this as a disadvantage as well as an opportunity. It was a disadvantage in that they would not be able to compare and contrast TupleSpace with other alternatives, and they did not have much knowledge about designing for multi-user interactions. It was an opportunity to be introduced to a new way of designing coordination directly, which was a good chance for us to examine the ways of teaching TupleSpace to novice programmers the development of socio-technical systems in a face-to-face environment.

3.3 Research Design Overview

3.3.1 Study Overview

The approach to investigating student programmers' cognition in the area of parallel-distributed socio-technical systems design and development was to study a semester-long

experimental class taught in the Department of Computer Science at Virginia Polytechnic Institute and State University (Virginia Tech). The class, *CS4984 — Designing Distributed, Networked Handheld Activities for Learning*, was offered in Fall 2005. The same class was taught again in 2006 as a normal pedagogical experiment, but was not a part of this study. The students in the class were asked to participate in all phases of the design and implementation of a collaborative game-like activity, e.g., classroom pedagogical activities using TupleSpace. The class was run in a highly participatory way, so the students were encouraged to talk about their thoughts and ask questions in the class. The students are often referred to as "participants" or "student programmers" in this dissertation.

The researchers included the author of this dissertation (refer to as "researcher" or "I" in this dissertation) and the author's doctoral advisor. Another graduate student helped teach the class, but did not engage in gathering and analyzing the research data. The researchers engaged both inside and outside the class throughout the semester to gather the essential data for research analysis. The data were collected primarily during the first class offering in 2005, with some supplemental data collected in 2006.

The following list summarizes the steps used to carry out this research. Following this list is a more in-depth discussion of each step.

- Preceding the actual collection of data, a selected review of the literature was conducted to study the contributions of other researchers in programmer cognition (a.k.a., programming psychology), distributed computing, collaborative learning, and TupleSpace uses and applications. More literature related to parallelism and parallel programming education in Computer Science was later reviewed in the data analysis stage.
- The researchers acquired approval from the Virginia Tech IRB to proceed with the research. The IRB approval process involved outlining all procedures and processes needed to ensure adherence to standards for the study of human subjects, including participant confidentiality and informed consent. The extension of IRB approval was obtained throughout the data analysis process.
- As the pilot study of the experimental class, a programming workshop (Section 3.3.2) was held in summer 2005. By focusing on more-experienced programmers, this workshop gave

us a chance to target the potential issues in presenting the core TupleSpace materials. Other purposes also included the test on interview protocols and the refinement of research procedures before the experimental class.

- The class was advertised in the department of computer science and education. The students who came to the first class were encouraged to participate in the study, as discussed in Section 3.2.2.
- Research data were collected throughout the semester, including a demographic survey, a semi-structured pre-class interview, video recording in each class, close observation in class and programming help sessions, programming source code collection, a semi-structured post-class interview, a post-class survey, and other related documents such as emails and class web discussions.
- Research data were organized, coded, and synthesized through a serial of qualitative analysis processes guided by Grounded Theory.
- Research findings and results are presented in this dissertation and were published as four peer-reviewed conference papers and one book chapter since 2005 (See Appendix J).

3.3.2 Pilot Study

Before the Fall 2005 experimental class, a three-week (July 11–29, 2005) summer programming workshop, considered as the pilot study of our research, was conducted at the research partner SRI International in Palo Alto, California.

Participants included five international graduate students, who originally came from Korea, China, Jamaica, South Africa, and Chile, and one professor in the computer science major from different universities. Of five graduate students, two came from Virginia Tech, two from Norfolk State University, and one from Pontificia Universidad Catolica de Chile. Three students were doctoral students at the middle or later stage of their Ph.D. programs; one was a second-year Masters student; and one was a first-year Masters student. The sixth participant was an assistant professor from National Hsinchu University of Education in Taiwan. Of the six participants, four participants were male and two were female.

These participants were considered intermediate-to-experienced programmers. Before the workshop, they had learned TupleSpace concepts more or less by themselves, and most of them had previous research experience in CSCL and HCI. They were able to use IBM TSpaces to program soon after the workshop began. In addition, many of them had their own projects in mind before they came to the workshop, and they were interested in designing and implementing those projects using TupleSpace technology during the three-week workshop. Some of these projects were later used as examples in the Fall experimental class. The interesting projects implemented in the workshop included:

- K-Map, a program which helps students to learn how to minimize Boolean expressions in digital design by sharing K-map drawings of multiple users.
- Match-My-Graph in SimCalc, a program which helps students to learn the mathematics of change and variation using a cooperative game. Students take distinct roles and work together to achieve success during the game.
- Rank-Task, a collaborative tool used in physics class to encourage student participation.
- Collaborative Draw-My-Molecule, a program which helps students to become more familiar with multiple representations of a molecule by generating chemistry representations using small-group and whole-class discussion.

Participants signed informed consent forms for this pilot study. During the workshop, they submitted daily reflection logs to report their design and learning progress. They were also required to submit daily programming code through a Concurrent Versions System (CVS). They participated in the semi-structured face-to-face open-ended interviews at the beginning and the end of the workshop. Participants talked about their expectations of the TupleSpace workshop and their previous programming projects at the first interview and reviewed their learning and programming processes with TupleSpace in this workshop at the second interview. Certain design discussion sessions that related to the design decisions of classroom interactions by using TupleSpace were video-taped.

The pilot study gave us a chance to target the potential issues in teaching TupleSpace. It also helped to refine the proposed research methods and procedures used in the later experimental class. The data collection procedure and methods were adapted to meet the new

requirements in the Fall 2005 class based on the feedback from the pilot study. First, the semi-structured interview protocol was modified to address the differences between the participants in the Summer workshop and those in the Fall semester class. For example, most of participants in the Fall experimental class had never heard of TupleSpace before; thus, we could not ask their expectations about using TupleSpace. Instead, we asked about their expectations of this class. Second, all class sessions were video-taped to collect more detailed data. Third, because it was very hard to keep students submitting their learning reflections every day, the daily reflection logs were changed to reflection questionnaires that were submitted after each major programming homework and project assignment. Also, the email messages and board discussion questions, which reflected the student actual questions and difficulties in leaning, were collected to monitor the students' progress in a timely manner. Finally, the teaching materials (e.g., Java and Eclipse Tutorial, and programming guide in TSpaces) for the experimental class were modified dramatically, based on audience difference between the Summer workshop and the Fall class. For example, we had to balance the introductory contents of the teaching materials, such as the programming IDE and Java, especially for those non-Java programmers. Also, we had to prepare additional examples to teach basic TupleSpace concepts such as template matching. Additionally, because of the insufficient Java knowledge that the students had, dedicated class time was spent teaching TupleSpace programming in Java.

3.4 Data Collection Methods

This study employed several data collection methods to obtain an in-depth understanding of the phenomenon under study and maximize the validity of the research. Multiple methods allow the uncovering of perceptions that might not have been revealed using only one method. The multiple method strategy also adds rigor, breadth, and depth to the study, and provides corroborative evidence of the data obtained (Denzin and Lincoln 2000; Creswell 2002). Five data collection methods were used to gather the essential research data from the experimental class. Brief descriptions of these methods and the corresponding outcomes are listed in Table 4. The detailed descriptions of each method are given in the following sections from 3.4.1 through 3.4.5.

Table 4. Data Collection Methods and Outcomes

Method	Description	Outcome (Raw Data)
M1	In-Depth semi-structured face-to-face open-ended interviewing	• Interview audio (RD1-1) • Interview notes (RD1-2)
M2	Close observation in class and programming help sessions	• Class session video (RD2-1) • Observation notes (RD2-2)
M3	Collection of students' email messages and class website board discussions	Email messages and website board discussions (RD3)
M4	Frequent programming code check-in and final turn-in products	Programming source code and design documentation (RD4)
M5	Questionnaires	• Demographic Information (RD5-1) • Programming Reflection (RD5-2)

3.4.1 Method 1 (M1): Semi-structured Face-to-face Open-ended Interviewing

An open-ended interviewing method was used to discover participants' views, experiences, and thoughts that cannot be easily obtained through observation and questionnaires. The fundamental principle of qualitative interviewing is to provide a framework within which respondents can express their own understandings in their own terms (Patton 2002). A semi-structured interview is a relatively informal and relaxed discussion based around a predetermined topic. It is comprised of guided conversations where broad questions are asked, which do not constrain the conversation, and new questions are allowed to arise as a result of discussions. The questions are prepared, but open, allowing the interviewees to express opinions in discussion. The questions are generally simple, with a logical sequence to help the discussion flow.

Each interviewee answered the same main questions in the interviews to facilitate easy comparison in data analysis. These questions were tested and modified based on the summer workshop interviews conducted at SRI International. The first-round semi-structured interviews was conducted at the beginning, and the second-round of interviews was at the end of the class. The interviews questions were designed to track the progress and changes in students' views, experiences, and learning. The interviews were held on campus in either a quiet conference room in Torgersen Hall or a quiet lab in McBryde Hall at Virginia Tech. Before each interview, participants signed the informed consent forms. The interviewer (i.e., the author of the dissertation) then presented the context of the study and its objectives to the interviewees. The interviews were audio-taped for later transcription. The researcher took interview notes to document her immediate reflections on participants' response to the questions.

3.4.1.1 First-round Interview (i.e., Pre-class Interview)

This round of interviews was conducted at the beginning of the experiment, i.e., at the beginning of the class. At that time, participants had only heard the term "TupleSpace" in the first class, but had not learned any details about it. The purpose of this round of interviews was to determine each student's background, previous programming experiences, project design skills, and previous knowledge of TupleSpace and pedagogy. The interview questions (in Appendix C1) consisted of two major parts. The first part related to participants' past programming experience. For example, the questions were asked, but not limited to, the reasons

that the students first got interested in programming, their self-evaluated professional expertise, programming projects that they had worked on before, prior teamwork experiences in collaborated projects, and their previous educational research and network programming experience, if any. The second part of the interview focused on the students' goals and expectations from this class.

Sixteen interviews were conducted at the time of the first-round interview. However, two participants dropped the class later. Therefore their interviews were not used in data analysis, but for reference only.

3.4.1.2 Second-round Interview (i.e., Post-class Interview)

The second-round interview was conducted after participants finished their group projects, and just before the class ended. This round of interviews (in Appendix C2) focused on understanding participants' learning outcomes throughout the semester. In general, participants' experiences of learning TupleSpace, coordination, and other related design principles were elicited during this interview. In particular, the challenges and difficulties they had when working on the group project design and implementation were reviewed in detail.

3.4.2 Method 2 (M2): Close Observation

The researcher attended every class session and acted as one of two teaching assistants for this class. A video recorder was used to record class lectures and participants' in-class activities. The important entries from the videos were identified to generate a class event log (Appendix E2). During class observation, field notes were taken through the whole semester. These notes were used as supplementary materials to highlight important events, as well as the participants' questions which were related to the process of learning and using of TupleSpace concepts to design and implement collaborative interactive systems. .

The researcher, together with the other teaching assistant, held programming sessions outside the class whenever needed by participants. The purpose of these help sessions was to solve participants' programming related problems, such as debugging. In some cases, the help sessions could last for hours, especially at the stage when participants finalized their group projects. The researcher had many opportunities to watch participants progress in learning because of the great amount of time that the researcher spent with participants.

3.4.3 Method 3 (M3): The Collection of the Participants' Email-messages, Class Website Board Discussions and Questions

The researchers encouraged participants to ask questions related to TupleSpace and programming via email and/or the class web discussion board at any time. This helped the researchers discover participants' questions and difficulties quickly. All of the questions raised by participants were promptly answered by email or on web discussion board, usually on the same day. If a question was considered as common in the class, this question or the related concepts would be brought up again to the whole class in the next class session. All of these questions were documented for detailed data analysis.

3.4.4 Method 4 (M4): Frequent Programming Code Check-in and Final Turn-in Products

Participants were asked to check in their programming code frequently, usually after each class exercise, in the middle of the process in which they worked on their homework or projects, and after they finished programming assignments. The purpose of frequent programming code check-in was to track students' learning and practicing progress. The unfinished source code was generally used to identify problems they had in learning so that the problems could be solved in a timely manner. The runnable programming code from each programming assignment and final group project (including design documentation and programming source code) were also collected and organized for further analysis.

3.4.5 Method 5 (M5): Questionnaires

Questionnaires and surveys were also designed and distributed to participants during the class as a supplement. In addition to the standard multiple-choice questions that asked participants' opinions by simple rating, the questionnaires and surveys also included open-ended questions. These open-ended questions provided participants a more flexible way to express their views and opinions in their own terms. Participants could spend ample time answering these questions with more careful thoughts.

3.4.5.1 Demographic Information Questionnaire (Appendix B)

This questionnaire asked participants' general demographic information, previous programming experiences, and teaching experiences at the beginning of the class.

3.4.5.2 Programming Reflection Questionnaires (Appendix D)

These questionnaires asked participants' opinions on specific concepts and programming issues in the learning process after major programming assignments and the final group projects. For example, participants rated the difficulties of using some important concepts and methods they had learned in TupleSpace and explained their opinions about the most challenging aspect in TupleSpace.

3.5 Data Analysis and Synthesis

The general qualitative data analysis strategy and the detailed procedure are discussed in Section 3.5.1 and Section 3.5.2. Sections 3.5.3 and 3.5.4 report on how researcher managed, organized, and analyzed the research data in preparation to report the findings in Chapter 5. Section 3.5.5 further describes how the researcher analyzed and interpreted the findings in Chapter 6 (Discussion).

3.5.1 Qualitative Data Analysis Strategy

As described at the beginning of this chapter, rather than starting with a hypothesis and trying to prove it, this research began with the design and implementation of an experimental class followed by comprehensive data collections. The next step was to let the results emerge from the primitive research data through qualitative data analysis (QDA).

The data analysis process followed the general QDA procedure, which contains three major processes: data reduction, data display, and conclusion drawing and verification (Miles and Huberman 1994). First, the mass of research data were organized and meaningfully reduced or reconfigured by data coding, in which themes and categories emerged. Second, the data were displayed as an extended piece of text or a diagram, chart, or matrix that provided a new way of arranging and thinking about the more textually embedded data. At this stage, additional and

higher order categories or themes emerged from the data that go beyond those first discovered during the initial process of data reduction. Last, the researcher stepped back to consider what the analyzed data meant and to assess the implications. Verification entailed revisiting the data as many times as necessary to cross-check and to verify these emergent conclusions.

A grounded theory data analysis procedure was followed in this research. The procedure included a three-stage coding and analysis (Strauss and Corbin 1990). The first coding stage was Open Coding, which examined the text for items of interest, with the ultimate aim of accumulating codes into themes and categories. The constant comparative approach were used until the saturation of the category (i.e., no new insights in the category could be gained from the data) was reached. The codes were based on themes, concepts, or ideas. The second coding stage was Axial Coding. It entailed relating categories to their sub-categories around the axis of a central category, based on linkages between their properties. The third stage, Selective Coding, entailed identifying a central phenomenon and relating central categories to it using statements of relationships. A "storyline" was generated that narrates the categories and their relationships in the third stage.

3.5.2 Data Analysis Procedure

Mass resourceful raw data, referred to as RD, were gathered throughout the data collection methods (M1-M5) discussed in Section 3.4. Figure 5 shows an overview procedure of how the data were processed and analyzed through a qualitative process. Raw data were first either organized to organized data, referred to as OD, or analyzed and summarized to analysis results (AR) directly; the OD were then coded to coded data, referred to as CD, through an iterative coding process; the coded data were synthesized to more meaningful analysis results; and the analysis results were finally used to answer the research question and its sub-questions. Table 5 lists the detailed descriptions of the intermediate data, including organized data (ODn), coded data (CDn), and intermediate themes and categories emerged from the first round coding, which are elaborated in Sections 3.5.3 and 3.5.4. Table 6 lists the final analysis results (ARn) generated during data analysis process, which are discussed in Section 3.5.4. Figure 6 shows the analytic coding and analysis procedure to each individual data and its sequential data and results. A detailed explanation of the procedure is discussed in Section 3.5.4. A qualitative research tool QSR NVivo was used to manage and organize various research data. Most research data were

imported to NVivo and coded for multiple rounds. The analytic codes, categories, and patterns were emerged through a qualitative analysis.

3.5.3 Data Organization

Part of the raw data were organized to formal write-up documentation, referred to as the organized data (OD) and shown as Table 5. The organization details are discussed in the following three sections.

3.5.3.1 Write-up Transcriptions with Notes (OD1)

The two rounds opened-end face-to-face interviews obtained through Interviewing (M1) were transcribed. The field notes taken in these interviews were attached to the transcripts. The organized transcriptions are referred to as OD1 in the data analysis. They were imported to NVivo with the name convention "pID#-interview-mmddyyyy" (e.g., p10-interview-12072005), in which ID# represents participant's identification number. The interview data are particularly resourceful because they captured and demonstrated the global changes in participants' thinking during the learning process.

3.5.3.2 Developmental Class Timeline (OD2)

The class video and field notes collected through Close Observation (M2) were reviewed. The major events and findings from each class session were documented in the form of organized logs, referred to as OD2. OD2 shows a developmental timeline that reflected activities in class sessions for each week. Each log entry provides a detailed description about what happened in the class. Major events and findings were recorded as main components in these logs. For example, cognitive confusion, which was shown when participants tried to compare and/or contrast the TupleSpace concepts with other techniques or concepts such as relational database, was documented. One of the challenges the researchers faced in the experimental class was that participants could not understand a technical system without rich models of how, why, and when coordination was desirable. To "bootstrap" the learning process, we asked participants to draw on their own coordination expertise by implementing familiar coordinative games (Lin, Tatar et al. 2006). The class timeline (OD2) is attached as Appendix E2 and also imported to NVivo for further analytic coding, with a name convention of "Lecture #-mmddyyyy," e.g., Lecture 06-09082005.

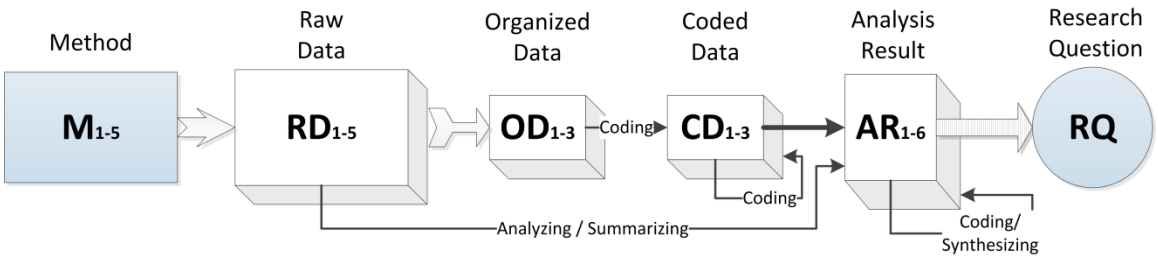


Figure 5. Qualitative data analysis procedure (brief view).

Table 5. Intermediate Data Description

Intermediate Data		Data Description
Organized data (OD)	OD1	Write-up transcriptions with notes
	OD2	Developmental class timeline
	OD3	Discussion documents
Coded data (CD)	CD1	Coded transcription
	CD2	Coded class timeline
	CD3	Coded discussion
Intermediate themes and categories		Themes and categories (generated during the first-round coding)

Table 6. Analysis Result Description

Analysis Result	Analysis Result Description
AR1	The patterns, themes and categories emerged from analysis of coded data
AR2	Programming code analysis results
AR3	Statistical results for demographic information questionnaire
AR4	Statistical results for programming reflection questionnaire
AR5	Sociological challenges in designing and developing parallel-distributed collaborative systems to support multi-user interaction
AR6	Technological challenges in developing parallel-distributed collaborative systems to support multi-user interaction

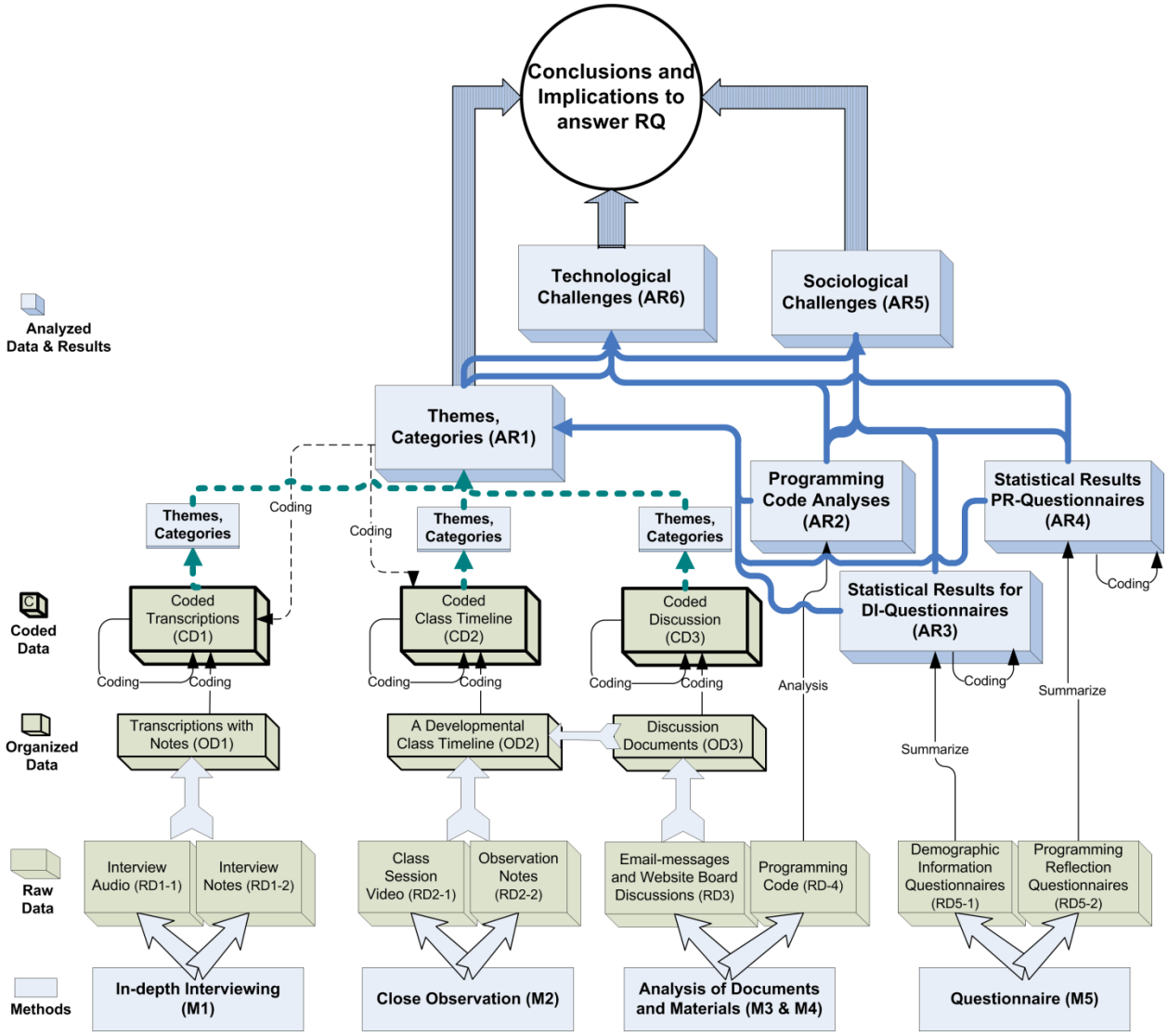


Figure 6. Qualitative data analysis procedure (detailed view)

3.5.3.3 Discussion Documents (OD3)

The researcher organized participants' email exchanges and website board discussions, obtained through M3 (Analysis of documents and materials - email and discussion), to discussion documents, referred to as OD3. Part of OD3 was inserted into OD2 to create a more complete developmental time line. OD3 was imported to NVivo for analytic coding as well.

3.5.4 Analytic Coding Processes

3.5.4.1 First Round Analysis

The data (raw or organized) were processed differently, and with the appropriated analysis method (e.g., analytic coding, checking and analysis, and summarizing) according to their inherent natures.

Open Coding technique, a constant comparative approach, was the major technique employed in this round. The coding process was an iterative and progressive analytic process, which was repeated as often as needed until the category reached saturation. During this process, OD1 (Write-up Transcriptions with Notes), OD2 (Developmental Class Timeline), and OD3 (Discussion Documents) obtained from the data organization stage were coded to generate CD1 (Coded Transcription), CD2 (Coded Class Timeline), and CD3 (Coded Discussion). More than seventy analytic codes, referred to as "Intermediate Themes and Categories", were generated in this process. Nine high-level categories emerged related to participant background, previous experiences, learning strategy, expectation from the class, design strategy, views about different aspects in learning, difficulties in learning, and problems in design and implementation. Each category has multiple sub-categories.

Participants' programming source code obtained at different developmental stages through M4 (Frequent Programming Code Check-in) was gone through thoroughly. The intermediate and uncompleted programming code revealed participants' cognitive learning process, while their final deliveries, such as project final source code and design documentation and reports, reflected the results of learning. The main objectives of analyzing programming code included: understanding how participants perceived TupleSpace as a coordination language and how participants utilized TupleSpace concepts to design collaborative systems; and

analyzing the TupleSpace related programming problems that participants faced, if any. Particularly, the researcher analyzed the Tuples designed in group projects. Through reading the programming source code, the researcher identified the purpose of each Tuple to check how each Tuple supported interaction and coordination in their program. The researcher also compared the initial and final designs to determine participants' learning and conceptualizing progress. The analysis result was referred to as Programming Code Analysis Result, i.e., Analysis Result 2 (AR2) (in Appendix G).

Statistical results from Demographic Information Questionnaire and Programming Reflection Questionnaire (gathered through M5) were summarized in Excel. Particularly, the standardized multiple-choice questions were analyzed statistically to check the statistical difference and significance through, for example, their average or standard deviation. The descriptive questions were coded with the open coding technique. These two results were referred to as Analysis Results 3 & 4 (AR3 and AR4), in Appendix H.

3.5.4.2 Second Round Analysis

The coded data (CD1-CD3), AR2, AR3, and AR4 were coded for another round. Axial coding technique was mainly applied to this round of data analysis. The focus was to relate codes (categories and properties) to each other via a combination of inductive and deductive thinking. This coding process was also an iterative and progressive analytic process. New higher-order themes and categories emerged from this coding process. These new-emerged themes and categories formed AR1 — themes and categories. AR1 is further categorized and elaborated in Chapter 5 Findings (Sections 5.2 to 5.4) as participants' major challenges.

3.5.4.3 Final Round Analysis – Storyline Generation

The final round of data analysis entailed identifying a central phenomenon by applying the Selective Coding technique. The central categories (AR5 and AR6) emerged when synthesizing the analysis results generated earlier (AR1 to AR4). The "storyline" of programmers' cognition in designing and developing parallel-distributed collaborative system with a coordination language was developed. By combining the analysis results obtained from AR1 to AR6, the proposed research question and its sub-questions were answered accordingly.

In summary, the results from the qualitative data analysis were rich descriptions of the important individual and class learning experiences. Various tables, figures, and diagrams extracted from analysis results (AR1 to AR6) were used to present the final results. Table 7 shows how the mass primitive data, through a set of data collection procedures (M1 to M5), were organized, coded, and analyzed. Table 8 shows how the research question can be answered through different data collection methods and data analysis procedures.

3.5.5 Conclusion Drawing

Based on the analysis and synthesis, the researcher considered what the analyzed data mean and also assessed the broader implications of this research, which are discussed in Chapter 6. The researcher further formulated several conclusions and developed various practical and research-related recommendations and proposes the future research directions in Chapter 7.

3.6 Ethical Considerations

Ethical issues (Marshall & Rossman, 2006) are a key concern in this research because it involved human subjects. The researchers are responsible for both informing and protecting respondents throughout the research. The recruitment process involved enlisting voluntary participation in the experimental class. A separation was maintained between participants' grades in the class and their participation in the research (refer to Informed Consent, Appendix A). Participants could drop out of the research at any time without losing class credit, but none did. The researchers treated all information in the research with respect to protect participants. Although it was anticipated that these would be no serious ethical threats were posed to any of the participants or their well-being, this research employed various safe measures to ensure the protection and right of participants. First, informed consent remained a priority throughout the study. Written consent to voluntarily proceed with the study was received from each participant at the beginning of the semester. Second, participants' rights and interests were considered as primary importance when reporting and disseminating data. The researchers committed kept the names and/or their significant identity characteristics of participants confidential. Cautionary measures were taken to secure the storage of research-related records and data; only the researchers had access to those materials.

Table 7. Data Processing Flow

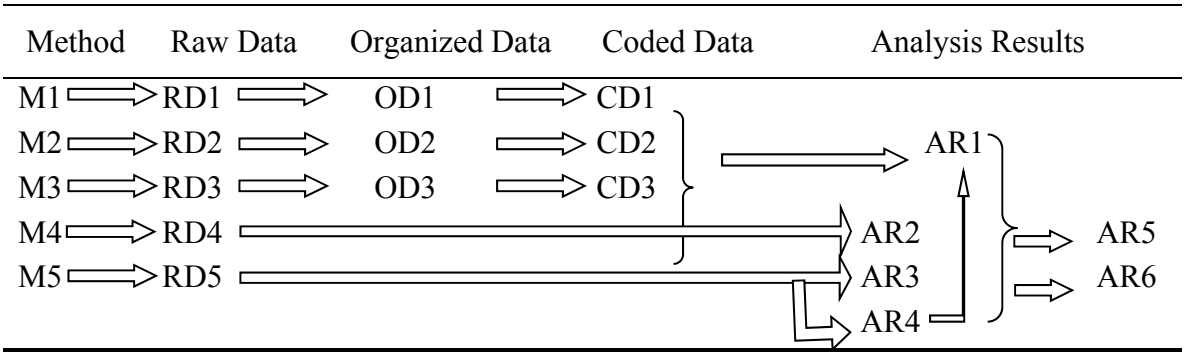


Table 8. Research Question in Relation to Research Methods and Data Analysis

Methods[1] ⇨	Analysis Results
M1, M2, M3, M5	AR1
M4	AR2
M5	AR3, AR4
M1, M2, M3, M4, M5	AR5, AR6

Analysis Results ⇨	Research Questions[2]
AR5, AR1, AR3	RQ(1)
AR6, AR1 , AR3, AR4	RQ(2)
AR6, AR2, AR3, AR4	RQ(3)

Note:

1. Research Methods

M1: Interviewing, M2: Close observation, M3: Analysis of class discussion documents, M4: Analysis of programming code and documentation, M5: Questionnaires

2. Research Questions

RQ: *What can we learn from the experience about how CS students conceptualize and approach parallel thinking?*

RQ-1: What are the challenges in designing parallel-distributed multi-user systems to support complex human interaction and coordination?

RQ-2: How do CS students conceptualize the coordination model through a specialized coordination language?

RQ-3: What are the technological challenges in building computational model through explicit coordination modeling?

3.7 Issues of Trustworthiness

In qualitative research, trustworthiness of the research refers to credibility, dependability, conformability, and transferability (Guba and Lincoln, 1998). It should be assessed differently from that in quantitative research validity (the degree to which something measures what it purports to measure) and reliability (the consistency with which it measures over time). Qualitative researchers must continually try to control for potential biases that might be present in the design, implementation, and analysis of the study.

"Credibility" (validity) is a key component of the research design (Maxwell 1996; Creswell 2002) and shows whether a finding is accurate and credible from the standpoint of researchers, participants, and readers. Methodological and interpretive validity are generally used to test the validity of conclusions reached. To enhance the methodological validity of this research work, the researcher triangulated data sources and data collection methods. Gathering data from multiple sources and by multiple methods yielded a fuller and richer picture of the phenomenon studied. To promote the interpretive validity of the study, the researcher used different strategies such as reflexivity and peer review to maximize validity (Johnson 1997). Reflexivity involved self-awareness and critical self-reflection by the researcher to identify the potential biases and predispositions, which may affect the research process and results. It also involved reviewing and discussing findings with professional colleagues through a peer review process.

"Dependability" in qualitative research emphasizes whether the findings are consistent and dependable with the data collected. Thus, it is important for researchers to document the procedures and demonstrate that coding schemes and categories have been used consistently. The researcher discussed the codes and categories with other colleges and kept a record of how the data were analyzed and interpreted.

"Conformability" refers to that the findings are the result of the research rather than the biases and subjectivity of the researcher. To achieve this, the researcher used different data sources to confirm findings, not by simply interpreting one incident from one source. For example, to confirm the finding that "participants had undeveloped imagination for coordination,"

the researcher found the evidences from interviews, class observation, design documentation, and programming source code, which happened in different learning stages.

"Transferability" emphasizes the reader determines whether and to what extent this particular phenomenon in this particular context can transfer to another particular context. By applying "context-bound extrapolations," (Patton 2002) which are defined as "speculations on the likely applicability of findings to other situations under similar, but not identical, conditions," the researcher attempted to address the issue of participants and the context. Section 3.2 describes participants' overall information and previous experiences in detail. These details can help readers to learn participants' knowledge and experiences in software design and development. Chapter 4 described the experimental class, teaching materials, assignments, students' performance, learning challenges, and how the researchers responded to the class as the instructor and teaching assistant. The researcher has tried to use a deep, rich, and detailed description to provide the basis for a qualitative account's claim to relevance in some broader context (Schram, 2003).

3.8 Limitations of the Study

This study contained limiting conditions, some of which were related to critiques of qualitative research methodology in general and some of which were inherent in the research design. Careful thought was given to ways of accounting for these limitations, and to ways of minimizing their effects.

Because qualitative analysis ultimately relies on researcher thinking and choices, qualitative studies in general are limited by researchers' subjectivity. Although the researcher took steps to maximize the trustworthiness of the research as discussed in the previous section, one of the key limitations of this study is the problem of subjectivity and potential bias in the researcher's participation in learning TupleSpace and being a graduate student in computer science program.

Another related limitation was that participants might have difficulty adjusting to their different roles as learners in the class and participants in the research. Although a separation was made between their grades in the class and participation in the research, because the researchers

in the research were the professor and graduate teaching assistant of the class, participants might have responded in ways they perceived we were seeking or in ways that they thought might be helpful to us. This was the phenomenon referred to as "participant reactivity" (Maxwell 1996).

To reduce the limitation of potential bias during data analysis, all participant names were replaced with participant IDs so the analysis was not associated with any material or data associated with any particular individual. And then all interview transcripts were coded blindly. To address the problem of participant reactivity, the researcher continued to reflect on in what ways she might be influencing participants. For example, during the interview, the researcher attempted to create an environment that was conductive to honest and open dialogue.

3.9 Chapter Summary

This chapter provides a detailed description of the research methodology. A qualitative research methodology was used to understand what cognitive challenges and difficulty affected student programmers to design and develop multi-user coordinated activities. Participants were six senior undergraduate and eight first-year graduate students in the CS program, who were considered as intermediate-to-experienced software developers in general but new to coordination system design. Multiple data-collection methods were employed, including semi-structured, open-ended, face-to-face interviewing, close observation, and artifacts and documents analysis. The data were reviewed against literature and emergent themes through the qualitative data analysis process. And the researcher identified key themes from the findings. The intent was that this research would make a contribution to the research area of programmer cognition by better describing how intermediate-to-experienced student programmers conceptualize the design and implementation of coordinated activities using a coordination language. In addition, it was hoped that this study would be value to the computer science curriculum designers and professors in the university level, shed light on how to train programmers who do not have much experience in this particular area, to approach pedagogy of computational parallel thinking. Credibility and dependability were accounted for using various strategies, including source and data triangulation.

Chapter 4

Experimental Class

The purpose of this study was to explore how student developers conceptualize and approach parallel thinking through the leaning of a specialized coordination language to design and implement parallel-distributed multi-user systems to support complex human interaction and coordination. The researchers believed that a better understanding of this process would allow developers to develop collaborative systems to better support human and humane group interactions. As described in Chapter 3, a qualitative study was conducted to gain more insights of the phenomenon. This chapter describes what happened during the experimental class, mainly from the teaching perspectives. It discusses the major class topics and their corresponding schedules, teaching materials, and various assignments inside and outside the class in the semester. It also describes the changes that the researchers made to cope with participants' learning difficulties.

4.1 Class Schedule

The experimental class consisted of 31 class sessions, held twice per week. Four of the class sessions (Lecture25 — Lecture28) were dedicated to participants working on the team projects. Each class session lasted about 75 minutes.

In the first half semester, participants learned TupleSpace, practiced programming collaborative activities through exercises and assignments, and reviewed and analyzed various collaborative activities developed in TupleSpace. Participants also learned

relevant technical knowledge (e.g., Java and its GUI design), programming tools, and design rationales in CSCL/CSCW and HCI. In the second half of the semester, participants mainly worked on a team project to design and develop a co-located, multi-user, and game-like collaborative system.

4.2 Teaching Materials

Table 15 (in Appendix E1) shows a detailed schedule of each class and the associating teaching materials. The teaching materials comprised TSpaces programming materials from IBM website, related published papers, the class lecture slides developed by the class professor and teaching assistants, and other related materials. Teaching plans were updated and modified based on the feedback from participants after each lecture.

4.3 Teaching TupleSpace Programming

Programming with TupleSpace was one of the major teaching components in the experimental class. The way of teaching TupleSpace started with simple skills and advanced to the most complex; from the necessity to extension. The detailed schedule of teaching TupleSpace programming is listed as below:

- Creating Tuples (fields).
- Creating a Space and connecting to it.
- Learning Template Matching.

  Template matching in TupleSpace refers to the way of accessing matching Tuples by specifying matching criteria for field(s) in a Tuple. A template is a Tuple, but has none or more fields that may be wildcards and consist of only the class type with no value. These fields are often called as "Formal Fields." Template matching is a form of associative matching that is common in Computer Science and related to regular expression parsing or pattern-matching. It is the most basic concept for Tuple based programming.

Two sets of Template Matching Exercises (see Appendix E3) were developed to teach the concept and the use in Java. A design exercise, "Tuple Learning Activity," an activity to support multi-user learning Template Matching, was discussed in the class. Participants designed necessary Tuples to support interaction in the learning activity. The details are discussed in Section 4.4.2.

- Learning operations on Tuples: write, take, or read Tuples to the Space through template matching
- Accessing multiple matching Tuples through Temple Matching
- Extending the SubclassableTuple class and use the ExplicitTuple (an extension class of SubclassableTuple implemented by SRI International)
- Using EventRegister and Callback in TSpaces to implement synchronized activities

The EventRegister and Callback feature is an extension of the original TupleSpace implementation in TSpaces. The feature enables a client computer to register with the Space and to be informed when specific types of events occur. For example, a client computer can be informed whenever a Tuple that matches a specific template is written to the Space by any client computer. In TSpaces this is done by using the TupleSpace.eventRegister() method to indicate the type of event that programmers are interested in and by specifying the Callback object that handles the event. This feature enables synchronized interactions in a networked and parallel-distributed environment. Here is an example code of how to use EventRegister and Callback, which is modified based on the programming code in the IBM TSpaces Client Programmer's Guide.

Step 1: Register an event to the Space;

```
Tuple template = new Tuple( "Key", new Field(String.class) );
    //sets up a template Tuple that describes the format of Tuples that we are interested in.

ExampleCallback callback = new Example3Callback();
    //creates an ExampleCallback object, which implements Callback interface.

boolean newThread = true;
    //default is false.
```

```
int seqNum = ts.eventRegister(TupleSpace.WRITE, template,
callback, newThread );
```

//tells the server to watch for a Tuple being written to the TupleSpace that matches the template Tuple. When this occurs, it should invoke the call() method for the ExampleCallback object. The setting of newThread to true indicates that a new Thread should be started to process the callback.

Step 2: Write a call method to indicate what the program needs to do when a desired event happens and notified by the Space.

```
class ExampleCallback implements Callback {

    public Boolean call(String eventName, String tsName,int
    seqNum, SuperTuple tuple, boolean isException)   {

        if (! isException) {
          // ... process the tuple passed to this method, like

                String data =
    (String)tuple.getField(1).getValue();

        } else {
          //... handle exception here

        }

        return false;

    } // call()

}
```

In the lecture that introduced EventRegister and Callback (in Lecture 09), the researchers first explained how the mechanism works on both the client computer and TupleSpace server, then we drew diagrams to show the changes on the client computer and Space before and after a registered event happened. Finally, we explained how to program in TSpaces using Java. During and after the lecture, participants did not seem have much difficulty with the topics. However, when participants started programming, many of them had trouble implementing their own programming source code in both the Clicker assignment and team project.

In addition, several visual programming tools were introduced to participants to ease programming tasks. First, a popular open-source integrated development

environment (IDE), Eclipse (http://www.eclipse.org/), was recommended to participants. The researchers handed out Eclipse tutorials, held a short lecture, demonstrated the major and most-useful features, and guided participants in finishing several simple programming exercises in various class sessions. Second, Jigloo, a visual Swing and SWT GUI design plug-in in Eclipse, was also recommended to implement user interfaces (http://www.cloudgarden.com/jigloo/). Third, Space Monitor (Section 5.3.2.3), developed by SRI International, is a visual tool to facilitate Tuple based programming. It lets programmers watch real-time changes in the server Space when a Tuple is added, removed, or changed, and modifications of a Tuple and its fields can also be observed. By using these tools, the researchers hoped that participants could focus on problem solving and spend less time dealing with programming minutiae.

4.4 Coping with Students' Learning Barriers

Because of the nature of the experimental class and the purpose of our research, frequent changes were made to teaching materials and strategy whenever needed, according to participant feedback and performance. The major changes consisted of teaching non-Java programmers, template matching, and coordination.

4.4.1 Teaching Non-Java Programmers

Unexpectedly, more than half participants (8 of 14) had not previously programmed in Java. We needed to teach Java at a basic level to help participants get started. Three class sessions were used to introduce the different aspects of Java programming: Introduction to Java (Jumping from C++ to Java) at the beginning of the semester, and Swing and GUI Design in Java (I & II) before participants started designing their team projects. Although Java was not a major component in the experimental class, many problems in implementation were related to Java and affected performance in the class. Related difficulties reported by participants through the post-class interview and surveys are discussed in Section 5.3.

4.4.2 Teaching Template Matching

The researchers did not anticipate that participants would have problems understanding and applying the Template Matching concept at the beginning of teaching. However, we soon found that some participants could not understand the concept very well, even in a simple Tuple programming exercise in Lecture04. Because this concept was basic to TupleSpace programming, we revised the teaching materials and spent three class sessions (Lecture05 — Lecture07) further teaching and discussing the topic.

Two sets of exercises (in Appendix E3) were developed after Lecture04 to help participants get a better understanding of the concept. The first exercise (Tuple Matching Exercise 1) emphasized three components of matching: the same number of fields, the same type of each field, and the same order of each field. Participants' goal was to find matches between 15 Tuples and 7 templates. They finished the exercise by themselves at first, and then discussed their results with another participant. The whole class finally went through the results and discussed some interesting cases. Participants did fairly well in this exercise. The second exercise (Tuple Matching Exercise 2) focused on introducing the use of Template Matching in the context of Java programming (TSpaces). Some difficulties related in Java were reported. Participants showed better understanding of the concept and the use of template matching after these two exercises.

In the next two lectures, participants were asked to design a classroom activity, Tuple Learning Activity (See the description and teaching goal in Table 9), to teach Template Matching. Many class discussions focused on design alternatives. The activity goal, end users (an instructor and students), activity sequence diagram for different users, and the Tuples used to support interaction were developed. More details are discussed in Section 4.5.1.

4.4.3 Teaching Coordination

The original plan for team project was to create collaborative pedagogical activities that used in university classrooms to promote learning. However, because there were no Education students enrolled in the class, the plan was changed to concentrating on using computer technology to support interesting coordination in a co-located

environment. The researchers soon found that participants had difficulty envisioning coordination if the design space was totally open. Familiar collaborative board games thus were brought into the classroom to help participants think about coordination issues. Various multi-player playground games were also discussed in the class. With the inspiration of playing and discussing the board games and playground games as well as analyzing the related coordination issues, many collaborative-activity ideas were brought up by participants and researchers through brainstorming and class discussions. These ideas included collaborative versions of board games and some other collaborative games, usually with some educational elements. The detailed description of the team projects is given in Section 4.5.3. Participants' difficulty of imagining coordination is described in Section 5.2.3 and discussed in Section 6.2.

4.5 Class Assignments

The major class assignments and the teaching goals associated with each assignment are shown in Table 9: Summary of the Class Assignments and Teaching Goal. Other simple in-class programming exercises were also given to participants during the semester. Because both design process and implementation process were important when designing a collaborative system, and in our study, some assignments were design-oriented (e.g., design a "Tuple Learning Activity," Observation of a fast food store) and others were programming-oriented (e.g., TupleSpace programming, Extend ExplicitTuple class). In particular, programming assignments and exercises in the first half of the semester aimed to help participants get familiar with various kinds of computational tools (e.g., Java, Eclipse, TupleSpace). The team project was an open-ended project. The goal of the project was to design and implement a game-like collaborative activity using TupleSpace. For promoting further discussion about participant barriers and discussing the opportunities that TupleSpace could bring to design in Chapter 5, Sections 4.5.1 to 4.5.3 elaborate the details of the assignments Design a Tuple Learning Activity, Clicker assignment (the first programming assignment), and the team project.

Table 9. Summary of the Class Assignments and Teaching Goal

Name	Description	Teaching goal
TupleSpace programming	Write, take, and read Tuples in local and remote Space in different ways.	• Practice programming in TupleSpace: - work with writing and taking in a local machine - work with TSpaces in a remote shared domain, to experience the need for coordination - experiment with more complex data structures (Tuple) and more complex coordination. • Get familiar with using Eclipse and Java programming.
Tuple Learning Activity	Based on the description (activity goal, sequence diagram, and UI) reverse engineer the Tuples and fields that are needed to implement the learning activity and describe how the code will work to create the activity using the Tuples.	• Understand Template Matching better • Design for a real collaborative activity that the students just experienced to promote learning by using Tuples • Start thinking about using Tuples to support coordination
Extend ExplicitTuple class	Design the basic elements (constructor, member variables, and the methods to access the member variables) for different classes by extending ExplicitTuple	• Learn how to extend ExplicitTuple class • Get more practice in Java programming of using Tuples.
Observation of a fast food store	Observe at least one interaction in fast food store drive-through and identify a problem, and think about the interactions.	Practice problem finding and problem solving for wicked problems.

Clicker	Implement a "Clicker" system that allows students to vote on an option in response to a question posed by the teacher. The teacher sees the results of the votes, and may adjust teaching instruction accordingly.	• Experience the coordination issues that come with the multiple user interaction • Practice using EventRegister and Callback • Get familiar with Java UI design by reading and modifying the given UI code (a single-user version of Clicker) • Get more programming practice in TupleSpace programming by using ExplicitTuple
Team project	Design and implement a coordinated game-like co-located activity in TupleSpace as a group of two.	Use the knowledge acquired from this class to design and implement an application that supports multi-user coordination and interaction.

4.5.1 Design "Tuple Learning Activity"

Section 4.4.2 describes that participants had difficulties understanding Template Matching, a core concept in TupleSpace, and its uses in Java. Two sets of Tuple Template Matching exercises (one set at the conceptual level, and another set within the programming context) were first designed for participants to practice individually and collaboratively in class. Then a design assignment, Tuple Learning Activity, was created to help participants understand the Tuple Template Matching topic deeply. In this assignment, participants needed to design a computer-based interactive system to support the Template Matching learning classroom activity. This design assignment provided participants an experience, for the first time in this class, of designing an actual collaborative activity to promote learning by using the coordination language. Because participants had actual experience with the activity in the class, the researchers thought that it would be natural for them to think about collaboration and coordination in a classroom setting. This assignment was also a good start for participants to think of user interactions in terms of Tuples. To inspire participants' ideas about the activity, many class discussions were conducted to analyze the design alternatives. The researchers especially encouraged participants to think about why a certain design was decided, e.g., whether using a computer mediated or a human social control to support a communication between end users, or if a design choice might make a difference to the end users. Based on the discussion results, participants designed Tuples to support the coordination and interaction in the Tuple Learning Activity. Below is a list of some interesting issues discussed in the class. These issues are elaborated in Chapter 5 and further discussed in Chapter 6.

- If security login (with user ID and password) feature is necessary in a classroom setting.
- What is the difference between exchanging ideas with other people (students or instructor) in two circumstances?
    - anonymous exchange through computer, and
    - social exchange through face-to-face interaction

- If different UI design options needed different Tuples design?

  One participant proposed that he would design Tuples differently for different forms of the matching exercise (i.e., True/False questions, multi-choice questions, or directly matching questions) because novice users and expert users may benefit differently from different forms of exercises. Particularly, he proposed that he would have different Tuple designs for different user interface designs.

- Using a database model to design Tuples

  When participants were asked to give examples of their Tuple designs in this Tuple Learning Activity, one participant said that he had used a database model to design his Tuples because he did not want to duplicate fields in the Tuple. He wanted the Tuples to be normalized as the tables in a relational database. A few participants agreed with him. To clarify the difference and confusion, the researchers summarized the different properties and uses for TupleSpace and database in Lecture08, as shown in Table 10 (more-detailed analysis is presented in Section 5.2.4.3).

4.5.2 Clicker

The Clicker assignment was the first programming assignment that required participants to use TupleSpace to create a working program. The goal of this assignment was to implement a simple multi-user voting application for classrooms. The application could reflect instant voting results from multiple students in response to a question posed by their teacher. The voting result can be used by the teacher to adjust further teaching progress. As shown in Figure 7, a student user can vote by clicking a button (A to E) on the screen, and all others can see the change immediately. Each user can vote only for one choice, and the user can change the vote at any time. The teacher user has the option of removing users' votes by clicking the Clear button.

Table 10. Comparison of TupleSpace and Database

Property	TupleSpace (TS)	Database
The purpose of use	• Support learning, a process • Create moments that students can encounter materials appropriate for them • Help students to interact with other people in that learning process	Manage mass of data effectively, secured, and robust
The way of design structure	• Do not really care if the data structure is dense. • The power of TS is really "taking", "take" from resource. You cannot do "take" easily in other systems.	• People always think about design dense structure (e.g., dense table). • Need to think about the data and their relationship with one another. And people care about "store data."
Robustness	"Can people come in and communicate with each other?" is more a more important issue than reliable and security. Even when the teacher's machine stops working, students should be able to keep working. Students should be able to work in pairs.	Many concerns about reliable and security
Scale	Scaling in TS is not about getting more data but about connecting students and machines	Scale data

As summarized in Table 9, there are four teaching purposes for this particular assignment: participants could experience the coordination issues that come with the multiple user interaction; practice using EventRegister and Callback; get familiar with Java UI design by reading and modifying the given UI code (a single-user version of Clicker); and get more programming practice in TupleSpace. The goal was to teach designing proper Tuples to support multi-user interactions and synchronization.

1. Clicker Implementation Process

Participants' tasks were to design Tuples to support posting, synchronizing, and clearing votes, then to implement a simple GUI in Java to accept users' vote through mouse clicking, and then to update the user screen by responding to Tuple changes in the Space. Because this was the first time that participants used TupleSpace to implement a working program, some programming components were given to participants, including: a single-user version of Clicker, the Java programming code for UI and their corresponding event listeners (to listen to user's mouse clicks); and an example of using EventRegister and Callback in TSpaces to synchronize different machines. Both programming code were fully runnable with detailed comments. Therefore, participants' major task was to first comprehend two given example programming code, combine the necessary code to create a program structure, and then modify and add the necessary code to implement the required functions. Participants particularly needed to consider the following issues when programming:

(1) Designing Tuples to support interaction

The vote would be needed to pass between multiple users. Thus, a Tuple such as VoteTuple was needed to represent a user's vote. Because one end user can vote for one choice and change their votes at any time, the first field in the Tuple was used to identify the source of each vote (i.e., whose vote). An end user can make a choice at any time on the screen. If it is the first time the end user makes a choice, an instance of the VoteTuple class is generated and inserted into the Space. If it is not for the first time (i.e., to update their previous votes), the old VoteTuple generated by the end user is removed and a Tuple containing the new vote is inserted into the Space. Each client machine is

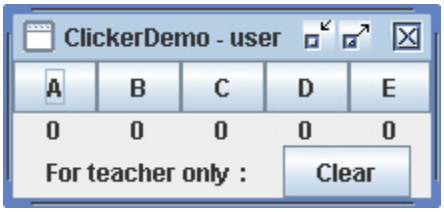


Figure 7. The screenshot of the Clicker program interface

notified by the Space that a new Tuple is inserted and the client machine's screen is updated accordingly.

(2) Synchronization

Participants needed to use the EventRegister and Callback features provided by TSpaces to implement synchronization after they designed the proper Tuples. Each client program registered to the Space for listening to certain Tuple change events (e.g., Tuple created or deleted) from the Space. When a desired Tuple event was triggered, the client program retrieved the necessary Tuples and made the necessary changes in the UI to reflect the current votes from the value of current Tuples.

2. Clicker Implementation Results

Among 14 submissions from participants, six were fully functional; six submissions were partially functional (their code was runnable, but either one or more client machine screens could not be updated correctly to respond to an end user's vote); and two participants, P08 and P12, failed to build the program structure almost completely. The self-reported average time to complete the job was 3.5 hours for design and 6.2 hours for implementation. Two participants reported that they had used more hours than others: P01 reported 10 hours for design and 30 hours for implementation, and P09 reported 10 hours for each. The self-reported challenges from the survey and post-class interview, not surprisingly, were related to a variety of problems. For example, in the implementation phase, some participants reported that they had difficulty using the Java basics and GUI in Eclipse, e.g., making the class constructor, using Java Exception class, working with Java GUI components, and utilizing functions such as CharAt(). Participants who had these problems were mostly first-time Java programmers or those who had not used Eclipse before. These problems were expected to decrease or disappear as participants spent more time practicing programming in TSpaces. The problems related to TupleSpace included, for example, how to structure a Tuple, how to match Tuples through Temple Matching, and how to use Callback to synchronize information between multiple clients.

All participants successfully designed Tuple(s) for the purpose of interaction. Two types of Tuple were designed:

- Example (1): (<String>userID, <Integer>vote) (13 out of 14)
- Example (2): (<String>userID, <String>currentVote, <String>previousVote) (1 out of 14)

The partially working programs had either one or both of these problems: Tuples were not managed correctly in the Space or the end user client's screen is not correctly updated. In the post-class interview and Clicker survey, both participants who failed to build the Clicker said that they had difficulty comprehending the example programming code, and thus could not implement the program, although one of them claimed, in the interview, that he did not get enough time to work on the assignment, but thought that it was very easy in retrospect.

4.5.3 Team Project

The team project was an open-ended project. It involved multiple rounds of design iterations and a stage of implementation. The major goal was to design and implement a game-like collaborative activity using TupleSpace. The procedure of accomplishing the team project included: team project idea creation, team formation (two participants per team), project preliminary design and discussion, project design act-out, project design revisions, project implementation, and project final demonstration and discussion. Because of time limitation, the developed systems were self-tested and used by the whole class, but not by actual end users. However, several projects were evaluated with end users or extended in different ways through subsequent studies in the later semesters.

4.5.3.1 Project Idea and Team Creation

As described in Section 4.4.3, to help participants to envision coordination, board games and different multi-player playground games were brought into class discussion. Many collaborative activity ideas were inspired through the discussion of the coordination issues in the board games and playground games. These ideas included collaborative version of board games such as Apples-to-Apples and Crossword Puzzle, with some educational elements or collaborative games to promote some learning objectives such as practicing simple math calculation in Math Bingo activity.

Fourteen participants were grouped into seven teams, two persons per team, according to their background and programming proficiency. Each team needed to decide on one collaborative activity and then design and implement it in an executable prototype. Table 11 shows the detailed description of each team project. The projects were all game-like, with different components of supporting coordination between multiple end users, many with a pedagogical goal.

4.5.3.2 Project Design Process

4.5.3.2.1 The Preliminary Design

The preliminary design required each team to create the activity goal, activity scenario, activity sequence diagram, UI sketches, and Tuple Design for the activity. After deciding the goal of the project, the first step in design was to create the activity scenario to describe how end users interact with each other in the activity. This was guided by scenario based design (SBD) (Rosson and Carroll 2002). Participants needed to describe a scenario (i.e., a story), from the end user point of view, of how end users carrying out an activity. A scenario generally includes social background, resource constraints, and background information. It also describes a currently occurring use or a potential use that is being designed and included text, video, pictures, story boards, etc. By using a narrative, designers can capture more information about the end user's goals and the context the end user was operating in. The context includes the details about the work place or social situation where the end user stays in, and the information about resource constraints. Scenarios should provide some insights in understanding why end users do what they do.

Based on the scenarios, participants conducted the preliminary design. Because team projects were open-ended, participants did not need to follow specific requirements. Instead, they were encouraged to discover interesting coordination components in an activity, and then create requirements. They also had the freedom to design innovative features to support coordination with TupleSpace. During the design, participants were encouraged to think in terms of Tuples to support user interaction and coordination. The scenario and preliminary design for each activity were then discussed in the class.

Table 11. Team Project Summary

Project name	Description
Algorithm Organ	A program which demonstrates the conceptual significance of the use of function call stacks for a recursive algorithm to compute a mathematical expression. Students who used the program are also viewed as memory space where an activation record of a function call can be placed.
Apples-to-Apples ™	A collaborative game in which each player maintains a hand of seven red apples cards (nouns); during a round, each of the players will select a card that he/she thinks is best described by a green apple card (adjectives) played by the judge. The judge awards one point to the player whose red apple card was selected, then a new judge is selected for the next round.
Crossword Puzzle	A collaborative crossword puzzle game in which groups of students can collaboratively solve puzzles through face-to-face discussion inside and between groups.
Krypto ™	A game in which pairs or groups of players play together to yield an objective value using five numeric cards and four arithmetic operator cards.
Hangman	A multi-player version of the popular word game Hangman in which multiple players can collaboratively play to guess a word.
Math Bingo	A handheld version of the popular Math Bingo game. A player, called picker, can generate an equation and send to other players. Other players calculate the result and fill it on their own board with a goal of claiming a Bingo.
Pictionary ™	An online game in which a poster posts a word, the drawers collaboratively create the picture of the word, and the guesser(s) look at the picture and guess what the word is.

Unfortunately, their initial activity scenario and preliminary design were neither innovative nor particularly fun, even compared to their own experiences and by their own standards. Major problems are listed below and the details are discussed in Section 5.1.1. Most of the initial designs:

- used simple turn taking to support end user coordination,
- used points to reward and setting time limitation to encourage end user engagement,
- used teacher control and management,
- used message passing to communicate with each other in a face-to-face environment,
- used a login feature to ensure system security, and
- maximized machine automation as often as possible.

Board games were brought into the classroom to encourage participants' imagination. Some played the games while others watched. In addition to board games, we encouraged them to think about coordination through playground game play (Lin, Tatar et al. 2006; Tatar, Lee et al. 2008) to help participants focus more clearly on a setting of coordination between multiple parties. The researchers also brought in videos of children playing playground games, and also read anthropological accounts of game playing that focus on negotiation as a crucial component of games, and point out that why a game's end is not as important to children as how it begins.

4.5.3.2.2 Project Activity Act-out

The second round of design was done after the class discussion. However, after this there were not many improvements to the preliminary design. The researchers realized that participants could not really understand the issues and questions we brought up. Three class sessions (Lecture21 — Lecture23) were spent acting out their own designs team-by-team (Nielsen 1990) to bootstrap participants' thinking. One of the sessions was conducted with our research partner, the researchers from SRI International, via video conference. These researchers were experts in CSCL, computer science, and

learning science, and also had a relatively longer experience using TupleSpace. They shared many helpful insights with participants about design and offered various design suggestions.

During the activity act-out, each team used paper, cards, and whatever was needed to demonstrate the whole activity. Volunteers from other teams participated and helped each act-out. For example, participants acted as TupleSpace to receive and send Tuples, or as end users to interact with the computer user interface. The act-out was very useful and helpful. As one participant said:

> One thing that I found was really useful is when we had like, I think it was the first step and we presented it to the class and we kind of did a run through the whole thing, the whole set up, how like Math Bingo would actually work. And the suggestions that people … gave a lot of insight to like to how we should think about mostly like the tuple designs and it kind of cut out a bunch of extra stuff we didn't need. There were at least two or three like real good ideas that I just didn't think of beforehand […]. (Post-class interview, P08, December, 07 2005)

Table 12 summarizes the interesting design issues and choices for each project that arose from the design act-out. For example, in the Krypto game, after reviewing their activity scenario, the researchers questioned whether the simple turn-taking scheme, in which each player makes a move by taking turns, would really work and be fun. In the act-out, Krypto team recruited six players who tried to use five numbers and four operators to come up with a solution for a target value. The Krypto team saw clearly what happened: some players did not seem to think hard to find a solution to the problem; some players were helping each other, while others tried to confuse each other by moving the cards back and forth; and multiple players could not achieve the common goal even after a very long time of play. This was because their design had no affordance to support problem solving processes, and no restriction to limit players' intentional disturbance during a game.

Table 12. Interesting Design Issues and Choices Arose from the Design Act-Out

Group name	Issues discussed in act-out
Algorithm Organ	One challenge in the design was to design a good user interface to facilitate the coordination between end users as well as to show the process of solving the recursive equations or functions through group and personal efforts.
Apples-to-Apples	• What are the differences between whether the judge is known or unknown by other players? • How to select a judge for each round: rotating or socially selecting according to a rule.
Crossword Puzzle	• Resource controlling: resource can be owned by one team at a certain time, i.e., one team "take" one puzzle to work on so that other users cannot get it before it is released to the Space. However, what happen if a team takes all of the questions? The professionals from SRI suggested: (a) change the rewarding system, e.g., a team will get more points by providing correct glues; (b) it may be better to think in letter level than in word level, players can get credits by completing partial answers. • Confliction solving - what happen if one team takes 1-down and another team takes 1-across in the puzzle, where there is a conflict in the shared cell? The participants in this group suggested that the players can use message passing when there is a conflict. • Implementation feasibility: the design may be too complex to implement for the semester.
Krypto ™	A design problem was revealed during the act-out: the players did not seem to think very hard and find a solution to solve a given problem. For example, some players were helping, while others were just trying to confuse each other. A player had no obvious goal to work for in his/her turn. The game design did not bring too much fun, competition, or coordination for players.
Hangman	• Who chooses the word for each game, either the system automatically picks one or selects one from the users' entries. • How to design the "Ask advisor" function, in which one player can ask other players' opinion when he/she cannot decide what letter to choose? Possible choices include: (a) the players who want to offer help click a letter within a period of time, and the player can decide whose advise he/she wants to take; (b) computer counts votes for the advises; and (c) social discussion to decide the letter.

Math Bingo	• What happen if more than one player claims Bingo simultaneously: the system automatically picks the winner or the players negotiate that? • Players need to know where they make mistakes (a) calculation error, or (b) claim Bingo error.
Pictionary	It is hard to look at the timer while drawing. The Pictionary team suggested that they could design beep sounds to indicate the end of the drawing is approaching.

4.5.3.2.3 Design Iteration

After the activity act-out, all teams re-thought and re-designed their activities according to the discussion and feedback. First, many projects became more interesting and interactive with more social interactions, either through computers or conversation, whichever is more appropriate in the given context. For example, the Collaborative Crossword Puzzle team removed the instant messaging feature. Instead, the players (students and teacher) were encouraged to communicate with each other through direct conversation in the activity. This change made the interactions more natural and flexible. Second, many of the designs were more open-ended, with minimal and tailored features to support coordination around rules and roles without simple enforcement. For example, the Math Bingo team removed the automatic Bingo claim feature, so players can take more responsibility for their answers. In addition, many teams removed the security login feature and retained the user name as an identity, if needed.

Not every team did well. The Krypto team still had difficulty designing coordination rules to make the activity more interesting and effective. During the design iteration, they added a list of card movement history. However, the way that they designed was still not helpful for the players in achieving their goals.

4.5.3.3 Project Implementation

Participants were given about two weeks to implement a working prototype for the collaborative activity they designed. Each team member took part of the responsibility. The way that they collaborated with each other and the time that they spent on programming were varied. Their programming experiences and the difficulties they encountered during implementation are discussed in Chapter 5. The researchers helped participants in programming during regular and extra office hours. Email communication and class website discussions were used through the whole implementation phase.

4.5.3.4 Project Demonstration

The last three class sessions were used to demonstrate participants' team projects. All teams developed a multi-user system to support a collaborative activity in a relatively

short period, considering they were new to multi-user socio-technical systems development, and had little programming experiences in Java and TupleSpace. Five projects (i.e., Algorithm Organ, Apples-to-Apples, Crossword Puzzle, Krypto and Hangman) were desktop applications, one (i.e., Pictionary) was a web-based application, and one (i.e., Math Bingo) was a handheld-based application. For each project, part or all of the class participated in the activity together as end users, and acted as different roles when necessary. The technical requirements and the social requirements were both checked and discussed to see if the designed system met the original goal. Most of the systems functioned well during testing. Some systems had minor technical problems such as synchronization between different users' screens. Suggestions and further work were also discussed in the class.

4.6 Chapter Summary

This chapter gives an overview of the teaching activities in the experimental class. The class topics, teaching materials, assignments, and responding schedule are listed, and described respectively. The chapter especially elaborates how the researchers dealt with unexpected participants' learning difficulties in the class. The major assignments in the class were also described in detail to support the discussion in Chapter 5 about participants' learning experiences and barriers.

Chapter 5

Findings: Students' Learning Experiences and Barriers

The researchers offered an experimental class to investigate how student programmers conceptualize the design and development of multi-user coordinated systems using a coordination language to approach parallel thinking. The researchers developed and followed the curriculum described in Chapter 4. The students in the experimental class participated in the research. They learned TupleSpace programming, the related technical information, and design rational and principle in CSCW, CSCL, and HCI. Participants developed seven multi-user, distributed, fine-grained collaborative systems by the end of the class. As described in Chapter 3, the research followed a qualitative based research methodology. During the class, the researcher gathered various qualitative data, including face-to-face interviews, class video, design documentation, programming source code, surveys, and other related materials. The data was organized and analyzed through a series of qualitative analysis procedures.

This chapter presents the results of participants' experiences and behaviors in relation to the events of the class. The data are presented as three major categories of findings according to the sub research questions. The first category of findings addresses the sociological challenges in programmers' understanding of the human coordination and interaction needed in multi-user coordinated systems. These findings came primarily from the design stage of the class. However, problems that emerged in implementation

stage showed that system design influenced approaches from the beginning. The second category of findings is related to TupleSpace as an expressive language for coordination. The third category of findings is related to the way that TupleSpace integrates with other system technical components. Each class of findings is reported with detailed data that support and explain them. Illustrative quotations taken from interview transcripts, class videos, students' surveys, and other documentation are used to show multiple participant perspectives.

5.1 Student Programmers Faced Sociological Challenges to Understand Complex Human Coordination and Interaction

The first category of findings is related to the programmers' perceptions of the social components in a coordinated interactive system. The findings reveal that the overwhelming majority of participants had difficulty envisioning complex human coordination and interaction.

The analysis result (AR5: Sociological challenges in designing and developing parallel-distributed collaborative systems to support multi-user interaction) obtained mainly through the in-class observation and analysis of design documents suggested that, in the design stage, participants' perceptions and assumptions about human social behaviors in a coordinated activity affected and limited the design of the target systems. The survey and post-class interviews provided more insight about the values that programmers' brought to the design task.

5.1.1 Participants Took Utilization of the Computer's Ability as a Design Brief

The overwhelming majority of participants took the design of computational features as an implicit design brief. All design proposals involved one or more computational features that were added because the computer could do it. The following

examples illustrated both the features and their consequences for the design of the systems.

5.1.1.1 Let the System Automatically Correct and Judge for Users

The proposal for the Math Bingo activity to be implemented on handheld computers involved two kinds of players: a picker who generates an equation and sends it to the other players, and players who calculates the result on their own board and, if correct, get a "token" on the correct place on their board. An assumption was that the system would automatically claim a "Bingo" for the player and deliver that information to the picker.

A second example is from the Apples-to-Apples activity. In this activity, on each round there is a judge who picks an adjective (Green Apple) card. The other players have hands of seven noun phrase (Red Apple) cards. During a round, players select a card that they think is the closest association to the card played by the judge. The judge picks the card that they believe is closest. A new judge is selected afterward for the next round. The design feature in question was that it was the computer's job to decide who would be the judge in the next round rather than players decide.

There is nothing intrinsically wrong with these design choices, but the design rationale turned out to be surprising. When asked, participants explained "It was more *efficient* for the computer to decide who," for example, had a Bingo; and "It's not *fair* if people decide who goes next." Not only did these design teams make the "efficient and fair" argument, but most participants did not see a need to consider alternatives. It was not easy for participants to understand that there could be reasons to restrict from using computer automation if it could be done by the computer. We reframed this question in terms of discussion of values by taking the issue to extremes; while efficiency has a role in games, it is most efficient not to play at all; while fairness has a role, normal adults do not run footraces with four-year olds for the satisfaction of winning. Fun is also a consideration.

Associated with these rationales was a surprising view of the machine. Many participants made the blanket assertion that "machines are fairer than people." This takes

on particular importance when we consider the claim in the Media Equation that people treat computers as "social actors." (Nass, Steuer et al. 1994) Our participants seemed to go beyond that, treating computers as privileged, powerful actors.

5.1.1.2 Designed an Instant Messaging (IM) Feature to Support Face-to-face Interaction between Players

Although every team designed for face-to-face interaction, almost every initial design had an IM-like component. For example, the Collaborative Crossword Puzzle was a game in which each team of players had copies of the board on their computer that updated in real time when other teams made changes. In the initial plan, communication between teams was to be accomplished via IM. The design document read, "A message dialogue box opens and team two tries to urge team one to update their answer via electronic messaging" and "the game continues with teams solving as many questions as they can and messaging other teams to resolve conflicts." (Collaborative Crossword Puzzle initial design documentation, Oct. 26, 2005) The designers realized that discussion between teams was important, yet they assumed that it was *better* to have people communicate via computers than orally, even in a co-located environment.

5.1.1.3 Used a Security Login as a Default Feature for Multi-user Systems

Four teams of seven started designing their games with a "security login" feature requiring input of user names and passwords. Security logins are common features that appear in almost every web application. When designing a website, this feature can prevent unauthorized accesses to users' accounts and protect the user's privacy. But participants did not articulate these justifications in design. The Hangman activity is a multi-player version of the popular word game. When participants were asked why players need security in this activity, some argued that they needed a user name to provide identification and scores to track different people. Yet, if desirable, scores for single games or sessions could easily be maintained without identification. Also, it was unclear who would want to maintain long-term track of the Hangman scores. In fact, most of the games were deliberately ephemeral, and many required collaboration to finish a task through social interaction (e.g., finish a crossword puzzle together). However, it

was difficult for participants to take that design brief down to implications that discarded standard features.

The downsides of implementing IM-like mechanisms and security login features are that these features distracted participants from thinking about the actual game activity, greatly added to the potential complication of implementation, and encouraged participants to think about the reason for the game being some future outcome rather than present play. In many cases, the need to remember user names and passwords can be a barrier to participation. The researchers encouraged critical analysis of design features and alternative representations of roles using visual techniques such as shapes, colors, or patterns to represent teams.

5.1.1.4 Over-Designed Teacher's Interaction with Computer

Three team projects had teacher-like roles. The initial designs delegated time-consuming computational tasks to the teacher role. Design features enabled teachers to *monitor* and *guide* students' activities using a computer. For example, the initial Math Bingo design read:

> The teacher then notices a popup on her screen saying "Sam thinks (s)he has a BINGO!" with a button that says "Check Board." She clicks the button, and it says that Sam did not in fact win. The game continues for everyone except Sam until a valid winner is found. (Math Bingo initial design documentation, Oct. 28, 2005)

Another example, Collaborative Crossword Puzzle, shows participant tendency to let the teacher use a computer system to *control* the game process:

> Mrs. Smith all the sudden has to make an announcement to the class. She clicks on the "pause" button on her screen. The game is temporarily frozen with all teams unable to make changes. The teacher gives her announcement then clicks on the "play" icon to continue the game. (Collaborative Crossword Puzzle initial design documentation, Oct. 26, 2005)

What was so striking about these designs was that the teacher was put in the middle in the first, and that existing teacher practices were ignored in the second.

All the preceding examples from Section 5.1.1.1 through 5.1.1.4 reveal that not only participants had a hard time designing user experience, but their inclination was to design computational features and automate the process.

5.1.2 Participants Brought Strong Assumptions about User Experience into the Design Task

Participants made strong assumptions about the meaning of user experience, especially winning experience, and brought that to the design task. They seemed to understand that engagement is important, yet their first thought of increasing user engagement was to use a reward or to challenge external to the activity itself. For example, in Collaborative Crossword Puzzle activity, "All teams are awarded one point each for all correctly answered questions they were responsible for solving. The team's list is updated with each team's score, a team ranking is calculated and the list is sorted by rank." (Collaborative Crossword Puzzle initial design documentation, Oct. 26, 2005) The design team thought it obvious that having an explicit score and engaging in competition made the game better for users. But the reward in Collaborative Crossword Puzzle lies in making a contribution. In fact, in many cases, participation and engagement are the reason to play collaborative games, and an explicit reward structure can discourage interaction between people who are not as competent. As with many playground games (e.g., rubber band rope jumping in Chinese, Guatemalan, and Indonesian cultures), the goal is having fun, and gaining points only help to the extent that they promote fun for the particular players.

Time limitation is another external factor that interested participants. Pictionary is an online game in which a player posts a word; several players (i.e., drawers) collaboratively create the picture of the word, and other player(s) (i.e., guesser(s)) look at the picture and guess what the word is. The Pictionary team initially designed two timer features to set time limitation, one for each drawer, and the other one for guesser(s). However, the team did not consider (a) whether it was fun to have a timer, (b) what people could accomplish within the time limitation, or (c) how external markers of progress affect different people (e.g., it was hard to look at the timer while drawing).

These proved to be self-correcting design issues in the project act-out, as no one could play, much less enjoy, the initial game in the given time frame.

5.1.3 Participants Had Difficulty Envisioning User Coordination through Computers

Software design is fundamentally about envisioning and facilitating new ways of doing things and new things to do. Our target systems used a distributed architecture to support interaction and coordination between end users. Therefore, software developers need to first envision how people interact and coordinate without a computer system and then to design technology to support and improve the user experience. Almost all participants showed an undeveloped imagination of what was entailed in the coordination process. This lack revealed itself primarily in the design phase but also required instructional attention in implementation phase. This section first describes the difficulty in design, mainly at the conceptual level. Section 5.2.2 then describes the difficulty in program design phase when modeling a coordination activity, and Section 5.2.3 describes the difficulty in the implementation stage. Section 6.2 summarizes these difficulties, analyzes the possible causes, and discusses the benefits of using a coordination language to model coordination activities to catalyze participant imagination.

Participants showed great interest in examples of coordinated activities, and started to acquaint themselves with the concept of coordination. However, their preliminary project designs revealed that they used a linear approach to address users' distributed activities. Almost all initial designs, for example, failed to take advantage of collaboration, manifested difficulty envisioning all the resources on which the system could draw, and failed to provide a coordinated mechanism.

First, several projects initially failed to consider the benefit of collaboration in a coordinated activity. For example, the Krypto game involved a group of players collaboratively arranging five numeric cards and four arithmetic operator cards to calculate a target number. The designers initially proposed the support of strict turn taking strategy, in which one person would lay down a card once; the next player lays down another card, and so forth. Like Krypto, Pictionary and the Collaborative

Crossword Puzzle games also initially followed a pattern in which multiple players acted in strict sequence. In designing these activities, participants did not seem to understand well that the primary benefit of making these games distributed was that multiple players could act at the same time. Considering end users as a resource of a coordinative system, the system did not fully utilize the resource. In addition, participants did not realize that other players might be waiting for something to do. For example, the first proposal for the Collaborative Crossword Puzzle activity only allowed one player to choose a clue and then enter a word, while all the other players waiting. The only way that one player would know what another player had accomplished was when the entire word appeared on their screens. Failure to envision the "wait state" of the players was one example of a system element that needed more consideration.

Second, participants did not have good understanding of the coordination path. Section 5.1.1.2 discussed participants took the IM-like design feature as a default feature in a coordinated system no matter if it was needed. Another aspect to see this design was that participants misunderstood the coordination paths. The collaborative games were intended to be played face-to-face. This meant that the most ordinary of human capabilities, such as talking, not only could be used, but were part of the fun, yet all participants initially proposed implementing separate IM-like channels as coordination paths among players. As described earlier, in games such as Collaborative Crossword Puzzle rather than telling the other person that they had made a mistake, one person would write a message to the person next to them. While there was nothing *a priori* wrong with implementing such systems, neither was it a rational option. It was an unexamined assumption that added a large implementation component without improving the coordinated nature of the game by anyone's criteria.

Third, participants did not provide a mechanism, either in storyboard form or via the design of the appropriate Tuples protocol, to present the necessary information needed by game players to take different actions or have a satisfactory game playing strategy at different stages during a game. For example, the Krypto implementation did not pay enough attention to the essential coordination among players. The game design also neglected the fact that laying down each card severely constrained the possible

solutions. The first players were under-constrained, whereas the last players were typically left with an unsolvable problem. A successful game implementation requires not only support for game moves, e.g., laying down a card, but also support for the interactions among multiple players. The initial Krypto design ignored the necessities of the supporting decision making process and the exploration of alternatives.

These design impulses suggest that participants did not initially include the perception of what the resources were in a coordinated environment, what others did when one user got a turn to perform their tasks, and the system resources needed to be passed back and forth among multiple parties. Participants did not have deep understanding of coordination and its mechanism.

5.1.4 Participants Had Difficulty Envisioning Coordination and Individual's Experience

In CSCW systems, there is often a conflict between what people want to see on their screen to support the local task at hand and what they want to support collaborative work (Greenberg and Marwood 1994). When a person is not doing something locally in his screen, he wants to know what others are doing, but they may be busy. Interrupting them could ruin the activity.

The Algorithm Organ team had particular trouble understanding the relationship between what was needed to be shared and what was needed to be kept private at different times. The Algorithm Organ demonstrated the conceptual significance of the use of function call stack to implement recursive algorithms, for example, to compute a mathematical expression. Each player was responsible for a stack element, asked to solve their small part of the problem according to the current stack status (using pseudo-code), pass their requests for solution to others, wait for the partial answer to be returned to them and then return it to the elements above them. For example, a group of students tries to solve a Fibonacci number, e.g., F_5, where

$$F_n = F_{n-1} + F_{n-2} \qquad (n > 1);$$
$$F_1 = 1 \qquad (n = 1);$$
$$F_0 = 0 \qquad (n = 0).$$

The first student (S1) realizes that the solution to F_5 is $F_4 + F_3$ (n=5), so he sends out (posts) the required calls to calculate F_4 and F_3 and waits. S2 sees the posted call and picks F_4 to solve. Using the first equation, he figures out that F_4 is $F_3 + F_2$ (n=4) and sends out the calls required to solve F_3 and F_2, and waits too. S3 picked the call for F_3 and posts $F_3 = F_2 + F_1 = F_2 + 1$ (according to the first and second equations). S4 picked the call for F_2 and posts $F_2 = F_1 + F_0 = 1 + 0 = 1$ (according to the second and third equations), he returned his answer to S3. S3 solves his question ($F_3 = F_2 + F_1 = 1 + 1 = 2$) and posts an answer to S2. With S3's and S4's answers, S2 can calculate $F_4 = F_3 + F_2 = 2 + 1 = 3$. Finally, S1 is able to solve the whole problem $F_5 = F_4 + F_3 = 3 + 2 = 5$. In this case, the end users play at being computer memory space where an activation record of a function call could be placed. The waiting that individual players do is equivalent to the waiting that computer memory computes do.

The initial UI design was simple but confusing: only the student who sent out their partial solution could receive the answers returned by other students. Thus, students who were waiting for returned answers were both idle, and also not aware of others' work. The modified design split the UI to a public view and a private view. Users could switch between the two views. The public view, representing the whole function call stack, showed all of the changes posted by students. Thus, end users could monitor the global changes. The private view provided an area where end users could work on their own solution.

5.1.5 Summary

The design impulses in Sections 5.1.1 to 5.1.4 suggested that participants brought strong assumptions and values about their target systems into the design process. These assumptions did not initially include the envisionment of passing resources that needed for coordination back and forth among multiple parties. In addition, participants did not have deep understanding of coordination mechanism and interaction among multiple users, which affected participants designing appropriate computational features to support coordination and interaction. Sections 6.1 and 6.2 further discuss the possible causes and implications.

5.2 Student Programmers' Conceptualization of Using TupleSpace to Form Coordination Model

Participants were encouraged to use TupleSpace constructs, as found in the TSpaces language, to conceptualize coordination and TupleSpace operations to form the internal communication protocols for each design and programming assignment. Participants' feedback about their learning experience with TupleSpace, and the analysis of their Tuple design show that TupleSpace helped all participants focus on thinking coordination issues.

5.2.1 Participants' Feedback about TupleSpace

The experience of learning and applying TupleSpace was reported through the class survey, post-class interview, email and web discussion, and researchers' observation during class and office hours. The survey results (see Q1.1-Q1.5 in Table 13) show that the overall learning experience with TupleSpace was positive, with an average ranged from 6.2 to 8.8 (see note under Table 13). Participants reported that the basic concepts and operations in TupleSpace were easy to understand and use. Almost every team reported that TupleSpace ultimately simplified their coordination design in the project. Evidence of growth in understanding coordination was that most teams reduced the number of Tuples in the final design as they realized that coordination was simpler than they originally thought. The most difficult aspect was the "EventRegister and Callback" feature in TSpaces (not a part of the TupleSpace Model). Six participants (42%) rated it less than or equal to 5. However, interestingly, five participants (36%) rated it greater than or equal to 9. This suggests that participants had very different learning experiences from one another with EventRegister and Callback. The second most difficult aspect in TupleSpace programming was "Template Matching." Four participants rated it less than or equal to 4. Post-class interview data gave more explanation of the statistic as shown in Table 13. The details are discussed in Section 5.2.4: Roadblocks in TupleSpace/ TSpaces Programming.

Table 13. Survey Results Regarding Learning and Using TupleSpace / TSpaces

Questions	Response range	Average response	SD[a]
Q1 Please rate the following TupleSpace/TSpaces operations/functions/concepts in terms of "easy to understand or/and program"			
Q1.1 Tuple/TupleSpace idea itself	5-10	7.5	1.3
Q1.2 Tuple basic functions: Read, Write, Take	6-10	8.8	1.2
Q1.3 countN, multiRead, multiTake	4-10	7.4	2.2
Q1.4 Template Matching	3-10	6.9	2.4
Q1.5 Callback and eventRegister	2-10	6.2	2.8

Note. "1" represents Absolutely Hard, and "10" represents Absolutely Easy.

[a] SD: Standard deviation

5.2.2 Explicit Modeling Coordination and Interaction through TupleSpace

In the post-class interviews, participants were asked their general purpose of designing Tuples for a program. Most participants summarized that Tuples were defined as information or resources need to be shared between people and used to support interactions between client programs or people who used those programs. Each team's programming source code and project design documentation were checked and analyzed to validate their claim. The name, structure, and the detailed description of each Tuple for seven team projects were summarized from Table 17 to Table 23 in Appendix G. The role of each Tuple in the activity was labeled by the researcher for analysis purpose. To summarize, each collaborative activity designed a couple of Tuples, ranged from 4 to 7. Almost every team reduced the number of use Tuples (from 1 to 2) in the implementation stage compared with their original design. This indicated that participants obtained better conceptualization in constructing the coordination model and that interaction was actually simpler than they organically imagined. TupleSpace facilitated participant conceptualization of coordination activities at both the system level and the end-user level.

As discussed in Chapter 3, the most significant benefit of using a coordination language such as TupleSpace is to separate the coordinative activities from computational activities and create separate models: a coordination model and a computational model (Gelernter and Carriero 1992). The role of a coordination model is to glue separate computational activities (e.g., generating or updating UI components in screen) into an ensemble. From the software engineering perspective, the advantage of separation is its orthogonality, which further enables the generality and portability of the language. From the programmers' perspective, the advantage of separation is that programmers can easily focus on thinking about coordination through the explicit exchange of resources and communication between users. The following sections summarize how participants used TupleSpace in design and implementation.

5.2.2.1 Manage and Allocate Resources through TupleSpace

Among the Tuples designed in team projects, about 70% Tuples were defined as system resources that were shared (e.g., viewed) by multiple users. In the interview, three participants said that they used Tuples as information or resources that need to be shared between people. These resources were either used for public display (in which contents will not be changed after posting), or shared use (in which contents may be changed by any user after posting), which indicated the needs for concurrency management. For example, Apples-to-Apples is a game that involves red and green "apple" cards, which are resources that may be passed from one player to the group or vice versa. The design team created seven Tuples that delineated the coordination they wanted to implement through the computer. Among the Tuples designed, five Tuples — i.e., RedAppleTuple, GreenAppleTuple, CurrentGreenAppleTuple, PlayerTuple, and PointTuple — were defined as public or shared resources, as shown in Figure 8. For example, RedAppleTuple and GreenAppleTuple were used to represent red apple and green apple cards in the game. And there were seven instances, represented by seven red apple cards (i.e., windows, pirates, bodies, buttons, cigarettes, interviews, and waterfalls), of class RedAppleTuple at that moment.

Three Tuple operations (i.e., *write*, *read*, and *take)* can be used to manage and allocate the shared system resources through the Space. The *take* operation is unique. It enables TupleSpace to represent limitations on resources. For example, if one person *take*s an instance of RedAppleTuple (e.g., the pirates card), that Tuple is no longer available for anyone else. This is very similar to the way customers might buy copies of the New York Times at a newsstand until the pile is used up. Learning and understanding TupleSpace operations were rated as 8.8 (in which "1" represents Absolutely Hard, and "10" represents Absolutely Easy), the easiest concept in TupleSpace.

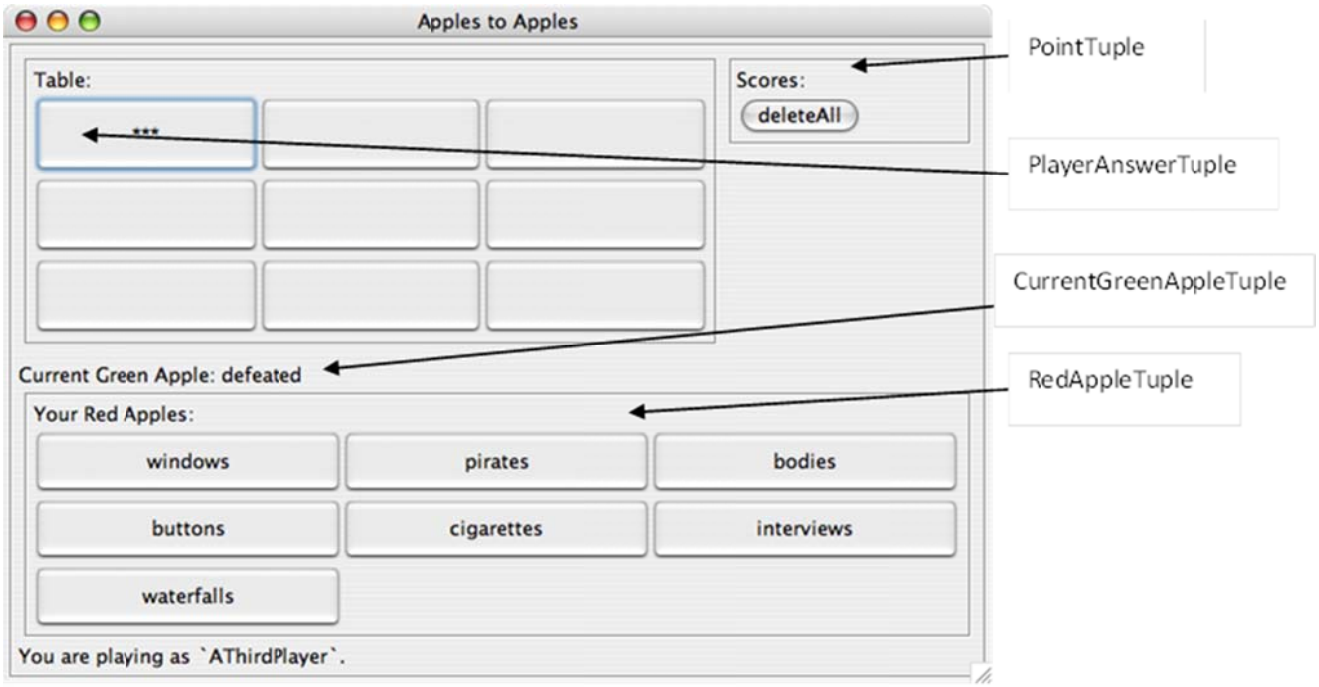


Figure 8. Apples-to-Apples screen shot with responding Tuples.

Note. PointTuple: the score for each player is not currently displayed; RedAppleTuple: there are 7 RedAppleTuples (7 instances of class RedAppleTuple) in the screen, at the moment.

5.2.2.2 Abstract Data for Interaction

In the post-class interview, about half of participants said that Tuples could be used to support interactions between client programs or end users who used those programs. Two participants mentioned that designing Tuples was like designing an object-oriented class. However, the difference was "*… for Tuples you are not abstracting data, but you are abstracting data that is for interaction.*" (P14, post-class interview, December 04, 2005) For example, in the Apples-to-Apples game, PlayerAnswerTuple conveyed information (i.e., player's selection of a red apple card) passed between players and the judge and among all players. RevealTuple was used as a signal by the judge to make the red apple cards visible to other player (in the game, the cards are face-down after submitted by each player until the judge reveals them). And PlayerAnswerTuples represented the players' answer.

5.2.3 Undeveloped Competence in Modeling of Coordinated Activities

As discussed above, participants used the coordination model to manage and allocate the shared resource in the coordinated environment, as well as support interaction and communication between end users. The advantage of using a coordination language like TupleSpace is to separate coordinated activities from non-coordinated and non-communicative activities. The separation is a process that programmers concentrate on in identifying coordinated issues and modeling them with Tuples. Although conceptually the line between coordination model and computational model is relatively clear, some participants encountered challenges envisioning the relationship between these two models in programming. The analysis of design documentation and programming code showed that several participants had difficulty differentiating these two models. For examples, some participants (1) used a shared resource, implemented as a Tuple, in both the computational model and coordination model, at different times; (2) used a Tuple to represent aspects of the program that were not coordination resources or that contained mixed information about different resources; and (3) used multiple Tuples to represent the same resource. This roadblock affected

participants to build appropriate communication protocols in the coordination model successfully.

First, Tuples form the coordination model and indicate coordination needs, thus should only appear in the coordination model. However, some participants used a Tuple to represent a shared resource in both the coordination model and the computational model, at different times. In Krypto, two types of Tuples, AnswerNumTuple and AnswerOpTuple were created to represent the number cards and operator cards. These Tuples were supposed to appear only in the coordination model when the end user made the solution public. However, they were also used in computational model when an end user moved a card around on the screen without showing the change to others. This design led to other users losing their own private work (e.g., they attempt to move cards in their own screen) without notification when one user broadcasted the change.

Second, a Tuple should represent the aspects of the program that are related to coordination and it should only contain information for one resource. However, some participants used Tuples to represent non-coordination related resources and used one Tuple to represent mix information of multiple resources. For example, in the Pictionary activity, GameCreatingTuple included mixed information about the game and users, which are not related to coordination either.

Third, one Tuple is usually enough to represent one resource for most cases. Without particular reasons, some participants used multiple Tuples to represent one resource. For example, the Hangman activity used four Tuples (i.e., WordToGuessTuple, LengthWordToGuessTuple, CurrentGuessTuple, and GuessLetterAsciiTuple) to represent a guessing word in different formats. This increased the distribution of interdependency between data. A similar general design problem was in using Tuples as global variables, which can lead to unnecessary race conditions or data inconsistency.

Furthermore, the difficulty in imagining the underlying structure also revealed that a few participants lacked the imagination of what actions should be counted as “local” events (things done by a single user’s machine), thus in the computational model; and what should be counted as “remote” events (things done by other users and reported to the central TupleSpace server), thus in the coordination model.

5.2.4 Technical Roadblocks in TSpaces Programming

Participants designed and implemented seven collaborative activities in a relatively short period as novices in this development area. These activities all contained different coordinated components that support collaborative tasks in a face-to-face environment. Yet, like most novice programmers, almost every participant had difficulties when learning and conceptualizing TupleSpace and its Java implementation TSpaces. The following sections elaborate the most significant issues.

5.2.4.1 EventRegister and Callback

Not an original feature in TupleSpace model, the EventRegister and Callback feature is an extension in IBM TSpaces to support automatic synchronization in a distributed environment. The similar mechanism is commonly used in event-driven programming architectures and distributed programming environments.

The post-class survey reported that EventRegister and Callback feature was rated as the hardest component of TupleSpace/TSpaces (See Table 13, with a mean of 6.21). The variation was 2.75 standard deviations, which means that different participants had different opinions about using it. Eight participants (58%) thought it was easy (that is, rated it greater than or equal to 9), whereas six participants (42%) had problems understanding and using it. In the Clicker assignment, more than half of the participants thought it was hard to understand or program. In this team project development, two participants in different teams still reported that they had difficulty or failed using EventRegister and Callback to implement certain features. This result was emphasized by the post-class interview and their programming source code. Here are two cases in which two participants had extremely hard time understanding and using this feature.

Case 1:

Participant P01 claimed that using EventRegister and Callback was the biggest challenge that he faced in the class. In the post-class interview and survey, he explained what was hard. While being asked the Clicker assignment, he wrote, “I did not understand Callback. I did not know what needs to be done locally or in TupleSpace. I

still do not understand the sequence. I still don't understand how exactly it is being carried out." (P01, student Survey) While being asked the team project, he said,

> Well, the logic was kind of hard to follow through. Like so, like, for example, the user logs in, Callback would happen, I know that part. But after Callback happens what happens next, you know, like there, what happens like locally what happens in TupleSpace… I understand it, but when it comes to implementing and debugging I think debugging is very hard… I don't like, that doesn't make sense to me how can I debug the Callback. (P01, post-class interview, December 04, 2005)

This participant finished the Clicker assignment successfully, and his program was fully functional. But in the process of implementing the team project, as the internal distributed structure became much more complicated, he had difficulty writing the code, as he indicated in the post-class interview.

Case 2:

Another participant, P11, did not seem to understand how to use EventRegister and Callback when it was needed in their team project. Instead, he used Event Listener, Action Listener, and thread programming provided in Java to implement his own way of "Callback" to have a synchronized update for all client machines. It was surprising to see that he implemented a similar function that Callback already provided. It required a quite amount of work and the use of Java. It was also sad he did not realize such an existing feature in TSpaces that he should have used. During the post-class interview, when he was asked why not use EeventRegister and Callback, he implied that he forgot this feature so that he had to figure out his own way to implement it. He said, "I don't (didn't) know that, (at) that time. … if we change the code, even I knew the solution is to use Callback function, (if) we change the code, we can't meet our deadline." (P11, post-class interview, December 06, 2005)

Apparently, P11's ignorance of the feature was not solely because of his bad memory in remembering the EventRegister and Callback. It might be due to the fragile knowledge and neglect strategies (Perkins and Martin 1986) among novices programmers (P11 was new to TupleSpace programming). Actually, their team project was modified

from a well-written example Whiteboard from IBM TSpaces (IBM), in which the program used this Callback function to ensure a real-time update for every user's screen. Literature shows that the ability to write a program and the ability to read one do not corresponded (Gilmore 1991).

Most of the problems were revealed in programming because there were no complaints from participants when learning the concept. Most of their difficulty related to building the necessary distributed structure: a *receiver* for a desired event or/and a *handler* for that event. A receiver in a client computer monitors the desired event (triggered by the Tuple creation or deletion events from the TupleSpace server); whereas a handler tells the client computer what to do when the "event" actually happens. The failure in building such a program structure revealed that participants had difficulty imagining the interactions between client machines through TupleSpace. It was quite difficult for some participants, such as P01, to imagine what actions should be counted as "local" events in client machine and what should be counted as "remote" events in the TupleSpace at a certain moment. This problem also related to the insufficient understanding of the event-driven programming paradigm, a more general case of this concept. Participants also reported difficulty in debugging such a distributed structure. Section 6.3.1 further elaborates this.

In summary, participants who had no explicit Callback programming experience in other languages tended to encounter this difficulty, whereas participants who used the same or similar concept or feature in other programming language, such as PHP, thought that this feature was easy to understand and use.

5.2.4.2 Template Matching in Java

As described in Section 4.4.2, some participants had difficulty with the concept of Template Matching. The teaching curricula were changed accordingly to help participants master the concept. The researcher's observations showed that some participants especially had difficulty understanding the matches between nested Tuples, and some participants had problems applying the concept in the Java programming context, e.g., Serializable class Tuples. The survey results showed that the self-reported rating of using Templates Matching was 6.9 ("1" represents absolutely hard and "10"

represents absolutely easy). Four of 14 participants rated the concept less than or equal to 5. Interestingly, although participants remembered this as a challenge when they were asked explicitly about it in the survey, no one mentioned it spontaneously in the post-class interviews. In retrospect, for all participants, Template Matching was not too hard to learn. However, some participants needed more time and practice to master the concept and its use at the beginning of learning. As P02 said in the student survey, "This is possibly the hardest part of mastering Tuples, but once the basics of pattern matching are understood, it is easy to extract interesting solutions to problems with it." (P02, student survey)

5.2.4.3 Confused TupleSpace with Database Programming

As described in Section 4.5.1, a few participants thought they designed Tuples as if designing a database. One participant emphasized that he wanted non-duplicate fields in the Tuples and wanted the Tuples to be normalized as the tables in a relational database are. A few participants agreed with him. Comparing the use of Tuples and the tables in a relational database, the similarity is that both of them use referenced associative search, i.e., contents matching (Carriero and Gelernter 1989). Except this similarity, however, the design principles are almost opposite. The flexibility and strength of TupleSpace design come from the possible repetition with variance of similar structures. Whereas database design follows a very important principle, *normalization*, a systematic way of ensuring that the database is as compact as possible and the database structure is suitable for general-purpose use and free of certain undesirable characteristics — insertion, update, and deletion anomalies — that could lead to a loss of data integrity (Codd 1990). It is understandable that participants may have thought of databases when they were first introduced to TupleSpace Template Matching. However, the researchers were surprised by how strongly those participants expressed the thought that it was *wrong* to design this way. And indeed that participant was so offended by the idea that designing Tuples used repetition that he was never able to bring himself to make his Tuples perspicuous in their own terms. We tried to distinguish TupleSpace and database via the discussion of design purpose and system rational in class and in the course of project development (see Table 10). Through some struggle, by the end of the class most

participants' description of TupleSpace did not reflect the view of database any more. For example, one participant said,

> … it (TupleSpace) was like a grocery bag and everything is sent to it, like apples and oranges… Then it stays there, but then it is waiting for some item pulls it out, or just looks at it. That's how I see it, is just a container almost. …At first I have sort of more as like a database, like that, but when you actually use it and everything I see especially with the monitor (Space Monitor) and everything you see it's not really, kind of get away from that idea and as now I think it more as a just container of information of data. (P08, post-class interview, December 07, 2005)

5.3 Student Programmers Faced Technological Challenges in Forming Computational Model through TupleSpace

TSpaces is implemented in Java, so participants encountered TupleSpace through the lens of various programming aspects, including Java, GUI design (Swing and SWT in Java) and different programming tools (Space Monitor, Eclipse, and Jigloo). Although participants were senior undergraduates and first-year graduates in the Computer Science program, their first challenge was to gain enough skills with these programming concepts and tools so that they were capable of, for example, connecting to a TupleSpace and drawing a user interface (UI). This section reports the technical challenges in the course in developing multi-user, distributed, fined-grained, collaborative systems.

5.3.1 GUI Design in Java

More than half participants (8 of 14) had not previously programmed in Java. Therefore, we needed to teach Java at the minimum level to help them get started, as described in Section 4.4.1. The analyses (mainly from the student survey, post-class interview, and email and web discussion) showed that about half of participants reported Java-related problems in the first programming assignment (Clicker), and almost every

participant reported Java-related difficulty at different levels in implementing their team projects. Some problems were simple syntax errors or unfamiliarity with certain methods in Java libraries. These problems usually disappeared quickly when participants had more practice using Java. P02 reported in the student survey, "My only issue with the Clicker assignment was with Java (I have never programmed in Java before), and I had issues with the way the Clicker constructors…. The Clicker constructor didn't behave as I had anticipated." (P02, student survey)

Many problems were related to GUI design in Java with Swing classes. In the student survey, P09 reported the problems he faced in Clicker assignment,

> Understanding the GUI interface: I am a newbie in Java programming. I wasn't familiar with Java methods. …we found the method updateUI()….To find that, we spent two hours only for that single thing." Another participant, P14, reported, "…interaction between the GUI components…the coding becomes hard when dividing the GUI into several JPanels. (P09, student survey)

P10, one of two advanced Java programmers, said that his biggest challenge in the class was to help his partner learn GUI in Java,

> … one was getting the GUI together since he (refers to his group partner P14) was not a Java programmer before this. I think he is now. I tried to really lay it out as much as possible for him to make sure that he understood what was going on. I think that was a challenge. (P10, post-class interview, December 07, 2005)

Some of these problems were due to unfamiliarity with Java methods and the Swing library. More problems revealed that participants did not understand the fundamental mechanism behind GUI design in Java — the event-driven programming paradigm, which is further discussed in Section 6.3.1.

5.3.2 Programming Tools

5.3.2.1 Understood Complicated Features in Eclipse

Eclipse is a very powerful Java IDE tool. It has been widely used by many professional Java developers as well as in university classrooms (Storey, Damian et al. 2003; Brian 2006). The main purposes for using Eclipse in our class were to eliminate

routine programming tasks like compiling, take the advantage of many automatic features, e.g., automatically highlighting of errors and suggesting corrections, and manage programming files and documentation.

All of participants used Eclipse from the beginning of the semester for every programming assignment. The survey results (student survey, Q2.1-2.8 in Table 14) show that most participants thought that Eclipse was helpful in that it was relatively easy to detect errors, automatically fix errors, and automatically prompt methods and junctions for classes and objects. However, 40% participants rated "debug program or manage project (source) files in Eclipse" less than or equal to 5 with the mean values of 5.8 out of 10 for both cases (where "1" represents Absolutely Hard, and "10" represents Absolutely Easy). For example, P01 complained, "Debugging mode is hard. I think .Net environment is much easier." (P01, post-class interview, December 04, 2005) His team partner P03 also complained, "We have huge problems debugging in Eclipse. We had no clue how to debug any, even the debugger we couldn't use it very well." (P03, post-class interview, December 02, 2005)

The class videos and programming help sessions also recorded participants' opinion that Eclipse provided many features through menu and configuration options, and it was not always clear what options should be selected to accomplish a certain task. This complexity also resulted in participants being frustrated by some behavior. For example, P06 complained,

> We could only get the code working on Mac OS X, because on Windows it kept giving some error about not being able to find the main class. Based on information gathered from Google, it seemed that the problem laid in the Java virtual machine being installed to a directory with a space in it. Sadly, the installer for the JVM did not give me a choice of where to install the JVM. (P06, student survey, December 11, 2005)

Table 14. Survey Results Regarding the Use of Eclipse (Question 2)

Sub-questions	Response range	Average response	SD [a]
Q2.1 Detect error(s)	2-9	7.2	2.0
Q2.2 Automatically fix errors	2-10	7.7	2.2
Q2.3 Automatically prompt methods / functions for class / object	4-10	7.9	1.9
Q2.4 Debug	1-9	5.8	2.6
Q2.5 Manage project/source files	2-10	5.8	2.8
Q2.6 Jigloo (the Swing/SWT Editor), choose "Other" if you didn't use it and fill the reason	6-9[b]	7.6	1.1
Q2.7 Other function(s)/feature(s) you would like to comment	N/A	N/A	N/A
Q2.8 Overall, is Eclipse helpful for your programming in Java?	3-10	6.9	2.1

Note. "1" represents Absolutely Hard, and "10" represents Absolutely Easy

[a] SD: Standard deviation

[b] The results of Question 2.6 were based on 5 responses from whom used Jigloo.

A virtual development environment such as Eclipse can help programmers write their programming code effectively in various ways. However, the functionality provided in Eclipse was too complex for programmers who were first introduced to Java programming. Although our participants had used IDEs such as VC++ or other similar ones in the past, there was still a steep learning curve for Eclipse for those who had never used it before. In the student survey, P04 suggested, "The developers (of Eclipse) should have made common functions obvious and obscure ones hidden." P09 also suggested, "It would be great if Eclipse's method suggestion can also give us some brief explanation for that method when the user wants to know it." Other research (Storey, Damian et al. 2003) also suggested the use of a small portion of the functionality provided by Eclipse for novice Java programmers. The difficulty in debugging with Eclipse was also related to participants' ability to debug in a distributed programming environment, which is further addressed in Section 6.3.

5.3.2.2 Jigloo

Jigloo was picked from many Eclipse GUI plug-ins because of its dual supports for two popular Java GUI libraries, Swing and SWT, which were used by participants to develop UIs for various platforms. It had a high user rating (about 9) on the Eclipse website. Jigloo can create Swing and SWT source code and show the GUIs as they are being built. It can also generate source code to handle events.

Five participants (38.46%) used Jigloo and agreed that it provided advantages for generating the GUI code and related event handlers so that it simplified their programming in Swing or SWT. The complaints about Jigloo were that the speed of refreshing interface was slow, and it was not designed to support GUI development in handheld devices. For example, P06 (whose group developed a Math Bingo game in handhelds using SWT) said,

> Reworking the interface for the clients was incredibly slow, and took forever….Jigloo did not accurately display how SWT converted the GUI to the handheld, there were many problems such as button size and text size that had to be worked out. (P06, student survey, December 11, 2005)

5.3.2.3 TSpaces and Space Monitor

Space Monitor (see Figure 9) lets programmers watch the real-time changes of the server Space when a Tuple is added to, removed from, or changed in the Space. The modification of a Tuple and its fields can also be observed. The original objective of designing such a tool was to benefit programmers from several perspectives. First, it can help programmers check their imagination of how the Space dynamically changes through the insertion and removal of Tuples, which are not very easy to trace without such a tool. Second, it can help programmers coding in TSpaces to understand the structure and value of a Tuple and its fields in the Java programming language. For example, multiRead() method (equals to scan(), which will return a set of matching Tuples, e.g., tupleSet[2]) is not very straightforward to use because it involves nested Tuples and fields, a special way of retrieve the matching Tuples in Java. By using the Space Monitor tool and Eclipse to watch the variable changes, programmers can get a much clearer picture of how the complicated procedure is carried out.

This tool was recommended to participants when they started implementing the team project. At that time, program complexity greatly increased compared with the relatively simple assignments at the beginning of the semester. Participants who used the tool in programming thought that it was very useful and helpful. For example, P06 appraised the tool when he was asked if he had any trouble to use any APIs or functions in TupleSpace:

> Not too much really. One thing that like we did was used the TSpaces monitor…where you can monitor everything that was in TupleSpace, like that helped a lot I noticed. And like any problem that we had could easily be like found out like what we are doing wrong through that TupleSpace model there. So we didn't really have too many problems with that. (P06, post-class interview, December 04, 2005)

---

[2] The returned tupleSet itself is a Tuple type, thus each field in the tupleSet is actually a matching tuple. In order to get each tuple from the tupleSet, an Enumeration class must be used to extract each of the returned tuples. Then each element in the enumeration object needs to cast as a Field type, and then cast the Tuple by using getValue() of the field object. A detailed example of using multiRead() or scan() method can be found in the IBM TSpaces Programmer's Guide.

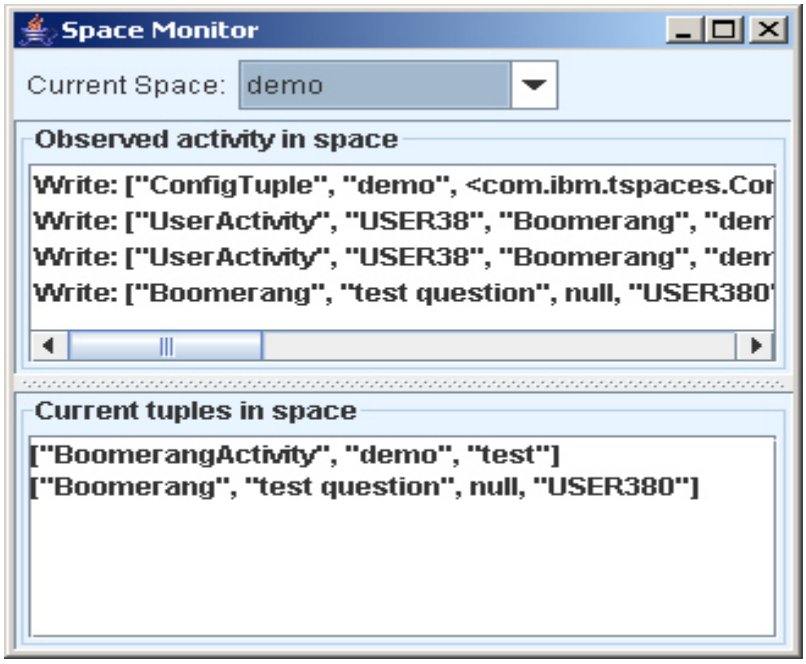


Figure 9. The screenshot of Space Monitor.

However, not every participant actually used it until they found unsolvable programming problems and came to the researchers near the end of the implementation phase. We would introduce the tool earlier and make its use mandatory if we taught the class again.

5.4 Chapter Summary

This chapter reports research findings about participants learning, design, and development experiences. It first presents three major categories of findings emerged according to the sub research questions. The analysis for individual interviews, class observations, surveys, documentation, and programming source code revealed participants' perceptions in relation to their experiences of learning and using TupleSpace to develop multi-user coordinated systems. Samples of quotations from participants were included to report the reality of the persons and situations studied.

The first category of findings was that participants faced challenges related to the understanding of the social and human aspects in a socio-technical system, most in the design stage. Participants took the design of computational features as an implicit design brief: almost every participant tried to maximize the use of computer without analyzing to determining if the features were necessary in a given context. Although participants were introduced to many examples of unusual possibilities for user experience, they brought assumptions from their previous experiences into design. These perception and assumptions did not initially include the envisionment of parallel structure and coordinated activities, which affected and limited the design of the target systems. Participants also faced challenges in envisioning coordination among multiple end users and the individual end user experience in design.

The second category of findings was about learning experience in using the coordination language to form the coordination model. Explicit modeling of coordination and interaction through TupleSpace provided opportunities for participants to concentrate on coordination and interaction issues in a multi-user coordinated system. Yet, as novice programmers in the socio-technical system development, participants also faced technical

roadblocks in TSpaces programming through class learning and software development. Some of the difficulties were related to the familiarity of specific language features and programming tools, while other difficulties further revealed participants' incompetence in imagining coordination and understanding some of the CS fundamental programming diagrams. Chapter 6 further discusses these issues.

The third category of findings was that participants encountered roadblocks while forming computational activities (i.e., creating non-coordinated portion of the program), such as GUI programming, through the learning and uses of Java programming language and libraries, IDE tools Eclipse and Jigloo, and Space Monitor. These difficulties may happen to general programmers as well, thus were considered normal. Yet they indicate particular challenges in learning multi-user, distributed computing programming.

The researcher believes these opportunities and challenges from above findings all affect participants' learning and programming experiences in designing and developing multi-user socio-technical systems. The next chapter, Chapter 6, explains what these findings mean, why programmers faced those challenges, the implications to HCI and computer science education, and the necessity to approaching parallel thinking.

Chapter 6

Discussion

This dissertation explored how intermediate to advanced CS students conceptualize and approach parallel thinking through learning explicit coordination modeling to design and develop coordinated computer systems to support parallel human activities. The purpose of this research work was to identify students' challenges in the process, analyze and understand the barriers they had, and further examine ways to help them. This research was guided by qualitative research methods, i.e., Grounded Theory and Research-design based theory. Multiple qualitative research data were collected through a semester-long advanced undergraduate class in the CS department at Virginia Tech. Participants were six senior undergraduate students and eight first-year graduate students in the CS program. The data were analyzed and synthesized according to the following research question and its sub questions.

Research Question:

"*What can we learn from the experience about how CS students conceptualize and approach parallel thinking?*"

Sub Questions:

RQ1: What are the challenges to students in designing parallel-distributed multi-user systems to support complex human interaction and coordination?

RQ2: How do CS students conceptualize the coordination model through a specialized coordination language?

RQ3: What are the technological challenges to students in building computational model through explicit coordination modeling?

These research questions were largely answered by the findings presented in Chapter 5 (Section 5.1 to 5.3). In summary, the findings in this study revealed that participants faced a web of sociological and technological challenges in designing and developing multi-user, distributed, fine-grained coordinated, interactive systems to support complex human coordination tasks. First, a contradiction occurred in participants' design process: student programmers showed insufficient understanding of the social and human aspects in socio-technical system design; instead of looking for ways to better understand those aspects, participants had a tendency to maximize the design of computational features and automated processes. Second, explicit modeling coordination with a specialized coordination language helped programmers concentrate on coordination and interaction issues. Yet the process of building the coordination model, computational model, and making a connection between two models was not effortless. Finally, the technological challenges in development were also part of the roadblocks that programmers had to conquer during the program design and development.

This chapter, Chapter 6, takes into consideration of the literature on general programmer cognition (programming psychology), distributed and collaborative system design and development, parallel thinking (multi-core education), and human-centered and value-based design principles in HCI. The implications of the findings are intended to augment the understanding of programmer cognition in the area of parallel-distributed, multi-user coordinated system design, promote an important mindset that CS students should have had to design and develop ubiquitous parallelism, and also shed light on further research on this area.

Section 6.1 summarizes participants' assumptions in coordinated interactive system design and tries to understand the relationship between the assumptions and the challenges participants faced. It further discusses the implications to HCI, CSCW, and general computer science education. Section 6.2 discusses participants' undeveloped imagination of coordination from three levels. Section 6.3 discusses the opportunities that the specialized coordination language, TupleSpace, could bring to students'

conceptualization and programming. It further argues that coordination should be a fundamental to any coordinated system design, whether human involved or not. Encouraging parallel thinking through explicit coordination modeling is an implication to multi-core and parallelism education. Section 6.4 highlights two major technological barriers that participants had, and argues that the related fundamental principles or paradigms may need more attention in CS education. The chapter concludes with a short summary in Section 6.5.

6.1 Engineering Thinking and Machine-centered Mindset Hinder User-centered and Value-based System Design

The findings in Section 5.1 describes that participants encountered various sociological challenges in the process of designing and developing multi-user coordinated interactive systems. This section further synthesizes these challenges, analyzes the causes, and then discusses the implications.

6.1.1 Going-in Assumptions in Coordinated System Design

Participants' initial design examples in Section 5.1.1 to 5.1.4 reflected their worldviews and experiences. They had assumptions about themselves, computer systems, and end users' coordination. These assumptions prevented them from understanding what they were taught in the class, what they needed to design for coordination, and further conducting effective and creative design and development.

First, participants assumed that their role in system design was to maximize computer utilization rather than designing appropriate technology for the situation. All initial design proposals had one or more computational features (e.g., system automatically correct for users) that were added into the system because the computer could do it. They unintentionally designed computer features to help users *control* or *monitor* a situation that are not necessary (e.g., teacher uses the computer to control the

game process, in Section 5.1.1.4), and thought that it was the way "it ought to be." It was not easy for participants to see that there could be reasons to refrain from using computer automation for tasks that could be done by computers. They emphasized efficiency and fairness, easily ignored human's intelligence, and thought that computers were more efficient and fairer than humans under all circumstances. It was difficult for them to imagine design alternatives.

Second, participants assumed that technology has intrinsic values. A system is valuable only when it can support the users' needs or system values. Unfortunately, it was hard for our participants to see that easily. For example, participants showed great enthusiasm in designing the instant messaging feature. However, they did not seem to understand the initial design purpose. An IM tool is a mediated communication method used for text-based communication between two or more people over any type of network connections. The purpose of using such tools was to support instant communication between people who are not in a co-located situation at the moment. If people are in a face-to-face environment, talking to each other directly is the natural form of communication (unless they use IMs for other purpose, e.g., sharing a long paragraph of text). Designing an IM for such a co-located environment does not really fulfill its purpose, but only can bring complexity to the system. Many participants had such a strong belief in that they persisted in designing an IM-like chat tool in their systems. Only after acting out their designed activities did they begin to question their assumption about the role of the feature they designed.

Third, participants assumed that external rewards such as timers and reward points lead to active user engagement (see examples in Section 5.1.2). It was extremely hard for participants to consider the idea of engagement as a moment-by-moment issue, apart from external rewards. Points and timers were seen as central to rather than peripheral to, action.

Furthermore, participants assumed that linear coordination mechanisms could support complex human coordination. They could not easily envision a range of possible human behaviors in a distributed and interactive environment. Therefore, they tended to design simple, linear coordination mechanisms to support complex coordination and

interaction. Often the result was that end users (e.g., players) were idle in an activity, bored and not joyful or engaged in fruitless activity.

6.1.2 Participants' View of Computing: Machine-centered

How participants perceived computing and themselves affected the systems they designed. The assumptions discussed in Section 6.1.1, especially the first two, reflect that participants' view of computing was mainly machine-centered, but not human-centered.

Tedre (Tedre 2006; Tedre 2008) summarized the view of machine-centered computing in the early years. Before the early 1990s, the fundamental question underlying all computing was "What can be automated?" (Forsythe, 1968, (Denning 1985) and "What can be (efficiently) automated and how?" (Denning, Comer et al. 1989). The first question, "What can be automated?", asks what can be, in principle, automated by any kind of machinery. It reflects the central question in computability theory, which studies what can be theoretically computed with any kind of machinery. The second question, "What can be (efficiently) automated and how?", asks how to automate various kinds of computational problems efficiently. It reflects the central question in computational complexity theory, which studies the amount of computing resources, such as time and storage. Both of these theoretical questions are fundamental to empirical research on computation and computers, and crucial to the design and implementation of computing systems. Also they both represent the view of "machine-centered" computing, which was dominant and widely accepted till early 1990s.

As computing becomes more ubiquitous and pervasive, almost every computer system requires certain forms of interaction with end users. Focusing only on machine itself is not sufficient any more. Grudin (Grudin 1990) pointed out that the focus in computing research has been gradually broadening from the machine and automation toward how and where computers are used, the actual activities of end users, and how end users collaborate and interact. Shneiderman (Shneiderman 2002) also argued that a clear shift from machine-centered computing toward human-centered computing (Lee 1989) has occurred. Human-centered views implicitly or explicitly incorporate the view

that technology has no intrinsic value, but that the value of any technology is measured by its impact on people. Recently, Tedre (Tedre 2006; Tedre 2008) argued that the fundamental question in computing should become "What *should* be automated?", but not limited to "What *can* be automated?" He further suggested that the idea of human-centered computing should, at least, "become a concept that people use to systematize, categorize, and collect other concepts." The researcher agrees with his suggestion because "this question may fundamentally belong to the domain of social sciences and applied philosophy," but as computer scientists, we at least should keep that direction and concentration in mind.

The design examples discussed in Section 5.1.1 (e.g., let a computer decide who goes next) show clear evidence that most participants still followed the idea of "machine-centered" computing: they had a tendency to maximize the use of computer throughout the design no matter what they were taught and what design alternatives were discussed in the experimental class. Although the researchers asked participants to consider the context of computer use, for example, end users may need to, and can, talk with each other while using computers in a face-to-face situation, when came into design, participants inclined to design IM-like computational features. And their reason was that this kind of features was available in many systems and could be implemented with TupleSpace. The efficiency arguments they made also reflect that they tended to think design values naturally from a computer system's perspective (i.e., efficiency, performance), but not from the end users' perspective. Bucciarelli's description of *object world* can further help understand the object world of the computer scientist:

> It is the object as they see and work with it that patterns their thought and practice, not just when they must engage the physics of the device but throughout the entire design process, permeating all exchange and discourse within the subculture of the firm. This way of thinking is so prevalent within contemporary design that I have given it a label—"object-world" thinking. (Bucciarelli 1994), p. 4

Furthermore, participants' view of computing also contributes to the way they perceived themselves, being software designers and developers, as people who should

utilize computers in a maximum way. They did not ask what should be automated very often and see it as needed. Using computers to automate things was a dominant idea in their minds. Such way of thinking has become "unintentional" thoughts that led their design and development process. This type of phenomena can often be seen in development society. One example is automated features, e.g., the automatic formatting feature in a word processing application. Rather than saving end users' time, this type of automation often causes unnecessary confusion and unpleasant user experiences.

This finding is not totally alone. Other researchers have also seen a similar phenomenon among industry software designers. For example, Norman found that designers often think of computer technology only "in terms of available technologies, instead of how people use those technologies" (Norman 1999). Shneiderman also saw that the challenge came from technology developers and designers who see no value in the user-centered approach to computing, and thus are resistant the user-centered way of thinking (Shneiderman 2002).

The current research thinks that this type of phenomena and resistance came from software developers' long-term thinking of "what can be automated with computers?" This fundamental question drove their view of computing and the perception of themselves, and also directed the way they designed computer systems. This may also suggest that CS curricula and many CS faculty members have not fully prepared for the shift from machine-centered to human-centered computing. The resistance to the human-centered view requires new foundational guidelines in CS education.

6.1.3 "Rigid" Engineering Thinking Constrains "Soft" Design Skills

The design issues described in Section 5.1 and participants' assumptions about designs also show their insufficient experiences in designing open-ended projects. This relates to the fact that our participants had often followed pre-defined technical specifications in most of class projects. When participants were asked to talk about interesting projects they previously developed, most participants described their experiences as "to implement requirements" in undergraduate classes. In particular,

when they were asked to explain how they went about designing and planning their programs, one participant said, "That's undergraduate class, so professors gave us some specifications. You know some specifications and some deliverables. […]" (P09, pre-class interview, Aug. 26, 2005) Another participant also said, "It was pretty much designed for us. We had a lot of specifications we had to follow." (P12, pre-class interview, Aug. 29, 2005) Furthermore, when participant P01 was asked "What do you consider your particular expertise?" he replied, "I don't consider myself as an expert programmer anything. So I don't even think myself a good programmer. But I guess I have my ability to figure stuff out in some way." (P01, pre-class interview, Aug. 26, 2005)

According to participants' description, their previous projects mainly required them to understand and fulfill a list of technical specifications that already created by the class instructors and professors. Participants' major task was to find a solution, ideally an optimal one, to the problem. What P01 meant by "figure stuff out in some way" most likely referred to the ability of implementing requirements. Participants were not involved in the process of generating such requirements. They were not required or encouraged to think of or ask "why" either in the process of implementing these specifications. This indicates and confirms the finding that most software developers' education and training mainly focused on implementing a solution to problems that reinforce "rigid" engineering thinking, but less to problems that also involves "soft" design and many other skills required to solve complex problems (Stubblefield and Carson 2007).

Robinson (Robinson 1998) discusses engineering thinking mainly focuses on "seeking good, and if possible, optimum, solutions, according to well-defined criteria" by "applying scientific knowledge and mathematical analysis" (e.g., in the CS context, applying data structure and algorithm knowledge to a certain problem). "Implementing specifications" is very much this kind of job. This ability is an important part of software engineering training, yet is far from enough. As well known in the computing community, software development has been defined as a process of solving "wicked" problems, as one that could be clearly defined only by solving a problem, or by solving part of the problem (Rittel and Webber 1973). Not like simple engineering problems,

most technology-related projects are about wicked problem solving. These problems cannot be solved in a traditional linear fashion, because the problem definition evolves as new possible solutions are considered and/or implemented. If most training in early years (in our case, three to four years in an undergraduate's) would focus on figuring out solutions for well-defined specifications, this way of thinking could have been limited to "finding a typical solution, optimum if possible." When design tasks had no indisputably correct answer, participants seemed to lose direction of their endeavor. Apparently, neither is this focus adequate for an industry development job (Begel and Simon 2008), nor for the increasing development need for ubiquitous and pervasive computing technologies.

Ideally, computer science education should prepare students for many skill sets that Denning and his colleagues proposed (Denning, Comer et al. 1989). Among many skills, design skill seemed an undeveloped one that needs the most attention. Compared with engineering skills, design skills require more creative thinking, and cannot be easily put in a list of rules. Training this type of skills requires long-term emphasis and concentration. If such training is inadequate, design will be most likely based on previous experience and intuition, yet participants' intuitive knowledge and skills had focused on machine-values instead of human-values.

6.1.4 Undeveloped Social and Human Science Knowledge Limited Participants' View about Computer Technology

Participants were considered as intermediate-to-experienced software programmers as most of them could have enough competence to develop complex computer systems. Yet participants showed underdevelopment in understanding social and human aspects when designing coordinated systems to support human's collaborative activities. Actually, it was common that they needed to learn these knowledge and perspectives as entering the new domain. Therefore, it was not surprising that they had difficulty understanding the human side of human-computer interaction and the socio side of socio-technical systems, such as insensitivity to situation and context as well as inadequate consideration of various user experiences and behaviors. Indeed, it was very understandable. However, what surprised most was that the strong tendency to design

computational features for maximum utilizing computational objects, and the resistance to think out of the object world. The inadequate training in social and human science may have also contributed to their limited view of computing and technology. Among 14 participants, only 15%, 15%, 35% students took sociology, art, and psychology classes respectively. And participants who took these classes did not show the adequacy of good understanding either. This implicates that engineering students should have more training in these related areas. It is also necessary for them to better understand the relationship between themselves, as technology designers, and the technology (systems they need to design). As future technology designers in our society, it is crucial to realize that their role is not only to design technologies, but design technologies that benefit human's life.

6.1.5 Other Possible Factors

There may be other possible factors that affected participants' learning. First, heavy daily use of technologies contributes to software developers' thinking and design priorities. IMing on computers and texting on mobile devices have been so popular (24/7) that there may be less value placed on face-to-face communication in young adults' minds. Pierce's study about social anxiety and technology reported: Among 280 high school students from a large western city, on average, 35-40% of participants reported using cell phones/text messaging and online social sites 1~4 hours daily, 24% reported using IMs 1~4 hours daily (Pierce 2009). Young adults communicate with friends or other people through technologies more than through face-to-face. Logging into an online system is so normal in the everyday world of college students and software developers that it is natural for them to design such features into any interactive systems. Media Equation theory explains why people unconsciously and automatically treat computers as human (Nass, Steuer et al. 1994). While the fact our participants thought "computers are *fairer* than people" indicates that they not only see computers as "social actors," but treat computers as special and powerful social actors who have privileges. With such a presumption, participants tended to offer computers more power, e.g., letting a computer to decide for the users. Computer technologies have affected people's life

and thinking dramatically in the past 20 years. Yet the consequence to technology designers may have a far-reaching effect.

Second, another explanation for the raft of difficulties our participants faced is provided by Hudson (Hudson 2009), who, based on a study of 441 men and women professionals working in the IT field, argues that people whose role was more technology-focused tend to have lower empathizing (EQ), thus have significant difficulty in recognizing and addressing the issues of users. His data show that men whose role was technological had significantly lower empathizing scores. The data also show that women on the average in his sample have higher EQ than men in general (which consistent with other studies). In current research, 13 out of 14 participants were male students, which might indicate why there was such as strong tendency to prioritize machine values.

Finally, of course, another possibility is that the learning sequence of the course and efforts of this research was seriously in error, that we failed pedagogically. This possibility does not, however, explain the phenomena. If these topics are so difficult to teach well that the bulk of our findings are caused by pedagogical failure, then it is in fact further demonstration of a systematic block to designer performance in these areas.

6.1.6 Implications to User-centered and Value-based System Design

Participants assumed their role as software designers and developers was to create computational features to maximize the use of computers. They mainly consider the system values from engineering perspective. They had relatively simple vision of how end users interact with each other through technology and how the user experience was affected by technology design. As a result, participants designed maximum automation and unnecessary features but simple coordination mechanism to support complex human interactions. This indicates that their previous training mainly focused on engineering values (e.g., efficiency, performance) but less on system values (e.g., to support collaboration). They did not have system-level thinking. This largely constrained their mind from considering the coordinated factors in designing coordinated systems.

Although our research is specifically about computer science students to design collaborative software, the data also illuminate how this can happen in general, though future research work is needed to support our claims. Iivari (Iivari 2006) and Viswanathan (Viswanathan 2008) described similar situations in software industry. Both studies show that professional HCI practitioners, who are responsible to "represent end users," often had difficulty communicating and training the professional software developers, who would eventually implement their design. Some examples were: software developers often ignore the design guidelines, do whatever they think is appropriate in design; they create functions without discussion and approval with HCI practitioners. This means that even after HCI practitioners finish user-centered design, the important detailed characteristics are often left to software developers' discretion. Although developers may be limited by development time, they may also have difficulty seeing the values that HCI professionals perceive and that permeate design guidelines. Without enough knowledge and techniques to comprehend user-centered design guidelines, developers cannot fully implement user-centered design. Instead, they accomplished their role as engineers by creating engineering values.

As discussed previously, the need for developing connectivity has made a transition from supporting raw machine-to-machine communication to human-to-human coordination and interaction. It is urgent and important that software developers realize the need and have the competence to build such systems. The implication from the research findings is that simply *telling* developers to focus on users' needs may not work in practice. Furthermore, neither HCI professionals nor end users can help software developers to envision the process of human coordination and interaction in the context of a single project. Setting new goals for HCI researchers and professionals might be needed. Explicit modeling human interaction and coordination may help developers.

6.2 Underdeveloped Imagination of Coordination Further Prevented System Level Thinking

6.2.1 Summary of Underdeveloped Imagination of Coordination

The findings in Section 5.2 show that participants had difficulty in envisioning coordination mechanism and processes between human and human with or without technology, modeling the coordinated behaviors, and further implementing the underlying program structure to support the coordination.

First, Section 5.1.3 discusses that participants lacked imagination of complex human coordination from the conceptual level in design stage. Many of them failed to envision end users and natural communication path as valuable system resources in the coordinated environment. End users in a collaborative environment should play an active role as a processer in a parallel computer, busy with their own work (e.g., performing a subtask), instead of idle and bored. Without a complete perception of user behaviors, most participants designed a simple strict turn-taking (sequential) mechanism to support the complex coordination. As a result, except one active user, all others are idle, waiting, because only one user could perform their tasks at a certain turn. From a coordinated system point of view, participants failed to take advantage of parallelism, end users' manpower and time are wasted. From an interactive system point of view, participants failed to think that many end users could be idle and bored, without engaging in any active participation. Participants also ignored the fact that direct conversion (social interaction) is a natural communication path in a face-to-face context. They also failed to provide a coordination mechanism either via storyboard or the design of the appropriate Tuple protocol, to support coordination.

Second, in the process of modeling coordination (Section 5.2.3), participants had difficulty envisioning the relationship between coordination model and computational model. Some participants could not separate coordinated activities from computational ones completely and caused potential display problems for shared resources; some failed

to identify shared resources as Tuples; and some had problems imagining "local" and "remote" events.

Third, in implementation stage (Section 5.2.4.1), participants manifested difficulty conceptualizing distributed structure and internal communication protocols of the coordination model through EventRegister and Callbacks (a request for the client program to be notified on the occurrence of certain events). About half participants experienced difficulty with EventRegister and Callback in the class. The underlying problem was that participants could not envision the sequences and flow (local vs. remote events) created by the combinations of normal TupleSpace operations, posting Callbacks, and receiving them.

The above summary shows that participants faced challenges imagining coordination at different levels: at the conceptual level, e.g., lacked imagination for user behaviors in coordinated activities; at the model level, e.g., had problems in separating coordinated activities from computational ones; and at the implementation level, e.g., encountered difficulty envisioning the sequences and flow created by the combinations of normal TupleSpace operations and distributed events. All these imply that participants had undeveloped imagination about coordination and its mechanism.

6.2.2 Understanding the Incompetence

Participants' incompetence in imagining human coordination may be partly due to the nature of the complexity of human activities. Suchman's Situated Action theory (Suchman 1987) thinks that human's action are highly contextualized, the context of specific situation determined what the next action is. Therefore, people's behavior in such situation is often undetermined. In CSCW, researchers often assume that social activity is "highly flexible, nuanced, and contextualized" (Ackerman 2000). Both views point out some nature of human social activities — complex and uncertain. This nature makes that imaging what end users may do while they interact with a system and each other is especially hard for engineering students who had not have many experiences in this area.

The challenges in modeling and implementing complex human coordination further lied in participants' insufficient understanding of the coordinated components and how they cooperate with each other. For example, participants needed to understand what would be shared, so that proper Tuple(s) could be designed. Then they needed to think how to share, so that the communication protocols and operations on the Tuple(s) could be defined. With Tuples and their protocols, the coordination model is formed. If not treating humans and computers as coordinated entities, when came into implementation, participants could easily get frustrated about local user vs. local machine, remote user vs. remote machine, and local vs. remote. Fundamentally, without concentrating on coordination issues in the system, programmers could get lost very easily.

6.2.3 Implications: Making Coordination and Interaction More Visible

Software infrastructures (such as toolkit, framework) are defined as "system-level software providing functions, capabilities, or services to other software" (Edwards, Newman et al. 2010). Many infrastructures that support distributed and collaborative work have made an assumption that programmers understand what users might be doing in a parallel-distributed environment. However, our participants' difficulty imagining complex human coordination and interaction processes indicates that the conceptualization itself should be supported. Failure of supporting the envisionment may lead to the failure of a promising product, such as Google Wave (Google 2009). It is arguable that the main reason for shutting down the service is that users could not conceptualize the complicated collaborative nature created by the waves, and thus cannot further imagine possible uses of the product.

Better understanding the nature of human and social activities in general requires longitude learning and practice. Our experience is that explicit modeling human coordination and interaction with a coordination language, such as TupleSpace, can help programmers' imagination. Infrastructures or tools that can support this need can ultimately allow developers, skilled or not in HCI, to run free in their exploration of truly sophisticated multi-way, human-centered design. Making interactions and coordination

more visible to developers would presumably improve the technical learning experience, and could allow other less articulate groups to improve faster. A participant whose group was particularly efficient described such visualization as follows:

> What I usually did is I just had two sides basically like one with one client, one with other client. And in the scope of Bingo, I would have the player's client and then we would have the picker's client. In the scope of some other thing where there is only one single client, I would still have two copies of it. And then basically in the middle, I have like arrows pointing from one to the other. That says like send this certain type of information and then they would have one going back that says like maybe respond with this type of information. Or I just basically have all the interactions between the different clients of what needed to be sent between the different people using the different clients. (P06, post-class interview, December 04, 2005)

This confirmed the idea that there is a great deal to be done in envisioning complex human interaction and coordination even with underlying support for coordination modeling.

6.3 Encouraging Parallel Thinking through Explicitly Modeling Coordination

The "complex and flexible" nature of human behaviors challenged the imagination of our novice coordinated system designers and developers. It was hard for participants, the engineering students, to think about how people actually coordinate and interact with each other to achieve a common goal, with or without computers. But the challenges were not solely because that the students were not told or taught about those social and human design issues. The challenges mainly come from the "engineering" thinking and "object" view, which emphasizes on engineering values and concentrating on designing computational features. They did not want to think out of the object box. Their thinking and views largely prevented participants from perceiving the system coordinated nature, which should be supported by their design. A specialized

coordination language like TupleSpace greatly helped them focus on the coordinated issues, instead of only thinking about computational "features." This section addresses the potential and opportunity that a coordination language can bring to computer science education in the future.

6.3.1 Benefits of Explicit Coordination Modeling with TupleSpace

The feedback about students' learning experience with TupleSpace was positive. All participants thought it was easy to learn and understand TupleSpace concepts. The analysis of participants' tuple design showed that TupleSpace helped all participants focus on coordination issues. All students produced working system, and were able to make decisions about the allocation and reintegration of work. Particularly, TupleSpace enables participants to (1) manage and allocate resources from a system level; and (2) abstract data for interaction from end-user level. The cognitive benefits of using TupleSpace, one among of many coordination languages, are summarized as Sections 6.3.1.1 to 6.3.1.5.

6.3.1.1 Virtual Space Represents Physical Space Analogues Naturally

Generally speaking, it is easy for people to conceptualize a multi-user physically shared space (Greenberg and Marwood 1994). This is because people can explicitly see the changes made by others and the physical constraints on particular types of user actions (e.g., the amount limits for certain cards in a card game). In contrast, when designing or using a computer-mediated shared space, people perceive the digital space as a single space containing distinct objects. Users see the space and objects through their own user interface. The changes in the space caused by one user's action are updated through network. Time delays can be expected when showing the actions for others. In such a circumstance, the properties owned by the physically shared space are still expected, but the physical constraints are a simulated, rather than a natural, property of the objects. In implementation, the computerized digital shared space and the objects inside usually have multiple copies, in different machines.

The advantage in TupleSpace programming is that the Space (containing Tuples) simulates the physically shared space naturally. The physical space is represented by the shared virtual space, Tuple Space. First, the different resources (the objects in the physical space) can be represented by different types of Tuples. Second, the amounts of each resource can be reflected by the number of each type of Tuple in the Space. Third, the operations (i.e., *read*, *in*, and *out*) on Tuples can be used to manage and allocate resources in the Space. For example, in the Apples-to-Apples activity, the designed Tuple Space contains different types and amounts of red-apple and green-apple cards, represented by different types of Tuples (e.g., RedAppleTuple, GreenAppleTuple, and CurrentGreenAppleTuple), with different properties. If a certain game has ten green apple cards, there will be ten instances of GreenAppleTuple class in the Space. And there is always one CurrentGreenAppleTuple instance in the Space to represent the current green apple card (adjective card) that players try to match. The Space and its operations enable programmers to simulate the manipulation of physical space and its resources, without the necessity of adding an extra layer of abstraction in programming. This also makes programmers manage the Space and allocate the shared resources from a system level.

6.3.1.2 Tuple Can Optimize the Grain Size in Coordination to a Right Level

TupleSpace helped participants to find appropriate level of data abstraction — defined a proper grain size for coordination. For example, in the Collaborative Crossword Puzzle activity, multiple players solved a puzzle by working on different clues. One person could take a clue and fill in the answer in the board while others worked on other clues. Thus, conflict could exist between two clues if they shared a cell in the grid. The design team changed the grain size of Tuple from "clue" to "cell" in the design stage, thereby simplifying the implementation of both the underlying data structure and the corresponding UI components.

6.3.1.3 TupleSpace Model Hides Parallel and Networking Details from Programmers

TupleSpace (and some extensions in TSpaces) hid many implemental details, such as sockets and threads. The communication between machine and machine was supported by the model as well, but without programmers' explicit concern. Similarly, many general parallelism terms such as race condition and locking rarely appeared in the class discussion. These simplifications helped programmers concentrate on facilitating coordination and other important issues in the design and development.

6.3.1.4 Tuples Builds Distributed Data Structure in MVC Model

Multi-user systems require that a user interface shows relevant information both from one's own and others' actions. Tuples simplify this at the UI level by reifying the important coordination elements. Most Tuples are used to represent public or shared system resources, and thus generally relate to certain UI components in the screen, as those shown in Figure 8. From a MVC model point of view, Tuples construct the model as a distributed data structure. When any shared resource (defined as a Tuple) is changed by one user, the TupleSpace is responsible for distributing the Tuple change in the Space. The model change leads to the view change in UI components.

6.3.1.5 Modeling Coordination Facilitates the Transition from a Novice to an Expert

As discussed in Section 2.1.1.2, program design involves knowledge from at least two domains, the application (or problem) domain and the computing domain. Programmers construct at least two types of mental models and establish a mapping between these two models. Mental model is "a representation which is capable of supporting mental simulations of the design in progress." Constructing a mental model goes through "a progression from an abstract to a concrete state" (Adelson and Soloway 1985). Building the bridge between two domains can be quite complex in some cases. For the target systems participants needed to build, i.e., the parallel-distributed multi-user systems that support complex face-to-face interaction and coordination, the mapping seemed not easy, even for senior and graduate CS students. TupleSpace model provided participants the tools to create mental models with two states: at the abstract state, it helps

to construct the coordinated entities and the protocols to communicate between them; and at the concrete state, it helps to construct data structure and how to manipulate those data in the Space.

TupleSpace model also facilitated programmers to build up a program plan — "a hypothesized mental structure corresponding to a *stereotypical program component*, such as a conditional expression or a counter mechanism" (Rosson 1996) — of similar programs. Although all team projects were different, the systems were all coordinated programs, thus had the similar coordinated nature. A stereotypical program structure, creating and manipulating Tuples, is a common structure when programming this type of programs.

Using TupleSpace model can also increase the role expressiveness (Green 1991) in a software program. In other words, a TupleSpace-based program can reflect the purpose of software components more explicitly. And the Tuples and its protocol reflect the coordinated nature of a program.

The ability to construct abstract and concrete mental model and conduct plan-like behaviors shows a process of developing expertise and taking the step towards an expert in the area (Adelson and Soloway 1985). Programming in TupleSpace model helped programmers to make such a transition.

6.3.1.6 Summary

TupleSpace, as the first coordination language, has been used to successfully solve different coordination problems in the past thirty years in parallel computing (e.g., (Carriero and Gelernter 1988; Whiteside and Leichter 1988)), distributed computing (e.g., (Omicini and Zambonelli 1998; Cabri, Leonardi et al. 2000; Lehman, Cozzi et al. 2001; Murphy, Picco et al. 2003)), and in Human Computer Interaction (HCI) (e.g., (Johanson and Fox 2004)). The concept was implemented as network middleware, e.g., TSpaces by IBM, JavaSpaces by Sun, in the past as well. Most of these efforts treated TupleSpace as an underlying infrastructure to support the distributed structure. Instead, the current research work explored TupleSpace as an infrastructure to support human coordination. By using TupleSpace, the student programmers were able to create coordination model to

explicitly address human coordination and interaction issues in a computer-mediated shared virtual space. The Space simulates the virtual space in a natural way in the sense Tuple reflects the existence of objects and the physical limitations of the objects. Tuple operations can be used to manage and allocate shared resources and support communication and interaction between end users. Participants were able to design and develop at a high level, while many details of connectivity at network level can be hidden from programmers.

In summary, a coordination language like TupleSpace encourages programmers to concentrate on coordination and interaction issues, more explicitly, in a natural and clean way.

6.3.2 Implication for CS Education in Multi-core and Parallelism

The rapid development of multi-core technologies and ubiquitous computing has made parallelism ubiquitous. Computer scientists need to be able to design and program for both parallel computers and parallel humans. As discussed in Chapter 2, the most recent pedagogical trend was to integrate parallelism into the entire CS curriculum (Graham 2007; Ernst and Stevenson 2008; Ernst, Wittman et al. 2009), from the entry-level courses. These were all good steps of training students with a parallel thinking mindset. However, to teach parallelism early, most curricula had to accelerate in teaching the basic programming, such as Java in the first month of the class, so students can learn multi-thread programming. Forcing parallel constructs could actually complicate the transition to parallel thinking in the sense that students may not learn either sequentiality or concurrency well. Furthermore, Wrinn (Ernst, Wittman et al. 2009) pointed out that curriculum design in parallelism education cannot keep up with the rapid development of hardware. Educators are trying to anticipate the state of parallel systems. SIGCSE community is still looking for ideas and solutions to "re-frame parallelism using simpler concepts, preferably ones that students can think about and understand in a natural way.

The findings from the current research suggest that even quite advanced CS students did not initially have enough competence imagining the parallel structure and coordinated activities. They designed the system based on their assumptions about coordinated system. However, unfortunately, their assumptions did not initially include the envisionment of the coordination nature of the system. They designed computational features and computer automation to reflect their engineering values, but not the system values. This suggests that the early focus should be on ubiquitous parallelism, and concentrate on coordination, a fundamental aspect of parallelism, rather than relatively mutable technical details. Parallel thinking, as defended below, should be encouraged through explicit coordination modeling:

a mindset that enables computer scientists to think about and implement computer systems that allow activities to happen concurrently, including parallel processing, network programming and multi-user systems — any system that involves the distribution and reintegration of work.

We believe that the difficulty imagining coordination underlies much of the pedagogical challenge. If coordination is understood well, other aspects of parallelism (syntax, optimization, efficiency, control, etc.) will be seen as solutions to the problems posed by the basic situation.

6.4 Fundamental CS Principles Need More Attentions

Besides the sociological roadblocks and challenges in imagining coordination, participants also faced technological challenges in building the coordination model and computational model through TupleSpace as described in Sections 5.2.3, 5.2.4, and 5.3. The following sections discuss two major challenges and their implications to computer science.

6.4.1 Event-driven Programming Paradigm

As described in Section 5.2.4.1, about half of the class had difficulty implementing distributed structure with EventRegister and Callback feature in TSpaces. Section 5.3.1 also describes that some participants thought it was hard to implement the interactions of GUI components. Both of Callback and GUI implementation employed a common model in computer science, event-driven programming paradigm. In an event-driven programming paradigm, the flow of the program is determined by events (e.g., user actions like mouse clicks and key presses) or messages from other programs or threads. The kind of programs usually requires two parts: event detection and event handling. This concept has been widely used in many fields of computing and is a key idea in GUI development in OOP (e.g., Java, VC++, Visual Basic), concurrency in operating systems, and distributed events over networks. It was one of five programming fundamentals in 2001 Computer Science Curriculum Guidelines (2001).

All participants had exposed to event-based programming environment before our class, either in an Operating System class or GUI development in OOP. However, a student may have used the concept to implement an event-based user interface function, such as "after a button is clicked, display the contents," but could not have realized the essential concept behind. Few realized that the Callback feature actually uses the same idea either. Their difficulty suggests that even quite advanced CS students may need opportunities to better understand event-driven programming systematically, in a deep way. Only utilizing it in applications may not be sufficient. Some participants especially had difficulty debugging in a distributed even-driven programming structure.

6.4.2 Model-View-Controller Design Pattern

Model-View-Controller (MVC) design pattern (Gamma, Helm et al. 1994) is a standard programming practice abstracting the user interface from the underlying data structure and representation. It was developed as part of the Smalltalk environment and consists of three parts: the Model is the application object, the View is its screen presentation, and the Controller defines the way the user interface reacts to user input. With this design pattern, it was not necessary to design different data structures (Model)

for different user interfaces (View). However, in the design assignment "Tuple Learning Activity," (Section 4.5.1), a couple of participants proposed that different Tuples should be designed for different user interface. It was surprising to know that they had not heard about MVC in the past. Teaching students the higher level programing design patterns, such as MVC, may be necessary.

6.4.3 Summary

In the process of implementing computer-based multi-user coordinated activities, participants had confusion and misunderstanding (e.g., event-driven programming and regular expression), especially when applying the concepts in the Java programming context. Most of the knowledge had been taught through different classes before participants took our class. Yet their ability to use these concepts in a relatively complex situation was insufficient. Many participants lacked the competence to recognize the basic concepts, decompose a large problem into smaller units, and further integrate them. This suggests that many of these important aspects especially event-driven programming diagram should be taught more intensively, possibly in an advanced class and in a more comprehensive way. The difficulty also suggests that teaching parallelism has dependencies to other CS topics.

6.5 Chapter Summary

This chapter illustrates the multifaceted and complex nature of computer science students' learning experience in the course of designing and developing parallel-distributed systems to support multi-user coordination and interaction through explicit coordination modeling. The chapter focuses on understanding the possible causes that participants faced from design to implementation. It also discusses the implications for computer science education and software development. The endeavor of analyzing the findings was to produce a nuanced and multitiered, but holistic and integrated synthesis to help understand the phenomenon.

Section 6.1 summarizes the participants' assumptions about computers, themselves, and end users in interactive system design. Those assumptions reflected participants' view of machine-centered computing and rigid engineering thinking. That further explains why participants had strong tendency to maximize the use of computers and automate different process as much as possible, but often ignored the social context that computers would be used in and the important coordinated features that a system should support. Section 6.2 discusses participants' difficulty in imagining multi-user coordination in design and development and also analyzes the incompetence. The implications for CS education and software development tools and environment are also discussed. Section 6.3 summarizes the benefits and opportunities that a coordination language like TupleSpace model can bring to encourage parallel thinking in CS education. Section 6.4 highlights two CS fundamental principles and paradigms in collaborative system development that may need more attention.

The next chapter, Chapter 7, draws the conclusion of this research, makes suggestions for CS education, software infrastructure and tools development, and further research direction. It ends the dissertation with a brief final reflection.

Chapter 7

Conclusion

The ultimate goal of this doctoral dissertation is to promote a crucial mindset, parallel thinking, in computer science, through explicit coordination modeling. The researcher thinks it is essential for every computer scientist, especially for every computer science student, to have such a mindset to design everyday connectivity in the world. This qualitative study explored how six senior undergraduate and eight first year graduate CS students perceived and approached parallel thinking in building coordinated systems to support parallel face-to-face activities through a specialized coordination language. The conclusions from this study follow the research questions and findings and therefore address the following areas: (1) the cognitive challenges in designing parallel-distributed multi-user systems to support complex human coordination and interaction, (2) the CS students' conceptualization of explicit coordination modeling through a specialized coordination language, and (3) the technological challenges in building computational model in coordinated system design. Following is a discussion of the major findings and conclusions drawn from this research. The discussion is followed by the researcher's suggestions and a final reflection on this study.

7.1 Conclusion

7.1.1 Cognitive Challenges in Designing Coordinated Systems

The first finding from this research is that participants, as intermediate-to-advanced computer science students, faced cognitive challenges in conceptualizing the target socio-technical system, i.e., the parallel-distributed multi-user coordinated systems, to support parallel face-to-face interaction. As novices in coordinated system design, participants in this study could not easily understand the social and human aspects in the target system. That is participants initially: (a) had difficulty envisioning user coordination with or without computers, (b) had difficulty envisioning coordination and individual user experience, and (c) brought assumptions about user experiences into design. Although participants were introduced to many examples of unusual possibilities to support coordination and user experience, they could not conceptualize the high-level system goal as "to facilitate multiple users to achieve a common goal." Instead, they took the design of computational features as an implicit design brief: almost every participant tried to maximize the use of computer without analyzing to determining if the features were necessary in a given context. They designed those features mainly because (a) the features could be implemented with given technologies (e.g., hardware and TupleSpace), (b) most current networked and interactive systems had those "standard" features, and (c) computers are fairer and more efficient than humans. The researcher thinks that these challenges were related to participants' education background and previous experiences in software design and development.

First, most participants' view about computing largely reflected the "machine" aspect of the target system, but not the "human" aspect. They had strong tendency to maximize the use and the automation of computer, without first thinking of or asking the necessities. It seemed that their most-often-asked questions in software design and development was "what can be automated and how to do that efficiently." This perception limited participants from seeing the values that a target system should support and further understanding the coordination between users and individual user experiences. It is very important for participants (and everyone working in computing field) to realize

that technology innovation is not about creating computer systems to replace human intelligence, but about extending it to create new possibilities.

Second, most participants were not exposed to open-ended projects adequately thus did not show system-level thinking in design and development. Most projects that participants previously did were to implement a list of well-defined specifications, with the concentration on engineering values and an ultimate goal of finding optimized solutions. They were not required and encouraged to ask how the specifications were created, and did not see a necessity to ask the reasons for creating these specifications either. They often thought the target system in terms of objects and computational features, but not the coordinated nature of the system. They valued the target system from the aspects of efficiency, performance, and effectiveness, without careful thinking about if these values reflected the user or system values. As a result, they were reluctant to think out of the "object box" to find design alternatives, did not treat a system as a whole to design a solution, and thus could lose track of their endeavors easily. Even though participants' future job is to implement specifications, as many professional software engineers do, they should still have a high-level system goal in mind to direct their program design and implementation. System-level thinking is important for solving any "wicked" problems in the real world. In this research work, in designing parallel-distributed multi-user systems, explicit coordination modeling helped participants stay on system-level thinking.

A conclusion to be drawn from this finding is that CS students should have a high-level system thinking, parallel thinking, to guide and help them to concentrate on the coordination nature of the system. Without this mindset, they could not able to spontaneously make appropriate design choices to create systems to support complex human coordination.

7.1.2 Conceptualization in Explicit Human Coordination Modeling

The second finding from this research is that most participants started with an undeveloped imagination for human coordination. Explicit coordination modeling

method facilitated participants focusing on the coordinated nature of the target system, from design to implementation. All participants successfully designed parallel-distributed systems to support parallel human activities. Yet the process was not effortless. Most participants initially did not consider the coordinated issues of the system, but only thought of computational features and computer automation. By using a coordination language to explicitly model human coordination, participants were able to think about the design issues they should have focused on. Furthermore, participants could not imagine the parallel user behaviors in collaborative activities, so they designed a sequential turn-taking mechanism for the parallel users, but failed taking users as a useful system resource. In the modeling stage, some participants had initial difficulties in distinguishing the coordination and computational models — one Tuple was used in both models at different times or contained mixed information of different resources. Some participants had difficulty envisioning the sequences and flow created by the combinations of normal TupleSpace operations and distributed events, confused by "local" and "remote" events. By constantly working on coordination explicitly, all teams produced working systems with coordination models by the end of the semester. The number of Tuples decreased from initial design to implementation. This indicated that participants' better conceptualization in constructing coordination model. It also indicated that interaction was actually simpler than they originally imagined. Focusing on coordination did simplify the coordinated system design. TupleSpace model provided a forum for learning, as one participants said "… for Tuples you are not abstracting data, but you are abstracting data that is for interaction." (P14, post-class interview, December 04, 2005) The approach kept students in the zone of proximal development (ZPD). This means that they were working at a level that was difficult for them, but not too difficult. We saw the growth in parallel thinking: participants were able to better imagine coordination and make decisions about the allocation and reintegration of work and how to design for it. TupleSpace model also helped participants make the transition from a novice to an expert in coordinated system design and development.

The research finding also shows that TupleSpace model itself was overall easy to learn and understand. The basic concept, template matching, was not hard to understand. But practice for its use in Java programming language was needed. Some participants

were confused with TupleSpace and database design at the beginning because they both involve content matching. A clarification for its design purpose and rational can clarify the confusion.

One conclusion to be drawn from this finding is that modeling human coordination explicitly by a specialized coordination language such as TupleSpace can foreground the coordination and coordination issues. I believe that better understanding of coordination and having the ability to imagining the coordinated activities can lay the groundwork for more complex parallel and distributed thinking.

7.1.3 Technological Challenges in Building Computational Model in Coordinated System Design

The third finding from this research is that participants had difficulty applying some of the CS fundamental knowledge and principles in building computational model. The difficulty in event-driven programming paradigm was revealed from their use of EventRegister and Callback feature in TSpaces and implementation of interactions of GUI components. Although all participants had exposed to event-based programming environment through previous class and programming experiences, few realized that the Callback feature and GUI design actually use the same idea. This suggests that even advanced CS students need opportunities to better understand event-driven programming systematically, in a deep way. It is also important that students know how to debug in such a distributed even-driven programming structure. This can be a common challenge for any distributed system design and implementation, thus more attention is needed when teaching related contents in CSCW, CSCL, and distributed computing courses.

In addition, some participants had misunderstanding of model-view-controller design pattern and had difficulty with regular expression parsing in Java programming context. These challenges raise questions about the nature of earlier parts of CS curriculum. Although some have changed since the experimental class, should students learn more about regular expression parsing in entry level class? The experimental class, at least, gave a chance for us to reinforce these important principles and knowledge in CS.

Furthermore, these challenges suggest that teaching ubiquitous parallelism has dependencies on other CS topics. If moving it earlier in the curriculum, careful thoughts should be put into these issues.

7.2 Suggestions and Future Work

This section summarizes the suggestions for CS education and software development from this research work. Future work is also briefly discussed.

7.2.1 Suggestions for CS Education

To be a computer science professional, it is very important to be able to think inside the object "box." However, only focusing on the box is not sufficient. Computer science education needs to emphasize the edges and the outside of the box. CS students should be exposed to more open-ended, "wicked" problems, rather than implementing well-defined specifications. Soft design skills and rigid engineering skills are both needed for solving real world problems. So professors can give students more problem finding and definition related projects.

Teaching parallelism early and broadly in the CS curricula is important. It is essential to train CS students to have "parallel thinking," so they can spontaneously think about and create support for parallel activities. This research suggests that the early education should be on ubiquitous parallelism and focus on coordination, an underlying concept of parallel, distributed, and multi-user programming. Furthermore, explicit modeling coordination is crucial to coordinated system design and development and can foreground the coordinated issues. The original TupleSpace (not TSpaces) is a simple model that does not require any programming experience, thus can be used in CS1 classes to discuss related issues at the conception and abstract level. By examining the coordination model, we can easily check students' perception about a given coordinated issue. Design guidelines, such as "Tuples should only contain resource that shared by multiple users," can also be given to students. In addition, familiar social coordination

activities such as collaborative games can be used to catalyze students' imagination of coordination in a natural way.

7.2.2 Use and Develop Software Infrastructure and Tools

The emphasis on education is important, but requires long time to see its impact. Software infrastructure or tools, that can provide system-level facilitate, can be developed as well to directly support the imagination and conceptualization process of complex human coordination and interaction. The infrastructure of tools can ultimately allow developers, skilled with human or social knowledge or not, to freely explore truly sophisticated multi-way, human-centered design. Making interactions and coordination process more visible to developers would improve their technical learning experience, and help their implementation and debugging processes.

7.2.3 Future Work

I suggest the following future work. First, this research work defined and created a scope for "parallel thinking." Future research work can focus on finding the detailed skill sets and knowledge that should be included in. Second, this research work identified that coordination should be a fundamental topic in teaching ubiquitous parallelism. This laid the groundwork for more complex parallel and distributed thinking. Future work can explore how coordination relates to the detailed concepts and techniques in parallel programming and distributed programming. Third, teaching parallelism depends on other CS topics. Future work can think about different strategies and approaches, but either the technique has to avoid dependencies on difficult prior issues or it has to embrace teaching them as part of learning about ubiquitous parallelism. Finally, researchers in HCI may create a new coordination language, specialized to human interaction and coordination, which may provide more powerful and direct abstraction of human behaviors.

7.3 Final Reflection

This research work chose TupleSpace, one of the specialized languages for coordination, to teach explicit coordination modeling. It was chosen for various reasons to serve the purpose of this research work as stated in Section 2.3.1. It hinted at the idea that prior understanding and beliefs about a particular topic shapes people's thoughts. By choosing TupleSpace, we have chosen one coordination language, but also have chosen one vocabulary. Then what about others, those languages and models for coordination and concurrency were discussed in Section 2.3.2? I imagine all of them would help participants focus on thinking of some aspect of coordination or coordinative effects. Yet, I think that the focus would be different, in the way of helping programmers to think of the different aspects of a system. TupleSpace model is a data-oriented model, which makes programmers think more about system resources and their exchange. Process-oriented models seem to let programmers think more of controlling processes to facilitate communication to synchronize individual work, or configuration of the system; and the details about resources are hidden inside a process. Petri-net-like models emphasize temporal sequence of events and do not explicitly express resources or processes; they help programmers see how states change as time passes in a concurrent system. This is very useful for supporting the design of more efficient systems, but not necessarily for thinking about overall functionality. Dataflow models let programmers think of currency in terms of the dependences of data values, and therefore actually downplay thought about the coordination itself. Concurrency is everything. It is possible that combining more different kinds of representations would help students learn better. Or, alternatively, introducing too many different views, along with the systems of representation required to use them might just confuse students. Regardless, I believe that the findings from this research still stand: participants would still need to imagine how individual computational activities cooperate through the allocation and reintegration of the individual work; and explicit coordination modeling can foreground the coordinated issues. Future research work is needed to identify the differences and examine how the differences will affect further learning.

Wing argued that everyone, not only computer scientist, should be trained to have a universally applicable attitude and skill set called "computational thinking." (Wing 2006) She explained the concept with a list of "what it is, and isn't." She said, computational thinking is "conceptualizing, not programming," "fundamental, not rote skill," "a way that humans, not computers, think," "ideas, not artifacts." The process of doing this doctoral research and writing this dissertation has made me agree with this vision. Yet I also see that the path to achieve the goal was not effortless. Even for our advanced-level computer science students, the engineering thinking and their object world had pushed them towards the "isn't" list rather than the "is" list. Explicit modeling coordination using TupleSpace model helped them focus on coordination issues. However, TupleSpace model cannot totally free students' engineering thinking to understand broader values of different systems. Our students need to be exposed to more creative design, thinking at the level of systems, not just fulfilling a technical specification. Professors in multi-user system design need to challenge their students to envision complex human interaction and coordination more often. And professors in CS need to challenge their students' design ideas more often.

References

. from http://www.eclipse.org/.

. "IBM TSpaces Version 3 Client Programmer's Guide." Retrieved March 24, 2009, from http://www.almaden.ibm.com/cs/tspaces/Version3/ClientProgrGuide.html.

(2001, December 15). "Computing Curricula 2001 - Computer Science Volume." Retrieved March 31, 2009, from http://www.sigcse.org/cc2001/.

Abrahamson, L. A. (1999). Teaching with a classroom communication system: What it involves and why it works.

Abrahamson, L. A., A. Davidian, et al. (2000). Wireless calculator networks: Where they came from, why they work, and where they are going. 13th Annual International Conference on Technology in Collegiate Mathematics, Atlanta, GA.

Ackerman, M. S. (2000). "The Intellectual Challenge of CSCW: The Gap Between Social Requirements and Technical Feasibility." Human-Computer Interaction **15**(2): 179-203.

Adams, J. C., D. J. Ernst, et al. (2010). Multicore education: pieces of the parallel puzzle. Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE'10), Milwaukee, Wisconsin, ACM, New York, NY.

Adelson, B. and E. Soloway (1985). "The Role of Domain Expenence in Software Design." IEEE Transactions on Software Engineering - Special issue on artificial intelligence and software engineering **11**(11): 1351-1360.

Agerwala, T. and Arvind (1982). "Data Flow Systems." Computer: 10-11.

Apple Computer Inc., C. (1995). Changing the Conversation about Teaching, Learning & Technology: a report on 10 years of ACOT research.

Beck, K. (2000). Extreme programming explained: Embrace change, Addison-Wesley Professional.

Begel, A. and B. Simon (2008). Struggles of new college graduates in their first software development job. Proceedings of the 39th SIGCSE technical symposium on Computer science education, SIGCSE'08, Portland, Oregon USA ACM.

Ben-Ari, M. (2004). A suite of tools for teaching concurrency. Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education (SIGCSE'04), Leeds, United Kingdom, ACM, New York, NY.

Bishop, P. and N. Warren (2002). JavaSpaces in Practice, Addison Wesley Professional.

Bollella, G., S. Graham, et al. (1999). Real-time TSpaces. Industrial Electronics Society, 1999. IECON '99 Proceedings. The 25th Annual Conference of the IEEE, San Jose, CA USA.

Bransford, J. D., J. J. Franks, et al. (1989). New approaches to instruction: Because wisdom can't be told. Similarity and analogical reasoning. New York, NY, USA, Cambridge University Press**:** 470-497.

Brian, H. (2006). "Using Eclipse in the classroom." Journal of Computing Sciences in Colleges **21**(3): 118-127.

Brooks, R. E. (1980). "Studying programmer behavior experimentally: the problems of proper methodology." Communications of the ACM **23**(4): 207-213.

Brown, A. L. (1992). "Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings." Journal of the Learning Sciences **2**(2): 141-178.

Bucciarelli, L. L. (1994). Designing Engineers, The MIT Press.

Burns, N., S. K. Grove, et al. (2004). The Practice of Nursing Research: Conduct, Critique, and Utilization, Elsevier Health Sciences.

Cabri, G., L. Leonardi, et al. (2000). "MARS: a programmable coordination architecture for mobile agents." Internet Computing, IEEE **4**(4): 26-35.

Cabri, G., L. Leonardi, et al. (2000). "Mobile-agent coordination models for Internet applications." Computer **33**(2): 82-89.

Carriero, N. and D. Gelernter (1988). Applications experience with Linda. Parallel programming: experience with applications, languages and systems, New Haven, Connecticut, United States, ACM Press.

Carriero, N. and D. Gelernter (1989). "Linda in Context." Communications of the ACM **32**(4): 444-458.

Carriero, N. and D. Gelernter (1990). How to write parallel programs: a first course, MIT Press.

Codd, E. F. (1990). The Relational Model for Database Management, Addison-Wesley.

Creswell, J. W. (2002). Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, Sage Publications, Inc.

Curtis, B. (1984). Fifteen years of psychology in software engineering: Individual differences and cognitive science. Proceedings of the 7th international conference on Software engineering. Orlando, Florida, United States, IEEE Press.

Curtis, B. (1986). By the way, did anyone study any real programmers? The first workshop on Empirical studies of programmers. Washington, D.C., United States, Ablex Publishing Corp.**:** 256-262.

Curtis, B. (1995). "Objects of Our Desire: Empirical Research on Object-Oriented Development." Human-Computer Interaction **10**(2&3): 337-344.

Curtis, B., E. M. Soloway, et al. (1986). "Software psychology: the need for an interdisciplinary program." Proceedings of the IEEE **74**(8): 1092-1106.

Davies, S. P., D. J. Gilmore, et al. (1995). "Are Objects That Important? Effects of Expertise and Familiarity on Classification of Object-Oriented Code." Human-Computer Interaction **10**(2&3): 227-248.

Denning, P. J. (1985). "The Science of Computing: What is computer science?" American Scientist **73**(1): 16-19.

Denning, P. J. (2004). Great principles in computing curricula. Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE'04), Norfolk, Virginia, USA, ACM, New York, NY.

Denning, P. J., D. E. Comer, et al. (1989). "Computing as a discipline." Communications of the ACM **32**(1): 9-23.

Denning, P. J. and J. B. Dennis (2010). "The resurgence of parallelism." Communications of the ACM **53**(6): 30-32.

Denzin, N. K. and Y. S. Lincoln, Eds. (2000). Handbook of Qualitative Research, Sage Publications, Inc.

Detienne, F. (1995). "Design Strategies and Knowledge in Object-Oriented Programming: Effects of Experience." Human-Computer Interaction **10**(2&3): 129-169.

Detienne, F. (2001). Software Design-Cognitive Aspects. New York, USA, Springer.

Dunican, E. (2005). A Framework for Evaluating Qualitative Research Methods in Computer Programming Education. The seventh Annual Workshop of Psychology of Programming. Brighton, England.

Edwards, W. K., M. W. Newman, et al. (2010). The infrastructure problem in HCI. Proceedings of the 28th international conference on Human factors in computing systems (CHI' 2010), Atlanta, Georgia, USA, ACM.

Ernst, D., B. Wittman, et al. (2009). Preparing students for ubiquitous parallelism. Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE'09), Chattanooga, TN, USA, ACM, New York, NY.

Ernst, D. J. and D. E. Stevenson (2008). Concurrent CS: preparing students for a multicore world. Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE'08), Madrid, Spain, ACM, New York, NY.

Eugster, P. T., P. A. Felber, et al. (2003). "The many faces of publish/subscribe." ACM Comput. Surv. **35**(2): 114-131.

Freeman, E., S. Hupfer, et al. (1999). JavaSpaces Principles, Patterns, and Practice, Addison-Wesley.

Gamma, E., R. Helm, et al. (1994). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional.

Garrett, J. J. (2005). "Ajax: A New Approach to Web Applications." Retrieved Oct. 28, 2009, from http://www.adaptivepath.com/ideas/essays/archives/000385.php.

Gelernter, D. and A. J. Bernstein (1985). "Generative Communication in Linda."

Gelernter, D. and N. Carriero (1992). "Coordination languages and their significance." Communications of the ACM **35**(2): 97-107.

Gilmore, D. J. (1991). Expert Programming Knowledge: A Strategic Approach Psychology of Programming. J. M. Hoc, T. R. G. Green, R. Samurcay and D. J. Gilmore, Academic Press**:** 223-234.

Google. (2009). "Introducing Google Wave." Retrieved Nov. 28, 2011, from https://sites.google.com/a/pressatgoogle.com/googlewave/.

Graham, J. R. (2007). "Integrating parallel programming techniques into traditional computer science curricula." SIGCSE Bulletin **39**(4): 75-78.

Green, T. R. G. (1991). Programming Languages as Information Structures. Psychology of Programming. J. M. Hoc, T. R. G. Green, R. Samurcay and D. J. Gilmore, Academic Press**:** 117-137.

Greenberg, S. and D. Marwood (1994). Real time groupware as a distributed system: concurrency control and its effect on the interface. Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW' 94), Chapel Hill, North Carolina, United States, ACM.

Greenberg, S. and D. Marwood (1994). Real time groupware as a distributed system: concurrency control and its effect on the interface. Proc. CSCW 1994, ACM Press.

Gregory, R. A. (1982). Distributed programming languages. Proceedings of the ACM '82 conference, ACM.

Grudin, J. (1990). The computer reaches out: the historical continuity of interface design. Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people, Seattle, Washington, United States, ACM.

Grudin, J. (1994). "Groupware and social dynamics: eight challenges for developers." Commun. ACM **37**(1): 92-105.

Guion, L. A. (2002). Triangulation: Establishing the Validity of Qualitative Studies. Gainesville, FL, University of Florida.

Halstead, M. H. (1977). Elements for software Science. New York, Elsevier North-Holland.

Hartree, D. R. (1950). Calculating Instruments and Machines, Cambridge University Press.

Hewitt, C., P. Bishop, et al. (1973). A Universal Modular Actor Formalism for Artificial Intelligence. International Joint Conference on Artificial Intelligence.

Hoadley, C. M. (2005). "Design-based research methods and theory building: A case study of research with SpeakEasy." Educational Technology **45**(1): 42-47.

Hoare, C. A. R. (1978). "Communicating sequential processes." Communications of the ACM **21**(8): 666-677.

Hoc, J. M., T. R. G. Green, et al., Eds. (1991). Psychology of Programming. Computers and People Series, Academic Press.

Hudson, W. (2009). Reduced empathizing skills increase challenges for user-centered design. Proceedings of the 27th international conference on Human factors in computing systems (CHI'2009), Boston, MA, USA, ACM Press, New York, NY.

IBM. "Whiteboard Example in TSpaces." Retrieved March 31, 2009, from http://www.almaden.ibm.com/cs/TSpaces/html/examples.html#whiteboard.

Iivari, N. (2006). Understanding the work of an HCI practitioner. Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles, Oslo, Norway, ACM Press, New York, NY.

Irvin, R. K., P. Marian, et al. (2001). "Editorial: empirical studies of programmers." International Journal of Human-Computer Studies **54**(2): 185-188.

Johanson, B. and A. Fox (2004). "Extending tuplespaces for coordination in interactive workspaces." Journal of Systems and Software- Special issue: Ubiquitous computing **69**(3): 243-266.

Johnson, R. B. (1997). "Examining the Validity Structure of Qualitative Research." Education **118**(2): 282-292.

Karen, J. R., R. C. Curtis, et al. (2000). WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation. Proceedings of the 22nd international conference on Software engineering. Limerick, Ireland, ACM Press.

Kim, K. (2007). THE EFFECTS OF HANDHELD NETWORK SERVICE "LOOK" ON THE ACQUISITION OF COMMON GROUND. PhD, Virginia Tech.

Lee, E. (1989). Some suggestions on a computer science undergraduate curriculum. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers. (COMPCON Spring '89), San Francisco, CA.

Lehman, T. J., A. Cozzi, et al. (2001). "Hitting the Distributed Computing Sweet Spot with TSpaces." Computer Networks: The International Journal of Computer and Telecommunications Networking **35**(4): 457-472.

Lehman, T. J., S. W. McLaughry, et al. (1999). TSpaces: The Next Wave. Hawaii International Conference on System Sciences (HICSS-32), January 1999.

Lewandowski, G., D. J. Bouvier, et al. (2010). "Commonsense understanding of concurrency: computing students and concert tickets." Communications of ACM **53**(7): 60-70.

Lewis, C. (1982). Using the "thinking-aloud" method in cognitive interface design. . Yorktown Heights, NY, IBM T. J. Watson Research Center.

Lin, S. and D. Tatar (2011). Encouraging Parallel Thinking through Explicit Coordination Modeling. The 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11), Dallas, Texas, USA.

Lin, S., D. Tatar, et al. (2006). Learning When Less is More: "Bootstrapping" Undergraduate Programmers as Coordination Designers. Proc. PDC2006.

Lin, S., D. Tatar, et al. (2006). Learning When Less is More: "Bootstrapping" Undergraduate Programmers as Coordination Designers. 2006 Participatory Design Conference (PDC2006), Trento, Italy.

Marshall, C. and G. B. Rossman (2006). Designing qualitative research. Newbury Park, CA, Sage.

Marshall, J. M. (2002). Learning with technology: Evidence that technology can, and does, support learning. Cable in the Classroom. San Diego, CA.

Maxwell, J. A. (1996). Qualitative Research Design: An Interactive Approach, Sage Publications (CA).

McCabe, T. J. (1976). "A complexity measure." IEEE transactions on software Engineering **SE-2**(4): 308-320.

McDonald, C. (1992). Teaching concurrency with Joyce and Linda. Proceedings of the twenty-third SIGCSE technical symposium on Computer science education (SIGCSE'09), Kansas City, Missouri, ACM, New York, NY.

Miles, M. B. and M. Huberman (1994). Qualitative Data Analysis: An Expanded Sourcebook, Sage Publications, Inc.

Miller, G. (1956). "The Magical Number Seven, Plus or Minus Two:Some Limits on Our Capacity for Processing Information." The Psychological Review **63**: 81-97.

Milner, R., J. Parrow, et al. (1992). "A calculus of mobile processes, I." Information and computation **100**(1): 1-40.

MURATA, T. (1989). "Petri Nets: Properties, Analysis and Applications." PROCEEDINGS OF THE IEEE **77**(4): 541-580.

Murphy, A. L., G. P. Picco, et al. (2003). LIME: A Coordination Middleware Supporting Mobility of Hosts and Agents.

Nass, C., J. Steuer, et al. (1994). Computers are social actors. Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence, Boston, Massachusetts, United States, ACM.

Nielsen, J. (1990). Paper versus computer implementations as mockup scenarios for heuristic evaluation. Proc. INTERACT 90, North-Holland Publishing Co.

Norman, D. A. (1999). The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution, The MIT Press.

O'Reilly, T. "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software." 2009, from http://oreilly.com/web2/archive/what-is-web-20.html.

Omicini, A. and F. Zambonelli (1998). "Co-ordination of mobile information agents in TuCSoN." Internet Research: Electronic Networking Applications and Policy **8**(5): 400-413.

Papadopoulos, G. A. and F. Arbab (1998). Coordination models and languages. ACM.

Patton, M. Q. (2002). Qualitative Research & Evaluation Methods. Thousand Oaks, CA, Sage Publications, Inc.

Perkins, D. N. and F. Martin (1986). Fragile Knowledge and Neglected Strategies in Novice Programmers. Empirical Studies of Programmers, Washington, DC.

Pierce, T. (2009). "Social anxiety and technology: Face-to-face communication versus technological communication among teens." Computers in Human Behavior **25**(6): 1367-1372.

Rallis, S. F., S. F. Rallis, et al. (2003). Learning in the Field: An Introduction to Qualitative Research, SAGE Publications Inc.

Rittel, H. and M. Webber (1973). "Dilemmas in a General Theory of Planning." Policy Sciences **4**: 155-169.

Robbins, S. (2003). Using remote logging for teaching concurrency. Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE'03), Reno, Navada, USA, ACM, New York, NY.

Robinson, J. A. (1998). "Engineering thinking and rhetoric." Journal of Engineering Education **87**(8): 227-229.

Roschelle, J., P. Schank, et al. (2005). From Response Systems to Distributed Systems for Enhanced Collaborative Learning. 13th International Conference on Computers in Education (ICCE 2005). Las Vegas, USA.

Roschelle, J., D. Tatar, et al. (2007). "Ink, Improvisation, and Interactive Engagement: Learning with Tablets." Computer **40**(9): 42-48.

Rosson, M. B. (1996). "Human factors in programming and software development." ACM Computing Surveys **28**(1): 193-195.

Rosson, M. B. and J. Carroll, M. (2002). Usability engineering: scenario-based development of human-computer interaction, Morgan Kaufmann Publishers Inc.

Salinger, S., L. Plonka, et al. (2007). A Coding Scheme Development Methodology Using Grounded Theory for Qualitative Analysis of Pair Programming. The19th Annual Psychology of Programming Interest Group Workshop(PPIG2007), Joensuu, Finland.

Scholtz, J. and S. Wiedenbeck (1990). "Learning second and subsequent programming languages: a problem of transfer." International Journal of Human-Computer Studies **2**(1): 51-57.

Sheil, B. A. (1981). "The Psychological Study of Programming." ACM Computing Surveys (CSUR) **13**(1): 101-120.

Shirky, C. (2003). A group is its own worst enemy. O'Reilly Emerging Technology Conference, Keynote.

Shneiderman, B. (1986). Empirical studies of programmers : the territory, paths, and destinations. Workshop on Empirical studies of programms. Washington D.C., Ablex Pulishers, Norwood, NJ.

Shneiderman, B. (2002). Leonardo's Laptop: Human Needs and the New Computing Technologies, MIT Press.

Soloway, E. (1986). What to do next: meeting the challenge of programming-in-the-large. The first workshop on Empirical studies of programmers. Washington, D.C., United States, Ablex Publishing Corp.**:** 263-268.

Sommerville, I. and G. Dean (1996). "PCL: a language for modelling evolving system architectures." Software Engineering Journal **11**(2): 111-121.

Storey, M.-A., D. Damian, et al. (2003). Improving the usability of Eclipse for novice programmers. Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, Anaheim, California, ACM Press.

Strauss, A. and J. Corbin (1990). Basics of Qualitative Research: Grounded Theory Procedures and Techniques, Sage Publications, Inc.

Stubblefield, W. A. and T. L. Carson (2007). Software design and engineering as a social process. Ext. Abstracts CHI 2007, ACM Press.

Suchman, L. (1987). Plans and Situated Actions: The Problem of Human-Machine Communication (Learning in Doing: Social, Cognitive and Computational Perspectives). New York, NY, USA, Cambridge University Press.

Sun Microsystems Inc. An Introduction to JavaSpaces Technology.

Tatar, D., J.-S. Lee, et al. (2008). Playground games: a design strategy for supporting and understanding coordinated activity. Proc. DIS 2008, ACM Press.

Tatar, D., S. Lin, et al. (2008). Playground Games and the Dissemination of Control in Computing and Learning. Educating Learning Technology Designers: Guiding and Inspiring Creators of Innovative Educational Tools. C. DiGiano, S. Goldman and M. Chorost. London, UK, Routledge, Taylor and Francis Group.

Tatar, D., J. Roschelle, et al. (2003). "Handhelds Go to School: Lessons Learned." Computer **36**(9): 30-37.

Tedre, M. (2006). What should be automated?: The fundamental question underlying human-centered computing. Proceedings of the 1st ACM international workshop on Human-centered multimedia (HCM'06), Santa Barbara, California, USA.

Tedre, M. (2008). "What should be automated?" Interactions **15**(5): 47-49.

Vahey, P., D. Tatar, et al. (2004). Leveraging Handhelds to Increase Student Learning: Engaging Middle School Students with the Mathematics of Change. ICLS, Santa Monica, CA.

Vahey, P., D. Tatar, et al. (2006). Using Handheld Technology to Move Between the Private and Public in the Classroom. Ubiquitous Computing in Education: Invisible Technology, Visible Impact. M. v. t. Hooft and K. Swan. Mahwah, NJ, Lawrence Erlbaum Associates.

Viswanathan, S. (2008). "Increasing design buy-in among software developer communities." CHI '08 extended abstracts on Human factors in computing systems: 2067-2070.

Weinberg, G. M. (1971). The psychology of computer programming, New York, Van Nostrand Reinhold.

Whiteside, R. A. and J. S. Leichter (1988). Using Linda for supercomputing on a local area network. Conference on High Performance Networking and Computing archive, Proceedings of the 1988 ACM/IEEE conference on Supercomputing, Orlando, Florida, United States, IEEE Computer Society Press.

Whiteside, R. A. and J. S. Leichter (1988). Using Linda for supercomputing on a local area network. Proceedings of the 1988 ACM/IEEE conference on Supercomputing, Orlando, Florida, USA, IEEE Computer Society Press.

Wing, J. M. (2006). "Computational Thinking." COMMUNICATIONS OF THE ACM **49**(3): 3.

Wyckoff, P., S. W. McLaughry, et al. (1998). "T Spaces." IBM Systems Journal **37**(3): 454-474.

Appendix A. Informed Consent Form

IRB# _____

Informed Consent to Participate in Research
Virginia Tech

You are being asked to participate in a research study associated with CS4984 or EDCI4984. This form provides you with information about the study. The Principal Investigator (the person in charge of this research) or his/her representative will also describe this study to you and answer all of your questions. Please read the information below and ask questions about anything you don't understand before deciding whether or not to take part. Your participation is entirely voluntary and you can refuse to participate without penalty or loss of benefits to which you are otherwise entitled.

Title of Research Study: Design and Use of Tuple-Space Architectures for Classroom Connectivity and Pedagogy

Principal Investigator:
Deborah G. Tatar, PhD,
Associate Professor of Computer Science and, by courtesy, Psychology
Member, Center for Human-Computer Interaction
660 McBryde Hall, MC 106
Blacksburg, VA 24060
(540)231-8457, Tatar@vt.edu.

Funding source: SubContractor to SRI International on NSF/IERI Grant #REC-0427783

What is the purpose of this study? This study will involve all students who enroll in CS4984 or EDCI 4984 in Fall Semester, 2005. Upper level undergraduates in Computer Science, Education and related fields will be invited to participate.

The purpose of this study is to test a computer architecture for classroom connectivity at two levels: (a) at the level of programming applications for use in pedagogy and, if possibly, (b) at the level of using the tasks in pedagogical settings. Using our Tuple-based architecture, you will work in teams, with a client if available, to design and program classroom activities which will if time permits be used in actual learning settings, either K12 or university, during the semester, or if time does not permit, in subsequent semesters with your involvement if possible.

What will be done if you take part in this research study?

You will participate in all phases of the design and implementation of a classroom pedagogical activity and, as time permits, in the initial testing of the activity.

You will be introduced to pedagogy principles, to the idea of Pattern Languages in programming, and to our experimental development system, including sample programs. After some initial exercises in programming and teamwork, you will form up into groups of 3-4, be guided through the process of finding a target customer (e.g. a professor or local teacher who has a problem he/she would like solved) guided through planning, implementing, testing, and revising a

classroom task. Each task will be tried with the class. If possible, you will also use and study the task in the pedagogical context for which it was created. If there is not time during the semester and maybe if there is, you will be encouraged to participate in subsequent research involving your team's task. You will write up your experience. The programs at various stages of development, starting from conceptualization to use, and the write-ups will constitute evidence about the long-term viability of this system for undergraduate programming. Programmers at VT and at SRI International will provide support for system functions during the semester. Background information about you will also be collected in order to characterize your prior experience and knowledge.

You will be graded on teamwork, class participation, responsibility especially in the timely articulation of problems, design, problem solving, delivery of a working program or a very good reason why not, response to critique, working well with a customer, and the write-up. You will not be penalized in grades for failure of the pedagogical success; however, you will be penalized for failure to make steady progress throughout the semester, letting your team members or customer down through negligence (not showing up at meetings, not either meeting deadlines or delivering an excuse, etc.), or not working with the other team members. With luck, there will be a good balance of computer science and education students. Computer Scientists will have a special challenge in working with people who may not be technical and in listening to the education student's expertise in other areas. Education students will have a special challenge in understanding the requirements and timeline in technology development. Both will have the chance to build systems that can make a real difference in classroom experience of children and/or university students.

What are the possible discomforts and risks?

Risks from participation in this study are minimal. If the system, currently under development, proves difficult to use, you will have substantial support from the two graduate student assistants, the professor, and the backup team from SRI International, where the development team consists of three highly experienced software engineers and three pedagogical experts.

Anonymity and confidentiality will be maintained in treating all data. No names will be used in write-ups.

If you wish to discuss the information above or any other risks you may experience, you may ask questions now or call the Principal Investigator listed on the front page of this form.

What are the possible benefits to you or to others?

The benefits are many. Challenges with the system, in working with clients, or working with teammates can provide important experiences relevant to the work world you are about to enter. You will benefit from working on multidisciplinary teams, in which you must articulate your disciplinary assumptions, goals, and limitations. Education students will have the opportunity to learn what is easy and what is hard to do with computation, and computer science students will have to work on simple, clear expression. Although students learning computer science in years past had ample opportunity to experience working with experimental systems, it is currently a

rare opportunity that provides excellent preparation for the kinds of debugging, testing, and development jobs that are your most likely first jobs out of college. Additionally, it is important for computer science undergraduates preparing for life-long learning to have the experience of (i) writing a real program for use by others and (ii) facing new and different technical challenges while doing so. The education students will have the opportunity to work with a team (Tatar and her colleagues at SRI International) that has had a real impact on math and science teaching and learning through many past and on-going projects.

Of course, benefits include those of helping people learn.

If you choose to take part in this study, will it cost you anything?

No.

Will you receive compensation for your participation in this study?

No. You will receive course credit for the class and a grade as for any other class.

What if you are injured because of the study?

There is no physical risk in this study.

If you do not want to take part in this study, what other options are available to you?

Participation in this study is entirely voluntary. You are free to refuse to be in the study, and your refusal will not influence current or future relationships with Virginia Tech or any project personnel. However, participation in the class is tied to willingness to be in the study.

How can you withdraw from this research study and who should you call if you have questions?

If you wish to stop your participation in this research study for any reason, you should contact: Deborah Tatar at (540)231-8457. You are free to withdraw your consent and stop participation in this research study at any time without penalty or loss of benefits for which you may be entitled.

You may, of course, drop the class, subject to Virginia Tech's rules, and would thereby be dropped from the study.

If you choose to withdraw from the study, but not from the class, you will be graded the same way as other students and receive the same course credit; however, you may have to work on a project by yourself or make other adjustments, if no other similar students are available. If working on a project by yourself, your grade on teamwork may suffer.

Throughout the study, the researchers will notify you of new information that may become available and that might affect your decision to remain in the study.

In addition, if you have questions about your rights as a research participant, please contact David M. Moore, IRB Chair/ CVM, Phase II (mail code 0442); (540) 231-4991; moored@vt.edu.

How will your privacy and the confidentiality of your research records be protected?

Authorized persons from Virginia Tech and the Institutional Review Board have the legal right to review your research records and will protect the confidentiality of those records to the extent permitted by law. If the research project is sponsored then the sponsor also have the legal right to review your research records. Otherwise, your research records will not be released without your consent unless required by law or a court order. You will be assigned an alias, and the mapping of alias to your real name will be kept in Dr. Tatar's office. Your materials will be photocopied or printed without your name for analysis during research. These materials will be kept in a locked area and checked out for analysis. They will be viewed only by researchers working on this project.

If the results of this research are published or presented at scientific meetings, your identity will not be disclosed.

Will the researchers benefit from your participation in this study?

Professor and student interests in any intellectual property (IP) developed in this course or commercialization thereof is subject to Virginia Tech's, SRI International's and the NSF's regulations and policies. Virginia Tech will have an interest in any and all IP developed in the context of this class. **It is highly unlikely that anything of financial value will be developed during this course**, nor is the development of financially valuable IP a goal of this class, the project or the study. However, if something of value is developed, negotiations will take place about financial interest similar to those in any partnership. The professor, graduate students, and undergraduate team members would generally be listed on any patent applications or patents.

Signatures:

As a representative of this study, I have explained the purpose, the procedures, the benefits, and the risks that are involved in this research study:

_____

Signature and printed name of person obtaining consent Date

You have been informed about this study's purpose, procedures, possible benefits and risks, and you have received a copy of this Form. You have been given the opportunity to ask questions before you sign, and you have been told that you can ask other questions at any time. You voluntarily agree to participate in this study. By signing this form, you are not waiving any of your legal rights.

Printed Name of Subject Date

Signature of Subject Date

Signature of Principal Investigator Date

Appendix B. Demographic Information Sheet

Participant Demographic Information Sheet

This form is solely for the use of gathering demographic information for TupleSpaces research project in class **CS4984/EDCI4984: Designing Distributed, Networked Handheld Activities for Learning**. This information will help our research group to describe the characteristics of the people who participate in our study. Please answer each question by either entering your answer, or selecting an appropriate response that describes your situation. Thank you once again for participating in our study.

Participant Name/ID #: _____

Date: _____ **Time**: _____

Age: _____ **Gender**: [ ] Male, [ ] Female

High School: _____

Native Country: _____ **Ethnicity:** _____

Length of Time in the United States if International: _____

Current Student Status: [ ] Senior, [ ] Master's, [ ] Ph.D., [ ] Other _____

Undergraduate Major (and Minor if any): _____

If you are a graduate student, please answer:

- **Undergraduate University:** _____
- **Current Graduate Program:** _____
- **Length of Time in Graduate Program**_____

Please list devices (e.g., Desktop/Laptop/Tablet/Handhelds/Cell Phone) you own/use currently and in the past:

	Own Device(s)				Device(s) at Work		
	Type	**Operating System**	**Programming (Y/N)**	**Owning now? (Y/N)**	**Type**	**Operating System**	**Programming? (Y/N)**
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

Please answer the following questions concerning your Previous Programming/Teaching Experience. We would appreciate as much detail as you are willing to provide.

(1) Do you have any programming experiences?

[ ] Yes (Please answer questions: a ~ b)

[ ] No (Please answer questions: c)

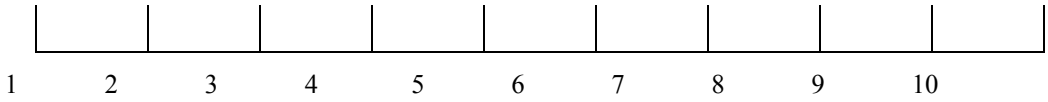
a. How many years programming experiences have you had overall (including school and job)? Please describe them briefly.

b. Please list three programming languages that you have most expertise in, and how many years you have used them.

(1) _____, _____Years
(2) _____, _____Years
(3) _____, _____Years

c. If you answer "No" to (1), please answer: How do you describe your computer proficiency?

Absolutely Beginner Absolutely Expert



(2) Do you have any teaching experiences?

[ ] Yes (continue answer the following questions: a and b)

[ ] No (skip the following sub questions)

a. How many years teaching experiences do you have? Please describe them shortly, e.g., which kind of classes did you teach, how many students in your class.

b. When you teach, have you ever thought about to use technology to facilitate your teaching? If you have done so, which kind of system did you use?

Appendix C. Semi-structured Face-to-Face Interview Protocols

C1. Semi-structured Face-to-face Interview Protocols (Pre-Class Interview)

SEMI-STRUCTURED INTERVIEWS PROTOCOL FOR TUPLE CLASS (PRE-CLASS)

(CS4984/EDCI4984: Designing Distributed, Networked Handheld Activities for Learning, Fall 2005)

Introduction

As you know, part of the purpose of this class is to study the properties of TupleSpaces architecture for programmers and pedagogy designers. We are engaging in several activities to assess these:

(1) We'll require you to frequently write reaction log and check-in your code;

(2) We'll video record certain class discussions;

(3) We'll monitor or check progress and difficulties as they arise during the class;

(4) And we'll interview all the people in this class at the beginning and then again at the end of the class to track their views, experiences, and expectations, which are important components of the success or failure of the project. We would like to record the interview in order to be able to accurately represent what you said. The tapes (actually digital files) will be heard only by the people on the research project who are involved in analyzing the success of the class. Your name will NOT appear in any published context associated with any opinions or statements you make during this interview. Of course, you are free to withdraw at any time from the interview if you are not comfortable and to refrain from answering any question that you do not wish to answer. You are under no obligation to tell us why and no punitive measures will be taken.

You've signed the consent form for the study, and here is a copy for you to keep for your reference. And do you have any other questions before we start? And shall I start to record?

I. Past Programming Experience

1. How did you first get interested in programming?

2. What do you consider your particular expertise?

3. Now we'd like to talk with you about particular programming projects that you have worked on. What projects have you worked on? If you have worked on a lot of projects, please describe the important ones, those that you have mainly worked on the design

(Note; for more experienced people, say graduate students, ask about *important* projects and the projects they have mainly worked on the design.

a. Iterate over each project as below. However, start with any collaborative projects and any education related projects.

i. Tell us about the program/project. What did it do? How did it work? What were the important classes?
ii. How did you go about Designing and planning the program?
iii. Did you work with other people on it? (Have you every worked with other people on a project?) How did that work out? Who did the work?
iv. Were there any API's (or other dependencies) you had to learn about to write the program? How did you do that? (What's your way to learn new APIs)
v. Did you ever have to revise or rethink your original idea? How did that happen? What did you learn (if anything)?
vi. Are there any other interesting programs & projects you've worked on?

b. The tuple project is of course concerned with education. Have you ever done any education research? Taught? Tutored? Tell us about those experiences.

4. If they haven't talked about the Network Programming, ask them if they have any and what they have.

5. Please describe how do you usually plan your project?

II. Prior Experience / Expectations with /for Tuples

1. What are your goals for the class? In other words, why do you want to take this class?
2. Have you ever heard of TupleSpaces before this class? And what's your impression during the 1st week of the class about TupleSpaces?

   If they know something about it, ask the following questions:

   a. In your view, what are TupleSpaces good for? What are your Hopes and Goals for TupleSpaces pedagogy?
   b. Are there any programs that you imagine writing with TupleSpaces? Pick one. What would it do? Tell me about it.
   c. How would you go about planning it / writing this?
3. What's important to you (if anything) about pedagogy?

Are there any other thoughts or questions about the class or research you would like to share or ask me before we end this interview?

Thanks for your participation again!

C2. Semi-structured Face-to-face Interview Protocols (Post-Class Interview)

SEMI-STRUCTURED INTERVIEWS PROTOCOL FOR TUPLE CLASS (POST-CLASS)

(CS4984/EDCI4984: Designing Distributed, Networked Handheld Activities for Learning, Dec, 2005)

Introduction

As you know, part of the purpose of this class is to study the properties of TupleSpaces architecture for programmers and pedagogy designers. As we mentioned at the beginning of the class, we will conduct another interview at the end of the class to track the programmers' experiences of learning and using TupleSpaces to design collaborative systems. Again, to let you know, we would like to record the interview in order to be able to accurately represent what you said. The digital files will be heard only by the people on the research project who are involved in analyzing the success of the class. Your name will NOT appear in any published context associated with any opinions or statements you make during this interview. Of course, you are free to withdraw at any time from the interview if you are not comfortable and to refrain from answering any question that you do not wish to answer. You are under no obligation to tell us why and no punitive measures will be taken.

Do you have any other questions before we start?

I. Group Project

1. Project design/redesign, implementation, teamwork, and general questions:
   - i. Tell me about your project. What does it do?
   - ii. How did you start to design the program?
   - iii. Did you ever have to revise or rethink your original idea? How did that happen? What did you learn (if anything)?
   - iv. Let's spend some time talking about the tuples you designed (from the last version of their design proposal). Let's go through each tuple, and please explain to me (1) what's the purpose to design that tuple? And (2) how the tuple is used in your program to implement coordination(s) and functionality (-ies)?

     ** If they mention they change the tuple design in implementation, ask which one and why, comparing with their original ideas.
   - v. What was the biggest programming problem or challenge you faced when coding? (e.g., stuck by some APIs) How did you solve it?
   - vi. During implementation, were there any API's (either Java or TupleSpaces) (or other dependencies) you had to learn about to write the program? How did you do that?
   - vii. Generally speaking, what is the biggest challenge when you work on the project? How did you solve it?
   - viii. How did you and your project partner work together? Who did what? Especially, how did you two distribute the programming work?
   - ix. Is there any other interesting thing about your project you would like to share?

2. Imagine if you may use other technology (-ies) you learnt before, e.g., client/server architecture, web service, other distributed technology, database, and thread to implement the same functionality as your project, what is the main difference you can think of? And which one do you think may be better (easier) and why?

II. Experience with Tuples and TupleSpaces

1. With your own definition, what is TupleSpaces?
2. In general, how do you design or define tuples for a program?
3. In your view, what are TupleSpaces good for?
4. Have you ever thought of any other programs you'd like to implement with TupleSpaces except your project?

III. What's the most important thing(s), if any, you learned from this class, and why? (e.g., Design principle, Java, general programming skills, and problem solving skills.)

Are there any other thoughts or questions about the class or research you would like to share or ask me before we end this interview?

Thanks for your participation again!

Appendix D. Programming Reflection Questionnaires

CS4984 Programming Questionnaires (Fall 2005)

1. Please rate the following TupleSpaces/TSpaces operations/functions/concepts in terms of "easy to understand or/and program" from 1 to 10. Please give a brief reason/comment for your choice.

1.1. Tuple/TupleSpaces idea itself

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

Reason(1.1)

1.2. Tuple basic functions: Read, Write, Take

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

Reason(1.2)

1.3. countN, multiRead, multiTake

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

Reason(1.3)

1.4. template and matching

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

Reason(1.4)

1.5. callback and eventRegister

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

Reason(1.5)

---

2. Please rate "Eclipse" from the following aspects in terms of "easy to use or helpful for programming."

2.1. Detect error(s)

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

2.2. Automatically fix errors

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

2.3. Automatically prompt methods/functions for class/object

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

2.4. Debug

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

2.5. Manage project/source files

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

2.6. Jigloo (the Swing/SWT Editor), choose "Other" if you didn't use it and fill the reason

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

☐ Other:

2.7. Other function(s)/feature(s) you would like to comment

e.g.,

☐ 1(Absolutely hard) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely easy)

2.8. Overall, is Eclipse helpful for your programming in Java?

☐ 1(Absolutely useless) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10(Absolutely helpful)

---

3. The following questions are related to the "Clicker" assignment.

3.1. How many hours did you spend on understanding and designing the program?

3.2. How many hours did you spend on programming (Coding and Debugging)?

3.3. What problems or challenges did you face (please list them one by one, and give a few words about each one)?

3.4. Did any of your problems or challenges have to do with (please check all that apply):

☐ 3.4.1. The programming environment(Eclipse)? What?

☐ 3.4.2. Java functionality? Which?

☐ 3.4.3. Design choices about the structure of the program? What?

☐ 3.4.4. Implementation choices? What?

☐ 3.4.5. Understanding the Tuples API? Which?

☐ 3.4.6. Understanding other API's? Which?

☐ 3.4.7. Understanding the relationship between different APIs' functionalities?

---

4. The following questions are related to your group project:

4.1. Project Name

4.2. How many hours did you spend on implementation by your own?

4.3. How many hours did you spend on implementation with your teammate?

4.4. In your program, did you use the following items? (Please check all that apply)

☐ Read, Write, Take
☐ countN
☐ multiRead, multiTake
☐ template and matching
☐ callback and eventRegister
☐ waitToRead, waitToTake
☐ transaction
☐ ExplicitTuple

Other:

4.5. When implement your project, which part did you mainly work on?

○ User Interface ○ Tuples ○ Both

4.6. What problems or challenges did you face (please list them one by one, and give a few words about each one)?

4.7. Did any of your problems or challenges have to do with (please check all that apply):

☐ 4.7.1. The programming environment(Eclipse)? What?

☐ 4.7.2. Java functionality? Which?

☐ 4.7.3. Design choices about the structure of the program? What?

☐ 4.7.4. Implementation choices? What?

☐ 4.7.5. Understanding the Tuples API? Which?

☐ 4.7.6. Understanding other APIs? Which?

☐ 4.7.7. Understanding the relationship between different APIs' functionalities?

4.8. Have you considered continuing your project or starting a new project using TupleSpaces for your independent study or/and in CSCW (Computer Supported Cooperative) course next semester?

○ Yes

○ No

○ Don't Know Yet

If you answer "Yes" to question 4.8, please answer question 4.9, otherwise, skip it.

4.9. What do you plan to improve current project? Or what else you would like to implement?

---

5. What would you like to suggest in order to improve the class? (e.g., help you have a better understanding of using TupleSpaces) Or any other comments if any?

Appendix E. Tuple Class Major Events and Teaching Materials

E1. Tuple Class Major Events and Teaching Materials

Table 15. Experimental Class Major Events and Teaching Materials

No.	Date [a]	Topics	Teaching materials
01	Aug.23	Introduction to the class	Class Syllabus
02	Aug.25	• Coordination and TupleSpace • Introduction to Java • TupleSpace Demos: Whiteboard and Collaborative K-Map • TupleSpace "Hello World" example in Eclipse	• Slides: Designing Distributed, Networked Activities for Learning • Slides: Jumping from C++ to Java
03	Aug.30	TupleSpace programming: Tuples	Slides: Tuples
04	Sep.01	TupleSpace programming (continued from the last class)	Slides: Tuples
05	Sep.06	Template matching exercises	• Tuple matching Exercise1 (focus on learning the concept) • Tuple matching Exercise2 (focus on programming in Java)
06	Sep. 08	• Review assignment 1 • Design exercise: design an activity to help Sep. 06, 2005 class (teach tuple) activity	Review of Assignment 1

07	Sep. 13	• Continuous discussion of class design activity from last class (concentrated on the tuple design) • Introduction of how to use ExplicitTuple	Slides: ExplicitTuple
08	Sep. 15	• Introduction of how to use ExplicitTuple (continued from the last class) • Class exercise • Database programming vs. Tuple-based programming	Slides: ExplicitTuple
09	Sep. 20	• General discussion • Clicker demo and discussion • Clicker user interface source code explanation • EventRegister and Callback	• Clicker user interface source code • Slides: Event Register and Callback (cs4984-09-20-2005-EventRegister.ppt)
10	Sep. 22	Demos (Boomerang, ImageMap) assessment	Slides: Demos
11	Sep. 27	detailed explanation of sequence diagram and Tuple design for Boomerang, ImageMap, and Match-My-Graph	Boomerang, ImageMap, and Match-My-Graph sequence diagram and Tuple design
12	Sep. 29	• Explained how the activity console and demos (e.g., Boomerang) work in detail (with sequence diagram and tuple design) • Discussed the possibility of using handheld in TS • Homework review: Tuple template design	• Sequence diagram for each demo: http://manleyhopkins.cs.vt.edu/CS4984/demo/ • Review of Assignment 1 Tuple Design for "Tuple Learning Activity" Slides (CS4984-09-29-2005_hw1Review.ppt)
13	Oct. 04	Education Design	• Slides: Education • Review of Assignment 2 – Clicker
14	Oct. 06	More demos: • Match-My-Graph in SimCalc • Draw Molecules • ImageMap	Demo WebPages
15	Oct. 11	Class project topic discussion	Slides: Implementing Games

16	Oct. 13	• Class project scenario discussion • A short lecture: scenario-based design	Slides: Scenario-based design
17	Oct. 18	• Claim Analysis as Design Rationale • Students introduced the game scenarios for their project	Slides: Claim Analysis as design rationale (CS4984-10-18-2005-Scenario Analysis.ppt)
18	Oct. 20	Talked about login feature designed in many students projects	
19	Oct. 25	Java Swing (1)	Slides: Swing_GUI_Notes_10-25-2005_-1.doc
20	Oct. 27	• Java Swing (2) • General discussion about homework, i.e., prepare a 2-side page to describe the project	Slides: Swing_GUI_Notes_10-25-2005_-1.doc
21	Nov. 01	Project act-out (section 1), including: (a) Apples-to-apples; (b) Math Bingo	
22	Nov. 03	Project act-out (continued, section 2: with SRI people), including: c) Crossword puzzle; (d)Algorithm Organ	
23	Nov. 08	• Showed the interface for Algorithm Organ, public space and private space • Project act-out (continued, section 3), including: (e) Pictionary; (f) Krypto	
24	Nov. 10	• Project act-out (continued, section 4), including: (g) Hangman; • Discussed design problems revealed from project act-out; and implementation suggestions	Slides: Implementing the game
25-28		Did not meet in the classroom, students worked on their own team projects. Professor and TAs were available for questions.	
29	Nov. 29	Project final demonstration and discussion: (a) Apples-to-apples	
30	Dec. 01	Project final demonstration and discussion (continued), including: (b) Krypto; (c) Algorithm Organ; (d) Hangman.	
31	Dec. 06	Project final demonstration and discussion (continued), including: (e) Pictionary, (f) Math Bingo, and (g) Crossword Puzzle	

[a] all classes were taught in year 2005

E2. Detailed Class Timeline and Events

Lecture 01

Date:
Aug. 23, 2005

Video Files:
NO video recorded

Topic:
Introduction

Teaching Materials:
Class Syllabus (ver1)

Instructor:
Dr. Tatar

Class Description:
This lecture was the first class in the semester. It was not video recorded because we had not got the students' permission yet. A class syllabus was distributed to the students. The lecture talked about the overview of the class, including TupleSpace, education, and collaborative learning activities. We did not show any demos in the first class because we did not want to limit students' creativity and imagination of the systems they would build.

The lecture also talked about the TupleSpace project research purpose, procedure, and the way that we would use to collect data during the class. We encouraged the students to actively participate in the research during the whole semester.

After the class, the most common asked questions from students were "what Tuplespace is indeed," "Is it just a socket?", and "Which kind of applications can we write with Tuplespace?" After the class, we found out that most of the students had little or no Java programming experience. We had to teach the Java basics at the beginning of the class so that the students were capable of writing programs with Tuplespace.

Other events:
Send an email to ask for students "programming background", see results in 001.ClassMemeberBackground.doc

Lecture 02

Date:
Aug. 25, 2005

Files:
1. Disc01_Title_01.avi
2. Disc02_Title_01.avi

Topic:
1. Coordination and Tuplespace (Disc01_Title_01.avi -00:00:30~00:16:31);
2. Introduction to Java (Disc01_Title_01.avi 00:17:00~00:58:00);
3. Tuplespace Demos: Whiteboard and collaborative K-Map (Disc01_Title_01.avi-00:58:00-end, Disc 02_Tiltel01.avi-00:00:00-00:02:30);
4. Tuplespace "Hello World" example in Eclipse. (Disc02_Tiltel01.avi)

Teaching Materials:
1. Slides: Designing Distributed, Networked Activities for Learning;
2. Slides: Jumping from C++ to Java; (by P10)

Instructor:
Lin & Kim, P10

Class Description:
1. The lecture first introduced the concept of coordination (e.g. how coordination happened between people, between people and machine, and between machines), the ways of coordination, and four coordination models (direct, meeting-oriented, blackboard-based and Tuplespace).
2. P10 gave an overall introduction of Java. To help students, most of whom were C++ programmers, to learn Java, he pointed out the similarity and/or difference between Java and C++.
3. To give students the basic idea of Tuplespace applications, two example programs implemented with Tuplespace were demonstrated, including the real-time Whiteboard and Collaborative K-Map.
4. "Hello World in Tuplespace" example was used to introduce how to write simple Tuplespace Java program in Eclipse. The idea of "template matching" was explored through the example.
5. The background information questionnaires were distributed to the students.

Lecture 03

Date:
Aug. 30, 2005

Video Files:
1. Disc 03: Title_01.avi
2. Disc 03: Title_02.avi (00:00:00~19:54:00)

Topic:
Tuples

Teaching Materials:
Slides: Tuples

Instructor:
Dr. Tatar

Class Description:
Set up Java 1.5 and Eclipse.

Introduced the basic knowledge of tuples, the concept, fields, the ways to create tuples, how to connect to Tuplespace, and Tuple operations.

Also talked about the concept of using "template" to search interesting tuples and why it is interesting that Tuplespace is capable of only "take"-ing one tuple.

Interesting discussion:
(1) Q: is Tuplespace a database? (Disc03_Title_02.avi, 00:09:50)

A: Tuplespace is more like a bag of tuples.

(2) Q: Is there a reason that only one tuple will be returned by default? (Disc03_Title_02.avi, 00:17: 22~00:19:25)

A: when we talk about Tuplespace, you want to think about data structure, but also want to think about the data structure as a resource management. If the default action would be "taking everything", that would be a very difficult to manage system. Tuplespace is kind of "operative" kind of resource management system. "I can just take something that matches my need and I take that."

Related Information:
As *bag* object is an unordered collection of zero or more elements of some type. If the all the elements are of the same type, it is a *homogeneous* bag, otherwise it is a *heterogeneous* bag.

Operations include adding an element to the bag, removing an element from the bag, testing if the bag is full or empty.

Representation

A bag of integers (HSM Program 3.3-4)
A bag template (HSM Program 3.5-6)
Applications

http://www.nist.gov/dads/HTML/bag.html

Lecture 04

Date:
Sep. 1, 2005

Files
1. Disc03_Title_02_part2.wmv
2. Disc02_Title_02.avi

Topic:
Continued to introduce Tuplespace programming

Teaching Materials:
Slides: Tuples

Instructor:
Dr. Tatar

Class Description:
1. The Good and bad about Eclipse (class discussion)

+ Compiles class files for users automatically;
− Too many features (need to setup a lot of things); it was not necessarily a bad thing, but need more time to figure things out;
+ Fill the code for users automatically;
  − but sometimes a little slow
+ Like the Debug window, because it has a graphical icon
− What the differences between "Debug" and "Debug As" (GUI confusing)
+ can run as "JPage", but we may not really use it too much
+ check Syntax error for users
+ useful shortcuts:
  - <Ctrl-Dot> (automatically to next Syntax error)
  - <Ctrl-Space> (content assistant)

2. Continued to introduce Tuplespace programming, including the difference between "take" and "read"

3. Class exercise (Disc03_Title_02_part2.wmv, 00:18:25~end, and Disc02_Title_02.avi 00~)

The students downloaded necessary software, created their own Tuplespace in the local host, and tried different tuple operations.

(1) Disc02_Title_02.avi_00:38:36~ clarified concept for "take" in the exercise

> In order to know what's in the tuple that you are reading, you "take" the thing match the template (associative memory mechanism) by specifying your interests, then you can examine what's inside the tuple, you can't search for portion of it.

(2) (Disc02_Title_02.avi_00:39:38) P01 didn't understand "template matching" to a tuple like <"key2", String>

Interesting discussion:
Q: (P02, 00:42:07) so we created an instance of a Tuplespace and call it “ts”, and we check some ts attributes. My question is how is using tuples better than using non-tuple?

A: the point is we’re setting up here to use the tuples. The above exercise did not do any coordination through a common space. We just went through local motions.

Problem found:
Some students had problem understanding “template matching”

Lecture 05

Date:
Sep. 6, 2005

Video Files:
1. Disc04_ Title_01.avi
2. Disc05_Title_01.avi (00:00:00~00:05:02)

Topic:
Two Template Matching Exercises

Teaching Materials:
1. Tuple Matching Exercise 1 (focus on learning the concept) (Disc04_ Title_01.avi_00:00:00~00:36:00)
2. Tuple Matching Exercise 2 (focus on programming in Java) (Disc04_ Title_01.avi_00:36:18~end)

Instructor:
Dr. Tatar

Class Description:
We handed out paper version exercise to the students. In Tuple Matching Exercise1, they finished the exercise by themselves first, and discussed their results as groups of two. And then we went over the results in the whole class and discussed some interesting exercises (Disc04_ Title_01.avi_00:21:00~). The students found the right answers most of the time.

In Tuple Matching Exercise2: We tried to explain some tricks in TSpaces (because of Java), elaborate what was in each template/ tuple, the type and the value of the field. (in order to have a match, must have the same number of fields, same type of fields, in the same order.)

> *Template1_2 = new Tuple(String.class); //the value is String.class, the type of the field is not a String class.*

Tell students they needed to use class (cast) when they tried to access the field value. Especially the transfer between int and Integer

Interesting Discussion:
1. (Disc04_ Title_01.avi_00:22:16~)When talked about what tuples match template3, P05 mentioned that it "depends on" the assumption, if template 2 already "take" tuple5, then no matches for template3. We made it clear that we only thought about one case at a time.

2. (Disc04_ Title_01.avi_00:33:16~) students described mistakes made by themselves:

- tried to take more than one tuples at one time;
- had difficulty in understanding nested tuples;
- did not understand the concept of Serializable in JAVA

4. discussion about how to create field

Field strFormalField = new Field(String.class);

template1_1 = new Tuple(strFormalField);

vs.

Field strFormalField = new Tuple(String.Class);

Programming Assignment:

Assignment 1 (09_06_2005_CS4984Ass1.pdf):

PART 1. Goal: Get experience working with slightly more complex writing and taking.

Task: (1) Write several of the simple Multiply Tuples to the space and then Take them. (2) Do this several times. What do you notice?

PART 2 Goal: To move the tSpace experiment into a potentially shared domain, to begin the need for coordination.

Task: (1) Connect to the activity/ tspace "RemoteTupleSpace" on the manleyhopkins server. (2) Improve the code that you wrote before to make lots more different kinds of Multiply Tuples. (3) Write many Tuples involving more different and bigger numbers. (4) Take fewer Tuples than you write. (5) Come back later, and take some more.

PART 3 Goal: experiment with more complex data structures (Tuple) and more complex coordination.

Task: Enhance your Tuples so that they contain a representation of an ID. Then Write and Take only "your" Tuples from the shared space.

Summary of the Assignment:

Most students were capable of writing several integers to the server by "write" and got back two numbers by "take." They reported the products of two returned numbers were different each time. They realized that the returned integers were in a random order, not necessarily in the order of writing.

The exercise showed their abilities of writing JAVA code, programming skills, and their status of programming practice. Some of them used loop or randomized function to generate multiple numbers, while some just wrote a couple of constants. Some students wrote source code without any comments, and some did not remove temporary code and did not realize that their code should be readable by other people as well.

Lecture 06

Date:
Sep. 08, 2005

Video Files:
1. Disc05_Title_01.avi (00:05:02~end)
2. Disc06_Title_01.avi (00:00:00~00:15:38)

Topic:
1. Review assignment 1 (Disc05_Title_01.avi_00:05:02~00:23:00)
2. Design exercise: design an activity to help Sep. 06, 2005 class (teach tuple) activity (Disc05_Title_01.avi _00:26:45~)

Teaching Materials:
Slides: Review of Assignment

Instructor:
Lin, Dr. Tatar

Class Description:

1. Reviewed assignment 1 and had discussion about "take" (return results were not guaranteed in order)

2. Design (and implement) an activity to teach students to learn template matching by using tuples
   (1) (00:29:00~00:31:50)Activity goal: To give students the chance to match tuples to templates (or templates to tuples) with many kinds of tuple templates. (We need to think about the activity as a class.)
   (2) (00:31:52)Design Steps:
      i. Sequence diagram: (Who has what event, what the teacher sees and what the students see, will they see the same thing? Same UI? Different UI? Or different UI stages? Then finally come up with a specification for the design)
         - Actors: instructor & students
         - Assume instructor has a big display that everyone can see
         - (00:38:00)(what the instructor starts with) The instructor selects a template, and sends a tuple problem set to the students(one student directly went to the level of tuple space, the professor reminded let's keep at the level of user experiences for now)
         - students work on their own problem
         - design choice of exchanging idea with another student (social component): (a) anonymity (b) with identity (see below for details)
            a. (00:49:00)if the instructor should send back the grading to the students? (see below for details)
      ii. (00:53:50)Design UI (simplest)
            a. Students' UI (let's assume for handheld screen)

multiple choices, check

iii. Tuples design

Interesting Discussion:

1. When talked about userID, some students mentioned security issues. We made it clear that security won't be our major concern.
2. (00:34:46~00:37:12) P01, Q: should we assume the number as "integer" or "string"?

   Professor: why does that make a design difference?
3. (00:43:50~00:46:59)two design choices of exchanging idea with another student between anonymity and partners: (think about other people's feeling under the two circumstances):

   (1) anonymous exchange: protects people's identity in some sense, but it can also hurt people by blaming a wrong answer without knowing who made mistake; people don't try as hard as if personal effort is not countable; it's important to let students know that other people may make the same mistake too

   (2) (social exchange, the way we did in the previous class) talk with other people: help to understand each other's idea; people have powerful affects on other people's attention; might be embarrassed by their mistake or performance differential
4. Q: the instructor can send back the grading to the students via computer?

   Professor: formal assessment. Not about grading, about information that allow students to know how they are doing, and the information allow the instructor to know that how well the students understand and how to fix things if students don't understand.
   a. the students need to know other people make the same mistake;
   b. the teacher needs to know how many students get the right answer; (90% of correctness might be good enough for her)
5. different UI designs: what are the concerns of each different idea can promote pedagogical goals?
6. (Disc06_Title_01.avi_00:03:25)when talked about the tuple design, a student thought different users (novice user vs. expert user) should have different user interface because he had a hypothesis that:
   (1) matching is harder than True/False question;
   (2) multi-choice question is harder than matching;

   (Disc06_Title_01.avi_00:09:00~00:10:50) and then talked about M-V-C
   (i) Model: data representation
   (ii) View: user see
   (iii) Controller: mediator

We have separation between tuple and the program interact on the client side that knows how to display the tuple for the device, building that is the major part of the work; actually 90% of the work in tuple based programming is to build that.

The design of tuples won't affect the UI (some students didn't realize this before we told them)

Homework:

Design a set of tuples for the class exercise and specify the tuple fields, types of the fields, and the names of each field

*Summary of the assignment: about half of the students did not really understand what the tasks were, for example, some only designed activities to teach tuples for the classroom. So we did not grade the first submission. Instead, we asked the students to do the assignment again by providing them a detailed specification on Sep. 20 2005.

Emails and Web Discussion After Class:

From p09, Date: 09-11-2005

For the homework #2, are we writing tuples such as having 'normal' fields (integer or strings), nested tuples and tuples with ID along with the templates to match those tuples? Or am I misunderstanding the homework?

From p02, 09-12-2005, 2:42pm, regarding tuple exercise

…

Some comments:

1) The order of the tuples taken is not always the same, so the results are almost always different (at least different orders).

2) I used my PID and birthday as a unique identifier for my tuples, so I could always return my own tuples. However, if somebody is greedy and takes tuples with a string at the front followed by two integers, they will steal my tuple. So while I can always return only what I want, I am not guaranteed that what I want will be there for me to take.

Lecture 07

Date:

Sep. 13, 2005

Video Files:

1. Disc06_Title_01.avi(00:15:38~end)
2. Disc07_ Title_01.avi(00:00:00~00:13:45)

Topic:

1. Continuous discussion of class design activity from last class (concentrated on the tuple design); (Disc06_Title_01.avi_00:15:38~00:45:45)

2. Introduction of how to use ExplicitTuple

Teaching Materials:

1. Slides: ExplicitTuple

Instructor:

Dr. Tatar, Lin, Kim

Class Description:

1. discussed what kind of tuples were needed (the data structure needed to implement the activity), different students' solutions

(1)P07 design

<Problem, ID, tupleToMatch >,

<Template, …>,

<AnswerTuple, ….>,

<StudentAnswerTuple, …>

He said he used a database model to design his tuples. We argued that this was a different kind of design from database and they wanted to think in terms of the "matching" properties.

The concept may be similar but it's much simpler than tables in database.

Not because we are not interested in scalability, it's different. We would come back to this later.

(2)P02 design

all of his tuples contain that userID as a field for identification purpose

a. security;

(3)P05 design

<”matching”, “pattern”, SomeKindTuple>, *instructor posts the pattern, the resource to the space, students read but not take it;*

<”matching”, ”possibleMatch”, Number, TemplateTuple> *instructor posts several possible matching tuples for students to read*

<”matching”, ”answer”, answerNumber> students send their answers back to the tuplespace

He had a field called “matching” for each tuple, was used to identify this activity from other activities (in the same tuple space), the second fieled for each tuple is always a String.

He did not have anything to match the problem, but he assumed that students would work on one problem at a time. When next problem starts, all tuples related to the previous problem would be taken out of the space.

2. introduced Tuple, SubclassableTuple, and ExplicitTuple classes; and also discussed why and how to use ExplicitTuple class;
3. demonstrated a design example: Collaborative Activities: K-map (1 of 2)
   (1) what is K-Map;
   (2) activity sequence diagram;
   (3) user interface

Interesting discussion:

1. database programming vs. Tuplespace programming, just touched the topic a little, not too much. More discussion and clarification will be in the next class

Lecture 08

Date:

Sep. 15, 2005

Video Files:

1. Disc07_ Title_01.avi(00:13:46~end)

2. Disc08_Title_01.avi (database vs. TS, starts from00:01:30)

Topic:

1. introduction of how to use ExplicitTuple (continued from last class)

2. Class exercise

3. Database programming vs. Tuple based programming

Teaching Materials:

1. Slides: ExplicitTuple

Instructor:

Kim, Dr. Tatar

Class Description:

1. told students that the class projects would be changed;
2. demonstrated a design example: Collaborative Activities: K-map (2 of 2 continued)
   (4) tuples design for K-Map, and how to use these tuples;
   (5) Exercise: (a) Create getter and setter for UserColorTuple; (b) Using Ovals instead of Rectangles, create UserKMapPointTuple
3. Database programming vs. Tuple based programming

Professor: this has to do with how you conceptualize what we are doing.

(a) Scaling, reliable

We are trying to support learning, a process; trying to create moments that students can encounter materials appropriate for them; trying to help students to interact with other people in that learning process. So scaling in TS is not about getting more data but about connecting students and machines

Database programming cares about reliable and security. We care about robustness too, but it's a different kind of robustness. We care about "can people come in and communicate with each other?" Even when the teacher's machine stops working, students should be able to keep working. Students should be able to work in pairs.

In that sense, we're scaling people, but not scaling data,

When design database, people always think about design dense structure (e.g. dense table). But in TS, we do not really care if the data structure is dense. In database, we really need to think about the data and their relationship with one another. And people

care about “store data.” The power of TS is really “taking”, take from resource. You can not do “take” easily in other systems;

(b) two ways of implementing

TS implementation way is very different from database programming

Interesting Discussion:

Q: Why Tuple is a final class?

A: Make it possible to change the tuple space properties.

Lecture 09

Date:

Sep. 20, 2005

Video Files:

Disc09_Title_01.avi

Topic:

1. Some discussion about something (can't tell from the video)

2. Clicker demo and discussion (00:02:26)

3. Clicker user interface source code explanation

4. Event Register and Callback (the last couple of minutes video wasn't recorded)

Teaching Materials:

1. Clicker user interface source code

2. Slides: Event Register and Callback (cs4984-09-20-2005-EventRegister.ppt)

Instructor:

Dr. Tatar, Kim, and Lin

Class Description:

Topic 2. Clicker demo

Students tried using Clicker and conducted discussions about what software like this can provide to teacher and students in classrooms.

Topic 3. Went through the single user version (non-collaborative) Clicker GUI source code with the students;

Topic 4. Explained how to implement synchronized connectivity by using EventRegister and Callback provided in TSpaces, so that the students can modify the example source code and implement the collaborative version Clicker.

Interesting Discussion:

The class had discussion about the tradeoff of using software or using social control. Here is an example,

Professor asked: How many people feel comfortable with eclipse?

And the feedback from students: "8:4" (comfortable: uncomfortable)

Discussion about what the answer might mean to the teacher and students:
- *What does the result tells?*
- *Help to make Instructional decision, and reveal satisfaction*

- *Professor asked a very open question, and students may response very differently. Future opportunity to learn in different levels.*
- *Students feeling: anonymous to see other people's opinion*

Conclusion: "the little information we got is very rich"

Homework

1. Redo the design activity for learning Tuples assignment by providing them a detailed specification

2. Distributed the Clicker Assignment
    - no difference between teacher and students user interface;

Email/Web Discussion after Class:

From p01, Date: 09-21-2005

I have some questions about yesterday's lecture.

First can you provide answers for slides 6?

I am thinking they are

1) A call back happens to B

2) A call back happens to A

3) A call back happens to B

4) Nothing

For <String, 6, 20> case

1) A call back happens to A & B

2) A call back happens to A & B

3) A call back happens to B & Z

4) Nothing

I am not sure if they are correct or not, so....

In eventRegister, what exactly is newThread ?? when it returns, what is the sequence number ?

Lecture 10

Date:

Sep. 22, 2005, Thursday

Video Files:

1. Disc10_Title_01.avi

2. Disc08_Title_02.avi

Topic:

Demos (Boomerang, ImageMap) assessment

Teaching Materials:

1. Lecture slides: Demos

Instructor:

Dr. Tatar

Class Description and Interesting Discussions:

1. We asked students if they had any question about the Clicker assignment due next week. Students asked about drawing GUI in Eclipse by using plug-ins.

2. Boomerang demo (00:13:00)

First we talked about the background of using Boomerang in the ethic class.

Then we encouraged the students to think about pedagogy from the following different aspects:

(1) Design purpose: teacher said their students could not ask questions.

(2) Discussion about asking a same question by different ways, like open ended questions or multiple option questions

(3) K-12 and college difference

(4) Gave students the impression that we were moving from "voting tool" (Clicker) to "a tool to support collaborative discussion" (Boomerang))

(5) Posted a question in Boomerang and used it to discuss the result. The question was about "what do you think you can implement in TS" or "what area you think might be interesting to implement in TS", or "if your answer is no, do you know a way to find an interesting area." Here are the inputs from the students.

- Students ideas: "checkers", "hangman", "tick-tac-toe", "math bingo"(a little bit more complex in coordination), "getting students' reflection on peer behavior has been reported to be problematic", "chat message system" "the game with consuming resources"
- simple coordination (checkers, hangman, tick-tac-toe): turn taking

3. ImageMap Demo

It was a different kind of representation, could be used to teach history

– where were you born, mark your born-place

Some discussion about the possible usage of ImageMap

Homework:

1. Observe at least one interaction in fast food store drive-through and identify a problem, and think about the interactions.

Purpose of this assignment: design problem solving, problem finding, wicked problems/ tame problems, think about how to identify a problem

2. Assignment 2: Clicker (09_20_2005_CS4984Ass2Clicker.pdf)

Emails/web Discussion after Class:

From: P10, Date: 09-24-2005

Subject: Assignment #2

Hi Guys,

A few items that you (and the students) may want to know about the assignment.

You can do whatever you wish with the info I give.

1. Took me 5 hours to complete.

(researcher's comments: For a highly experienced JAVA programmer, it took him 5 hours to finish the assignment. The understanding of concept not necessary related to the proficiency of JAVA. Maybe other non-JAVA programmers spent more time.)

2. The students should REWORK and RENAME S's "MultiplyTuple.java" to fit what tuple they need to create for the assignment.

3. Many parts of the code need to be inside "Try/Catch" blocks.

4. specific functions from S's "ExplicitTupleExampleCallBack.java" that

should be used are:

a. eventRegister

b. write

c. creating a "template" of a Explicit tuple

d. creating a "tuple" of a Explicit tuple

5. They should ALMOST copy and paste the "call" function.

6. Make sure that they import the correct files

Well, I hope they do well. Depends on how much they know GUIs as well. Let me know if I can be of further assistance.

09-25-2005, 11:33am, another reply after we responded his first email

Here's answering a few question you had:

1. Why did it take so long?

a. Most of it was my own doing, I tried to create a REGULAR tuple, and use it with the callback TA gave us. That obviously doesn't work.

b. remember that I'm a design freak, so that takes up a lot of time.

c. I forgot this part, THIS IS IMPORTANT!! When I first had everything together, I would hit a few of the buttons in a SHORT AMOUNT OF TIME! I was getting error after error back from the Tuple Classes. The problem is the"server" does not update quickly. If I hit a button, then pause, then hit another button, THEN it would work correctly. So.. in a nutshell, pressing buttons need to have a delay in between. THAT TOOK AN EXTRA HOUR!! I thought it was me!!

2. The try /catch blocks.

a. You said it was "ok" if they didn't have them. BUt if we didn;t, we get a ton of errors and are unable to compile. It's not that hard to put in, so have them do it.

…

From P07 (Web), Sunday, 25 September 2005, 01:52 PM

Subject Clicker Demo assignment

Hi,

Can we assume that all students are connected to the tuple space before students start choosing their answers? I believe this assumption was made in the demo we saw, therefore people who connected later got the negative numbers.

Otherwise is there any "shared" tuple implementations available that have a mutex lock on them?
Thanks

From P14, 25 September 2005, 08:39 PM

Hi, when you say it supports later login, does it mean it resets the votes?

According to my observation, if a new user logs in, the numbers are set to zero. That is what I did to my implementation.

From P07, Sunday, 25 September 2005, 10:45 PM

ya.. the new version has it fixed. Following up on my earlier post, I was wondering if there is more than an atomic read/write that you pointed out. I have my program working except for new people signing in and seeing old scores... I need a blocking read/write to do that... without that I can implement but can't gaurantee correctness... but I guess blocking would be against the basic tuple space principle? ... so some tuple way of doing things..

From P10, 09-27-2005
Subject: Help

Of all things, I can't get the Clicker to UPDATE if someone logs on late. I'm using the very simple example on pg 13 of the Programmer's Guide. I am try anything so "countN" will have a value, but it never does. Will any of you be around at 1:30??

Yeah, I did check the template. You'll see the code when I stop by. I'll be there in 30 minutes. Don't go anywhere ;)

>P10,

>Make sure your template is correct. And the following sample code should work, obviously you >need to modify it since you use your tuple.

```
>Tuple tryTemplate = new Tuple();
> int count = mTupleSpace.countN(tryTemplate);
> System.out.println("count is: " + count);
```

P06 also had questions and sent his source code, 09-27-2005 1:26pm

Another email from him:

I was not able to complete the assignment, even in class. I am almost done though. However, I still need a little help. Are you going to be in your office tomorrow anytime before 4? Is it worth it for me to turn in late? I am just having an incredibly tough time understanding Java, having learned C++.

Thanks for all the help.

Lecture 11

Date:

Sep. 27, 2005

Video Files:

1. Disc11_Title_01.avi

2. Disc12_Title_01.avi (00:00:00-00:10:00)

Topic:

1. detailed explanation of sequence diagram and Tuple design for Boomerang, ImageMap, and Match-My-Graph

Teaching Materials:

Boomerang, ImageMap, and Match-My-Graph sequence diagram and Tuple design

Instructor:

Lin, Dr. Tatar

Class Description:

1. Explained how Tuplespace stores tuples. How multi-read works.

2. Boomerang: (sequence diagram and Tuple design)

Interesting Discussion:

1. P01 requested to explain Callback again

2. Flowchart

After talked about the Boomerang, P11 asked if we have a flowchart, so that he could understand the whole process. Even after we drew a simple diagram, he insisted seeing a flowchart. He said he did not understand the scenario without a flowchart. We asked him to draw one, but he wanted to draw Clicker's flowchart. We insisted him drawing Boomerang's. Dr. T told him if he tried to understand through a flowchart, he would not be able to understand.

It seems he had problems to visualize/conceptualize the activities/interactions between the TS and the application client

(what does mean the student join the activity)

Email discussions between SRI fellows and us are attached in Lecture 13

Homework:

After the class, we felt some students had difficulty visualizing how Tuplespace works (e.g. the interactions between server, space, and client). So we drew a detailed sequence diagram for each demo program. In the diagram, we showed what happened on client

side (both interface and inside the program), and the changes in the space after each operation. (e.g. http://manleyhopkins.cs.vt.edu/CS4984/demo/Boomerang.html)

Emails/web Discussion after Class:

From p01, 09-27-2005, 10:49pm

When I was turning hw2 today, I knew I had a bug with callback.

I didn't quite understand how to use 'callback', but now I think I do.

I knew it would be probably too late, but I tried and fixed the problem.

I am attaching corrected version of my hw2.

Thank you.

Sincerely, p01

From p13, 09-28-2005, 11:50pm

Hello

Sorry for late submission. It's been a hectic week. The program does not work properly and is documented in the java file. The problem is that the program does not take a needed tuple most of the time. Will continue to try to fix it.

thank you

From p02, 09-29-2005, 1:08am

I have an ongoing bug that I can't seem to fix. Clicking a button will not display the updated votes correctly. It is on a delay. For example: click one, on A, returns no result on the GUI. Click two, on B, returns a vote for A. Click three, for E, returns a vote for B. The voting results are always off one "turn." I am not sure how to fix this, and I think part of the problem is my misunderstanding of the java language. I think it might be correlated to the fact that I had to make the tuplespace variable static in the clicker UI. I seem to have some sort of scope resolution issue, but I can't resolve it.

If you could tell me what I am doing wrong, I would greatly appreciate it.

Thanks,

P02

From p14, 09-29-2005

Hi TA,

When I mentioned that I only had Java 1.5 installed, I was thinking that multiple version of Java may be causing the problem. However, I understood how you wanted me to modify the code.

I changed all my switch statements to switch

(((Integer)(((Tuple)(list.getField(i).getValue())).getField(1).getValue())).intValue())

The attachment is the modified code. I hope it would work on your computernow.

P14

>===== Original Message FromTA=====

>

>It IS weird, I'm using Java 1.5 too. And another student who is using a MACmachine met the similar >problem.

>

>In your code the final value you put into the switch is an Integer class object, right? Let's say it is called myInteger, then if you use myInteger intValue(), it will return an int type value. I think that will work. I

>can help you to change the code, but I'd like you to do that.

>

>Let me know if there's more problem.

>

> >

>-------Original Message-------

>

>From: p14

>Date: 09/29/05 03:46:37

>To: TA

>Subject: RE: CS4984 clickerdemo submission

>

>Dear TA,

>It is very weird. I am using a PC (IBM laptop) for my homework. I only have Java 1.5 installed on it, so I >don't think this is a problem since we all agreed to work with the most recent Java version. My code on >my laptop works perfectly fine without any compile error messages. Also, my code shows me that

>the line number is 185 instead of 187. Even so, I tried to modify my code as you suggested using >intValue().

>However, intValue is not defined for what getField() returns nor what getValue()returns. So I wonder >what you have meant by suggesting me to use intValue().

>Am I not understanding where to use it?

>P14

>

>

>>====== Original Message From TA=========

>>p14,

>>

>>Do you use a Mac machine? The following code shows error in my machine. Please modify it >>accordingly, e.g. intValue()... Otherwise your code will not run...

>>

>>Line 187: switch ((Integer)(((Tuple)(list.getField(i).getValue())) getField(1).getValue()))

>>

>>Thanks,

From p06, 09-29-2005, 3:14pm

I finally got it working... almost. My only problem is that the clear function only clears for the user that clicks it, it does not clear out everyones votes. I tried using tuplespace.deleteAll() as well as a multiTake for every tuple existing on the server side, however they all have the same result of only deleting that certain user's tuple. Oh well, it seems like better functionality anyway. I'll see you in class and try to fix it there.

P06

Lecture 12

Date:

Sep. 29, 2005

Video Files:

1. Disc 12, Title_01.avi, 00:11:06~end

2. Disc 13, Title_01.avi, 00:00:00 ~00:17:16

Topic:

1. Explained how the activity console and some demos (Boomerang) work in details (with sequence diagram and tuple design)

2. the possibility of using handheld in TS;

3. Homework review: Tuple template design

Teaching Materials:

1. Sequence diagram for each demo: http://manleyhopkins.cs.vt.edu/CS4984/demo/

2. Review of Assignment 1 – Tuple Design for "Tuple Learning Activity" Slides (CS4984-09-29-2005_hw1Review.ppt)

Instructor:

Lin, Dr. Tatar

Class Description:

Topic2. talked about the possibility of using handheld in TS

(1) There were no educational students enrolled in class due to registration issues. (2) Due to the unsolved problem using TS on handheld devices at that time (before IBM sent the updated jar files), we told the students that we were going to implement games, populating the space with coordinated activities (not necessarily in handheld devices). With those games, we could see the opportunity to change them to educational activities afterwards. We would use the coordination in the games as examples to build the educational contents. Some potential projects were proposed, e.g. hangman, math bingo, crossword puzzle, etc..

Topic3. Homework Review

(1) Reviewed the key elements for tuple design;

(2) Reminded them the problems existing in their homework submission:

- used list to represent multiple answers, not a flexible data structure
- used unmeaning names for tuples, fields, and field type

(3) Emphasized that any tuple could be a template, and a template is also a tuple.

Interesting Discussion:

When talked about building a connection between the client and tuple space, we discussed what level the connection refers to, i.e. network level or higher abstraction level.

Homework:

N/A

Lecture 13

Date:

Oct. 04, 2005

Video Files:

1. Disc 13, Title_01.avi, 00:17:16~end

Topic:

Education Design

Teaching Materials:

1. Slides: Education

2. Review of assignment 2 - Clicker

Instructor:

Dr. Tatar

Class Description:

1. Talked about some important concepts in education:
   - learner-centered (help students deep understand knowledge, build bridges between different ideas); People learn new knowledge build on existing knowledge;
   - knowledge-centered (sense making);
   - formative-assessment centered(students should be life-long learners)

Design techniques to manage complexity - abstraction: model-view-controller (MVC), OOP, design patterns

Design patterns for collaborative coordinated software: e.g. Moon Phase; Elephant

2. Review of Assignment 2 - Clicker

(1) discussed what should be in TS, i.e. which kind of information should be designed as tuples;

(2) reviewed how to use eventRegister and callback in the program;

(3) discussed how to count total votes;

Discussions between SRI and us: Flowchart

P11 expressed difficulties understanding Tuplespace, tuples and interactions between them/ client machines without using a flowchart presentation. We tried to help him in the

class by drawing different graphs, explained how tuples work with detailed sequence diagram, which showed the status change in Tuplespace, the interactions between Tuplespace the client machine. But our efforts seemed not help P11 too much. We thus discussed the issue with SRI researchers to analyze what the real problem was to prevent P11 from understanding the fundamental concepts without a flowchart as claimed by him.

1. Discussion: flowchart.doc
2. Charles mapping chart: StudentFlowChart1.pdf

Lecture 14

Date:

Oct. 06, 2005

Video Files:

1. Disc14_Title_01.avi, 00:00:00~00:26:26

Topic:

More demos:

- Match-My-Graph in SimCalc
- Draw Molecules
- ImageMap

Teaching Materials:

Demo webpages

Instructor:

Lin & Kim

Class Description:

Reviewed the details of design and tuples for each demo. There was not much discussion in the class.

Lecture 15

Date:

Oct. 11, 2005

Video Files:

1. Disc14_Title_01.avi, 00:26:27-end

2. Disc15_Title_01.avi, 00:00:00~00:25:12

Topic:

Class Project topic discussion

Teaching Materials:

Slides: Implementing Games

Instructor:

Dr. Tatar

Class Description:

- Talked about development process; forming teams; working in teams;
- Discussed the important components for each game (crossword puzzle, hangman, apples-apples, Krypto/telephone, and some other coordinated games);
- Encouraged students to think about coordination during their project design;
- Students discussed the project with their partner.

Homework:

Decide which project to work on with the group partner

Lecture 16

Date:

Oct. 13, 2005

Video Files:

1. Disc15_Title_01.avi, 00:25:13~end

2. Disc16_Title_01.avi, 00:00:00~00:38:36

Topic:

1. Class Project scenario discussion

2. A short lecture: scenario-based design

Teaching Materials:

Scenario-based design

Instructor:

Dr. Tatar, Kim

Class Description:

1. a lecture about scenario-based design

2. assigned project for each group according to their first or second choice; repeated the development process;

3. had group discussion about their projects (some discussion with Dr. Tatar was recorded, but the background was very noisy, the voice was not clear)

Interesting Discussion:

SBD vs. PD (not talk about technology at all, just talk about the work the users do)

Homework:

Project: Game Scenario due next Tuesday (10-18-2005)

Lecture 17

Date:

Oct. 18, 2005

Video Files:

1. Disc16_Title_01.avi, 38:37~end

2. Disc17_Title_01.avi, 00:00:00~00:37:20

Topic:

1. Claim Analysis as Design Rationale

2. Students introduced the game scenarios for their project

Teaching Materials:

Slides: Claim Analysis as design rationale (CS4984-10-18-2005-Scenario Analysis.ppt)

Instructor:

Kim

Class Description:

1. Students demonstrated the scenarios for their projects

(1) Hangman;

(2) Apples-apples: social control to pick a judge, skip of being a judge

(3) Math Bingo (Disc17_01, 00:01:00): teacher and students have different user interface, user name and password to log-in, notification of claims of Bingo (how many people don't choose), race win condition (who clicks first)

(4) Crossword Puzzle (Disc17_01, 00:11:33): had a very completed activity flow chart (a better name?), teacher and students have separate user interface; different teams will compete for the highest scores

(5) Krypto (Disc17_01, 00:18:00): turn taking, teacher screen, students screen

(6) Guess what it is (Disc17_01, 25'36"): drawer takes turn to draw the picture; system automatically pick a guesser

(7) Algorithm Organ (Disc17_01, 30'):

Interesting Discussion:

(The following discussion was related to each project with the same number)

(3) No help for a student either from other students (since they compete) or teacher;

(4) Q: what kind of mechanism will be provided for students to discuss with other groups?

A: the communication between teams/team members will through instant messaging, not social control

(5) Krypto: Q: is that possible to overwrite other students' solution to trap in a dead-lock?

A: this situation will only happen between 2 people, but not happen when more than 3 people

(7) Q: what's the advantage of combing social activity to the system, students stand up and sit after solve the problem.

Emails/web Discussion after Class:

From P05, Date: 10/19/05 11:50:02

Subject: Re: CS4984: Game scenarios uploaded, and think about your GUI design needs

If P10 talks about drawing images to the screen, it would be nice for us to know about transparency (e.g., does Java support drawing PNGs with an alpha channel, or is it a magic color, or what?). We don't absolutely need image support, though, so consider this low-priority. Everything else we need is just your standard text entry, text box with scrollbars, buttons, and clickable areas on the screen. Thanks.

Lecture 18

Date:

Oct. 20, 2005

Video Files:

1. Disc17_Title_01.avi, 00:37:23~end

Topic:

1. talked about login feature designed in many students projects;

Teaching Materials:

N/A

Instructor:

Dr. Tatar;

Class Description and Interesting Discussion:

Professor's Comments before class:

(1) a lot of you came up with the idea that "users came in and log in", want to let you think if log in is necessary, do you need to know the person's name; any other more general ideas more interesting than login idea. What's the mean for?

A from students: a. identification

b. differentiation: track of score: why need a name for tracking of score?

c. (Professor) Why wouldn't you use your own name? What's the function to use your own name?

(p10): trace

(Professor): trace->time

(p01): more friendly

(Professor) always friendly? Can be friendly, also "accountability, uniqueness, acknowledgement (blame and appreciation), separation from other people"

(Professor) use token when you participant in a games; or have roles within teams; use shapes, color, or patterns to represent each teams. Encourage you to think visual techniques to implement the ways to know people (not only name, also presents)

Homework:

N/A

Lecture 19

Date:

Oct. 25, 2005

Video Files:

(should be in Disc18)

Topic:

JAVA Swing (1)

Teaching Materials:

Swing_GUI_Notes_10-25-2005_-1.doc

Instructor:

P10

Class Description:

Designing the Layout (JFrame, JPanel, Layouts); the Java components; Java file architecture;

An email From P10, *sent 10-20-2005*

HI All,

It's P10. I will be lecturing on the creation of windows, layouts, and basic GUI Components (buttons, textfields). I need everyone that can, bring a laptop. We will be using an Active Learning approach to today's material that is online. (vincent.cs.vt.edu) We will team up (the team you are currently in), and log onto the toolkit. I will give you an instructions sheet in class. I will also be handing notes, and other dittos out in class so you can follow my notes.

One last item, if possible, please see if you have Mozilla on your computer. The toolkit runs the best with it. Thank you all, hope you all learn a lot today.

Interesting Discussion:

Homework:

Lecture 20

Date:

Oct. 27, 2005

Video Files:

1. Disc19_Title_01.avi,

Topic:

1. JAVA Swing (2) (Disc19_Title_01.avi, 00~00:29:20)

2. talked something about homework, i.e. prepare a 2-side page to describe the project

Teaching Materials:

Swing_GUI_Notes_10-25-2005_-1.doc

Instructor:

P10

Class Description:

Introduced how to use the following Java Swing components: ActionListener; JButton, JTextField; setting the size for Java components; color selection; Fonts.

Interesting Discussion:

P05 Q (about the act-out next class): every time our client is supposed to write a tuple, do you have to necessarily to see the tuple, or do we just need to give the general sketch of what is going on?

A: This is from the user experience. For most of the time, the person (the user) should not be thinking "I am writing a tuple." The person is thinking "I am contributing to X"

Homework:

(1) Prepare a 2-side print, with picture, talking points;

(2) Prepare to run the activity in a simulated way, e.g. act out each of the project, like if any information needs to be passed through computer, we want to see it passed through paper physically.

Lecture 21

Date:

Nov. 01, 2005

Video Files:

1. Disc20_Title_01.avi, 00:00:00~end

Topic:

1. project act-out (section 1), including: (1) Apples-to-apples; (2) Math Bingo

Teaching Materials:

N/A

Instructor:

N/A

Class Description and Interesting Discussion:

Each group gave an overview of how their projects worked and acted out for the main activities by using papers, cards, etc. as if the students were interacting with the computer user interface.

Project 1. Apples-to-apples (00:00:00~00:23:20)

(1) Introduction: they had a mock user-interface, with a button "start a round", a list of player names on the right side, a couple of red-apple cards, and a green-apple card.

(2) Act-out: Players log in. Except the judge, each of them automatically gets seven noun cards. The judge starts a new round by clicking the "start round" button. One student acts as the tuple space. First he shows an adjective card (a green apple card) to other students as if the system automatically put a green apple card into the tuple space. Each student picks a red apple card and puts it into the space. The judge picks his favorite card and gives it to the space.

Players can decide if they want to discuss the reason that the judge picks a certain card or not. And the players need to figure out what cards the judge might like.

(3) Discussion:

Q: why the card was showed to the player one by one?

A: because in the physical game, we had to show one by one, but in the computer game, they should be sees at the same time. When the judge clicks reveal the cards, callback will notify each player the green apple card.

Q: What are the differences between whether the judge is known or not? Make the game better or worse?

Using people's name to login has disadvantage; use different ways to tell who the judge is on the screen, or symbol.

Q: Do all the players see the cards back flipped over together?

A: not finally decided yet.

Q: what are the coordination patterns?

A: rotating judge, judge selected, information that goes from the judge to other people; reaction from different people;

Q: how to quit the game?

A: close the client, socially notice other people leaving;

A classical kind of game coordination with explicit rotating rules.

Project 2. Math Bingo (0:24:00~1:01:00)

(1) the game:

Players (roles): picker, guesser

The picker needs to create an equation; sends it to the guessers. Each guesser has a board to work on the problem. There is also a sign to indicate when the guessers finish a certain problem (so the picker can continue to post problems). After a person claims a "Bingo", he/she needs to type in his/her name and notify the picker.

(2) Discussion: (Q represents Question, and A represents Answer)

Q: what happen when 2-3 people say Bingo simultaneously? Need time to type their name in

A: all of them win, or the first one claims,

Dr. T: teachers do not always pick the first one who raises hands. Have different reasons, some students always raise hand immediately then think again, some students won't raise hand until double check their answer.

Q: How to decide the winner? Let everyone to get a Bingo before the game ends;

P11: users can give the teacher a clue when the round takes too long;

The distribution of the numbers;

Students need to know where they make mistake; a). sb. calculates wrong; b). sb. thinks she has a bingo, but makes a mistake, her answer is not in the diagonal; can have an animation to show the errors; can have a list of the questions and marked wrong ones.

Homework:

N/A

Lecture 22

Date:

Nov. 03, 2005

Video Files:

1. Disc21_Title_01.avi, 00:00:00~end

2. Disc22_Title_01.avi, 00:00:00~end

Topic:

Project act-out (continued, section 2: with SRI people), including: (3) Crossword puzzle; (4) Algorithm Organ

Teaching Materials:

N/A

Instructor:

N/A

Class Description and Interesting Discussion:

Each group gave an overview of their how their projects work and act out for the main activities by using papers, cards, etc. as if the students were interacting with the interface.

Project 3. Algorithm Organ (00:02:00~00:37:29)

Here was the problem they tried to work on during the act-out activity:

f(x) = f(x-3) +2 (x>1)

f(x) = x+3

received x=10, f(10)=?

The game: Students stand up when they are picked to resolve the problem. And they tried to contribute the activity by resolving part of the problem, like this:

Student 1: f(10)=f(10-3)+2=f(7)+2

Student 2: f(7) = f(7-3) + 2 = f(4)+2;

Student 3: f(4) = f(4-3) + 2=f(1)+2;

Student 4: f(1) = 4;

So we could see that the fourth student got the base case. That was the first time that someone could solve part of the problem and submitted back to the server. Before that,

everyone could only solve part of the problem and submitted an equation to the server. They needed to request other's help to solve the whole problem.

Professor: need to discuss the interface, the representation of the equation;

SRI: what's the reasoning for having the original parameter passed back? Does it matter if a different person receives a returned function?

A: it's important to track down if a student makes an error during the calculation

Project 4. Crossword Puzzle (00:38:15~)

There were three teams. And each team had two persons and one laptop. They first showed login screen, including login name, password, selection of puzzle, and a description of a puzzle. When a team clicks a puzzle, e.g. 3 down, the client machine "take" that tuple from the space, write the answer, and then write back to the space. The quicker team will get more points. Ranking will be showed in the screen. Teacher can stop the game by pressing a stop button at any time. The winner team will be congratulated. Students can print the final answer and clues.

Q (SRI Ch): when they "take" the clue and "write"s the answer back, other people won't be able to "take" it any more, right? Can I change the answer?

A: it's taken from the list, so can't be taken any more.

Before scoring, only the team that submitted the answer can change their answer until it is marked correct or wrong. If wrong, any team can change it. Our system also supports message passing, so teams can send message to each other. If there's a confliction, they can message each other until both teams have an agreement.

Q (Professor): what happened if one team takes 1-down, and another team takes 1-across, where there is a conflict?

A: when there is a conflict, they can use message passing.

Q (P02): game play issue question: what happen if a team takes all of the questions, tuples so that other teams can't work on them?

SRI J: you could also change the rewarding system; e.g. sb. gets bigger points by providing correct glues.

P04: maybe it enhances the collaboration.

SRI Ch: so it may be better to think in letter level than in word level, you get credit by completing partial answers, i.e., a couple of letters in a word.

Professor: this may be too complex to implement for the semester;

SRI J: they can take off the messaging.

Lecture 23

Date:

Nov. 08, 2005

Video Files:

1. Disc22_Title_02.avi, 00:00:00~end

Topic:

1. Showed the interface that Charley designed for Algorithm Organ, public space and private space.

2. project act-out (continued, section 3)

(5) Pictionary (00:15:40); (6) Krypto

Teaching Materials:

N/A

Instructor:

Dr. Tatar

Class Description and Interesting discussion:

1. (00:00:00~00:14:20) Public/private pane, handwriting based system. Users can drag things back and forth between two to make them public side or private side; Charley doesn't care who "takes" the problem, just whoever is faster.

Professor: Interface is really simple, (a) maybe put in handheld; (b) distracting from the recursive.

2. Each group gave an overview of their how their projects work and act out for the main activities by using papers, cards, etc. as if the students were interacting with the interface. (continued)

Project 5. Pictionary (00:15:40)

5 drawers, 1 guesser, each drawer has 10 seconds to draw the picture; other drawers can't see when a drawer is drawing. The guesser can't see the drawing either before the guessing time is up.

Q: How to tell the drawing time is over?

A: look out for the timer while drawing is hard, can use audio or picture fading, etc.

Project 6. Krypto (00:35:00~end)

Six players try to use 5 numbers and 4 operators to come with a result. Players take turns.

Problem: students did not seem to think very hard and find a solution to the problem, some students was helping, while others just tried to confuse each other. This problem revealed design problems in the game. The game design did not bring too much fun and competition for players. The players had no obvious goal to work for.

(Lost some videos about the discussion and comments.)

Lecture 24

Date:

Nov. 10, 2005

Video Files:

1. Disc23_Title_02.avi, 00:00:00~end

Topic:

1. project act-out (continued, section 4), including: (7) Hangman (00:00:00~00:18:30)

2. discussed some design problems revealed from project act-out; and implementing suggestions

Teaching Materials:

Slides: implementing the game

Instructor:

Dr. Tatar

Class Description and Interesting Discussion:

Each group gave an overview of their how their projects work and act out for the main activities by using papers, cards, etc. as if the students were interacting with the interface.

(7) Hangman (00:00:00) (didn't record the beginning of the act-out)

Q: who chose the word?

A: system automatically chooses.

They had a "help" function. If someone clicks that button, people can vote for a letter. May be more interesting to let audience to vote for letters.

Dr. T: I'd like to see that somebody enters a word instead of picking a word from list.

2. implementing the game (Dr. Tatar)

last submission of 2-page report missed one of the components:

(1) goal: coordination;

(2) sequence diagram;

(3) major UI;

(4) tuple design;

Problems found in design

We made this very clear: tuples must be used to coordinate in the project.

Pictionary: how the communication between drawers is going to work? We didn't see the tuple components used.

Krypto:

a. The tuple design had problem. What are tuples for? Same as database? Not really. We want our data structure to communicate the movement.

b. Then we had some discussion about the possible tuple design for Krypto.

c. Extensibility: e.g. only 5 hands in the game, bad programming practice

Apples-to-apples:

a. Display fixed number of cards, and fixed number of players in the program. It is not user-tolerable; ideally dynamically reconfigure the display, but at least only one place to change the number.

b. Use people's name in the screen, you want to have a separate representation of the user name.

Order of the implementation:

(1) use of the tuples for coordination; (test UI)

(2) basic UI; UI for central coordination task;

(3) group setup;

(4) extras: timer, etc.

extra tuples

Crossword:

grain size: letter level, word level.

Emphasize: (1) use tuples work for the coordination; (2) design your data structure extendable; (3) make tuples findable

Email/web Discussion after Class:

by P07 - Sunday, 13 November 2005, 12:57 PM

Subject: Java Problem

Hi..

I cant create a new thread in the forum so replying to this one.

I have a problem with Java. I am trying to set the background color for a JLabel.

basically:

JLabel label = new JLabel("text");

label.setBackground(Color.YELLOW);

Doesnt work! If I use the same method for a JPanel it works! What am I doing wrong?

I want to color labels differently. Any hacks?

Sunday, 13 November 2005, 01:13 PM

I found it! Need to set the label Opaque first! label.setOpaque(true);

Hope it helps out other ppl

Lecture 29

Date:

11-29-2005

Video Files:

01. Disc24: Title_01.avi

Topic:

1. project final demonstration and discussion: (1) Apples-to-apples

Teaching Materials:

N/A

Instructor:

N/A

Class Description and Interesting Discussion:

(1) Apples-to-apples (00:00:00~00:15:00 demo; 00:15:00~00:37:00 discussion)

the program worked very well in overall

a. scores not updated correctly when multiple people select the same card;

b. only judge can reveal the cards, it seems other people won't have enough time to review the cards;

c. later-login not supported;

d. if judge loses the network connection, after he re-logs in he will still be the judge, but the session is lost, he has to restart the session for everyone;

e. can't see who gets the score in the latest round;

How to make this game more educational?

00:37:01~end: random discussion

Lecture 30

Date:

12-01-2005

Video Files:

01. Disc25: Title_01.avi

Topic:

project final demonstration and discussion (continued), including: (2) Krypto; (3) Algorithm Organ; (4) Hangman.

Teaching Materials:

N/A

Instructor:

N/A

Class Description:

(because we had three demos in one class, we did not have much discussion time for each team)

(2) Krypto (00:00:00~00:15:00)

a. use social control to decide the turns;

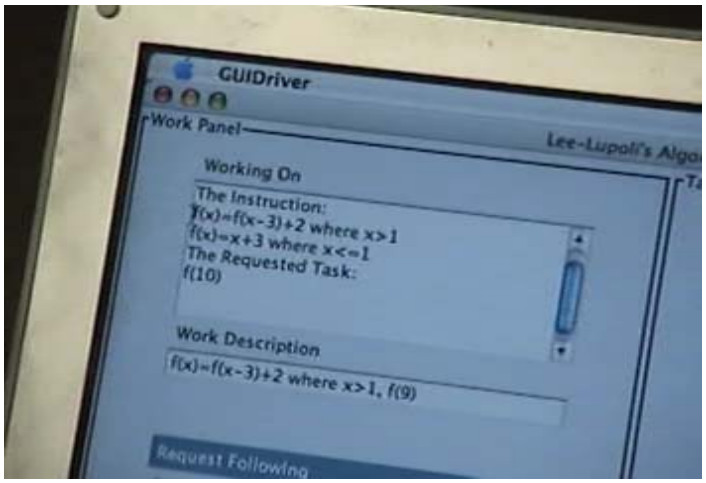
b. had to fill the numbers and operators one by one (bug), couldn't skip cards; target card on the left;

c. robust issues

(3) Algorithm Organ(00:16:20~)

limited graphic representation, but also use physical representation

please "raise your hand"



multiple tasks can be supported simultaneously

only the people who make the request will get the response

separate a way to introduce, see the whole stack instead of a list

(4) Hangman (00:41:00~00:45:15)

a. slow response to the clicking (after the demo, bugs were found in program);

b. when game is over, if user logs out, then logs in again, the program still tells the game is over

Lecture 31

Date:

12-06-2005

Video Files:

01. Disc25: Title_02.avi

02. Disc26: Tilte_01.avi

Topic:

Project final demonstration and discussion (continued), including: (5) Pictionary, (6) Math Bingo, and (7) Crossword Puzzle

Teaching Materials:

N/A

Instructor:

N/A

Class Description:

(5) Pictionary (00:05:00~end game setup, 00:02:00~00:21:00 demo and discussion)

a. 3 drawers totally

b. one drawer can't see other people's drawing, his drawing didn't show on the screen either. If press "refresh" in the web browser, he may see the update, but he would lose buttons in the screen to draw or do anything else.

c. 2 second drawers

d. there were lots of bugs in the program

(6) Math Bingo (00:23:00-00:44:00)

run successfully in handheld (3 machines); others could join with laptop, didn't find too many bugs

(7) Crossword Puzzle (00:48:00-end)

some students tried to break the program, but didn't get many bugs, might be lag issues between the red and xxx

didn't have a class discussion

only had one easy puzzle

E3. Template Matching Exercises

E31. Tuple Template Matching Exercise 1

Sept. 06, 2005

Tuple1: <"George","1 Main St,Santa Cruz,CA,95062"> Tuple2_1: <"George","305 Wall Street, Blackburg, VA, 24060"> Tuple2_2: <"George","305 Wall Street", "Blackburg, VA, 24060"> Tuple3_1: <"George","Blackburg, VA, 24060"> Tuple3_2: <<"George","Blackburg", "VA", "24060">> Tuple3_3: <"George","Blackburg", "VA", "24060"> Tuple4: <"George","24060"> Tuple5: <"George",24060> Tuple6: <"George"> Tuple7: <<"George">,"24060"> Tuple8: <String, "VT", 94060> Tuple9: <"student 1", <"John", "VT", 94060>> Tuple10: <"student 2", <"Tom", "VT", 94060>> Tuple10_1: <"student 2", <"Tom", "VT", "94060">> Tuple11: <"24060", "George">	Template1 <"George", String> Template2: <"George", Integer> Template3: <String, Integer> Template4<"George", Serializable> Template5<<String>, String> Template6: <String, Tuple8> Template7: <String, String >

Tuple1: <"George","1 Main St,Santa Cruz,CA,95062"> Tuple2_1: <"George","305 Wall	Template1 <"George", String > Template4<"George", Serializable>

Street, Blackburg, VA, 24060"> Tuple3_1: <"George","Blackburg, VA, 24060"> Tuple4: <"George","24060">	Template7: <String, String >
Tuple5: <"George",24060>	Template2: <"George", Integer> Template3: <String, Integer> Template4<"George", Serializable>
Tuple6: <"George">	N/A
Tuple7: <<"George">,"24060">	Template5<<String>, String> *not match tuple4 or tuple5
Tuple8: <String, "VT", 94060>	N/A
Tuple9: <"student 1", <"John", "VT", 94060>> Tuple10: <"student 2", <"Tom", "VT", 94060>>	Template6: <String, Tuple8> *tuple10 not match tuple10_1
Tuple3_2: <<"George","Blackburg", "VA", "24060">>	N/A *a tuple inside tuple
Tuple3_3: <"George","Blackburg", "VA", "24060">	N/A
Tuple2_2: <"George","305 Wall Street", "Blackburg, VA, 24060">	N/A
Tuple11: <"24060", "George">	Template7: <String, String > *not match tuple4
Tuple10_1: <"student 2", <"Tom", "VT", "94060">>	N/A *Not match tuple10

Answer:

Tuple1: <"George","1 Main St,Santa Cruz,CA,95062">

Tuple2: <"George","305 Wall Street, Blackburg, VA, 24060">

Tuple3: <"George","Blackburg, VA, 24060">

Tuple4: <"George","24060">

Matching Template1: <"George", String.class>

==========

Tuple5: <"George",24060>

Matching Template2: <"George", Integer.class>

Matching Template3: <String.class, Integer.class>

==========

Tuple4: <"George","24060">

Tuple5: <"George",24060>

Matching Template4: <"George", Serializable.class>

==========

Tuple6: <"George">

Tuple7: <<"George">,"24060">

Matching Template5: <Tuple7, String.class>

==========

Tuple8: <String.class, “VT”, 94060>

Tuple9: <“student 1”, <"John", “VT”, 94060>>

Tuple10: <“student 2”, <"Tom", “VT”, 94060>>

Tuple9 & 10 Matching Template6: <String.class, Tuple9>

E32. Tuple Template Matching Exercise 2

Sept. 06, 2005

Tuple1 *Tuple tuple1 = new Tuple("simpletest");*	**Template 1_1:** *Field strFormalField = new Field(String.class);* *template1_1 = new Tuple(strFormalField);*	Template1_1 matches tuple1; the field type is a String class, so it will match all of tuple with a string value field
	Template 1_2: *Template1_2 = new Tuple(String.class);*	Template1_2 will not match, since field value is String.class, but the type of the field is not String class, so the right format for a template to match a string value is: Template1_2 = new Tuple(new Field(String.class));

Appendix F. Participant Demographic Information and Self-reported Experiences

Table 16. Participant Demographic Information and Self-reported Prior Experiences

ID	Age	Gender	Ethnicity	Student status	Programming length (Year)	Programming experiences	Top-three programming languages	Teaching experiences
P01	23	M	Korean	Master	4	• C++, 4 years, however, not too much programming in last 2 years • Java, about 2 months, very little experiences	• C++, 4 years • Java, 1/6 years	No
P02	21	M	White	Senior	5	• 2 years, high school, basic and oracle 9i • 3 years at VT	C/C++, 3 years	No

P03	21	F	Indian	Senior	5	5 years	• C++, 5 years • Java, 1 year • C, 5 years	CS 1054, CS 1705, CS2604, 1 year
P04	23	M	Other	Senior	6	• Visual Basic 6 in high school • C++, C# for college courses • Develop in PHP/MySQL on my own	• C++, 5 years • PHP, 2 years • C#, 1 years	taught informally guitar and Kendo for about 1 year each. Guitar: 1 student, kendo: about 10
P05	22	M	Caucasian	Master	15	nearly 15 years, started with basic/visual basic, progressed to C++, currently mostly web programming with PHP; learning Ruby on Rails	• PHP, 2 years • JavaScript, 6 years • C++, 4 years	• taught gifted students to program in scheme-teaching assist only, 15 students; • helped teach HTML to middle-schoolers, 8-12 students

P06	21	M	Caucasian	Senior	7	7 years, which included making Mac programs that kicked people off AOL, making games on my TI-82 calculator C++ in high school, and college, learning PHP on my own, and applying my knowledge of PHP in a job where I created a website with functionalities similar to blackboard.	• C++, 5 years • PHP, 5 years • Basic for TI-82, 2 years	No
P07	24	M	India	Master	6	5-6 years, mostly at school, various CS classes	Java 5 years	1 semester, TA for CS networking class, 10 students, lab session, explaining experiment, helping with problems, grading homework
P08	23	M	African-American	Master	5	5 years, C++, VB, HTML, etc., most came from my undergraduate. I also gain work experience with HTML attending my undergrad and during summers	• C++, 5 years • Visual Basic, 1 year • HTML, 5 years	No
P09	32	M	Korean	Master	3	3 years	C++, 2-3 years	No

P10	28	M	Italian/ German	Ph.D.	12	12 years, oracle, JSP, Basic, VB, c/C++, HTML, JAVA, PHP, JavaScript	• C++, 8 years • Java, 3 years • PHP, 1 Year	7 school years, 2 high school, 5 college/community collage, class size 25, VT, 105 students
P11	33	M	Asian	Ph.D.	8	8 years	• C, 3 years • VC++/C++, 4 years • Java, 1 year	3 years: Multimedia-50 people, C Language-50 people, Circuit Design-70 people, MS Access-80 people
P12	22	M	Vietnamese	Senior	6	• 6 years: 11th grade, Pascal class • 12th grade, C++ class • 4 years college, C++, some C#	• C++, 5years • C#, 1-2 years • Java, 1 year	No
P13	21	M	Chinese	Senior	3	3 years, taking and passing all required course for CS major, C++	C++, 3 years	No
P14	23	M	Asian	Master	6	6 years, school projects and homework in varies languages for fun	• C++, 5 years • Java, 2 years • Scheme, 1 years	No

Appendix G. Tuple Design

Table 17. Tuple Design for Algorithm Organ

Tuple definition	Structure	Description	System role
MemberTuple	<”Member”, int Id, string State>	Used for keeping track of list of students and their state.	used for identify user’s working state
TaskTuple	<”task”, string Text, string Request>	Used for describing a task.	Public resource, written to the space as part of StackTuple
StackTuple	<”Stack”, <Task Tuple>, <Task Tuple>, ….>	Used for representing the stack logically. The history of all the requests made up to the when a student receives a request to complete a task. The list is in the order of oldest to most recent.	Public resource, written to the space as part of RequestTuple
RequestTuple	<”Request”, int Id, <Stack Tuple>>	Used for when a request to complete a task is made. The last Task Tuple in the Stack Tuple is the actual task that the student who received the request is to complete.	Public resource
ResponseTuple	<”Response”, int Id, string ReturnValue>	Used for when a response is being made to a request.	Public resource

Table 18. Tuple Design for Apples-to-Apples

Tuple definition	Structure	Description	System role
CurrentJudgeTuple	<String:userID> the name of the current judge	a game is on-going	signal the state of the game (system character)
CurrentGreenAppleTuple	<String:word>	the current “green apple” (adjective) for the round	Public resource
RevealTuple	<>	tuple which is posted by the judge's client to let the player clients know it's time to reveal the other players’ “red apples” (nouns)	Signal for synchronization
PointTuple	<String:userID>	shows player’s score. every time a player scores a point, another Point tuple containing their name is added to the space	Shared resource
GreenApple	<String:word>	“green apple” word (adjective)	Public resource
RedAppleTuple	<String:word>	“red apple” word (noun)	Public Resource
PlayerAnswerTuple	<String:userID, String:word>	each player adds his/her “red apple” answer for each round	Public resource

Table 19. Tuple Design for Crossword Puzzle

Tuple definition	Structure	Description	System role
LoginTuple	<String PuzzleID, String userID, String pw >	Contains user's information.	Public resource
PuzzleTuple	<String PuzzleID, String title, String description>	Contains the puzzle title and the description.	Public resource
GridTuple	<String PuzzleID, Integer X_dimension, Integer Y_dimension>	Stores the puzzle dimension.	Public resource
CellTuple	<String PuzzleID, <Integer X, Integer Y>, String cellType, String acrossLetter, String downLetter>	Contains the location and display information of a cell. cellType can be a clue number such as String "1", "White", or "Black." acrossLetter and downLetter are used to display both letters involved in a conflict in a cell.	Shared resources: show confliction
ClueTuple	<String PuzzleID, String ClueID, String clueNumber, String direction, <Integer X, Integer Y>, Integer wordLength, String clue>	Contains a clue, and all the location information of the corresponding answer. direction is either "ACROSS" or "DOWN"	Public resource
SolutionTuple	<String PuzzleID, String ClueID, String "CorrectLetter", <Integer X, Integer Y>, String oneLetter>	Contains a letter of a correct answer.	Public resource

Table 20. Tuple Design for Krypto

Tuple definition	Structure	Description	System role
NumCardsTuple	<"NUMCARDS", Integer numberOfCards>	The number of cards for a problem	Public resource, global variable
PlayerTuple	<"Player" Integer ProbID, Integer PlayerID>	Stores all the players associated with a certain game	Not have any real meaning in the program
ProblemTuple	<"Problem", Integer ProbID, Integer Content, Integer SeqNum>	Stores the current state of the hand (i.e., which cards have been used, the order to display the cards) The target value will always be stored in sequence number 0. This design allows for a problem to consist of any number of cards.	Resource (producer/consumer) synchronization, interaction between user and computer Transfer (transferred from the producer activity to consumer)
AnswerNumTuple	<"AnswerNum", Integer ProbID, Integer Content, Integer SeqNum>	Along with AnswerOp, stores the current state of the solution. Works like Problem tuple	
AnswerOpTuple	<"AnswerO", Integer ProbID, Integer Content, Integer SeqNum>	Along with AnswerNum, stores the current state of the solution. Stores the operators used and the order they are in	
HistoryValueTuple	<"HISTORY", Integer HistoryID, String Content, Integer SeqNum>	Stores all the moves made for a particular game. Used in conjunction with History.	Shared resources: to create awareness (of history)

StatusTuple	<“Status”, Integer ProbID, Integer HandSize, Integer AnsSize, Integer OpSize>	To keep track of when a client has finished writing	Shared resource, global variable

Table 21. Tuple Design for Hangman

Tuple definition	Tuple Structure	Description	System role
UserTuple	<username>	One String field presents user name	Public information
UserWordTuple	<username, “wordChoice”, and wordChoice>	The word user contributes to the words bank	User Contribution
WordToGuessTuple	<"wordToGuess", wordToGuess>	There can be only one, and it represents the word which needs to be guessed	Shared resource
LengthWordToGuessTuple	<“size”, int size, string word>	There can be only one, and it shows the length of the word	Public resource, used as a global variable
LetterClickedTuple	<"UserClicked", String letterClicked, "YES">	“YES”means it is being used in the word, “NO” means it’s not being used.	Shared Resources
CurrentGuessTuple	<"WordArray", __A__>	This shows the current word with dashes and letter that user guessed correctly.	Shared resource
GuessLetterAsciiTuple	<”letterUsed”, Integer ascLetter, Integer seqNum>	ascLetter is the Ascii code of the letter needs to guess, the seqNum is the position inside the word to guess	Public resource, used as a global variable

Table 22. Tuple Design for Math Bingo

Tuple definition	Tuple Structure	Description	System role
ImAPlayerTuple	<string “ImAPlayer”, string clientID, string PlayerType>		User identification
EquationTuple	<string “Equation”, string number, string equation>	Equation need to solve each time	Information for sharing, task
BingoTuple	<string “mb_bingo”, string clientID, string playerName, string B, string I, string N, string G, string O>		
ChosenTuple	<string "mb_chosen", string userID>	The user who claims Bingo successfully, used to check whether the claimed Bingo is correct or not	Message passing

Table 23. Tuple Design for Pictionary

Tuple definition	Tuple Structure	Description	System role
Game Creating Tuple	[UserType<String>, GameName<String>, DrawingTime<String>, NumberofDrawers<String>]	(To register game space) game related information	Mixed information
Game Joining Tuple	[UserID<String>, UserType<String>	(To join the game) user role in the game	User identification and game stage transfer
Posting Tuple	[Word<String>, Drawing Time<String>]	(To post a WORD to drawers) word to guess and time	public resource
Answering Tuple	[Answer<String>, UserID<String>]	(To post an answer) guesser's answer	Public resource
Next Drawer Tuple	[FinishingKeyword <String>, Next Drawer'sNumber <String>]	(To tell next drawer to start) notify the next drawer that he needs to start	Signal (peer-to-peer message)
Send Drawing Tuple	[Class<Class>, Lines<Vector>, Colors<Vector>]	(To send current drawing) drawing	Information for sharing, synchronization

Appendix H. Statistical Results from Programming Reflection Questionnaire

	P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11	P12	P13	P14	Ave.	SD
1.1. Tuple /Tuplespace idea itself	7	9	7	7	7	6	7	9	7	8	5	9	7	10	**7.5**	**1.3**
1.2. Tuple basic functions: Read, Write, Take	9	9	8	7	9	8	10	9	9	10	9	10	6	10	**8.8**	**1.2**
1.3. Tuple Operations: countN, multiRead, multiTake	8	8	10	5	8	9	4	7	9	6	5	10	4	10	**7.4**	**2.2**
1.4. template and matching	9	8	6	4	7	4	3	7	9	4	9	10	7	10	**6.9**	**2.4**
1.5. Tuplespace callback and eventRegister	2	10	4	5	7	9	2	6	9	4	7	9	4	9	**6.3**	**2.8**
2.1. Eclipse Detect error(s)	5	7	2	9	7	9	7	9	8	7	9	9	5	8	**7.2**	**2.0**
2.2. Eclipse Automatically fix errors	9	8	10	8	8	9	7	9	8	2	9	10	5	6	**7.7**	**2.2**
2.3. Eclipse Automatically prompt methods/functions for class/object	9	9	10	8	7	4	7	9	9	6	9	10	5	9	**7.9**	**1.9**

2.4. Eclipse Debug	1	7	1	7		5	5	9	6	6	9	8	4	8	**5.8**	**2.6**
2.5. Eclipse Manage project/source files	2	8	10	3	4	3	4		8	6	9	9	3	7	**5.8**	**2.8**
2.6. Jigloo (the Swing/SWT Editor), choose "Other" if you didn't use it and fill the reason		7				8		6			9			8	**7.3**	**1.1**
2.8. Overall, is Eclipse helpful for your programming in Java? (10-Absolutely Helpful)	6	7	8	7	7	5	3	10	9	6	8	10	4	7	**6.9**	**2.1**
3.1. How many hours did you spend on understanding and designing Clicker?	10	4	2	3	0.5	2		2	10	3	3	2	1	1	**3.3**	**3.1**
	10	4	2	3	0.5	2		5	10	3	3	2	1	1	**3.6**	**3.1**
3.2. How many hours did you spend on programming (Coding and Debugging) on Clicker?	30	4	4	6	0.5	2		5	10	3	5	3	4	3	**6.1**	**7.5**
	30	4	4	6	0.5	2		7	10	3	5	3	4	3	**6.3**	**7.5**
4.2. How many hours did you spend on implementation by your own for group project?	100	10	100	5	15	36		40	6	12	100	30	1	12	**35.9**	**38.4**
4.3. How many hours did you spend on implementation with your teammate for group project?	50		100	25	10	36		15	36	2	50	8	28	1	**30.1**	**27.9**

Appendix J. Related Publications

- **Sirong Lin,** D. Tatar, *Encouraging Parallel Thinking through Explicit Coordination Modeling*, SIGCSE '11, The 42nd ACM Technical Symposium on Computer Science Education, Dallas, Texas, USA**,** March 9-12, 2011
- **Sirong Lin**, D. Tatar, *Beyond Current Social Computing: Challenges to Complex Coordinated Systems Design?*, CSCW' 2010, CSCW Horizon, Savannah, Georgia, USA, Feb. 6-10, 2010
- Book Chapter: D. Tatar, **Sirong Lin**, et al., "*Playground Games and the Dissemination of Control in Computing and Learning*", in Educating Learning Technology Designers: Guiding and Inspiring Creators of Innovative Educational Tools, Routledge Press (Taylor & Francis Group), 2008
- **Sirong Lin**, Tatar, D., Paper Abstract, "*Programmer Cognition in Tuple Spaces for Collaborative Learning*", The 6th International Conference on Education and Information Systems, Technologies and Applications: EISTA2008, Orlando, FL, USA, July 2008
- **Sirong Lin**, Tatar, D., Harrison, S., *Learning When Less is More: "Bootstrapping" Undergraduate Programmers as Coordination Designers*, 2006 Participatory Design Conference (PDC2006), Trento, Italy, August 1-5, 2006