The Pennsylvania State University

The Graduate School

College of Information Sciences and Technology

USING COGNITIVELY INSPIRED AGENTS
AND INFORMATION SUPPLY CHAINS
TO ANTICIPATE AND SHARE INFORMATION
FOR DECISION-MAKING TEAMS

A Thesis in

Information Sciences and Technology

by

Shuang Sun

© 2006 Shuang Sun

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2006
The thesis of Shuang Sun was reviewed and approved* by the following:

John Yen  
Professor in Charge of College of Information Sciences and Technology  
University Professor of Information Sciences and Technology  
Thesis Advisor  
Chair of Committee

Michael McNeese  
Associate Professor of Information Sciences and Technology

Akhil Kumar  
Professor of Smeal College of Business

Tracy Mullen  
Assistant Professor of Information Sciences and Technology

Madhu Reddy  
Assistant Professor of Information Sciences and Technology

Joseph Lambert  
Senior Associate Dean  
Associate Professor of Information Sciences and Technology  
Chair, Graduate Programs Advisory Committee  
Head of the College of Information Sciences and Technology

*Signatures are on file in the Graduate School
ABSTRACT

September 11 and hurricane Katrina have shown that timely information is important not only for disaster prevention but especially valuable in effective disaster response. From the point of view of information and communication technologies, the challenge is how to coordinate information sharing effectively among members of a complex decision-making team (e.g. the first responders for a disaster). A common difficulty is to provide useful and time-sensitive information to team members quickly but at the same time not overwhelm them with irrelevant information. This problem has also been encountered in other application domains that require effective communication in a team environment: examples include military command and control, health care, and global enterprise.

Research in the area of team cognition suggests that anticipating information needs of other teammates is a key behavior for achieving highly efficient and effective teamwork. Guided by this finding, a framework of Information Supply Chain (ISC) is proposed and implemented in this thesis research. ISC contains three novel features. First, it anticipates information requirements using a cognitively inspired decision model. Second, it consolidates and prioritizes the information requirements using a novel planning algorithm. Third, it integrates inference, information seeking, and auction for satisfying the information requirements. The ISC framework is formalized using existing agent theories as well as implemented in an agent architecture called R-CAST. The efficiency and the formation of ISC are evaluated using an experiment in a simulated “information market”.

This research has made two major contributions in addressing the challenges of information sharing among decision-making teams. First, more accurate information needs can be anticipated using a high-level cognitive model of decision-makers. This avoids “pushing” irrelevant information to a decision-maker, which often leads to information overload. Second, the cost associated with information seeking and distributing activities can be greatly reduced because these activities can now be well-coordinated within the ISC framework. In summary, the work presented in this thesis can help a human team to make better decisions under time pressure, especially in a distributed environment where an immense amount of information and knowledge are scattered among members of the team.
# TABLE OF CONTENTS

Chapter 1  Introduction ................................................................. 1  
  1.1 Research Motivations ............................................................... 2  
  1.2 Present Models for Information Sharing and Their Limitations ........ 5  
  1.3 Research Questions ............................................................... 7  
  1.4 Research Scope .................................................................... 8  
  1.5 Information Sharing: An Information Usage Perspective .......... 11  
  1.6 Major Research Results ............................................................ 14  
  1.7 Thesis Outline .................................................................... 15  

Chapter 2  Background ................................................................. 16  
  2.1 Introduction ...................................................................... 16  
  2.2 Cognitive Models of Decision-making .................................... 17  
  2.2.1 Team Cognitions .............................................................. 18  
  2.2.2 Decision-making Models .................................................. 19  
  2.2.3 Recognition Primed Decision-making Model ...................... 19  
  2.3 Workflow Process Models ........................................................ 22  
  2.4 Agent Technologies ............................................................... 23  
  2.4.1 What is Agent? ................................................................. 24  
  2.4.2 Agent Theories ................................................................. 26  
  2.4.3 Knowledge Representation ............................................... 28  
  2.4.4 Agent Communication ..................................................... 30  
  2.4.5 Cognitive Architectures .................................................... 32  
  2.4.6 Agent Oriented Methodologies ......................................... 36  
  2.4.7 Information Agents .......................................................... 38  
  2.4.7.1 Broker ...................................................................... 39  
  2.4.7.2 Matchmaker ................................................................. 40  
  2.5 Market and Agents ............................................................... 41  
  2.5.1 Agent Auctions ................................................................. 41  
  2.5.2 Contract Net Protocol ....................................................... 42  
  2.6 Conclusions .................................................................... 43  

Chapter 3  Task-Oriented Information Supply Chain Framework .......... 44  
  3.1 An Overview of the Information Supply Chain (ISC) Framework .... 44  
  3.2 Formal Foundations ............................................................... 47  
  3.2.1 Notations .................................................................. 48  
  3.2.1.1 Basic Logical and Mathematical Notations: .................... 48  
  3.2.1.2 Notations for the Information Supply Chain Framework ... 48  
  3.2.1.3 Notations about Tasks and Actions .............................. 49  
  3.2.1.4 Notations about Agents .............................................. 49  
  3.2.2 Research on Proactive Information Exchange in Agent Teamwork .... 50
3.2.3 Two Fundamental Concepts ......................................................... 51
  3.2.3.1 Task ...................................................................................... 51
  3.2.3.2 Information............................................................................ 56
  3.2.4 Assumptions ........................................................................... 60
3.3 Anticipating Information Needs in Task Contexts .................................. 62
  3.3.1 Information Need........................................................................ 62
  3.3.2 Information Needer ................................................................... 65
  3.3.3 Recognize Information Needs ...................................................... 66
    3.3.3.1 Anticipating Information Needs ............................................. 67
    3.3.3.2 On-demand Information Needs ............................................. 69
    3.3.3.3 Comparing Anticipating Information Needs with Waiting for
    On-demand Needs ............................................................................ 70
  3.3.4 Satisfying an Information Need .................................................... 72
  3.3.5 Committing to Information Needs .............................................. 73
3.4 Information Requirement Planning ....................................................... 74
  3.4.1 Transforming Information Needs to Information Requirements ............. 75
  3.4.2 Consolidate Information Requirements ......................................... 77
  3.4.3 Determine Sources .................................................................... 80
  3.4.4 Knowledge-based Information Requirement Decomposition ................. 82
  3.4.5 IRP Algorithm .......................................................................... 85
  3.4.6 Challenges .............................................................................. 87
  3.4.7 Evaluation of Information Management ......................................... 88
3.5 Information Supply Chain ................................................................. 91
  3.5.1 Information Partner .................................................................... 91
  3.5.2 Definition of Information Supply Chain .......................................... 93
  3.5.3 Basic Communication Modes ...................................................... 94
  3.5.4 Basic ISC Protocol ................................................................... 97
  3.5.5 The Benefits of ISC .................................................................. 99
3.6 Establishing Information Partnership and Forming Information Supply
  Chain ................................................................................................... 102
  3.6.1 Extending Contract Net for Information Auction ............................... 102
  3.6.2 Chain Auction, an Example ......................................................... 105
  3.6.3 Bidding Behavior ..................................................................... 106
  3.6.4 Discussions ............................................................................. 107
3.7 Developing ISC Framework from SCM .............................................. 108
  3.7.1 ISC differs from SCM ................................................................. 113
  3.7.2 ISC Framework Unifies Existing Methods ...................................... 114
3.8 Conclusion ...................................................................................... 115

Chapter 4 Realizing the ISC framework within R-CAST: an Agent Architecture 117

  4.1 The R-CAST Architecture .............................................................. 120
    4.1.1 Framework ......................................................................... 122
    4.1.2 Realizing Decision-making Process Model in R-CAST ................. 123
    4.1.3 Anticipate Information Needs in R-CAST ................................. 125
4.1.4 An Integration Perspective ................................................................. 127
4.1.5 Control and Interface ........................................................................ 129

4.2 The R-CAST Components .................................................................... 131
4.2.1 Active Knowledge base ................................................................. 132
   4.2.1.1 AKB Key Features ................................................................. 132
   4.2.1.2 AKB Syntax ........................................................................ 134
   4.2.1.3 AKB Interface Functions ...................................................... 137
4.2.2 Process Manager .............................................................................. 138
   4.2.2.1 Process Manager Key Features ............................................. 138
   4.2.2.2 Process Knowledge Syntax and Characteristic ....................... 141
   4.2.2.3 Process Manager Interface Functions ..................................... 145
4.2.3 RPD Decision-maker ....................................................................... 146
   4.2.3.1 RPD Model design ............................................................... 148
   4.2.3.2 Experience Knowledge Syntax ............................................. 152
4.2.4 Task manager .................................................................................. 154
4.2.5 Information Manager ....................................................................... 157
4.2.6 Communications Manager ............................................................. 159
4.2.7 Auctioneer ...................................................................................... 163

4.3 Lessons Learned .................................................................................. 165
   4.3.1 Configurability Leads to Flexibility ............................................. 166
   4.3.2 Component-based Design Leads to Robustness ......................... 167
   4.3.3 Two Perspectives to Knowledge Engineering ........................... 168
   4.3.4 General Implementation Guidelines ........................................... 170

Chapter 5 Experiments and Results ......................................................... 173

5.1 Experiment 1: Using R-CAST Agents to Model and Assist Decision-
making Tasks ......................................................................................... 175
5.1.1 Introduction .................................................................................... 175
5.1.2 Scenario Design ............................................................................. 176
   5.1.2.1 The Blue Team ................................................................. 177
   5.1.2.2 The Red Team ................................................................... 179
5.1.3 Agent Models ................................................................................. 180
5.1.3 Procedure ....................................................................................... 182
   5.1.3.1 The Blue Team Configuration ............................................. 182
   5.1.3.2 The Red Team Configuration and Scenario Settings ............ 186
   5.1.3.3 Equipments ....................................................................... 186
5.1.4 Results ........................................................................................... 187
5.1.5 Summary ......................................................................................... 191

5.2 Experiment 2: Forming Information Supply Chains (ISC) ................... 191
5.2.1 Introduction ..................................................................................... 192
5.2.2 Color Block Game Settings ........................................................... 193
   5.2.2.1 Game Design .................................................................... 193
   5.2.2.2 Game Monitor ................................................................. 196
5.2.3 Agent Models ............................................................................... 197
LIST OF FIGURES

Figure 1-1: Timeline of information sharing in emergency response (Chen et al 2005). .......................................................... 3

Figure 1-2: A research roadmap. .................................................. 9

Figure 1-3: Three perspectives on information sharing. .................... 12

Figure 2-1: RPD model (Klein 1989). ........................................... 20

Figure 2-2: Agent technologies in seven areas. .............................. 23

Figure 2-3: W3C semantic Web stack (W3C 2006). ......................... 30

Figure 2-4: CAST agent architecture ........................................... 35

Figure 2-5: An information broker architecture (Martin et al 1997). .... 39

Figure 3-1: Three stages of a task and their time points. ................... 54

Figure 3-2: What information an agent needs v.s. what needs the agent knows. ......... 66

Figure 3-3: Information must remain valid .................................... 72

Figure 3-4: Multiple seeking plans to cover the duration of a need. ........... 74

Figure 3-5: Basic IRP process .................................................... 75

Figure 3-6: Overlapping information requirements ............................ 78

Figure 3-7: Close (in time) information requirements.......................... 78

Figure 3-8: Consolidated information seeking actions .......................... 79

Figure 3-9: An agent should plan within its capacity constraints ............ 80

Figure 3-10: Multiple types and recipes of information seeking task .......... 81

Figure 3-11: A BOM tree and an IDR tree ................................. 83

Figure 3-12: Multiple information fusion rules .................................. 85
Figure 3-13: Information need sets. ................................................................. 88
Figure 3-14: An information supply chain ...................................................... 94
Figure 3-15: Three communication models .................................................... 95
Figure 3-16: Duplicated and circular demands ............................................... 98
Figure 3-17: Comparing ISC with other communication models ...................... 100
Figure 3-18: Information auction protocol ..................................................... 103
Figure 3-19: An information auction example .............................................. 105
Figure 3-20: A material supply chain and an information supply chain ............ 109
Figure 3-21: Developing ISC from SCM ....................................................... 110
Figure 3-22: Unifying information sharing methods with the ISC framework ... 115
Figure 3-23: 3rd party ordering and 3rd party inquiry .................................... 115
Figure 4-1: Using an agent to model and support a cognitive task .................. 117
Figure 4-2: R-CAST agent and its environment ........................................... 121
Figure 4-3: R-CAST architectural framework ............................................. 122
Figure 4-4: R-CAST cognition ................................................................. 124
Figure 4-5: Managing information requirements ....................................... 126
Figure 4-6: R-CAST component integration ............................................... 129
Figure 4-7: An R-CAST agent interface ..................................................... 130
Figure 4-8: AKB interface ........................................................................ 134
Figure 4-9: Process state transitions ......................................................... 139
Figure 4-10: PM interface ........................................................................ 141
Figure 4-11: Enacting contingencies ......................................................... 144
Figure 4-12: Computational RPD model .................................................... 147
Figure 4-13: RPD state transitions ................................................................. 148
Figure 4-14: Identifying experiences in R-CAST through RPD. ...................... 149
Figure 4-15: A collaborative decision space .................................................... 150
Figure 4-16: RPD interface ........................................................................... 151
Figure 4-17: TM interface ............................................................................. 155
Figure 4-18: Overall information management process .................................. 158
Figure 4-19: IM interface ............................................................................. 159
Figure 4-20: CM interface ............................................................................ 160
Figure 4-21: Conversation management ....................................................... 161
Figure 4-22: Auction process ....................................................................... 164
Figure 4-23: Auctioneer interface ................................................................. 165
Figure 4-24: The Living Lab framework (McNeese et al. 2005). .................... 171
Figure 5-1: A screen shot of S2 interface. .................................................... 177
Figure 5-2: Team configuration ................................................................... 183
Figure 5-3: Interface for moving pattern inputs ............................................ 186
Figure 5-4: Performance evaluation ............................................................... 188
Figure 5-5: Variant human performance v.s. stable agent performance .......... 190
Figure 5-6: An IDR in CBG game ................................................................. 194
Figure 5-7: Color block game monitor ............................................................ 196
Figure 5-8: An ISC in a CBG ....................................................................... 199
Figure 5-9: Three supply models ................................................................. 200
Figure 5-10: Comparing performances of three supply models ...................... 205
Figure 5-11: Comparing average utilizations of three supply models .......... 206
Figure 5-12: Formation of information supply chain ........................................ 208

Figure D-1: Whiteboard class diagram ................................................................. 244

Figure D-2: AKB class diagram ................................................................. 245

Figure D-3: AKB sequential diagram ................................................................. 246

Figure D-4: PM class diagram ................................................................. 247

Figure D-5: PM sequential diagram ................................................................. 248

Figure D-6: RPD class diagram ................................................................. 249

Figure D-7: RPD sequential diagram ................................................................. 250

Figure D-8: TM class diagram ................................................................. 251

Figure D-9: CM class diagram ................................................................. 252

Figure D-10: CM sequential diagram ................................................................. 253

Figure D-11: IM class diagram ................................................................. 254

Figure D-12: IM sequential diagram ................................................................. 255

Figure D-13: Auctioneer class diagram ................................................................. 256
# LIST OF TABLES

Table 2-1: Background Overview..........................................................................................26
Table 2-2: BDI Interpreter (Rao et al. 1995). ........................................................................26
Table 2-3: KQML Performative Example (Finin et al. 1994). ..............................................31
Table 2-4: MaSE Methodology (Wood 2000). ......................................................................37
Table 3-1: Comparing the On-demand and Anticipated Needs.........................................72
Table 3-2: Comparing Basic Communication Modes ..........................................................97
Table 4-1: Anticipate Information Needs in R-CAST.........................................................127
Table 4-2: AKB Knowledge Definition Syntax....................................................................135
Table 4-3: AKB interface functions....................................................................................137
Table 4-4: An Example of Process That Count Numbers...............................................139
Table 4-5: Process Knowledge Syntax................................................................................142
Table 4-6: Experience Knowledge Syntax...........................................................................153
Table 4-7: Task Specification Syntax...................................................................................156
Table 4-8: R-CAST Communication Message Types.........................................................163
Table 5-1: BFA Properties....................................................................................................178
Table 5-2: A Rule Example. ...............................................................................................181
Table 5-3: A Plan Example...................................................................................................182
Table 5-4: An Experience Example........................................................................................182
Table 5-5: Comparing S3 Performance. ..............................................................................187
Table 5-6: Standard Deviation Comparison between Human and Agent S4 Player. ..191
Table 5-7: An Example of Agent Capabilities in CBG. .....................................................198
Table 5-8: Parameters Used in a typical CBG. ................................................................. 202
Table C-1: R-CAST Commands .................................................................................. 242
Table D-1: R-CAST Design UML Overview ................................................................. 243
Table E-1: Overview of the File Names and Purposes ................................................. 257
ACKNOWLEDGEMENTS

Many people have played significant roles in the successful completion of this dissertation research. First, I sincerely thank my adviser, Dr. John Yen, for five years of support, care, dedication, and guidance. I am very fortunate and grateful to have Dr. Yen as my mentor. His help will never be forgotten.

I also thank members of my doctoral committee. Dr. Michael McNeese taught me cognitive science and provided great help on RPD model. Dr. Tracy Mullen worked closely with me on agent auctions and the trading agent competition. Dr. Akhil Kumar provided me lots of insights on workflow modeling. Dr. Madhu Reddy guided my writing and presenting. I appreciate their efforts and constructive suggestions throughout my doctoral program.

My research benefits from many other faculty members. A special note of gratitude goes to Dr. Frank Ritter who taught me cognitive modeling and gave me advice on how to be a professional scholar. I also thank Dr. Peng Liu for giving advice on research about information security. Dr. Chao-Hsien Chu, in my proposal committee, also advised me on how to get through stages of my PhD study.

In addition to faculty members, my fellow group-mates also provided inspiration through countless hours of discussion and debates. I thank Dr. Xiaocong Fan for useful comments on Chapter 3. I am also thankful for fellow graduate students: Rui Wang, Cong Chen, Kaivan Kamali, Guruprasad Airy, Shizhuo Zhu, Viswanath Avasarala, Bingjun Sun, and Po-Chun Chen.
The R-CAST process syntax and the DDD agent adapter are extended from CAST 2.0, designed by Michael Miller. Mathew Davis offered prodigious support in software development. Professional editing services from Roger Dudik improved the quality of this thesis. Their help saved my time tremendously and is highly appreciated.

I dedicate this thesis to my family: my parents, my bothers, and finally my wife Ying. Your love makes my achievement possible and makes my life meaningful.
Chapter 1
Introduction

The crisis of 9/11 and hurricane Katrina has shown that timely information is important for preventing and responding to disasters. The 9/11 Commission [1] found that “poor information sharing was the single greatest failure of our government in the lead-up to the 9/11 attack.” Failure to share information adequately, within and across agencies, was a significant factor that led to missing opportunities to disrupt the 9/11 plot. The 9/11 Commission recommended a better information sharing system for connecting the “dots” and helping intelligence analysis teams to draw on all relevant sources of information [1].

After four and half years, however, hurricane Katrina revealed this problem once more. The catastrophic disaster overwhelmed the decision-makers for an initial period of time. The Committee to investigate the preparation for and response to hurricane Katrina published a similar finding to the 9/11 Commission that suggested “better information would have been an optimal weapon against Katrina” [2]. The Committee urged for a system that can share information “within agencies, across departments, and between jurisdictions of government” [2]. The system must enable the information to be sent to the right people at the right time in a secure and efficient fashion.

From the point of view of information and communication technologies, the challenge is how to coordinate information sharing effectively among members of a complex decision-making team. A common difficulty is providing useful and time-
sensitive information to team members quickly without overwhelming them with irrelevant information. This problem has also been encountered in other application domains that require effective communication in a team environment: examples include military command and control [3], heath care [4, 5], and global enterprise [6, 7].

1.1 Research Motivations

Information and knowledge are distributed across people, systems, and locations. For example, the contents in a distributed information system may be retained by professionals specializing in various fields, or distributed via software systems of different platforms, or stored in hardware media in geological locations farther apart. Consequently, people must share information effectively in order to support the tasks they want to accomplish. Sharing information requires effective integration of elemental steps that include seeking\(^1\), processing, and distributing useful information in a timely manner. Recent development of new technologies such as the Web and large-scale database systems are examples ways to make information sharing easy and efficient. An unintended consequence of the Web, however, is that the amount of information available is also increasing at an explosive rate. This creates another key requirement in information sharing, that is, it has to be accurate so that only the most relevant information is selected, and efficient so that a large volume of information can be processed quickly for decision-making.

\(^1\) In general, information sharing and information seeking are separate activities. In this thesis, information sharing refers to any activities related to supporting decision-makers with useful information. Therefore, information sharing encompasses information seeking.
The need for information sharing and the associated challenges for a decision-making team can be illustrated by an example where decisions have to be made based on time-sensitive information from multiple sources. Chen et al. [8] analyzed a coordinated emergency response system where teams of responders are divided into three tiers according to their dispatch sequence: the 1st tier consists of FBI, police, firefighters, and emergency medics; the 2nd tier includes hazardous material workers and medical doctors; and the 3rd tier is made up of security inspectors, waste disposal technicians, and government agencies. Figure 1-1 shows the timeline for the available information and the timeline for the needed information during the entire course of an emergency dispatch.

Figure 1-1: Timeline of information sharing in emergency response (Chen et al 2005).
Clearly, this diagram shows an uneven distribution of the available information and the needed information for supporting decision-making. A peak in the timeline of the available information may represent a situation of information overload or too much noisy information; whereas a peak in the timeline of the needed information may indicate that some responders are overwhelmed by information seeking tasks. The lack of needed information may result from insufficient capability or from lack of resource and coordination. For example, fire fighters do not have the capability of assessing potential hazard materials, but they can delegate this task to the hazmat team. However, if the hazmat team has been assigned to other tasks, then it will be unable to provide the information in a timely manner. Therefore, resources must be allocated properly to ensure that time-sensitive information is provided quickly and that coordination of various tasks is made based on the available resources and anticipated information.

Overloading decision-makers with irrelevant information should also be avoided because interpreting information consumes time and other resources. Sometimes, collecting information is less important than interpreting the information with experience and intuition [9]. For example, sending a book to someone takes only a minute, reading the book may take days, and fully understanding the content can take a much longer time. Information overload is partially caused by the volume of available information but, more importantly, it can also be caused by the limited capacity to process and interpret information [10].

In summary, both information deficiency and information overload are two key problems that prevent effective information sharing for collaborative tasks.
1.2 Present Models for Information Sharing and Their Limitations

In general, information sharing methods can be categorized into push or pull models [11, 12]. In a pull model, an information consumer (info-consumer) sends request to an information provider (info-provider) for the needed information. If the info-provider has a perfect rating, i.e., does not send irrelevant information, then the information overload problem can be addressed effectively as long as the info-consumer does not over-request. However, in some situations, methods based on the pull model can lead to an information deficiency problem.

First, in a pull method, an info-consumer cannot obtain needed information because of not knowing either the existence of certain relevant information or who can provide the information. For example, without a search engine, the Web would not be as useful as it is. In addition, an info-consumer has to know how to use a search engine and how to search within a relevant Web site.

Second, an info-consumer may not know the relevance of certain information. As mentioned before, interpreting information requires knowledge. Without sufficient knowledge, the info-consumer may not even begin requesting the relevant information. The 9/11 Commission report [13] shows that the lack of such critical information could lead to detrimental consequences:

“In the 9/11 story, for example, we sometimes see examples of information that could be accessed-like the undistributed NSA information that would have helped identify Nawaf al Hazmi in January 2000. But someone had to ask for it. In that case, no one did.” (p. 417)

Third, the info-consumer may not know whether a piece of information is up to date. In dynamic situations, such as an emergency response, a great deal of information is
changing constantly, e.g., the status of a fire, the availability of fire engines, and the progress of a rescue mission. Continuous updating could be one way to solve this problem; however, it can be impractical because of high communication cost.

Fourth, seeking information requires time and other resources; thus, the info-consumer may not be able to get information soon enough to make reasonable decisions. This can cause severe problems in time critical situations. The needed information can be prepared before carrying out a time critical task. For example, firefighters can get a map of the emergency site before they get there. If the needed information is changing frequently, however, the prepared information can become obsolete when used for making decisions. Therefore, one can only prepare with relatively stable information.

Methods based on the push model have been proposed to address some of the problems described above. In a push model, the info-provider sends information to the info-consumer without being asked. A common method is to tailor the contents according to a consumer profile, which specifies what information the consumer needs and under what constraints the consumer needs such information. A push method often has the opposite problems to those in a pull method, namely, irrelevant information is often provided because it is difficult to update the consumer profile dynamically and intelligently. For example, suppose a student was taking Class A and purchased two text books from the online merchant, Amazon. Amazon might continue recommending to the student books related to Class A, even if the student had already finished the class and the student’s interests had changed. The above is an example for personalizing product interests. We can easily extend the concept to personalizing the information needs for decision-making. In summary, the problem of updating the consumer profile efficiently
in a push method often causes the info-provider to send irrelevant information that either
takes away valuable resources for interpreting information or obscuring the mining of
other useful information.

Methods based on a hybrid of the push and pull models have also been used.
Subscription is an example where a consumer is responsible to keep the consumer profile
up to date. Here, the consumer “pulls” the information indirectly by frequently updating
the profile, and the provider “pushes” the information by tailoring according to the
updated profile.

Pull and push models are simple concepts, and can only act as guides in designing
an information system. In complex situations, the direct link between the info-provider
and the info-consumer cannot be established because each may not be aware of the
existence of the other. In this case, an information broker, e.g., a search engine, can help
establish the missing link. Present brokers or search engines can provide the link through
search terms, but they cannot handle dynamic or real-time information sharing that is
needed for a complex decision-making process.

1.3 Research Questions

The above discussion suggests that simple profile-based models are inadequate
for analyzing and solving the information overload and information deficiency problems
in tasks requiring complex decision-making. The key problem is that the consumer
profile is inadequate in describing and modeling information needs in complex processes.
We propose to build a better model to address information sharing problems in a complex
environment. For example, the model must provide means (1) to forecast or anticipate information needs of the info-consumer and (2) to update time-critical information efficiently, which is accomplished by close integration with the decision-making process so that flow of information between the consumers (decision-makers) and the providers is regulated according to the dynamic status of decisions. Additionally, in an information rich environment, our model must be efficient so that a large volume of information can be handled. This is especially true in supporting tasks of complex decision-making. The center piece of our model involves an effective information planning function that is responsible for coordinating information management activities. This function must be able to coordinate multiple info-providers and optimize the overall performance of information seeking activities.

The research described in this thesis is focused on two research questions: 1) how to *accurately* anticipate the information needs of decision-making teams in a dynamic environment? and 2) how to *effectively* coordinate and improve the efficiency of the information sharing activities?

1.4 Research Scope

This section briefly introduces my research scope and activities. Figure 1-2 illustrates a roadmap that outlines my research efforts in addressing the information overload and deficiency problems. Specifically, we would like to build a better model to handle any dynamic task involving distributed information sources and decision-makers. This goal may be broken down into 5 steps: (1) analyzing the task, (2) modeling the
decision-makers who perform the task, (3) anticipating information needs according to
the model, (4) planning the information seeking activities, and (5) seeking the needed
information.

Figure 1-2: A research roadmap.

First, cognitive analysis can be used to understand the task environment, the
performer’s capabilities, the organizational structure, and the knowledge on how a
decision is made. In particular, information that is required must be captured along with
the decision-making process.

Second, cognitive task analysis will lead to a cognitive model, in which task
performers are abstracted by encoding and interpreting their capabilities and knowledge
regarding how to make decisions. The models must be able to simulate collaboration and

---

2 This thesis uses “cognitive model” to refer to the abstraction of cognition into models and “agent” to refer
the realization of cognitive models through a computational architecture called “agent architecture”.
Furthermore, “cognitive architecture” refers a subset of the agent architecture that simulates cognitive
processes. Agent architecture, however, may contain functions other than those for modeling cognitive
functions: e.g., those for implementing the information management system.
expert decision-making as well as predict the information required for relevant decision-making point.

Third, one should be able to predict information needs from the cognitive model. Compared with a profile-based push or subscribe method, methods based on the cognitive model are more accurate, more efficient, and require no explicit communication. A profile is static and deterministic: it reports the same information needs regardless of the context. In contrast, a cognitive model takes task context into account and can reflect a consumer’s true mental state. A profile or subscription is suitable for regular information requirements, but not suitable for tasks that are distributed and dynamically adjusted according to changing situations. Using subscription for these tasks can result in high communication cost because decision-makers have to constantly request or cancel subscriptions. In contrast, a cognitive model updates the true needs for information constantly and without decision-makers’ intervention. Hence, it is more efficient and can handle a large volume of data.

Fourth, the anticipated information needs must be prioritized. This helps plan or coordinate various activities in a complex decision-making process. Also planned are various routes for seeking the needed information. Information may be needed at various times or decision points, and different pieces of information require different resources and amounts of time. Therefore, one must plan the information seeking activities to maximize the number of satisfied information needs but minimize the cost associated with information seeking and processing.

Finally, information seeking plans are carried out, and obtained information is shared with the info-consumers. The information needs that are accurately anticipated in
step 3 allows the info-provider to provide the information that is relevant to the decision-making tasks.

This thesis concerns the step 2, step 3, and step 4 as described in the roadmap. I do not include step 5, on how information is obtained by a seeking action, which can be observation or retrieval from an information system. This topic is too broad to be included in this work: for example, pattern recognition, machine learning, search engine, and data mining are all areas in information retrieval. This thesis also excludes cognitive task analysis because it is not relevant to the problem of sharing information for decision-making.

1.5 Information Sharing: An Information Usage Perspective

A broad range of technologies have been developed to address information sharing problems. According to their purposes, technologies can be categorized into three groups: access, retrieval, and usage (Figure 1-3).
The first category includes technologies that enable users to access information sources. Information sharing, in this category, often involves low-level operations, including database management that can store information, network connection that can transport information from a source to a destination, and interfaces that can present information to human users. Development of these technologies has direct impacts on storage volume, network speed, and availability of information systems. For example, before 1997 retail giant Wal-Mart used a cluster with 768 processors and 16 terabytes of online storage [14].

The second category includes technologies that allow users to retrieve information according to their selection criteria. Information sharing, in this category, is affected by information explosion (better accessibility to more information) which results from the development of the technologies in the first category. Technologies in this category depend upon the basic storage and networking functions of the first category. In general, information retrieval systems must find the best way to index and select information so
that any needed information can be retrieved. In addition, data mining technologies can “discover” useful information such as association relations from a large data set.

The third category includes technologies that support users to make decisions with the relevant information. Information sharing, in this category, consists of seeking useful information, interpreting information, and making the relevant information available for the info-consumers. These are the technologies studied in this research.

The technologies that enable accessibility often address system and data level issues. They often care little about how information is used. Access systems can hold a large volume of data that are difficult to be used by human directly. The information retrieval technologies bridge the gap between humans and low level information to achieve better information usability. However, retrieval systems are passive, respond only to users’ requests, and usually have little knowledge on how a piece of information is going to be used. By contrast, systems that support information usage are often designed to help humans make better decisions. These systems have explicit knowledge about how the information is used. Information, in this case, must be understandable to humans.

The technologies in the three categories are not clearly separated. Instead, they influence and impact each other. On one hand, the development and evolution of information usage motivate and influence the development of technologies that it depends upon. For example, to make better decisions requires better information retrieval systems, which in turn requires faster, reliable, and more capable lower level systems. On the other hand, the advance in information access can impact new information usage. The
Web has provided a huge infrastructure and has profound impact on information retrieval technologies and on people’s social and economical life.

1.6 Major Research Results

This research has made two major contributions to addressing the challenges of information sharing among decision-making teams. First, through this research, I developed an agent architecture called R-CAST for modeling high-level decision-making processes. R-CAST models can accurately anticipate information needed in dynamic decision-making processes. This can avoid “pushing” irrelevant information to a decision-maker, which often leads to information overload. The R-CAST architecture can also model complex behaviors in decision-making and team collaboration. For example, it has been used to model team decision-making in a combat simulation [15, 16] and collaboration problems in intelligence analysis [17-19]. This research also created a computational model for the recognition primed decision (RPD), a naturalistic decision-making model for experts [20]. With the help of these models, one can anticipate information needs and design better decision support systems.

Second, I developed a framework called information supply chain (ISC). This framework was inspired by supply chain management, and its major strengths include identifying information needs with a task model and satisfying the needs with comprehensive ISC solutions. Using ISC can reduce the cost associated with information seeking and distributing activities by consolidating information requirements with a novel
planning algorithm. The framework utilizes a market-based strategy to implement information supply chains.

Simulation experiments suggest that information supply chains can achieve high efficiency in information management and avoid information overload commonly encountered in models that have limited cognitive capacities.

1.7 Thesis Outline

Following this introductory chapter (Chapter 1) is a literature review in agent technologies (Chapter 2). Then, the task oriented information supply chain framework is formalized in Chapter 3. Chapter 4 gives detailed design rationales of R-CAST, which realizes the task oriented information supply chain framework as an agent architecture. Chapter 5 describes two experiments and reports the corresponding results. Finally, Chapter 6 concludes the thesis with discussions on the major results, limitations, and further research.
Chapter 2

Background

2.1 Introduction

This chapter gives a survey on research and technologies for information sharing in decision-making teams. Seeking and sharing information are two very important topics in information sciences. Wilson [21] defined basic concepts such as information seek, search, and usage, gave an overview of the field, and reviewed human information behavior models. This research, however, will take a high-level cognitive modeling perspective and tackle the problem with decision modeling and artificial intelligence technologies. As described in Chapter 1, this research focus on (1) how to implement high-level cognitive decision process models for identifying information needs and (2) how to design information systems for coordinating information request and delivery so that those needs are met. Therefore, this survey concentrates on cognitive models of decision-making, process models, and collaboration technologies for effective information management.

This chapter gives a wide range of overview of related research. Table 2-1 summarizes the key technologies that are related to the goal of this research: creating high-level cognitive models and proving management for efficient information seeking activities. The table explains why those technologies are relevant to this research and
shows their limitations. The remainder of this chapter reviews in detail each of these technologies.

Table 2-1: Background Overview.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Section</th>
<th>Relevance to this research</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team cognition</td>
<td>2.2</td>
<td>Provide foundations for effective information sharing</td>
<td>Not concrete enough in anticipating information needs</td>
</tr>
<tr>
<td>Decision-making model</td>
<td>2.2</td>
<td>Rational and naturalistic decision-making models</td>
<td>Not designed for anticipating information needs</td>
</tr>
<tr>
<td>Workflow process model</td>
<td>2.3</td>
<td>Collaboration models for regular workflow processes</td>
<td>Not suitable for dynamic and complex decision-making</td>
</tr>
<tr>
<td>Agent technology</td>
<td>2.4</td>
<td>Theoretical foundations for agent collaborations and information sharing</td>
<td>Not focused on accurate anticipation of information needs</td>
</tr>
<tr>
<td>Cognitive agent architecture</td>
<td>2.4</td>
<td>Create computational decision-making models</td>
<td>Not designed for anticipating information needs and information management</td>
</tr>
<tr>
<td>Information agent</td>
<td>2.4</td>
<td>Basic information sharing methods: broker and matchmaker</td>
<td>Inadequate for efficient information sharing when capacities are limited</td>
</tr>
<tr>
<td>Market-based agents</td>
<td>2.5</td>
<td>Efficient collaborations for task assignments</td>
<td>Not used for coordinating information seeking behaviors</td>
</tr>
</tbody>
</table>

2.2 Cognitive Models of Decision-making

This section reviews cognitive decision-making models. Since this research is aimed at studying information usage for high-level decision-making, it will only include cognitive processes for collaboration and decision-making but not other processes such as memory, attention, perception, and learning or refined psychological constraints.

Specifically, this section gives an overview of team cognition and naturalistic decision-making models.
2.2.1 Team Cognitions

Team cognition is constructed through distributed and emerging activities using various sources [22, 23]. It emerges from the interplay of the individual cognition of each team member and team process behaviors [24]. Both team cognition and team knowledge determines the team performance. Communication can rapidly consolidate information distributed between various team entities to make effective team decisions [25]. Effective communication decisions, however, relies on an overlapped team cognition and team knowledge [26, 27], called shared mental model (SMM) [26, 28]. A SMM represents each team member's understanding of the global team state. It can be measured in terms of the degree of overlap or consistency among team members' knowledge and beliefs [26].

Research in team cognition suggests that teams with high degree of SMM can result in a high performance [26]. SMM produces a mutual awareness, with which team members can reason about other’s status and belief. This mutual awareness is the key for guiding communication [16, 29] and interactions [30] within the team. Shared mental models include 1) static knowledge about the team organization, roles, capabilities, goal, plan, and policies and 2) dynamic information about workload, situation, current task assignments, status of tasks, and progress toward its goal [29, 31].

The research findings on team cognition and SMM indicate that accurate anticipation of information needs for a human decision-maker must be based on certain degree of sharing and understanding of decision-makers’ mental models.
2.2.2 Decision-making Models

Rationality has long been accepted as a fundamental assumption for most well developed decision-making theories in economics [32], social sciences [33], and computer sciences [34]. It assumes that people calculate the likely costs and benefits of any action before deciding what to do. Rational decisions are normally done in four steps [35]: 1) identify problem; 2) define criteria; 3) generate and evaluate alternatives; and 4) implement the decision.

Herbert Simon pointed out that most people are only partly rational [10]. He proposed the concept of bounded rationality, which argues that agents experience limits in formulating and solving complex problems and in processing, receiving, storing, retrieving, and transmitting information [10]. Bounded rationality models can overcome some of the limitations of the rational-agent models, e.g. in economics [36]. Compared with rational decision-making, decision-makers often choose satisfactory, but not optimal, solutions. Supporting human decision-making process should be based on naturalistic decision-making models, which describe how humans make decisions not on rational decision-making models, which define how humans should make decisions.

2.2.3 Recognition Primed Decision-making Model

Recognition primed decision (RPD) is a type of naturalistic decision-making model, which is motivated by explaining expert decision-making [20]. Unlike rational decision model theories, RPD focuses on the decision process and situation assessment rather than evaluating options. Figure 2-1 shows a typical RPD decision process.
An RPD decision starts from recognizing the current situation by comparing it with the decision-maker’s past experiences. If there is not enough information, the decision-maker will try to seek additional information. If the current situation is familiar, the decision process produces four byproducts: plausible goals, relevant cues, expectancies, and courses of actions (COAs). Then, the decision-maker will evaluate each COA with a mental simulation. The decision-maker will pick the first one that works and implement the COA. After recognition, the decision-maker will monitor any expectancies, and if they are violated, the decision-maker will seek additional information to clarify the recognition.

Figure 2-1: RPD model (Klein 1989).
Compared with rational decision-making models, decision choices of a RPD model are not always optimal [20]. This is because a decision-maker picks the first COA that works, which is not necessarily the best one. However, RPD decision choices are effective [20]. RPD model uses user experiences to recognize situations. When a decision-maker is familiar with a situation, the decision-maker can recall a specific instance when something similar was faced. Sometimes, however, the decision-maker has to develop stories (not specific instances) to explain the current situation and apply to a problem.

Case-based reasoning [37, 38] (CBR) is another decision-making model that solves a problem by recalling past experiences based on the current situation. CBR has storage, index, and retrieval of cases as central activities, whereas RPD focuses more on developing better situation awareness through information gathering and expectancy monitoring. In short, CBR is a decision function, and RPD is a decision process.

There have been several attempts to implement the RPD model [19]. For example, long-term memory structure [39] and neural networks [40] were used to represent experiences. There are also attempts in integrating RPD with agent technologies. Norling, et al. [41, 42] explored ways of using RPD to enhance BDI agents so that simulation of human societies would be more realistic. These attempts are limited because the phase of finding additional information and evaluation of COAs are ignored. That means the existing models are not designed to study information sharing problems.
2.3 Workflow Process Models

In addition to decision-making models, workflow process models can also describes human collaborations processes. Many research efforts have investigated methods for modeling workflow processes. Dumas and Hofstede tried to specify workflows with activity diagrams of the Unified Modeling Language (UML) [43]. They demonstrated that activity diagrams can provide the expressive power that is required by most applications, and showed that an activity diagram is more powerful to express processes than most of the languages found in commercial workflow systems. A recent study by Aalst and Kumar [44] demonstrated that the Extensible Markup Language (XML) can be used to model inter-organizational workflows. The main contribution of that research is to support process exchange through the Internet. Aalst [45] mapped the workflow concepts into Petri nets, providing a more formal way to represent and verify processes. Dussart et al. [46] compared several workflow modeling methods such as Petri-nets, WfMC, UML, ANSI, and EPC on criteria such as formal basis, executability, and ease of visualization. Their study showed that Petri-nets satisfies most criteria and thus are desirable. However, in general, workflow processes are mainly used to manage well-defined processes and routine decisions such as for business control and transactions. Therefore, they are not suitable for guiding complex decision-making processes in a dynamic situation.
2.4 Agent Technologies

Previous sections introduced the models for team cognition, decision-making, collaborative processes. Typically, these models can be realized with agent technologies. Furthermore, agents are also used to management effective information sharing. This section reviews current agent technologies. Agent technologies have been developed for modeling human behavior [27, 47-52], developing intelligent systems [53-56], and reducing human workload [57-60]. These technologies have been reviewed and analyzed at different levels and in different domains [53, 56, 61-65].

![Agent Technologies Diagram]

Figure 2-2: Agent technologies in seven areas.

Figure 2-2 shows a 7-area analysis of agent technologies, and lower areas provide guidance and foundations for top areas. The first (bottom) area includes basic concepts, taxonomies, and properties on agent and multi-agent systems. This area addresses and debates on philosophical questions [53, 56] such as “what is an agent.” The second area, containing agent theories, deals with fundamental topics on how to formally represent agent intentions and behaviors [66-75]. This provides guidance and foundation for realizing concrete mechanisms and functions. The third area, or the area of knowledge
representation [76-79], is about how to encode and use knowledge for interpreting information, solving problems, or making decisions. Agent communication, the fourth area, addresses how to handle conversations among agents and how to implement information exchange in general [80, 81]. The fifth area concerns agent architecture [27, 47, 49, 51, 52, 82, 83], which involves integration and realization of the general principles obtained from above four areas: theory, knowledge representation, and communication. For example, agent architectures define knowledge representation syntaxes, knowledge interpretation mechanisms, agent communication languages (ACL), and learning mechanisms. The sixth area includes studies of methodologies that can save time and ensure quality during agent engineering [84-86]. Finally, the last area contains specific applications where agents can facilitate information sharing [87, 88].

The remaining section will review the main technologies of each area in detail, and the connections of these technologies to the research described in this thesis.

2.4.1 What is Agent?

Agent technologies represent a new research paradigm that result from contributions from many fields: e.g., artificial intelligence, object-oriented programming, cognitive psychology, sociology, and human-computer interaction [56]. Given the complex origin of this new paradigm, it is understandable that researchers have not agreed on the definition of an agent. Wooldridge and Jennings defined an agent as a computer system in some environment, capable of autonomous actions in order to meet its design objectives [53, 56]. This definition highlights three properties: 1) agents must be
intelligent to solve problems, 2) be autonomous to run by themselves without supervision from humans, and 3) be able to represent to serve humans’ needs [89]. Nwana and Wooldridge proposed a taxonomy of agents-based on several attributes including mobility and deliberative/reactive. This also led to names such as collaborative agents, interface agents, mobile agents, information/internet agents, reactive agents, hybrid agents, and smart agents [62, 90].

Multi-Agent Systems (MAS) are another agent classification that comprises loosely coupled network of agents. This class was developed to solve complex problems in a distributed system that are beyond the capabilities or knowledge of individual agents [56]. In a MAS, each agent has incomplete information or capabilities for solving a problem, and there is no global or central control system. In order to accomplish a task, agents must interact with each other through cooperation, coordination, and negotiation [89].

Team-based agents are a special kind of MAS that are implemented with explicit collaboration models. They have been used to simulate human users, support complex decision-making, or provide simulated peers for teamwork training [27, 91-98]. Research showed that effective teams must share common goals and approaches [26, 27] and that a team of agents can be embedded in human organizations [29, 99], either by delegating users’ tasks or by cooperating with them.

A philosophical question intensely debated is which software system can be called an agent. According to the so-called “weak notion”, agents must, at the minimum, be autonomous, social, reactive, and proactive [53, 56, 62]. In comparison, the strong notion requires demonstration of human-like attributes such as cognitive entities, feeling,
perception, and emotion [100, 101]. This thesis adopts the strong agent notion, and our agent may be characterized as intelligent, autonomous, capable of anticipating human needs, and “strong” in high-level collaboration among agents. From the taxonomy perspective, agents in this thesis may be classified as collaborative information agents.

2.4.2 Agent Theories

BDI (belief-desire-intention) is a practical model of human reasoning developed by Michael Bratman [66-68]. Belief is the information that an agent has about its environment. Information may be incomplete or incorrect. Desires are states that the agent would bring about, and they should be consistent with one another. Intentions are the desires that the agent has committed to achieve. In general, the agent can’t achieve all its desires. Table 2-2 shows a standard BDI interpreter proposed by Rao et al [68].

Table 2-2: BDI Interpreter (Rao et al 1995).

<table>
<thead>
<tr>
<th>BDI-interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize-state ()</td>
</tr>
<tr>
<td>Repeat</td>
</tr>
<tr>
<td>Options := option-generator (event-queue);</td>
</tr>
<tr>
<td>Selected-options := deliberate (options);</td>
</tr>
<tr>
<td>Update-intentions (selected-options);</td>
</tr>
<tr>
<td>Execute();</td>
</tr>
<tr>
<td>Get-new-external-event();</td>
</tr>
<tr>
<td>Drop-successful-attitudes();</td>
</tr>
<tr>
<td>Drop-impossible-attitudes();</td>
</tr>
<tr>
<td>End repeat</td>
</tr>
</tbody>
</table>

To emulate teamwork collaboration, BDI has been expanded to create Joint Intention Theory (JIT) [66, 69-72], which focuses on team formation and dispersion, and Shared Plans Theory (SPT), which focuses on task fulfillment.
JIT provides a theoretical framework for joint commitments and communication. An intention is viewed as a commitment to perform an action while in a mental state. A joint intention is a shared commitment to perform an action while in a group mental state [71]. Communication is required to establish and maintain mutual beliefs and join intentions. A team of agents jointly intend to do an action if and only if the members have a joint persistent goal. An agent has a persistent goal relative to some context to achieve a goal state if and only if 1) the agent believes that the goal state is currently false; 2) the agent wants the goal state to be true eventually; and 3) “2)” will continue to be true until the agent comes to believe either that the goal state is true, or that it will never be true, or that the context is false [72].

SPT provides mental-state specifications of both shared plans and individual plans. Shared plans are constructed by groups of collaborating agents and include subsidiary shared plans formed by subgroups as well as subsidiary individual plans formed by individual participants in a group activity [73-75]. SPT emphasizes the need for a common high level team model (shared plan) that allows agents to understand all requirements for plans that might achieve a team goal, even if the individuals may not know the specific details of the plans (individual plan).

JIT and SPT provide foundations for other research in agent collaborations. STEAM [102] and CAST [27] are frameworks that are designed according to these two theories. STEAM was implemented in Soar architecture [49]. It specifies an agent organizational hierarchy and team tasks, but not coordination plans [103]. At runtime, a team model can monitor team and individual performances and imply needs for communication and coordination [102]. The model contains both team operators that
execute together and individual operators that work independently. In addition, STEAM assumes communication has cost and uses a decision tree for communication decisions: if a potential recipient probably already knew a piece of information, and the cost of collaboration is high, then agents don’t communicate. STEAM has been applied in several simulation domains such as a team of pilot agents in an interactive environment and a team of virtual players for RoboCup soccer [104].

JIT and SPT, two theories on agent teamwork collaboration, provide a foundation for creating the task-oriented information supply chain described in this thesis. This part of thesis work emphasizes the need of communication when implementing collaborative agents, and is a direct extension of the proactive communication theories that were developed together with the CAST architecture.

### 2.4.3 Knowledge Representation

A knowledge representation requires design of both syntax and semantics, which can greatly affect the behaviors of an agent [76]. Knowledge representation based on logic offers just enough expressiveness for representing information and much more avenue for reasoning. Logic is more precise than natural language, which requires a great deal of background knowledge to understand. Logic includes enough operators for reasoning such as “not”, “and”, “or”, “all”, or “some”. Compared with object-oriented language, logic leads to better knowledge reuse because of better preservation of implicit knowledge [76, 105]. Logic has many branches and extensions such as propositional logic, fuzzy logic, first order logic, higher order logic, temporal logic and modal logic.
Many agent theories choose modal logic [72, 106] for representing the agent’s rationalities; however, most implemented systems, often called “production systems”, are still based on first order predicate logic due to computational cost considerations [49, 51]. A production system consists of a collection of rules, facts, and a forward chaining algorithm that “produces” new facts from old ones. A rule becomes eligible to fire when its conditions match some facts in current memory. A conflict resolution strategy determines which of several eligible rules (the conflict set) fires next [49, 51].

Spurred by the success of HTML and XML in internet for sharing and parsing information, traditional knowledge-representation languages start to adopt them for low-level syntax [76]. As an early attempt, SHOE (Simple HTML Ontology Extensions) is an HTML-based knowledge representation language. SHOE is a superset of HTML: adding tags to embed arbitrary semantic data into web pages. SHOE was designed partly because of the need for handling large volume of data in the Web. SHOE can be used to embed data from a variety of sources [77].

After the creation of XML, the Semantic Web [78, 79, 107] representation was invented to allow easy information sharing and processing over the internet, given the fact that the Web is highly contextual and manifests itself in the form of HTML files. The Semantic Web uses URIs to represent data with RDF (Resource Description Framework) syntax. Strictly speaking, the Semantic Web is not designed with agents’ knowledge representation needs in mind but for general knowledge sharing purposes. For this reason, agent knowledge representation with Semantic Web language must include reasoning capabilities. The Web Ontology Language (OWL) [108] adds more vocabulary for describing properties, classes, and cardinalities such as disjointness, exactly one, equality,
symmetry and enumerated classes. Figure 2-3 shows the W3C standards related to the Semantic Web.

Figure 2-3: W3C semantic Web stack (W3C 2006).

The agent architecture that is developed in this thesis uses a diverse set of symbolic knowledge representations. For example, it uses logical rules for conditional reasoning, a high-level process language for planning, and a tree structure for agent experiences. Currently, it does not convert the knowledge specifications to those that conform the XML standard.

2.4.4 Agent Communication

Multi-agent architectures require an agent communication language (ACL), which defines the form (syntax) and the meaning (semantics) of messages (also called speech acts) routed from one agent to another [109]. The semantics in an ACL is more important than the syntax because semantics carries the specific purposes of a message and results
in consequences [110, 111]. This section briefly introduces two ACLs: KQML and FIPA-ACL.

KQML (Knowledge Query and Manipulation Language) is a language and a protocol for exchanging information and knowledge [80]. It is the result of DARPA’s knowledge sharing efforts aimed at developing techniques and methodologies for building large-scale sharable and reusable knowledge bases. KQML focuses on an extensible set of *performatives*, which defines the communicative operations according to agent's knowledge and goals. Each performative represents a speech act as well as its associated semantics, protocol, and list of attributes. Table 2-3 gives an example, in which a stock server agent tells Joe about the IBM stock price [80].

Table 2-3: KQML Performative Example (Finin et al. 1994).


FIPA (Foundation for Intelligent Physical Agents) is an organization making standards for agent-based systems with strong preference on design process life cycle [81]. FIPA-ACL [81] is the agent communication language associated with FIPA’s open agent architecture. The syntax of FIPA-ACL is almost identical to that of KQML. However, FIPA-ACL prohibits direct manipulation of other agents’ knowledge base hence it does not include performatives such as *insert* and *delete*. FIPA also established
several working groups for projects on Web services, human-agent communications, and mobile agents, etc.

In this thesis, agent communication language is not the central topic. Instead, emphasis is placed on communication strategies such as what to communicate and when to communicate. FIPA-ACL guidelines are followed in implementing the basic communication performatives.

2.4.5 Cognitive Architectures

A cognitive architecture is a computational design that allows a researcher to simulate cognitive processes such as perception, attention, and decision-making. According to the Unified Theory of Cognition (UTC), an agent is composed of architecture and knowledge: the term architecture refers to the mechanisms and structures that allow functions for processing knowledge and for generating behaviors [50].

Cognitive architectures can be symbolic, connectionist, or hybrid (combination of the previous two). Symbolic architectures encode knowledge using rules and procedures which are understandable to humans. In contrast, connectionist architectures are based on interconnected networks of simple units such as neural networks. Within this architecture, knowledge is a result of learning and is encoded in an implicit form, hidden from humans. Connectionist architectures are often used to model refined cognitive behaviors that are impractical to model with rules and procedures, and good results have been seen in modeling high-level mental behaviors and complex tasks. In this research, only symbolic architectures are used for modeling decision-making tasks, and the
following section introduces three symbolic architectures: Soar [49, 50], ACT-R [51], and CAST [27, 112].

Soar, an architecture for problem solving, was developed by John Laird, Allen Newell and Paul Rosenbloom [49, 50]. Soar uses problem spaces for managing all tasks and sub-tasks, productions for representing permanent knowledge, automatic sub-goaling for generating goals, and chunking for learning. Soar uses operators to perform actions, and in each decision cycle, Soar proposes, selects, and then applies the operators. If no operator or multiple operators are encountered, Soar will create an impasse (lack of knowledge) and cast the impasse as a sub-goal to resolve the problem.

ACT-R [51] architecture consists of multiple cognitive modules, which are integrated to produce coherent cognition. The cognitive modules include a perceptual-motor module, a goal module, a declarative memory module, and other specialized systems. ACT-R has a production system that responds to patterns of information. At any point in time a single production rule is selected to respond to the current pattern. ACT-R is based on the cognitive architecture in a production system that represents knowledge as rules (i.e., productions). In ACT-R, a production must be selected at each step of performing a task. When more than one production matches the current goal, the systems selects one of them via a process called conflict resolution. In general, conflict resolution selects a production by weighing the cost and benefit for each matching production and then selecting the best candidate. When, however, there is a lot of noise in the process, less optimal productions may be selected.

CAST (Collaborative Agents for Simulating Teamwork) is an architecture for modeling teams [27, 112]. Its teamwork model is based on Joint intention and SharedPlan
theories and inspired by a shared mental model (SMM). The SMM allows different levels of common goals, which are designed for sharing belief, responsibility, and plan [19, 26, 28-30, 74, 99, 113]. CAST also implemented communication mechanisms to ensure efficient information sharing among team members via anticipation of teammates’ information needs. CAST formally defines a new performative, called ProInform [114], which allows a speaker to establish a joint belief that the speaker believes in information $p$ and that the speaker believes the receiver needs $p$. Based on the anticipated information needs of its teammates, a CAST agent continuously monitors whether the information it senses matches any needed information. If a match is detected, the agent uses a decision-theoretic communication strategy to decide whether to proactively inform those teammates who need the information.

The CAST architecture is designed to model well-structured agent teams capable of adapting to dynamic environments. There are five major integrated components in the CAST architecture to support this objective: (1) teamwork knowledge specification, (2) coordinated plan execution, (3) world model, (4) communication, and (5) domain adapter. The teamwork knowledge specification allows agent designers to use MALLET (Multi-Agent Logic Language for Encoding Teamwork) to design team structures and team processes. Meanwhile, domain knowledge is captured in the agent’s world model. The coordinated plan execution component makes action decisions and communication decisions. The CAST agents operate in a distributed fashion, and communication is implemented via JAVA RMI (JAVA Remote Method Invocation). Each agent interacts with a domain through a domain adapter, which integrates the four other components and
makes CAST agents adaptive to different task environments. An overview of the CAST agent architecture is shown in Figure 2-4.

Figure 2-4: CAST agent architecture.

There are other agent architectures such as EPIC [38], COGENT/iGen [7], and subsumption architectures [1] for cognitive modeling. However, Soar and ACT-R are the two most popular ones. Both have been used in a wide range of applications. Compared with Soar and ACT-R, CAST is less powerful in modeling sophisticated cognitive processes. CAST has a better mechanism to anticipate information needs and support team communication, but only limited capability to manage information needs. The limitation is caused by the lack of coordination and planning functions for managing information seeking and delivering activities. Thus, CAST models can be inefficient for
supporting a team that requires a large volume of information. This research includes a
design and implementation of a new architecture called R-CAST, which models high-
level cognitive processes, and more importantly, provides an effective mechanism for
managing information requirements specific to collaborative tasks that involve complex
decision-making.

2.4.6 Agent Oriented Methodologies

Agent oriented software engineering has been growing along with agent
architectures and theories. Wooldridge [84] presented the Gaia methodology, which
supports both micro-level (agent) and macro-level (agent society and organization) agent
development. Using Gaia, software designers can systematically develop an
“implementation-ready” design based on system requirements. In the Gaia design
process, the first step is to map roles into agent types, and then to create the right number
of agent instances of each type. Roles consist of four attributes: responsibilities,
permissions, activities and protocols. The second step is to determine the services needed
to fulfill a role in one or several agents. The final step is to create the acquaintance
model that represents communication among agents [84]. Wood proposed a more refined
methodology called MaSE (Multi-agent Systems Engineering) [85], which consists of
seven steps (Table 2-4).
Table 2-4: MaSE Methodology (Wood 2000).

1. Capture goals and transform the initial system specification into structured system goals.
2. Create use cases and sequence diagrams based on the initial system specification.
3. Refine roles that are responsible for the goals into state diagrams.
4. Create agent classes and map roles to agent classes in an agent class diagram.
5. Construct conversations by defining a coordination protocol in the form of state diagrams.
6. Assemble agent classes with internal functionalities of agent.
7. Design system and create actual agent instances based on the agent classes.

In addition to design methodologies, AUML [86], an agent design diagram schema is also proposed to formally specify an agent design. AUML is an extension of UML (Unified Modeling Language) [115], a graphical notation for object-oriented analysis and design developed by the Object Management Group. AUML can specify agent interaction protocols, internal behaviors, interfaces, and deployment diagrams. AUML abstracts the system design by defining and extending the basic building blocks such as agent, belief, commitments, and agent communication acts such as inform and request.

In this research, R-CAST architecture is developed to be highly configurable; thus, at agent design stage, decisions on how to configure the architecture must be made before encoding the agent functions and knowledge. A designer should determine which components to include and how each should be configured and tailored to specific domain requirements. In addition, a domain adapter must be implemented to connect the agents to their environments because the architecture itself is domain independent. Chapter 4 gives a detailed guide on how to apply the R-CAST agent architecture.
2.4.7 Information Agents

Agents can be applied to various domains: e-commerce [58, 64, 116-126], business process management [59, 60, 127-129], manufacturing planning [130], supply chain management [127, 131, 132], and simulations [53, 96, 98]. Maes showed that agents can reduce workload and assist users in a variety of ways: they can hide the complexity of difficult tasks [57], perform tasks on a user’s behalf [54, 127, 133, 134], train or teach the user [96, 98], help different users collaborate [133], monitor events and procedures [57], and overcome information overload [57, 135].

Due to a rapid advance of the Internet technologies, modern information environments are becoming increasingly large, open, unorganized, distributed, and dynamic [136]. Information agents [15, 30, 137-139] have been proposed to address the challenges brought about by the above attributes. Being autonomous, intelligent, and equipped with knowledge to process information, information agents have been applied to identify and find information sources [87, 140]. In addition, mobile agents have been proposed to achieve efficiency and flexibility for information discovery under certain circumstances [141]. However, to apply mobile agents to solve real world problems, one must provide mobile agents with appropriate run-time environments. Furthermore, security and privacy must be considered [142]. The following section introduces two types of information agents: brokers and matchmakers.
2.4.7.1 Broker

A broker agent links info-consumers to info-providers and provide information access transparently to info-consumers (who are either not aware of the presence of the broker, or unimpeded by it during information seeking). Martin et al [87] implemented an information broker system with the Open Agent Architecture (OAA) framework [82]. The broker agent realized its service by 1) mapping the query schemas to source schemas and 2) gathering and integrating the results from information sources [87]. Figure 2-5 illustrates the basic information broker architecture. BQ, BR, SQ, SR refers to broker query, broker response, source query, and source response, respectively.

Figure 2-5: An information broker architecture (Martin et al 1997).

Although the broker architecture can connect info-consumers to info-providers, it cannot address the issue of information sharing in a broader sense. If multiple brokers can be integrated, for example, how do info-consumers identify which brokers to contact? What
if there are duplicated queries? This thesis addresses these issues that are beyond the abilities of an agent based on the broker architecture.

2.4.7.2 Matchmaker

A matchmaker [88] can be viewed as a market, in which info-providers advertise their information seeking capabilities whereas info-consumers post their information requests. The matchmaking agent will then match the requests to the info-provider and share the requested information with the info-consumer. It differs from double auction in that info-providers and info-consumers will not include a bidding price and that a match does not take the pricing information into consideration. Matchmakers, however, are not generally able to prioritize the services of info-providers to info-consumers with the most urgent needs.

Realized in RESINTA (Reusable Environment for Task-Structured Intelligent Networked Agents) architecture [30, 143], a matchmaker communicates with other agents using LARK (Language for Advertisement and Request for Knowledge Sharing). Matchmaking agent can determine the information source by considering the context, interest similarities, and constraints. To be more specific, it has five filters: context matching, profile comparison, similarity matching, signature matching, and constraint matching [88]. Designers can configure a matchmaker by selecting and combining the appropriate filters.

Matchmaking offers a solution to specify and understand the information needs in an open environment. However, the context is often static and too general to make sense
of by a matchmaker. For example, it cannot capture the dynamic context of a task, which often involves complex task decomposition, changing environments, and variable decision points. Therefore the matchmaker can serve as a sophisticated source for information discovery but cannot anticipate the information need of a task performer at a certain point in time. In essence, a matchmaker is an enhanced information broker with more flexible match criteria.

2.5 Market and Agents

Section 2.4 gives a systematic review on agent concepts, theories, representations, communications, architectures, methodologies, and applications. This section gives additional methods on efficient agent collaboration using a market-based approach. Market-based systems have become an increasingly popular approach for distributed and self-interested agents to solve decentralized problems. Since both information and decision-making can be distributed across people, a market-based approach has the potential to optimize and coordinate information seeking activities among multiple agents. This section will review the actions of such an agent in general, with a focus on the contract net protocol [144].

2.5.1 Agent Auctions

One approach to optimizing an agent’s decisions is to implement a multi-agent system as a virtual market, which defines goods and prices. Agents make price decisions
according to “competitive equilibrium” [145]. Although economic theories mostly assume multiple self-interested agents, in computer science applications, agents are not strategic. They are designed not only to maximize an individual utility, but also to maximize overall performance. This approach is called market-oriented programming [145], and it has been used in distributed resource allocation [146], information services [147], trading systems [117, 148, 149], and sensor networks [150].

Some auctions can involve a large set of related commodities. It is important to allow bidders to express their preferences on complementary items or substitutes. For example, bidders in wireless radio spectrum would gain an advantage if they could be awarded with a block of connected band segments [151]. These kind of issues can be addressed by using combinatorial auctions [152-154], in which bidders can submit bids on bundles of items.

2.5.2 Contract Net Protocol

The contract net protocol [144] is a high level protocol for solving identification problems—selection of the most suitable manager-contractor pair: 1) how a manager can find the most appropriate contractor to accomplish a task and 2) how a contractor can find a suitable task when it is idle. When a manager needs a contractor to execute a task, it initiates an auction to find the most suitable contractor [144]: 1) the manager announces a service demand to a set of potential contractors; 2) those contractors that can accomplish the service submit a bid that details its offer; and 3) the manager analyses the submitted bid and awards the contractor who can best accomplish its expectations.
A contract net is a distributed protocol because any agent can be a manager or contractor. Unlike broker or matchmaker, it has no central node that handles all the interactions and makes most interaction decisions. In addition, its auction mechanism can optimize the overall services of providers by responding to the most urgent and valuable information.

In this thesis, an info-consumer holds an auction to identify the most efficient info-providers. The contract net protocol is extended to handle sub-contracts: a potential info-provider may hold additional auctions to determine the cost of needed information before it creates an offer for the info-consumer. Over a “long” time, agents, representing either an info-consumer or an info-provider, build long-term relationships called information partner relationship. Established partnerships help an agent to avoid the cost of holding auctions (e.g., an info-consumer can contact its info-providers within its partnership network directly).

2.6 Conclusions

This chapter reviewed research in cognitive models of decision-making, workflow process models, agent technologies, and market-based collaboration technologies. It focused on agent technologies that model cognitive tasks, solve problems, and enable collaborations. Existing technologies, however, are inadequate in providing architectural models that can a) precisely anticipate what information is needed and b) efficiently seek and satisfy the needs. Subsequent chapters will address these two problems at conceptual, theoretical, and architectural levels.
Chapter 3
Task-Oriented Information Supply Chain Framework

3.1 An Overview of the Information Supply Chain (ISC) Framework

High performance information sharing is critical for complex decision-making tasks\(^3\), especially when the task is distributed and under time pressure. Chapter 1 introduced two related research questions. First, the information needs of decision-making teams must be \textit{accurately} anticipated so that decision-maker can have the relevant information without being overloaded with irrelevant information. Second, coordination of information sharing activities including seeking and fusing of information, must be efficient so that information providers can better supply the needed information when their capabilities and capacities are limited.

Much research in information science, cognitive science, and artificial intelligence are studying human information seeking behavior [21], information in team cognition [22-24, 92, 155], and information sharing methods [27, 30, 102, 111, 114, 156] respectively. Yen, et al. proposed a proactive information sharing method based upon a shared mental model [16, 26, 28, 29, 94, 157]. Fan et al. [158] gave a formal definition of information need, along with proposed theories and semantics on proactive information sharing in teams. This method for anticipating information needs emphasizes guiding agent communication. The relation between information needs and decision-making

\(^3\) In this thesis, a task contains a decision-making process.
tasks has not been fully explored. In addition, the research does not offer functions to support efficient coordination for information seeking activities.

Some other methods such as information broker and information match maker can provide an efficient mechanism for identifying information providers and information needers. However, these methods do not consider the limitations on information agents’ cognitive capacities. As a consequence, the performance of both methods can suffer when the capacities are limited. Therefore, they can not be used to support decision-making tasks that are under time pressure and with limited information sources. Finally, existing methods have no planning functions that can consolidate redundant information requirements that waste time and effort during information seeking tasks.

My research studies how to accurately identify information needs and how to efficiently seek and share information. The task-oriented information supply chain framework organizes a set of concepts, methods, and theories that pertain to the research questions that have emerged from information sharing in a task context. The framework studies information sharing from both intra-agent perspective and inter-agent perspective. The intra-agent perspective looks at the basic concepts of information needs, information requirements, and information requirement planning. The inter-agent perspective studies the relation among information agents and how to establish relationships among a large number of information agents.

In this framework, information needs are defined as a property that is closely related to a decision-making task. Within this context, an agent can tell what information is needed, who needs it, and when it is needed. This definition enables accurate anticipation of information needs.
After information needs are captured, an agent will transform them into information requirements. In this framework an information requirement is an abstract view of an information need. An information requirement hides its decision context that is irrelevant to information seeking actions. In other words, who needs this information and how the information is used will not affect how the information will be found and fulfilled. An information requirement only contains information type and time, which are important for guiding information seeking actions.

Based on the concept of an information requirement, the ISC framework proposes a coordination plan for information seeking activities, including capturing, consolidating, and sourcing for information requirements. This planning process is called information requirement planning. It is the core function for efficient coordination among agents fulfilling the information requirement. Without the planning function, an information agent will waste a lot of resources finding duplicate information.

From an inter-agent perspective, a key concept is information partnership, which defines a mutual understanding between two agents regarding their provider and needer roles in exchanging information. A large collection of information partners form an information supply chain. It is worth noticing that this framework does not adopt a centralized planning mechanism to identify information partners and form information supply chains. This research solves this problem by using an extended contract net protocol, which uses a market-based auction mechanism to identify information providers.

In the end, this chapter gives a discussion about the relationship between information supply chain framework and supply chain management. It shows that despite
some fundamental differences between the two, supply chain management can serve as a useful metaphor in guiding new information management research.

The remaining chapter will describe the framework in detail. Section 3.2 gives preliminaries regarding notation, basic concepts, and assumptions used in this research. Formal notations are used mainly for concise descriptions. Section 3.3 gives definitions and properties about information needs. Next, how to manage information needs so as to plan and satisfy them is introduced in Section 3.4. Section 3.5 extends the methods of information sharing from individual agent perspective to an information supply chain (ISC) perspective. Then, a market-based approach is proposed to form information supply chains by using auctions. Section 3.7 explains how the idea is developed from the material supply chain management. Section 3.8 concludes this chapter.

3.2 Formal Foundations

This section introduces notations for describing the ideas in the framework, defines two fundamental concepts: task and information, and lays out assumptions. Then, it briefly introduces previous research that provides a foundation on how to anticipate information needs.
3.2.1 Notations

The description of the framework includes commonly used logical and mathematical symbols, notations in agent related theoretic papers [69-72, 74, 75, 110, 111, 158, 159], and special symbols used by this framework.

3.2.1.1 Basic Logical and Mathematical Notations:

\( \triangleq \): defines;
\( \models \) \( \varphi \): \( \varphi \) is valid;
\( \rightarrow \): imply;
\( \Box \): always;
\( \Diamond \): possible;
\( \min(\{x,y,z,...\}) \): the minimum of set \( \{x,y,z,...\} \);

3.2.1.2 Notations for the Information Supply Chain Framework

\( A, B, C,..., \): an agent or a group of agents;
\( \Lambda \): every one;
\( \alpha \): a physical action or a mental action (decision);
\( \eta \): an information need;
\( \delta \): an information requirement;
\( N \): a set of information needs;
\( i \): a piece of information;
\( I \): an information type;
\( s \): a source of information;
3.2.1.3 Notations about Tasks and Actions

- $\Gamma$: a task;
- $R_\alpha$: a recipe for $\alpha$;
- $t_0$: the current time;
- $t_a$: the start time of $\alpha$;
- $t_a'$: the estimated end time of $\alpha$;
- $\Theta_\alpha$: the context for the action $\alpha$;

3.2.1.4 Notations about Agents

- $Bel(A, p, t_0)$: agent believes on $p$;
- $MB(A, B, ..., p, t_0)$: mutual belief;
- $Int.To(A, \alpha, R_\alpha, t_0, t_a, t'_a, \Theta_\alpha)$: agent intention;
- $CBA(A, \alpha, R, t_0, t, t', \Theta)$: capable of;
- $pre$: precondition;
- $post$: rational effects;

The framework also makes the following two extensions from previous research. Agent intention and capability are extended from a time point to a time period. Instead of saying that an agent is intended to do something at sometime in the future, we say an agent is intended to do something during a period of time in the future. Likewise, an agent is capable of doing something during a period of time. The extensions allow better planning and time management when the agent works on multiple tasks.

- $Int.To(A, \alpha, R_\alpha, t_0, t_a, t'_a, \Theta_\alpha)$ is extended to $Int.To(A, \alpha, R_\alpha, t_0, t, t', \Theta_\alpha)$
- $CBA(A, \alpha, R, t_0, t, t', \Theta)$ is extended to $CBA(A, \alpha, R, t_0, t, t', \Theta)$
3.2.2 Research on Proactive Information Exchange in Agent Teamwork

Proactive information exchange is critical to teamwork [27]. Fan, et al. [158] formally defined information and information need, formally identified four types of information needs for agent teamwork, proposed anticipating others’ information needs based on shared team processes, and established a formal foundation for proactive information delivery behaviors.

Information need is defined as $\text{InfoNeed}(A, N, t, C_n)$: an agent $A$ needs to know $N$ by time $t$, under a context $C_n$. Although the definition is reasonably complete in capturing the properties of an information need, it is still not concrete enough to direct how to satisfy the need. Furthermore, the context $C_n$ is not clearly defined. Fan, et al. also group information needs into four categories according to their purposes: for action-performing, for decision-making, for goal protection, and for goal escape. In comparison, the purpose of information needs is generalized in the ISC framework to be simply accomplishing tasks, including tasks that involve decision-making.

Anticipating information needs is based on shared team processes. The agent who anticipates information has a peer-to-peer relationship with the agent who needs the information. Anticipating and sharing information are mutually beneficial behaviors among team members. In addition to these behaviors, ISC also look at supporting behavior between an agent and a human. In this type of supporting relationship, the agent will anticipate the human’s information needs by modeling the cognitive behaviors of the human who makes decisions.
Finally, Fan, et al. take a communication perspective that tries to understand how agents communicate. In contrast, the research in ISC will take an information management perspective that studies how agents can coordinate their information seeking activities so that agents can optimize their resources to better fulfill anticipated information needs.

3.2.3 Two Fundamental Concepts

Since this research is about information sharing within a task context, task and information are two fundamental concepts that should be defined beforehand.

3.2.3.1 Task

Merriam-Webster online dictionary defines task as “a usually assigned piece of work often to be finished within a certain time”. People have various motivations to work, but the ultimate goal of performing a task is to accomplish a goal or get a result. For example, cooking a dish is a task, and its goal is to make a meal.

Performing a task requires making decisions and initiating actions. Often a task requires multiple decisions and multiple actions. For example, cooking a dish is a process that may require shopping, preparing, cooking, and serving as parts of cooking process. For this reason, in the ISC framework, we use the word “task” to refer to a process that may involve multiple actions or decisions. For example, we may say that a task has multiple decision points.
Performing a task requires performers (humans or agents), knowledge, resources, time, and information. Knowledge is the method and process of how the task can be performed. A recipe is knowledge because it provides instructions for cooking a dish. Without knowledge, one cannot efficiently accomplish a task.

Performing a task also requires resources. In the cooking example, ingredients and necessary utensils are the resources required. In addition to knowledge and resources, performing tasks also requires time. One part of time is consumed by directly applying physical actions or making decisions. Another part of time is needed for preparing and waiting for the availability of resources or environmental conditions. Reducing the unnecessary time waste is the key to high efficiency.

Last, performing a task requires information. In the cooking example, if we are cooking a steak, the temperature of the meat in the pan is information we need to make a good steak. By knowing the temperature, we decide whether the meat is ready and when to turn off the heat. Within a task process, there can be multiple decision points. Each decision point needs certain information. Simple tasks may contain only a few decision points that need little information. In contrast, complex tasks may require a great deal of information. For example, a complex task such as handling an emergency situation requires far more decisions, collaboration, and information than what a simple task, such as cooking, needs. This research concerns how to accurately pinpoint these needs.

Tasks can be categorized in different ways [160]. A task can be either complex or simple. The complexity of required knowledge reflects the complexity of a task. A complex task requires more knowledge and usually takes more time than a simple task. In some cases, a complex task can only be accomplished through a collaboration of multiple
people. Tasks can also be categorized according to the types of work: decision-making tasks or operation tasks. In addition, tasks can be categorized according to the stages of a task: to be performed, being performed, or finished. At different stages, a task requires different levels of information availability.

The above analysis leads to the definition of a task as a meta-predicate of its performers, the action, the knowledge, the current time, the start time, the estimated end time, and the environmental context.

Definition 3-1: Task
\[ Task(A, \alpha, R_\alpha, t_0, t_\alpha, t_\alpha', \Theta_\alpha) \triangleq Int.To(A, \alpha, R_\alpha, t_0, t_\alpha, t_\alpha', \Theta_\alpha) \wedge (t_0 \leq t_\alpha) \]
\[ \lor Do(A, \alpha, R_\alpha, t_0, t_\alpha, t_\alpha', \Theta_\alpha) \wedge ((t_\alpha < t_0) \wedge (t_0 < t_\alpha')) \]
\[ \lor Done(A, \alpha, R_\alpha, t_0, t_\alpha, t_\alpha', \Theta_\alpha) \wedge (t_0 \geq t_\alpha'), \text{ where} \]
\[ A \] is an agent or a group of agents;
\[ \alpha \] is an action;
\[ R_\alpha \] is a recipe for \( \alpha \);
\[ t_0 \] is the current time;
\[ t_\alpha \] is the start time of \( \alpha \);
\[ t_\alpha' \] is the estimated end time of \( \alpha \);
\[ \Theta_\alpha \] is a context for the task \( \alpha \). It specifies the progress and environmental conditions of the task.

It is worth noting that in the Shared Plans theory [73-75], “Do” has very similar form but refers to actions in general. In this framework, “Do” represents a task that is being performed. “Done” denotes a task that is finished. “Done” often refers to an information source that is achieved by an information seeking action.

The definition can represent different types of tasks. Multiple performers indicate a team task; single performer indicates an individual task. Here an agent can be either a human or a software agent.
The recipe contains knowledge and resource requirements. A recipe is assumed to be independent of performer, time, or environmental conditions. Therefore, when performing a task, different performers can access and adopt either the same or different recipes to achieve the same goal. However, recipes can affect costs and qualities. The framework also assumes that it is possible to identify the information needs based on the recipe knowledge.

The context stores the environmental conditions and other status information regarding the task such as the progress status. The start and end time in an intended task and the end time in the task being performed are all “expected time”: they can be changed when the task is prolonged or finished ahead of schedule. Figure 3-1 illustrates the three possible stages of a task and their time relations.

![Figure 3-1: Three stages of a task and their time points.](image)

Every task has a start time and an end time. One may wonder how this definition handles routine tasks which take place regularly and thus have multiple start and end times. An easy way to handle this problem is breaking a routine task into a series of individual tasks. For example, suppose a teacher needs to lecture every Wednesday
morning for a semester. The task is a routine task. One can break the task into a set of individual tasks according to the class schedule so that each task is one class, thus each task has only one pair of start and end time.

Being autonomous, an agent should be able to identify what task it intended to do. The intended tasks are agent intentions, which are fundamental concepts in belief-desire-intension (BDI) model [66-68] that analyzes the agent behaviors. Current research in agent teamwork is also heavily based on the notion of joint intention [29, 72, 102]. Agent intentions are defined in an evolving process from $\text{Int.Th}$ to $\text{Int.To}$. The former carries meaning of desired property, but requires means-end reasoning to identify actions that can make the property true. While the latter carries a meaning of commitment to make it happen [72]. Such notions can explain interesting agent behaviors such as the mentalities of communication [69, 110] and teamwork [27, 30, 72, 97, 102, 161].

The ISC framework adopts a simplified agent mentality process with three assumptions. First, the issue of converting “$\text{Int.Th.}$” to “$\text{Int.To.}$” is simplified: all “$\text{Int.Th.}$” will leads to “$\text{Int.To.}$”: that is, the framework eliminates the notion of “$\text{Int.Th.}$”

Second, if an agent intends to do an action and the scheduled time arrives, the agent will carry out the action if it can:

$$\text{Int.To}(A, \alpha, R_a, t_0, t_a, \Theta_a) \land (t_0 = t_a) \land$$

$$CBA(A, \alpha, R_a, t_0, t_a, \Theta_a) \rightarrow Do(A, \alpha, R_a, t_0, t_a, \Theta_a).$$

Finally, the framework assumes that when the end time arrives, the task is done. Often, a task may not be completed exactly on the scheduled end time. A task can be ahead of or behind the schedules. In this case, the agent can constantly update the task end time.
In summary, a task has three stages: 1) Int.To. represents committed task; 2) when the scheduled time arrives, Int.To. becomes work-in-process task Do; 3) when a task terminates the task becomes Done. A task is performed according to a recipe, which specifies individual actions, decision points, and/or needed information in a process.

### 3.2.3.2 Information

How information is processed and used has been studied as sense-making [162-164], a well-defined research paradigm. Sense-making studies the information processing activities including scanning, interpretation, and action [162]. Scanning involves information gathering and determining which information is the right information to be gathered. A scanning process corresponds to an information seeking task which is often referred to in this work. Interpretation involves making sense of the information within the context of the organization and its environment. Actions are changes that result from interpretation, and may involve developing new products or services or other strategic considerations.

Information can also be discussed at three different levels: data, information, and knowledge [21, 165]. Data represent the information that is obtained through direct perception. Data can not be easily used in human decision-making. For example, “the temperature is 5°C” can only help to make decision if one interprets it as a piece of
information such as “it is cold outside”. Data can be stored and accessed through database systems, and it is easy to share data with other systems.

Information is created by identifying, correlating, and summarizing the relevant data. Standard methods such as statistics and data mining are generally used. Compared with the data, information is more compact, more stable, and often easier to be processed by human cognition. Unfortunately, information is more difficult to share than data because information is subjective and often sensitive to context and time. “The temperature is 5°C” can mean warm in winter or cold in spring.

Finally, knowledge is contextualized information and is used to make sense of information. In order to retain and use valuable expertise, people often rely on establishing shared knowledge bases containing key documents on practical methods. However, normally such systems require users to pull the needed information from them. If a user is not aware of the knowledge base system, the system will not offer the information in a proactive and timely manner. Unlike information, knowledge is established and evolved slowly.

In this research, a piece of information (or data) has four basic properties: type, value, source, and validation time. Many pieces of information can belong to the same kind (type). For example, “location (of a building)” is a type of information that has properties such as longitude and latitude. Instance of this type of information, or a particular building’s location (property values), is independent of, and often different from, the locations of other buildings. For example, the location of the White House differs from the location of the Washington monument. The relation between information
and information type is similar to object and class in object-oriented terminology, proposition and predicate in logic, and data and schema in database.

In addition to information type and property values, information has sources, which indicate how the information was obtained. In general, information can be obtained through perception, decision-making, and communication. People can obtain information by observing or measuring. A person can look outside and get the information about basic weather condition or read the current temperature from a thermometer. Information can also result from decision-making. A judgment can be made by analyzing other information sources. By comparing the prices and quality of products from different stores, people can reach a conclusion regarding what the best deal is. Although perception and decision-making can obtain information individually, sometime it is easier or even necessary to obtain information from others by communication. All these types of sources are tasks and therefore have performers (who seek the information), actions (observing, decision-making, or communication), recipes (how the action is performed), time (current, start, and end), and a context (in what environment).

Some information is only valid in a certain period of time after which the information expires and is no longer good for making decisions. Depending on the types of information, validation periods can vary from a few seconds to days, months, or longer. For example, the location of a moving target is changing constantly; while an

---

4 Sources of information and sources of information types are different concepts. The former indicates how information was obtained, whereas the latter indicates methods or how information can be obtained. The former is an instance of the latter.
The overall weather condition can last up to a few days. Of course, some information is constant and can last ‘forever’.

The above section discussed the main properties of information including type, value, source, and validation time. The formal definition of information is given as follows:

Definition 3-2: Information

\[ i \triangleq < I, p, s, t, t'> \], where

\[ I \triangleq \text{Info.Type}(i), \]

\[ p \] is a proposition or a value of the information type,

\[ s \triangleq \text{Done.Seek}(A, \alpha_i, R_i, t_0, t_i, t_i', \Theta_i), \] where

\[ A \] is the agent,

\[ \alpha_i \] is a seeking action,

\[ R_i \in \text{recipe}(I), \] where

\[ \text{recipe}(I) \] is a set of recipes that can obtain information type \( I \)

\[ t_i \] is the time that starts the seeking action \( \alpha_i \),

\[ t_i' \] is the time that finished the seeking action \( \alpha_i \),

\[ \Theta_i \] is the context of the seeking action.

\( s \) is a task.

\( t \) is the time when the information is obtained, so \( (t_i < t < t_i') \)

\( t' \) is the time when the information will expire.

Duration of the information can be calculated according to Eq. 3.1.

\[ \text{duration}(i) = t' - t. \] 3.1

Durations of information of the same type are approximately the same, which is called type duration. A type duration is a moving average duration of all the information of that type. Once a new piece of information is obtained, the type duration will be updated. The type duration gives an estimated validation time to the information that is needed. Having
type durations is critical in planning the information seeking actions because the information has to be valid at the time when it is used.

\[\text{duration}(I) = \pi \times \text{duration}(i) + (1 - \pi) \times \text{duration}(i), \text{where} \quad \pi\;\text{is an update factor between 0 and 1.}\]

In this framework, an information type is represented as a logical predicate, and a value is represented at a logical proposition. The source of a piece of information is always a result of an information seeking task, which contains performers, actions, a seeking recipe, time (start and end), and its context. In general, a piece of information is obtained during an information seeking task, i.e., the time when the information is obtained is between the start and end time of the information seeking task.

### 3.2.4 Assumptions

The framework assumes there are multiple agents and an agent may not have perfect knowledge on others. Each agent may have multiple tasks. Each task has a time duration: a start time and an end time. Similar to task durations, validity of information can be time sensitive that information can have time durations. A piece of information expires when the time is up.

Presumably, seeking, processing, and sharing information cost time and resources [10, 166]. If the number of information needs is big or the needed information is very complex, one can become overwhelmed by the information seeking task. In addition, when multiple methods to seek a piece of information are available, making the right choice can improve the information sharing performance.
Agents may have different methods, knowledge, and capabilities to seek information. Under these assumptions, communications are necessary but they can also be costly. Agents have to carefully exam the trade-offs among multiple information seeking methods such as investigating by themselves or inquiring other agents.

Furthermore, one can assume that agents have perfect information seeking precision that all information the agent find is needed. Although in real world, it is almost impossible to have perfect precision, making this assumption is necessary to simplify the problem. Otherwise, the research has to deal with the complex information retrieval problem, which is a less-related research area albeit a well-defined one. This research will focus on the problem of information need identification and information seeking within a task context, but not on information retrieval.

Finally, one can assume that agents can exchange information without any ambiguity and an agent believes the information that is obtained from a trusted source:

\[
\forall A, s, I, p, t, t', t_0, i =< I, s, p, t, t' >. \\
i \land Trust(A, s, t_0) \land (t_0 < t') \rightarrow Bel(A, p, t_0).
\]

Strictly speaking, having a piece of information does not mean an agent will trust or believe it. Evaluating how credible a piece of information requires comprehensive analysis regarding how the information fits in the known information or knowledge, the credibility of the source, and the potential impact of the information. The information credibility issues in the ISC framework has been elaborated in [167]. For now, let’s assume that agents will trust all of its sources.
3.3 Anticipating Information Needs in Task Contexts

Performing a task can involve multiple decision points, each of which needs certain information. In a complex and dynamic task, it is important to identify what decision is about to be made, who will make the decision, and when it is going to be made, so that an agent can determine what information is needed, who needs it, and when is it needed. This section defines what is an information need, how to identify information needs, when agent will commit to satisfying the needs.

3.3.1 Information Need

Assuming knowledge\(^5\) (recipe [74] or experiences [20]) on how to perform a task specifies the information needs, one can infer the potential information needs based on such knowledge according to an information need function (Eq. 3.3).

\[
N_{r_\alpha} = \text{Info.Needs}(R_{\alpha}).
\]

The function returns all the potential information needs for a task recipe. The information need for a particular information type can be defined as follows:

Definition 3-3: Information need

\[
\text{Info.Need}(I, \Xi, R_{\alpha}) \in \text{Info.Needs}(R_{\alpha}), \text{where}
\]

- \(I\) is an information type that is needed, e.g. \((\text{DoorStatus \ ? \ status})\)
- \(\Xi\) is a set of satisfaction conditions, e.g. \(\Xi = (\text{DoorStatus \ Open})\)

When the condition is met, the information need is satisfied.

\(\Xi = \phi\) means no satisfaction conditions can be found.

\(^5\) The concept of information needs refers to a set, which contains one or more information need elements.
The satisfaction conditions represent criteria for when a need can be satisfied. Assume there is a simple task of getting out of the room for a robot that has no perception capacity. Its decision on whether or not to move depends on the information about the door status: whether it is open or closed. One can inform the robot about the door status regardless if it is open or closed. If the door is open then the robot gets out. Otherwise, the robot stays and waits. Although the door’s being closed matches the information type, it is not useful to accomplish the task. Alternatively, one can just inform the robot when the door is open. In some cases, the information value is not important for accomplishing a task. For example, the robot needs path information that can lead it outside. It does not matter which route to take. In these cases, the satisfaction condition can be set to empty: any information that matches the type can satisfy the need.

A potential information need for a recipe has no contextual properties such as when it is needed or who needs it. In a task context, however, these properties become critical to determine the actual information needs. Thus the definition of information need should include a task context:

**Definition 3-4: Information Need and Task Context** ($C_T$)

$$\text{Info.Need}(I, \Xi, C_T), \text{ where } C_T \triangleq (A, R_a, t_a, t'_a, \Theta_a).$$

At a certain time, an agent can derive the information needs and know its information needs for a task. Since the task context is changing as a task is being performed, the information needs of the task will change accordingly: from potential needs to actual needs, or from “unsatisfied” to “satisfied”.

Previous research argues that a task performer may not necessarily know its information needs from its recipe [158]. Instead, agents other than the task performer
may know the information needs and offer help. It is true that a task performer may not have enough capacity due to high time pressure or workload and requires assistance from other agents. This framework adopts the same assumption that task performers may not always be aware of their information needs:

\[ \forall \eta, \eta = Info\_Need(1, \Xi, C_\Gamma) \cdot \neg(\eta \rightarrow Bel(A, \eta, t_0)). \]

In [158], an information need is defined as \( Info\_Need(A, N, t, C_n) \), where \( A \) is an agent, \( N \) is a proposition, \( t \) is a deadline, and \( C_n \) is a context. Distinctions between information type and information are implicit. From an information seeking perspective, it is difficult to tell whether two information needs are the same type. An agent may end up launching duplicated information seeking actions. Thus, this framework makes both information type and satisfaction conditions explicit in the definition of information need. According to definition 3-4, the information needs function Eq. 3.3 for a task can be generalized (Eq. 3.4).

\[ N_t = Info\_Needs(\Gamma), where \]
\[ \Gamma = Task(A, \alpha, R_{\alpha}, t_0, t_\alpha, \Theta_\alpha, \Theta_a \). \]

The needs for a task must belong to those derived from the recipe of the task (Eq. 3.5).

\[ \equiv Info\_Needs(\Gamma) \subseteq Info\_Needs(R_{\alpha}) \].
3.3.2 Information Needer

The needer of an information need is determined by the following needer function Eq. 3.6.

\[ \text{Needer}(\eta) = A. \]  \hspace{1cm} 3.6

In general, the needer of an information need is one of the task performers. An individual task performer is always the needer. A collaborative task has multiple performers, thus the information needers are determined by the task assignments. In other words, those who make decisions need the information.

At any time, an agent knows its information needs. Eq. 3.7 gives the information needs that A currently has.

\[ \text{Info.Needs}(A,t_0) = \bigcup_{B \in \text{Bel}(A,\eta,t_0)} \eta. \]  \hspace{1cm} 3.7

\text{Info.Needs}(A,t) \text{ and } \text{Needer}(\eta) = A \text{ represent information that } A \text{ anticipated and the information that } A \text{ needs respectively. The two are different because an agent can anticipate their own as well as others’ information needs. On one hand, an agent can know others’ information needs. Suppose } A \text{ anticipates both agent } A \text{ and } B’\text{’s information needs so, } A \text{ knows information needs from two needers including itself.}

\[ \models \exists \eta, t \cdot \neg(\eta \in \text{Info.Needs}(\Gamma) \land \eta \in \text{Info.Needs}(A,t) \rightarrow \text{Needer}(\eta) = A). \]

\[ \models \exists \eta, t \cdot \neg(\text{Needer}(\eta) = A \rightarrow \eta \in \text{Info.Needs}(A,t)). \]

On the other hand, the needer may not know its own needs [158]. Before the needer is aware about the needs, others who have the task recipe and know the needer’s assignment may already anticipate the needs.
3.3.3 Recognize Information Needs

A task can involve multiple steps according to its recipe. At a step, the performer may need to determine conditions or make decisions. Therefore, it is very natural that the task performer recognizes some information needs at a decision point. If the performer needs some information that the performer doesn’t know, it must suspend the current task and seek the missing information. Obviously, the delay caused by seeking needed information can slow down a task progress undesirably. An information need that is recognized at the decision point is called an on-demand need.

Alternatively, a task performer can look ahead and anticipate the information needs according to the recipe. Having the information needs before the decision point allows a task performer to seek the information in advance so that the information is made available when it is needed. By anticipating the information needs, a task performer can reduce the time spent on waiting. How early an agent can anticipate an information need depends on the recipe structure and specification.
3.3.3.1 Anticipating Information Needs

Information needs can be anticipated by task performers or other agents. The anticipation is denoted as a function in Eq. 3.8. The two functions correspond to anticipation by self and by others respectively.

\[
\Gamma = \text{Task}(A, \alpha, R_\alpha, t_0, t_\alpha', \Theta_\alpha),
\]

\[
\text{Anticipate}(A, \Gamma, t_0) = Bel(A, \text{Info.Needs}(\Gamma), t_0); \tag{3.8}
\]

\[
\text{Anticipate}(B, \Gamma, t_0) = Bel(B, \text{Info.Needs}(\Gamma), t_0).
\]

At any time, an agent can anticipate information needs if the agent knows about the task (performer, recipe, time, etc) and the information needs of the recipe.

\[
\models Bel(A, \Gamma, t_0) \land Bel(A, \text{Info.Needs}(R_\alpha), t_0) \rightarrow Bel(A, \text{Info.Needs}(\Gamma), t_0).
\]

\[
\models Bel(B, \Gamma, t_0) \land Bel(B, \text{Info.Needs}(R_\alpha), t_0) \rightarrow Bel(B, \text{Info.Needs}(\Gamma), t_0).
\]

From Eq. 3.8, above axiom and be extended to the following.

\[
\models Bel(A, \Gamma, t_0) \land Bel(A, \text{Info.Needs}(R_\alpha), t_0) \rightarrow \text{Anticipate}(A, \Gamma, t_0).
\]

\[
\models Bel(B, \Gamma, t_0) \land Bel(B, \text{Info.Needs}(R_\alpha), t_0) \rightarrow \text{Anticipate}(B, \Gamma, t_0).
\]

The above axioms imply that an agent can anticipate the information needs \textit{without} full knowledge about the recipe. An agent can anticipate with partial recipe knowledge, from which the agent can tell that the task needs some information [158]. But the agent is not exactly clear about how to perform the task, so the second axiom is modified as follows.

\[
\models Bel(B, \Gamma, t_0) \land Bel(B, \text{Info.Needs}(\tilde{R}_\alpha), t_0) \rightarrow \text{Anticipate}(B, \Gamma, t_0), \text{where} \tilde{R}_\alpha \text{ is partial recipe for } \alpha.
\]

When two agents mutually believe that they are about to do a task, and they know how the task is assigned, then the agents know the performer of the task.
When 1) two agents mutually believe that they are about to do a task, 2) they know how the task is assigned, and 3) they know the information needed by the recipe, then the agents can anticipate the information needs of the task.

One can group anticipation into potential needs anticipation and actual needs anticipation. In potential needs anticipation, an agent anticipate the information needs before a decision point is selected. A task performer may not pass all possible decision points. Therefore, it is often difficult to tell which decision points are going to be passed. An agent can anticipate potential needs by analyzing task schedules and overall needs for each task. Potential needs anticipation can result in an overall understanding about who needs what and at what time. Therefore, the agent can determine an overall seeking capacity requirement. Potential needs may or may not be actually needed since some decision points will not be visited. As potential needs anticipation can be inaccurate, they should be further refined as the task proceeds along.

Actual needs anticipation is a way to refine the potential needs when the decision points and the needer can be accurately determined. In complex and dynamic tasks, actual needs can only be determined after the task is executed and, in some cases, shortly before the decision point. Actual needs are ideal for planning the information seeking actions because unlike potential needs, seeking information that is actually needed will not cause any waste. In this framework, anticipated information needs refer to the potential needs.

\[ \models MB(\{A, B\}, \tilde{\Gamma}, t_0) \land MB(\{A, B\}, \text{recipe}(\text{Assign}(\tilde{\Gamma})), t_0) \rightarrow Bel(A, \Gamma, t_0) \land Bel(B, \Gamma, t_0), \text{where} \]

\[ \tilde{\Gamma} = \text{Task}(X, \alpha, R_a, t_0, t_0, t_0, t_0, A, B), \text{where } X \in \{A, B\}. \]
3.3.3.2 On-demand Information Needs

On-demand information needs are actual needs identified at the time when they are needed. Thus, compared with anticipated needs, an on-demand need is always an actual need. Eq. 3.9 denotes the On-demand information need.

\[ \Gamma = Do(A, \alpha, R_a, t_0, t_a, t_a', \Theta_a), \]
\[ OnDemand(A, \Gamma, t_0) = Bel(A, Info.Needs(\Gamma), t_0). \]

On-demand needs are a subset of the potential needs.

Theorem 3-1:
\[ \forall A, \Gamma, t, t_0, t < t_0 \cdot OnDemand(A, \Gamma, t_0) \subseteq Anticipate(A, \Gamma, t). \]

Proof:
At time \( t_0 \), anticipation requires prediction of the context \( \Theta_a \) may have multiple possibilities \( \Theta_{a_1}, \Theta_{a_2}, \Theta_{a_3}, ... \). Thus
\[ Anticipate(A, \Gamma, t) = \{ Info.Needs(R_a, \Theta_{a_1}), Info.Needs(R_a, \Theta_{a_2}), Info.Needs(R_a, \Theta_{a_3}), ... \} \]
As \( OnDemand(A, \Gamma, t_0) \) is based on an actural context, e.g. \( \Theta_{a_1} \)
\[ OnDemand(A, \Gamma, t_0) = Info.Needs(R_a, \Theta_{a_1}) \subseteq Anticipate(A, \Gamma, t), \]
Thus, \( OnDemand(A, \Gamma, t_0) \subseteq Anticipate(A, \Gamma, t) \)

In addition to anticipated needs and on-demand needs, agents can have communicated needs. An agent will believe the information need once it receives an information inquiry from another agent. The framework assumes inquiries are synchronized: the time of sending a query message is the same as the time of receiving it.

\[ \models Query(A', B, \eta, t) \rightarrow Bel(B, Info.Need(\eta), t). \]

Even though the communicated needs seem to be “on-demand”, they may not be actually needed in performing a task, because a requestor can ask about anticipated information needs (not necessary actual needs). Therefore, a communicated need is “on-demand” to
the agent who receives the need, but not necessarily “on-demand” with respect to a decision point.

In summary, information needs are identified in three situations: 1) on-demand needs at a decision point, 2) anticipated needs before a decision point, and 3) communicated needs.

3.3.3.3 Comparing Anticipating Information Needs with Waiting for On-demand Needs

Anticipating needs can avoid unnecessary waiting at a decision point. Thus anticipating the needed information and making the information ready at decision points are critical for tasks that have high time-pressure. When a team is working on a complex task, anticipating teammates’ information needs and proactively delivering the information can also improve the team performance [16, 27, 97, 158]. Some teammates may have the information that others need. If they can proactively share the information with the needers, then the needers will not have to seek the information, and the overall information seeking cost will be reduced. In contrast, waiting for on-demand needs can result in delay when the needed information is not available at the decision points. Furthermore, proactive behavior will lose its characteristic if team members just wait for the demands of information needs.

Although anticipating information needs appears to be a better solution than waiting for on-demand needs, anticipating needs can have drawbacks. First, anticipated information needs can be inaccurate. Anticipation can only determine the potential information needs but it cannot determine which ones are actually needed. Investigating
the information that turns out to be not useful is a waste. In contrast, on-demand needs are accurate. For this reason, waiting for on-demand needs can avoid waste in seeking unnecessary information. Second, when no other agents are available, seeking information for anticipated needs can cost a task performer as much time as the on-demand needs. In this case, anticipation offers no benefits in time cost. Finally, information can be time sensitive. Even if the needed information is made available before they are needed, they can become invalid at the decision point. By contrast, on-demand needs allows an information seeking action be scheduled at the decision point. Therefore the information will be used shortly after it is obtained.

<table>
<thead>
<tr>
<th>Properties</th>
<th>On-demand</th>
<th>Anticipation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of accuracy</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

In summary, anticipating information needs or waiting for on-demand information needs yields a trade-off situation: a trade-off between responsiveness and efficiency. One has to select the best strategy according to the particular task requirements. In time sensitive situations, waiting for on-demand information needs can delay a decision-making process. In this case, anticipating information needs can be more responsive. By contrast, for tasks that have relaxed time schedules, waiting for the information needs requested at a decision point can avoid unnecessary information seeking and information overload due to inaccurate anticipation.
3.3.4 Satisfying an Information Need

Information that can satisfy a need should satisfy the satisfying condition set (see Definition 3-3). Additionally, information must remain valid at the time when it is needed, for example, in Figure 3-3, \(i_2, i_3\) can satisfy the need.

Sometime, the satisfying condition can not be satisfied even with all the available information. For example, a robot needs to know door status information to accomplish its task- getting out of the room. Its information need can be satisfied when it gets the information that the door is open. However, the door may remain closed all the time. Assuming that the robot knows the door has never been opened, it is reasonable to say that the robot’s information need is satisfied even though the task has failed. The task failed because of the unfavorable environmental conditions, not because the information need is not satisfied. In other words, satisfying all the information needs can not guarantee accomplishing a task successfully.

Figure 3-3: Information must remain valid.
Definition 3-5: Satisfy Information Need

\[ \forall I, \Xi, \Gamma = Task(A, \alpha, R_t, t_0, t_a, t_a', \Theta_a), C_t = (A, R_t, t_a, t_a', \Theta_a). \]

\[ Sat.(Info.Need(I, \Xi, C_t)) \triangleq \exists i_x, i_x = (I, s_x, p_x, t_x, t_x'), (t_x' > t_a), \exists t, (t_a < t < t_a'). \]

\[ (Bel(A, \bigcup p_x, t) \rightarrow Bel(A, \Xi, t)) \lor (\forall t_x, (t_a < t < t_a') \square Bif(A, I, t_x)). \]

An information need is satisfied when the needer knows a set of information that leads to the satisfying conditions becoming true or the needer knows the information about the needed type at any time during the task.

### 3.3.5 Committing to Information Needs

Information needs are special goals. When an agent has an information need, it has a desire to satisfy it. Cohen et al. [70, 71, 110] defined weak persistent goals to handle cooperative intentions. Weak persistent goals specify the conditions under which an agent holds a goal: 1) an agent believes that the goal is not true and desires the goal to be true at some future time; 2) the agent believes that the goal is already true; and 3) the agent believes that the goal is not satisfiable.

Following the definition of weak persistent goal, an information need will be held until the it has been satisfied, unable to satisfy, or the context of the information needs is no longer valid (i.e., the “needed” information becomes irrelevant).

\[ \vdash \forall t, \eta = Info.Need(I, \Xi, C_t) \cdot Bel(B, \eta, t) \mathcal{U} (Sat.(\eta) \lor \neg CanSat.(B, \eta, t) \lor \neg Bel(B, C_t, t)). \]

A need can be satisfied if there is a source that can lead to the satisfaction of the need.
Definition 3-6: Can Satisfy

\[ \text{CanSat}(B, \text{Info.Need}(I, \Xi, C_r), t) \overset{\triangleq}{=} Q \lor (\exists s, t_s \cdot \text{LEAD}(B, s, Q, t, t_s, \Theta_s)), \text{where} \]
\[ Q = \text{Sat}(\text{Info.Need}(I, \Xi, C_r)). \]

If the duration of the information can not cover the duration of the task, an agent should plan multiple information seeking actions. Figure 3-4 shows that the information need is covered by three information seeking plans because the information can expire before it satisfies the need. The information seeking tasks will terminate when the need is satisfied, can not be satisfied, or becomes irrelevant.

---

Figure 3-4: Multiple seeking plans to cover the duration of a need.

---

3.4 Information Requirement Planning

An agent can have multiple information needs. Some can duplicate with others because the same type of information can be required at multiple decision points. Some can be more urgent or more critical than others, because needs have different requirement time or impacts on decisions. Information requirement planning or IRP refers to a set of planning functions that anticipate, consolidate, source, and schedule the information seeking activities.
IRP manages the demand and supply of information for each information type. An IRP has three basic steps: transform, consolidate, and sourcing. Before IRP, information needs will be anticipated. An agent can anticipate self needs and it can also anticipate other’s information needs. At first step of IRP, the information needs will be transformed into information requirements. Next, duplicating information requirements will be consolidated so that the agent can avoid duplicating efforts in seeking information. The consolidated requirements can also be sorted according to their time and criticalities. Finally, IRP will determine an optimum information source and schedule a seeking plan. Figure 3-5 shows the basic IRP process. After the planning stage, an agent can launch seeking actions according to the plan and then fulfill the information needs with obtained information.

![Figure 3-5: Basic IRP process.](image)

### 3.4.1 Transforming Information Needs to Information Requirements

Unlike information needs, an information requirement does not associate with a decision context. An information requirement is a simplified view of an information need: it includes only properties that are useful for information seeking: information type and time when it is needed. Information requirement does not include needer because it is not helpful to the task of information seeking. This implies that all needers are treated
equally. An information requirement does not include satisfaction condition because the seeking actions are independent to the content of the information. For example, a robot needs the door status information when it needs to get out of a room. The satisfaction condition is “the door is open”. Suppose another robot may need the door status information too, but the satisfaction condition is “the door is closed”. The two information needs have the same information type but different needers and different satisfaction conditions. From planning perspective, only one information seeking action is needed for the door status. Therefore having satisfaction conditions will not help the information seeking actions. Likewise, recipes and contexts are not included in an information requirement. Formally, the definition of an information requirement is given as follows.

Definition 3-7: Information Requirement

\[
\forall \eta, \eta = \text{Info.Need}(I, \Xi, C_r), C_r = (A, R, t, t', \Theta).
\]

\[
\text{Info.Req}(\eta) \triangleq < I, t, t' >.
\]

An information requirement is a triple of information type, a start time, and an end time. Information requirement hides unnecessary details for better demand visibility. It is also useful for consolidating information needs. Each information need has an information requirement. When an information need changes, the corresponding information requirement will be updated as well.

Chapter 3.6 will introduce using auctions to determine an efficient information provider. When a provider responds to a request for quote (RFQ) with an offer, the offer can generate a requirement and some capacity of the provider must be reserved for seeking the information. Otherwise the provider agent may never respond to the request.
for quote because it may be overwhelmed by other tasks all the times. Thus, reserving
time or capacity of information seeking task is important to the fulfillment of a promise
made by a provider agent.

Since seeking information for an information requirement yield the same
information as that resulted from information seeking for an information need, satisfying
information requirements is equivalent to satisfying the corresponding information needs.

\[ \Downarrow \text{Sat(Info.Req(\(\eta\)))} \rightarrow \text{Can.Sat(\(\eta\))}. \]

### 3.4.2 Consolidate Information Requirements

When consolidating requirements, an agent needs to tell if two information
requirements overlap. Overlapping requirements may cause redundant information
seeking, which is inefficient. Two requirements overlap if they share the same
information type and overlapped requirement time periods.

**Definition 3-8: Overlap**

\[
\delta_x = \text{Info.Req}(\eta_x) = \text{Info.Req}(I_x, t_x', t_x'),
\delta_y = \text{Info.Req}(\eta_y) = \text{Info.Req}(I_y, t_y', t_y'),
\text{Overlap}(\delta_x, \delta_y) \triangleq (I_x = I_y) \land (\neg (t_y' < t_x \lor t_y > t_x'))
\lor \text{duration}(I) > \max(t_y', t_y') - \min(t_x', t_x)).
\]

In Figure 3-6, the two information needs have overlapping requirements and
should be consolidated.
If a piece of information lasts long enough to cover two information requirements, the requirements should also be consolidated because they can share a single information seeking action. Figure 3-7 shows that two information needs are close to each other in time so that they can share a single information seeking task that generates a single piece of needed information.

When the resource for seeking information is limited, IRP should prioritize the requirements according to the criticalities and urgencies of the needed information. How to set the criticalities of the information needs, however, is out of the scope of this
research. Section 3.6 will describe an auction-based approach that can determine the most efficient information provider. One can apply a similar auction mechanism to bid for provider’s services, so that the provider will serve the most critical information needs.

Supply of information is independent to a particular information requirement or information need. Consolidating information needs of the same type or similar time can improve the efficiency of an agent information seeking task. IRP is about planning how to satisfy the requirements so as to satisfy the needs. Eq. 3.10 gives the requirement consolidation function.

\[
\sigma_x = \text{Info.Req}(I_x, t_x, t_x') \\
\sigma_y = \text{Info.Req}(I_y, t_y, t_y') \\
\text{Cons.Req}(\sigma_x, \sigma_y) = \begin{cases} 
\{\sigma_x, \sigma_y\} & \text{Overlap}(\sigma_x, \sigma_y) \\
\text{Info.Req}(I_x \vee I_y, \min(t_x, t_y'), \max(t_x', t_y')) & \text{Overlap}(\sigma_x, \sigma_y)
\end{cases}
\]

Figure 3-8 illustrates the two IRP steps: consolidate the information requirement and then plan the information seeking actions.
An agent may have multiple seeking tasks. It should avoid committing to tasks beyond its capability limit. Suppose an agent can only execute one seeking operation at a time, the agent cannot schedule two overlapped information seeking tasks. Figure 3-9 shows two information needs that are beyond an agent’s seeking capacity.

![Figure 3-9: An agent should plan within its capacity constraints.](image)

### 3.4.3 Determine Sources

An agent can seek information by fusing low level information, by inquiring other agents, or by direct observing, measuring, or testing. Since fusion requires multiple pieces of low-level information, an agent must check if the required low level information is available. If required information is complete, the agent can directly apply knowledge to transform the low-level information into the information that is needed. If required information is incomplete, the agent cannot fuse the needed information. In this case, the agent must diagnose and find out the missing information. After making the missing information available, the agent can continue the fusion process. Fusion is a special task, and the low-level information is the information needs of the fusion task.
In addition to fusion, inquiring other agents can also obtain the needed information. With inquiry, a task performer does not need to seek the needed information, so the agent can concentrate on accomplishing other tasks. In a multiple-agent environment, agent specialization in certain types of information coupled with effective information sharing can improve the overall performance. A query can be either a one-time ask or a subscription that requires multiple updates. If there are multiple potential providers, an agent can hold an auction to determine the most efficient one.

Finally, an agent can launch an operation to observe or measure the needed information. This is called information seeking operation, which may require complex processes, thus can cost more than fusion and communication methods mentioned earlier. Depending on the problem domain, an information seeking operation can also include various information needs.

In summary, information seeking has three types: information fusion, inquiry, and investigation. Each type is a special kind of task, which can have multiple recipes (see Figure 3-10) and information needs.

Figure 3-10: Multiple types and recipes of information seeking task.
Since information seeking can have multiple options, an agent must make a decision to select an optimum one. This decision is called seeking strategy determination. The decision can be made according to domain specific criteria such as investigation cost or credibility. Seeking strategy determination (Eq. 3.11) is the most important and also the most complex function in IRP.

\[ \text{Seek}(A, \alpha, R_t, t_0, t_1, t_1', \Theta_t), \text{Info.Need} (\eta) \]

\[ R_t = \text{SeekStrategy}(A, \text{recipe}(I), t_0), \text{where} \]

\[ \text{recipe}(I) = \bigcup_{\gamma \in \text{IDR}(I)} \text{Fuse}(\text{Dep.Needs}(\eta, \gamma)) \bigcup \]

\[ \bigcup_{B \in \text{Info.Providers}(A, I, t_0) \times \text{Needer}(\eta)} \text{Query}(A, B, \eta, t_0) \bigcup \]

\[ \bigcup_{\partial \in \text{SeekPlans}(A, I, t_0)} \text{Investigate}(\eta, \partial, t_0), \text{where} \]

SeekPlans(A, I, t_0) is a set of plans (recipes) that can result in getting the type of information.

A is the requestor.

3.4.4 Knowledge-based Information Requirement Decomposition

Information fusion is a function that transforms low level information to high level ones. In other words, high level information depends on the low level information. An information type has dependent information types if there is a fusion rule that can fuse the information of the type.

Definition 3-9: Dependent Information Type

\[ \text{Dep.Info.Type}(I, \gamma) \triangleq \{I_1', I_2', \ldots, I_n'\} \text{ such that} \]

\[ \text{Fuse}(\bigcup_{\text{Info.Type}(i) \triangleright \{I_1, I_2, \ldots, I_n\}} i, \gamma') = i \wedge \text{Info.Type}(i) = I. \]

An information type can be decomposed to dependent information types with a fusion rule. An information need can be decomposed to a set of dependent information
needs in a way that is similar to decompose a product into parts. The high level information need is called independent information need and the decomposed needs are called dependent information needs.

Definition 3-9: Dependent Information Need

\[ \text{Dep.Needs}(\eta, \gamma) = \bigcup_{t \in \text{Dep.Info.Type}(t, \gamma)} \eta_d \land (\forall \eta_d : \text{Sat}(\eta_d)) \rightarrow \Xi, \text{where} \]

\[ \eta = \text{Info.Need}(I, \Xi, C_T); \eta_d = \text{Info.Need}(I_d, \Xi_d, C_T), \text{where} \]

\[ \Delta = \text{Seek}(A, \alpha, R_0, t_0, t_1, t_1', \Theta_1). \]

In business management, bill of materials (BOM) lists the components needed to produce one unit of a product. Checking each component’s availability can reveal the shortage for desired productions. Figure 3-11 shows a simple BOM in a tree structure: a computer is composed of a machine, a monitor and a keyboard. The machine is composed of a main-board, a CPU, and a hard-disk.

![Figure 3-11: A BOM tree and an IDR tree.](image)

Similar hierarchical composition or dependency relationships can be found among information types. For example, a piece of information may depend on several supporting information or evidences, each of which may further depend on other supporting information. Such dependency relationship is called information dependency relation (IDR), which can also be represented in a tree-like structure.
In Figure 3-11, next to the BOM tree example shows an IDR tree about anti-terror intelligence analysis: 1) a group is labeled as “has key insurgents” if the group has a member who is on the wanted list; 2) a group is considered as “dangerous” if the group has a key insurgent and its size is large. Each node in the tree corresponds to the application of an antecedent-consequent rule. Suppose a group is large and its members are known. Diagnosing the IDR can identify the missing information (dependent needs) — “if the members are on the wanted list”.

When all dependent information needs becomes available, the needed information can be fused. Therefore, when dependent information is satisfied, the independent need can be satisfied.

Theorem 3-2:
\[ \exists \gamma_s \cdot Sat.(Dep.Needs(\eta, \gamma_s)) \rightarrow Sat.(\eta). \]

Considering there may be multiple rules to fuse a type of information, it is desirable to choose a rule that has high availability and low seeking cost of dependent information. Determining the best fusion rule can be a complex task that requires traversing the IDR tree recursively. An IDR tree is none-circular. In other words, an information type can not be an ascendant and a descendant node for another type at the same time. Figure 3-12 shows an information type that has multiple fusion rules.

---

6 We use logical rules as an example for IDR. However, IDR can also be used to capture other dependences such as the aggregative or selective relations among views and data sources.
3.4.5 IRP Algorithm

The degree of complexity of the basic IRP algorithm is $O(n)$, or the computational cost is in linearly proportional to the number of the needs. IRP is based on an assumption that a determined strategy will work and can satisfy the need. However, in the real-world, a strategy can easily fail. For example, an agent may ask another agent about an information need. Although the agent that is being asked has the necessary capability it may be too busy and cannot seek the information. In this case, the determined strategy failed. As a consequence, IRP must re-plan or activate a backup plan.
Notations:

*Info.Reqs.Table(A,t₀)* is a table that keeps the mappings between information needs and information requirements.

*Info.Con.Table(A,t₀)* is a table that keeps the mappings between information requirements and consolidated information requirements.

*Info.SeekTable(A,t₀)* is a table that keeps the mappings between consolidated information requirements and seeking action.

* refers to anything.

Updates of the needs, requirements, and seeking actions (satisfied, canceled, failed) can be done through the association relations above. Therefore algorithms for them are ignored here.

Begin

For each new information need \( \eta \in \text{Infor.Needs}(A,t₀) \land (\eta,*) \) \( \notin \text{Info.Reqs.Table}(A,t₀) \) do

create a new information requirement: \( \sigma = \text{Info.Req}(\eta) \);
add \( \sigma \) to \( \text{Info.Reqs}(A,t₀) \);
add \( (\eta,\sigma) \) to a table \( \text{Info.Reqs.Table}(A,t₀) \);
End

For each new information requirement \( \sigma \in \text{Info.Reqs}(A,t₀) \land (\sigma,*) \) \( \notin \text{Info.Con.Table}(A,t₀) \) do

determine a consolidated information requirement:
\( \sigma' = \text{Cons.Req}(\sigma,\text{Cons.Reqs}(A,t₀)) \);
if \( \sigma' \notin \text{Cons.Reqs}(A,t₀) \), add the new consolidated information requirement \( \sigma' \) to \( \text{Cons.Reqs}(A,t₀) \);
add \( (\sigma,\sigma') \) to a table \( \text{Info.Con.Table}(A,t₀) \);
End

For each new consolidated information requirement \( \sigma' = \text{Info.Req}(I,t,t') \cdot \sigma' \) \( \notin \text{Cons.Reqs}(A,t₀) \land (\sigma', *,) \) \( \notin \text{Info.SeekTable}(A,t₀) \) do

determine a seeking strategy: \( R_j = \text{SeekStrategy}(A,\text{recipe}(I),t₀) \);
create a new seeking intensity: \( \Gamma = \text{Int.To.Seek}(A,\alpha,R_j,t₀,t,t',\Theta) \);
add \( (\sigma',\Gamma) \) to a table \( \text{Info.SeekTable}(A,t₀) \);
End

End
3.4.6 Challenges

Performing a task, especially a decision-making task, requires information and its availability, which can have profound impact on the task outcome. Considering that, sharing either too much or too little information with a task performer is undesirable. On one hand, sharing too much information can cause information overload. A piece of crucial information may be easily ignored if it is buried in a large volume of irrelevant noise.

On the other hand, sharing too little information can cause information deficiency. A task can easily fail if some critical information is missing. Information deficiency can result from two reasons. First, an agent will not seek information for unrecognized information needs. From individual agents’ perspective, if an agent has incomplete recipe knowledge about the task, the agent can not recognize what information a task performer needs. An agent also cannot recognize information needs if the agent does not have enough time to anticipate the needs. In both situations, it is important for other teammates to anticipate the decision-maker’s information needs and proactively seek and share the needed information. The second reason is that agents cannot obtain the needed information. An agent can obtain the needed information only if the agent has enough capability or time. When an agent is working on tasks with high time pressure, it is likely that the agent does not have enough time to seek the needed information. Furthermore, if an agent cannot accurately anticipate its information needs, the agent may waste time seeking information that is not actually needed.
In addition to the problem of information deficiency, overwhelming a task performer with too much information can be equally detrimental because the newly obtained information has to go through an interpretation process before it can be used. A task performer has limited capacity for comprehending information. Therefore, too much information, especially irrelevant information, can easily overload the task performer.

To combat the problem of information deficiency and information overload, one must accurately identify the information needs and efficiently seek the information to satisfy the needs. The following section introduces key performance indicators for these information management activities.

### 3.4.7 Evaluation of Information Management

Anticipation-accuracy and anticipation-coverage are two measures for evaluating the quality of identified information needs. They are defined based on three need sets shown in Figure 3-13: information needs of a task, needs that an agent identified, and needs that are satisfied.

![Figure 3-13: Information need sets.](image)
An anticipation-coverage is a ratio of the number of actual needs anticipated to the total number of needs for a task (Eq. 3.12).

\[
\text{Anticipation-coverage}(A, \Gamma) = \frac{\text{Count}(\text{Info.Needs}(\Gamma) \cap \text{Info.Needs}(A, \Gamma))}{\text{Count}(\text{Info.Needs}(\Gamma))}. \tag{3.12}
\]

An anticipation-coverage measures how completely an agent has identified the needs for a task. A high anticipation-coverage value (close to 1) indicates that the agent has anticipated most of the needs, whereas a low anticipation-coverage value (close to 0) indicates that the agent did poorly in its anticipation. A low anticipation-coverage can result when the agent does not know the needs, and this will lead to information deficiency.

An anticipation-accuracy is the ratio of the number of actual needs to the total number of needs that the agent identified (Eq. 3.13).

\[
\text{Anticipation-accuracy}(A, \Gamma) = \frac{\text{Count}(\text{Info.Needs}(\Gamma) \cap \text{Info.Needs}(A, \Gamma))}{\text{Count}(\text{Info.Needs}(A, \Gamma))}. \tag{3.13}
\]

It measures the quality of needs anticipated. Similar to anticipation-coverage, the value can range from 0 to 1. A high anticipation-accuracy value suggests that most of the anticipated needs are actually needed for performing the task. By contrast, a low anticipation-accuracy suggests that what an agent identified are not needed. Seeking information for unnecessary needs can waste the agent’s time and the unnecessary information can cause information overload.

Anticipation-accuracy and anticipation-coverage are similar to precision and recall in the information retrieval research field. However, it is worthwhile to note the differences. In the information retrieval field, precision and recall are the basic measures
used in evaluating search strategies. Good searching strategies can retrieve all information yet with little irrelevant information. By contrast, anticipation-accuracy and anticipation-coverage are about needs anticipation strategy not about information retrieval.

In addition to measurements for evaluating the identification of information needs, fill-rates evaluate overall information seeking performance. Fill-rate is the ratio of the number of actual needs satisfied to the total number of actual needs for a task (Eq. 3.14).

$$\text{FillRate}(A, \Gamma) = \frac{\text{Count}(\text{Sat.(Info.Needs(\Gamma)))}}{\text{Count}(\text{Info.Needs(\Gamma))}.}$$ \hspace{1cm} 3.14

Fill-rates are based on the actual needs for a task. Poor fill-rate may be caused either by poor anticipation-coverage or by poor anticipation-accuracy. First, poor anticipation-coverage means that a large number of needs are unidentified, hence they cannot be satisfied. Second, poor precision can also cause low fill-rate. Poor precision means the identified needs are actually not needed by a task performer, and the agent has wasted its time. Poor precision will also affect the performance of the agent with regard to other tasks since the agent cannot work efficiently and has spent a lot of time seeking useless information.

In summary, in order to have better information management for a task performer, the anticipation of information needs must be complete and accurate, and the information seeking must be efficient.
3.5 Information Supply Chain

Previous sections define information needs, information requirements, and basic IRP functions. They are in an intra-agent perspective, which exams an individual agent. Section 3.5 and 3.6 will switch to an inter-agent perspective, which studies information sharing among a collection of agents.

Agents often have to collaborate and share information to fulfill their information needs. In a multi-agent environment, it is important for an agent to know who provides what information and who needs what information. The following section gives a definition to the concept of information supply chain and introduces its properties, models, and functions.

3.5.1 Information Partner

If a pair of agents mutually believes that one can provide information to the other who needs the information, the pair is called information partners. The party that provides information is called an information provider and the party who receives it is called an information needer. Each agent can have multiple providers and each provider can have multiple needers.

An information provider is an agent that can satisfy the needs of a type of information.

Definition 3-10: Information Provider
\[ \text{Info.Provider}(A, I, B, c, t) \equiv \forall \Xi, \Gamma \cdot \text{Bel}(A, \text{CanSat}.(B, \text{Info.Need}(I, \Xi, C_{\Gamma}) \land \text{Cost}(B, I, t) = c, t)). \]
An agent is qualified as an information provider because of the following reasons:

- The agent has provided the type of information.
- The agent has the capability (knowledge or recipe) that is required for seeking the type of information.
- The agent is committed to seeking the type of information.

The cost for seeking and sharing the information indicate how efficiently the provider is in providing the information. The cost is a moving average cost, which is updated after each communication. Both the provider and the needer can keep track of the cost, and this framework assumes that all agents are honest and will share information with the real cost. Whether or not an agent can deceive in order to better his performance is a philosophically interesting question. However, this question is out of the scope of this research. Of course, the unit of measurement may not be a monetary one. Instead, which provider is more efficient is determined by a numerical value so that the overall system performance can be optimized by giving the information seeking tasks to the most efficient agents.

A needer is an agent that may have information needs of a type of information.

Definition 3-10: Information Needer

\[
\text{Info.Needer}(A, I, B, t) \triangleq \exists \Xi, \Gamma, t \cdot \text{Bel}(A, \eta \wedge \text{Needer}(\eta) = B, t),
\]

where

\[
\eta = \text{Info.Need}(I, \Xi, C, t).
\]

\[
\text{Info.Needers}(A, I, t) \text{ is the needer set.}
\]

An agent becomes a needer because of the following reasons:

- The agent has requested the information of the type as a needer.
- The agent has worked on a task that may need the information.
A partnership refers to that two agents mutually believe that potentially one can provide needed information to another. The agent should update the partnership according to its collaboration experiences. For example, an information provider may deny an inquiry when it is too busy. Then, the requestor must find a new provider and avoid overwhelming the existing one with any additional request.

**Definition 3-11: Information Partner**

\[ \text{Info.Partner}(A, I, B, c, t) \triangleq MB([A, B], \text{Info.Provider}(B, I, A, c, t) \land \text{Info.Needer}(A, I, B, t), t), \text{where} \]

- \( A \) is the provider,
- \( B \) is the needer.

### 3.5.2 Definition of Information Supply Chain

A collection of information partners forms an information supply chain.

**Definition 3-12: Information Supply Chain**

\[ \text{Info.SupplyChain}(\Lambda) \triangleq \bigcup_{A \land B \land \Lambda \land (A \neq B)} \text{Info.Partner}(A, I, B, c, t), \text{where} \]

- \( \Lambda \) is all the members of the supply chain.

In an information supply chain, agents collaboratively anticipate information needs, plan, and seek the needed information. The information, in general, is fused from low level data to high level information. As a result, an information supply chain (ISC) fulfills users’ information requirements by a chain of information agents that include a) scanning agents that gather information and provide information to other agents, b) interpretation agents that make sense of the information, and c) broker agents that collect users’ requirements and satisfy the requirements with proper information. Figure 3-14 shows an information supply chain.
Usually, an ISC is about one information type for a particular agent. Thus, in a group of agents, different agents may have different ISC views. For example, \(Info.ISC(A,I,t)\) and \(Info.SC(B,I,t)\) may be different.

### 3.5.3 Basic Communication Modes

Communication is about exchanging both information needs and information. The first step in communication is to identify the information needs. There are three ways an agent can obtain information needs. Figure 3-15 illustrate the basic processes of the three modes.
First, an agent can communicate an information need directly. If agent $A$ has information need $X$ and $A$ believes that agent $B$ can satisfy the need, $A$ will *ask* $B$ about the information. As a result, $B$ will *reply* $A$ with the needed information or let $A$ know that it cannot be satisfied.

\[
\forall I, \Xi, \Gamma = \text{Task}(A, \alpha, R_{\alpha}, t_0, t_\alpha), \Theta_\alpha), C_I = (A, R_{\alpha}, t_\alpha, t_{\alpha}', \Theta_\alpha).
\]

Ask($A, B, \eta, t_0$), where

\[
\eta = \text{Info. Need}(I, \Xi, C_I)
\]

Pre : $\text{Bel}(A, \eta \land \neg \text{Sat.}(\eta) \land$

$\text{Bel}(B, \text{CanSat.}(\eta), t), t_0), where t_\alpha > t > t_a$

Post : $\text{Sat.}(\eta) \lor \text{Bel}(A, \neg \text{CanSat.}(\eta), t), t_0)$.
The benefit of ask-reply is that needs are accurate. However, ask requires clear knowledge about who can provide the information. Additionally, ask requires explicit communication for every information need.

Second, an agent can communicate the task and let the information provider decide what information is needed and when it is needed. We call this communication mode task information subscribe (TIS).

\[
\text{TaskInfoSubscribe}(A, B, \Gamma, t_0) \quad \text{Pre :}\ Bel(A, N \land \neg Sat.(N) \land Bel(B, CanSat.(N), t), t_0), \text{where}
\]

\[
t_a > t > t_a, N = Info.Needs(\Gamma)
\]

\[
\text{Post :}\ Sat.(N).
\]

TIS differs from subscription. A TIS shares task information whereas a subscription shares information needs. Furthermore, a TIS implies multiple information needs whereas a subscription only requests one type of information. As a task process is dynamic according to its environment, the information provider must monitor the task progress and adjust the needs accordingly.

Finally, an agent can anticipate others’ task and information needs. Being able to share needed information without any explicit request is called \textit{ProInform} [15, 19, 31, 97, 99, 158, 168, 169]. \textit{ProInform} differs from TIS in that \textit{ProInform} needs to reason about the needer’s task. \textit{ProInform} is based on the shared mental model theory (SMM) [26, 28, 94, 99, 113, 157, 170] and does not need any communication about the task.

\[
\text{ProInform}(B, A, t_0) \quad \text{Pre :}\ \forall \eta, \eta = Info.Needs(I, \Xi, C_\pi), \eta \in Info.Needs(\Gamma). \quad \text{Bel}(B, \Gamma \land N \land \neg Sat.(N) \land Bel(B, CanSat.(N), t), t_0), \text{where}
\]

\[
t_a ' > t > t_a, N = Info.Needs(\Gamma)
\]

\[
\text{Post :}\ Sat.(N).
\]
Compared with the other two methods, ask-reply requires little collaboration and is the most accurate way to share needs. However, it requires significant communication, which can cause problems when the number of information needs is large or communication cost is high. In those cases, it is better to choose TIS or ProInform, which do not require communication for sharing the information needs. However, they require the communication parties to form a high degree of SMM, which requires significant amount of training. Table 3-2 summarizes a comparison of the three communication modes.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Ask-reply</th>
<th>TIS</th>
<th>ProInform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of accuracy</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>Low-high</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Number of communications</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Requirement of SMM</td>
<td>Low</td>
<td>Mid</td>
<td>High</td>
</tr>
<tr>
<td>Collaboration requirement</td>
<td>Low</td>
<td>Mid</td>
<td>High</td>
</tr>
</tbody>
</table>

### 3.5.4 Basic ISC Protocol

An ISC provides a collaborative environment for the information agents to share and fulfill information needs. On one hand, each agent has limited capability for planning, seeking, fusing, and communicating information. On the other hand, decision and information requirements can be enormous in a complex real-time task environment. An ISC cannot have a centralized node that has global view or responsibilities, because
the centralized node will be overloaded easily. Therefore, the planning and seeking actions in an ISC are distributed: there is no central control agent. The distributed structure can avoid potential bottlenecks due to the limited information processing capacity.

However, a distributed collaboration may cause two problems. First, an information need may be passed to multiple agents and cause duplications in demands. In Figure 3-16, for example, C inquiries information I from both A and B. After receiving the query, B may relay the request to A. Hence, A has the same requests from both C and B. When A knows the information, A will send it to both C and B. Then B will pass the information to C. So, C received the same message twice: once from A and once from B. This process has duplicated information exchange. It involves 6 communications, which can be reduced to two.

In addition to duplication, a more serious problem is circular demand. In Figure 3-16, the information needs is transferred among A, B, and C in a circle. Unlike duplication, circular demands will keep A, B, and C busy all the time. So, an ISC must eliminate circular demands.

![Figure 3-16: Duplicated and circular demands.](image)
In order to solve the duplicated and circular demands, the framework adopts a simple protocol change in a normal inquiry: an inquiry will include the needer(s)’ identity. For example, when C passes the request to A and B, C will specify that the needer is C. Likewise, B will specify the needer, C, when it passes a query to A. When A receives a query, it will compare whether there is a duplicated query (same needer, information type, and time). If the query is a duplicating one, the agent will not take any further action to fulfill it. Instead, the agent will pass the needed information directly to the needer, C, and notify the requestors B that its inquiry has been fulfilled.

The same mechanism can avoid circular demands as well. Suppose C receives a duplicated query from A, it will not pass the query any further, thus avoiding any circular demands. Finally, as a commonsense, one should never pass a query to the needer.

3.5.5 The Benefits of ISC

Forming partnerships in an information supply chain can help an agent to choose the best partners to request or share information. When there are a large number of agents, deciding who can provide what information can be difficult. Agents adopt some strategies to help their decision-making. There are three basic kinds of information sharing strategies: free communication, broker facilitated communication, and ISC based communication (Figure 3-17).
In the free communication model, an agent can randomly inquire a set of agents or inquire everyone. If an agent chooses to inquire everyone, this model can involve a lot of communication that is not useful. If the agent has limited capacity to process communication, this model can overload the agent with unnecessary communication. If the agent chooses to ask a set of few agents, this model can be less effective in seeking the information because the agent may not be asking the right information provider.

The second model is broker facilitated communication, in which an agent will inquire an information broker, who has full knowledge on who can provide what information. Once a query is received, the broker will contact the appropriate information

Figure 3-17: Comparing ISC with other communication models.
provider and let the provider respond to the information needer. This method assumes that there exists an agent who has full knowledge of every agent’s capabilities and observabilities. In a large team and an information rich environment, however, this is very difficult to attain. In addition, the broker can easily become a bottleneck. Overloading the broker will jeopardize the whole information sharing activities.

In contrast, information supply chain can manage information flow and with a market mechanism it can quickly identify an information seeking plan for new information types. ISC is both responsive and efficient. With a partnership relation, both the provider and the needer understand each other’s responsibilities and capabilities. Choosing an information provider from a small set of agents can save a lot of time and communication. The next section introduces a market-based method for determining information providers. From a provider’s prospective, the fewer agents it serves, the better it anticipates their needs. Keeping track of everyone’s tasks for an agent is not realistic. So the ISC helps an agent to reduce its collaboration space and helps it to focus on its partners.

An agent will always choose the best information providers who can provide information at lowest cost. Several factors can make an agent be a better provider in certain types of information. First, the agent always has the information. Some information can last for a long period of time. If an agent has the information, it can offer the information at no cost. Then the agents can gradually become specialized in providing that type of information. Second, the agent has information fusion rules that require fewer dependent types (thus lower cost to satisfy these dependent information needs). Consequently, it can offer the information at a lower cost. Finally, good observability can
also make an agent an efficient provider. If an agent is close to the object, it has a better chance to observe the object. Hence, it can easily provide the information about the object.

In summary, an ISC can improve the overall efficiency of the information seeking task for a large set of agents. It can reduce the cost of provider determination by reducing the number of contacts, and it can optimize the allocation of information seeking tasks by making the agents specialized.

### 3.6 Establishing Information Partnership and Forming Information Supply Chain

Information supply chain is formed through discovery and formation of information partners. However, how can agents efficiently form a supply chain without predefined knowledge is still a question.

#### 3.6.1 Extending Contract Net for Information Auction

Since the agents are distributed, there is no agent that can know all other agents’ status. Sharing the status of everyone to a single agent requires expensive communication. In addition, agents’ availabilities and observabilities are constantly changing. Hence, keeping information updated requires extra communication cost. Therefore the decision on choosing a best provider should be distributed. This section extends contract net protocol [144] to identify a chain of contractors and sub-contractors for independent information requirements and dependent information requirements. Since
the goal of using auction is efficient collaboration, all agents will adopt honest bidding policies for determination of the best information providers (Figure 3-18).

Figure 3-18: Information auction protocol.

When an agent doesn't know who the potential providers are, the agent (auctioneer) broadcasts a request for quote (RFQ) to everyone (bidders). A bidder should respond to the RFQ by determining an offer price. A bidder can refuse the RFQ if the bidder cannot obtain the needed information. Otherwise, it will determine the lowest price by examining all the information seeking methods. In Figure 3-18 the bidder determines the price of direct observations and fusion. The prices can set in three ways:

1) if history data is available, it can be estimated according to the history data; 2) it can
be calculated based on the price of the observation plan; or 3) if the provider needs help on dependent needs from its providers, the provider can calculate the price by adding its sub-contractor’s price, which can be determined in a recursive auction. Since recursive auction could be computationally expensive, agents should avoid excessive auctions by determining prices according to history prices.

After the auctioneer receives bids, it will award the information seeking task to the winner, whose bidding price is the lowest among all the bidders. Then the winner will seek and delivery the needed information. If the winner involves any sub-contractors, the winner will award the dependent needs to its sub-contractors. Therefore, both auction announcing and awarding can be a recursive process. Compared with contract net protocol [144], which can establish 2-party contracts, this auction protocol allows agents to identify a contractor and a chain of sub-contractor to collaboratively working on a task.

In this process, agents can establish understanding about who needs what information and who can provide what information. On one hand, a provider can establish profiles for information needers. When a provider receives an auction announcement, it can update the needer’s profile, which will help the provider to prepare for information request from the same needer. The more often a needer requests certain types of information, the more likely the needer will ask for them again. Such need likelihood values can be used to represent a partnership value.

On the other hand, a needer can establish profiles for information providers. After a needer receives the information from a provider, the needer enhances the information provider’s partnership values. The more a provider wins and provides the needer’s information requests, the higher the partnership value will become.
After the partnership value increased to certain level, the needer will no longer hold auctions for that type of information needs. Instead, the needer will inquire the best provider directly.

### 3.6.2 Chain Auction, an Example

Figure 3-19 gives an auction example. Assume $A$ knows $\gamma_i$ and $\gamma_j$ for fusing the information $I$, and $B$ knows $\gamma_x$ for fusing information $I_1$. $A$ needs information $I$.

---

**Figure 3-19:** An information auction example.

---

$A$ holds two auctions $I_j$ and $\{I_2, I_3\}$. Once receiving $A$’s RFQ, $B$ holds an auction $\{I_4, I_5\}$. Bids are listed below each information type. In this example, $A$ awards $B$ the information seeking job because $B$ has the lowest bidding price, 16. Only after getting the awards, can $B$ award $A$ and $F$ for its dependent needs. It is worth to note that $A$ is the winner for $I_4$. $A$ must send $I_4$ to $B$ because $A$ doesn’t know how to fuse $I_1$ with $I_4$ and $I_5$. A potential supply chain can form after these tractions:
3.6.3 Bidding Behavior

In this framework agents will bid at prices that are based on the true seeking costs and the agents’ current utilization (Eq. 3.15).

\[ \text{price} = f(\text{utilization}) \times (\min(\text{information seeking cost}) + \text{penalty}). \]

A bidding price should reflect both the bidder’s capability and free capacity. The capability is reflected through the information seeking cost. If the bidder is ‘good’ at getting the information, the overall seeking cost should be lower than others. The bidder’s capacity is reflected through the utilization rate and penalty cost. The utilization rate can be defined as the ratio between the agent’s current workload and the agent’s overall capacity. A high utilization rate suggests that the agent is busy; a low utilization rate indicates that the agent is idle and can work on new tasks. In other words, the agent will bid at high prices when it is busy and bid at low prices when it is idle. This mechanism avoids overwhelming an agent with too many investigation actions: the more the utilization the higher the bidding price, thus the lower probability of getting new information seeking jobs.

The penalty refers to the cost if an agent defaults an agreed information seeking task. Default allows an agent to work on more important (high prices) information seeking task. How to set penalties affects an ISC performance. If penalties are set high the risk of losing a partnership is low. However, it is less likely to achieve optimum
efficiency because agents are bounded by their promises and can not work on more important jobs. If penalties are set low then a commitment is very easy to break. However, agents can work on the most suitable jobs. If an agent is working on a lot of tasks, the agent’s partnership will be disbanded when a provider revoke its seeking commitment. Loss of commitment through this mechanism implies that the old partnership is not optimum.

3.6.4 Discussions

As agents form and update an information supply chain, the agents can coordinate and optimize the information seeking activities. If an agent has already been seeking a piece of information, the agent can promise other needers with a low bidding price and it likely to win the seeking task for the same information type. This can save overall information seeking effort because this auction mechanism can discourage unnecessary or duplicating information needs. The auction mechanisms also help agents identifying the true cost of an information requirement. If a type of information has many needs or there are only a few agents who can seek the information, the price of getting the information will be high. In this case, more agents are needed to improve the capabilities for providing this type of information. Since auctions require additional communication other than the one for sharing information, an auction can worsen the problem of information overload. Therefore, after the information supply chain is formed, an agent should not hold auctions anymore.
This auction mechanism is extended from contract net protocol [144]. It allows a manager use auction to identify the best contractors. This thesis applies the same principle to identifying the best information providers. The contract net protocol has been extended to make it suitable to handle auctions that are required in information supply chain. First, the contract net protocol is extended to handle sub-contracts: a potential contractor may hold additional auctions to determine the cost of sub-contracts before it create an offer for the manager. In addition, it is extended to form long-term partnership relations: the manager and contract will establish a partnership so that the manager will not need hold auctions for every jobs.

3.7 Developing ISC Framework from SCM

The ISC (Information Supply Chain) framework is inspired by ideas in supply chain management (SCM), which has been widely used in business management science [171]. This section explains how the idea of ISC is developed.

A supply chain can provide value-added services and fulfill its customers’ demands by the formation of a network of services/products providers. Those providers include a) suppliers that provide materials, b) manufactures that make products, and c) distributors that sell products to customers. Figure 3-20a shows a typical supply chain.
Similar to a material supply chain, an information supply chain (ISC) can provide value added services to information users and fulfill users’ information requirements by a network of information sharing agents that may include a) scanning agents that gather information and provide information to other agents, b) interpretation agents that analyze the information and make sense of it, and c) broker agents that collect users’ requirements and satisfy the requirements with proper information. Figure 3-20b shows an information supply chain.

A material supply chain has two primary targets: to balance demand and supply, and to improve efficiency and responsiveness [171]. These are also the primary goals of an information supply chain. So, creating an Information Supply Chain framework offers us an opportunity to look at the information sharing problem from a new perspective, and to better leverage the existing research efforts in the SCM framework to find new solutions to information sharing.

The ISC framework is developed from SCM from six aspects: goals, problems, concepts, methods, transaction models, and evaluation criteria (as shown in the left two columns of Figure 3-21). First, the ultimate goal of both ISC and SCM is to balance
demands and supplies. Unbalanced demands and supplies can lead to poor supply chain performances: either high cost due to over supplies or poor customer service due to stock outs. Information sharing has the same goal: unbalanced demand and supply can cause problems such as information overload due to supplying too much irrelevant information or information deficiency due to inefficient information investigations.

![Diagram](image.png)

**Figure 3-21**: Developing ISC from SCM.

Second, a rich set of concepts for ISC can be developed by finding a counterpart for each concept in SCM. For example, basic activities and objects (or entities) in SCM such as purchase, sales, product, supplier, customer, and warehouses correspond to those in ISC: query, inform/answer, information, supplier, requester, and knowledge-base respectively. Even some complex concepts in SCM have their counterpart in ISC. For example, bill of materials (BOM) lists the components needed to produce one unit of a product. Checking each component’s availability can reveal the material shortage for desired productions (Eq. 3.16)
Similarly, diagnosing the IDR can identify the missing information (Eq. 3.17).

\[
\text{Material shortage} = \text{Required Material} - \text{On hand Stocks} \quad 3.16
\]

\[
\text{Missing Information} = \text{Required Information} - \text{Known Information} \quad 3.17
\]

Furthermore, warehouses, machines, or vehicles have capacity limitations for material storage, production, or transportation. Information sharing agents have capacity limitations in a similar way: they have limited memory, time for investigation, reasoning power, or communication bandwidth. In addition, human users have more cognitive constraints than agents. People can only read very limited amount of information at a time, thus they can be easily overloaded by an overwhelming amount of information from many poorly designed systems.

Third, business models in SCM can be adapted to handle information sharing problems. For example, vendor managed inventory (VMI) is a business model that specifies vendors to manage their customers’ inventories. After a customer sets its demands over a period of time, the vendor monitors the customer’s stock and decides to refill when the stock level is low. It is an effective model that can reduce the workload of a company from inventory management and spend more time for customer service. Similarly, we can adopt the VMI model to share information by subscription, in which an information provider updates its subscribers about any new or changed information. By using a subscription, a user can save time on querying information and spend more time on processing information. We call the subscription model a counterpart of the VMI model.
Other than the VMI model, some business models that have no counterparts for current information sharing solutions can suggest new ways of sharing information. In Section 3.5, we introduce a new information sharing method called Just-in-time (JIT), which is developed on the basis of Just-in-time in SCM. We should point out that unlike goals or terms, developing business models from SCM to ISC requires a great deal of understanding on information sharing challenges, on differences between handling material and information, and on evaluation criteria. The examples such as subscription or JIT are introduced to inspire readers to pursue new information sharing approaches from an information supply chain perspective.

Finally, criteria such as fill-rate and total cost that are used to evaluate material supply chains can be used to evaluate information supply chains. In ISC, fill-rate is defined as the ratio between the total number of satisfied requirements and the total number of requirements. Fill-rates measure responsiveness— the more demands are fulfilled (the higher the fill-rate), the better the performance. Total cost measures efficiency by considering the total numbers of information seeking actions and communication. Fill-rate and total cost are often contradictory to each other. Over-supply can often yield a better fill-rate. However, over-supply can cost more and indicate inefficiency. Thus, setting a performance target for an ISC is often a trade-off decision: choosing a balanced point between high responsiveness and high efficiency.

The above analysis a) identified the information sharing problems such as information overload and information deficiency, b) set goals such as to balance demand and supply and to improve efficiency, c) defined terms and concepts such as information
customer, vendor, and IDR, d) adapted business models such as subscribe and JIT, and e) selected evaluation criteria such as fill-rate and cost.

### 3.7.1 ISC differs from SCM

It is worth to note that SCM differs from ISC in many ways. It is unwise to borrow everything from SCM to ISC in a stiff manner. SCM deals with the flow of materials, while ISC deals with the flow of information. When we borrow strategies and methods from SCM to ISC, the differences between material and information should be considered. First, quantity is used to measure material requirements. One material cannot fulfill demands from two requests. In contrast, one cannot use quantity to describe information. A piece of information can fulfill all demands for this kind of information, no matter how many requests are about it. The only cost increase associated with an increase in number of requests is the cost for transmitting the information to needers. Furthermore, processing activities for handling material and information are different. Finally, material handling includes ordering, producing, packing, loading, and shipping, whereas information processing includes query, observing, reasoning, and transforming. This difference leads to different challenges for SCM and ISC.

In spite of the differences, I believe that some high-level concepts, goals, methods, and the philosophy of SCM are useful for managing information requirements and improving information sharing results.
3.7.2 ISC Framework Unifies Existing Methods

The ISC framework serves as an information sharing platform regardless of complexity of information contents. The framework is general enough to manage various information sharing activities, from scanning and interpretation to information delivery. Many existing information sharing methods can be unified and incorporated into ISC by matching a counterpart method in the SCM framework. For example, FIPA “Query Interaction Protocol” [81] specifies how to handle a query between an initiator and a participant as shown in Figure 3-22a. We can find a counterpart process in SCM, such as PIP 3A1 (Request Quote as shown in Figure 3-22b) from Rosettanet [172-174] in which a seller can choose to confirm a request or refer to other suppliers if it cannot satisfy the request. It is easy to notice that the referral option is ignored in FIPA’s specification. We thus can extend the current query protocol to incorporate the choice of referring alternative suppliers as 3rd party inquiry confirmation (Figure 3-22c). The 3rd party order process and the 3rd party query process are shown in Figure 3-23. Similar to the query interaction protocol, many other information sharing protocols can be unified in ISC such as subscription, third-party subscribe” [99], and “ProInform” [27].
3.8 Conclusion

This chapter developed a conceptual framework that defines the basic concepts of information need, information seeking, and information partners in a task context. It also
gave information requirement planning (IRP) algorithm to anticipate, consolidate, and source the needed information. It suggested using information supply chains to organize and manage information sharing activities. Finally, a market-based approach has been proposed to form such information supply chains by using a procurement auction protocol.

The next Chapter will follow up the discussion about realizing the ISC framework as an agent architecture called R-CAST. Then, Chapter 5 reports results of experiments that are designed to evaluate the framework and the agent architecture.
Chapter 4

Realizing the ISC framework within R-CAST: an Agent Architecture

As introduced in Chapter 3, this research concerns questions of how to implement (1) cognitive models for identifying information needs, and (2) information management for coordinating information request and delivery so that those needs are met. Figure 4-1 shows the schematic on how to accomplish the above goal using intelligent agent technologies. The core technology requires an agent architecture that incorporates cognitive models with an integrated information management system.

![Diagram](image)

Figure 4-1: Using an agent to model and support a cognitive task.

Unfortunately, agents are difficult to build because, according to the definition of agent, some minimum and unique requirements must be met before a software system can be called an agent [86, 175]. According to the “weak” agent notion, an agent must be autonomous, social, reactive, and proactive [53, 56, 62]. To be autonomous, an agent must be able to store and interpret knowledge. To be social, an agent must be able to
communicate and understand meanings within contexts. To be reactive, an agent must be aware of its environment including the past and current situations. To be proactive, an agent must be able to anticipate and fulfill the requirements of its team and of its own.

The notion of a “strong” agent requires even more human-like attributes: agents are seen as conscious and cognitive entities that have feeling, perception and emotion [100, 101].

Implementing some or all of the above requirements demands a great deal of expertise in a wide range of areas such as rule-based systems, theory of perception, knowledge representation and engineering, communication protocols, and decision-making. To reduce engineering efforts, the first step in building an agent is to design an agent architecture which provides domain- (or application-) independent mechanisms and structures. After this step, a set of domain-specific agent knowledge is then encoded and integrated with the agent architecture to form an application-specific agent model.

This chapter begins by reviewing the existing agent technologies relevant to this work. Soar is an agent model or architecture based on the “unified theory of cognition,” which claims that all intelligent mechanisms can be represented as productions [49, 50]. Soar maintains the information in I/O links, which are implemented as a tree structure. The usage of information is represented in LHS (Left Handed Side) productions either as knowledge or as state values in problem spaces. Soar creates impasses when it cannot propose any production to fire or have multiple candidates to fire. Soar adopts both declarative knowledge (facts) and procedure knowledge (rules). Act-R is another popular production system, which may be classified as a strong agent with some important psychological constraints [51]. However, both Soar and Act-R do not have information management capabilities, and they especially lack communication functions.
It will become clear later that information management functions are crucial for the thesis work described here. They can be realized using existing agent design tools such as JADE (Java Agent Development Environment) [175] and JACK [176]. JADE is a FIPA [81] compliant agent platform, providing an agent development environment and FIPA-ACL communication functions. JACK [176], an agent building tool based on the BDI framework [66, 67], also provides communication functions through an extension geared toward team oriented programming. Strictly speaking, both JADE and JACK are design tools for specifying and creating agents: they provide great graphical interfaces but very little functionalities for effective modeling of tasks that are needed in a typical agent design. ABLE (Agent Building and Learning Environment) is a component-based platform that does include some intelligent components such as data beans, learning beans, and rule beans [177]. These components, however, are computational functions not suitable for modeling high-level decision-making processes.

In summary, in the beginning of this research, no cognitive architectures and agent design tools were available that provide functionalities for both high-level cognitive modeling and information requirement management. Thus, there need to be a new architecture built for this research.

The result of the building efforts is RPD-enabled Collaborative Agents Simulating Teamwork (R-CAST), an agent architecture designed to a) model high level collaborations and decision-making processes [19, 178], and b) manage information that is crucial for the making processes. The design of R-CAST adopts an integration of loosely coupled intelligence components including a forward chaining system, a plan execution module, and agent communication management.
R-CAST can reduce the agent design efforts with a diverse set of knowledge representation syntaxes, which encourages designers to choose the most direct knowledge representation. This makes R-CAST different from many other agent architectures that choose simple and generic syntax sets. For example, Soar uses productions as the single knowledge representation. Consequently, some commonly needed mechanisms, such as communication, must be implemented as a large amount of productions. In contrast, R-CAST models hide the details at the architecture level, and designers can realize agent communication with little implementation of knowledge representation. Following the concept of distributed knowledge representation, R-CAST also interprets knowledge with multiple components. In short, R-CAST can be viewed as a framework within which intelligent components can be independently applied according to system requirements. This design concept allows the architecture to scale-up by “simply” adding new components.

The rest of this chapter will discuss the R-CAST architecture with highlights of design features, then describe each R-CAST component in detail, and finally give a reflection on lessons learned. Low-level implementations, such as class diagrams and sequential diagrams, represented using the unified modeling language (UML), are shown in Appendix D.

4.1 The R-CAST Architecture

This section describes the R-CAST architecture from framework, cognition, integration, control, and interface perspectives. The main objective is to provide a
detailed description on design principles, features, and functionalities but not too much detail on the usages as seen in a typical design documentation or software manual.

Figure 4-2 illustrates the basic structure or composition of an R-CAST agent and its interactions with human (users), other agents, and the problem domains.

Figure 4-2: R-CAST agent and its environment.

Each agent is built within a framework that contains intelligent components such as knowledge base and decision-maker. The component integration or interaction is achieved through a whiteboard where requests needed for proper functioning of each component are posted. An agent normally resides in some environments. First, the environment can be a problem domain, manifested either as a real world situation or as a domain simulator. Second, the agent can interact (communicate, collaborate, and negotiate) with other agents in the same problem domain. Finally, the agent can interact with human users through a graphical interface or a command shell window.
4.1.1 Framework

R-CAST realizes its flexibility through a whiteboard structure, which connects components. Loaded with proper knowledge specifications, each component can form a part of the overall agent team. A component can either yield behaviors such as actions or generate requests for other components. Figure 4-3 shows the framework and its relationships with components.

Figure 4-3: R-CAST architectural framework.

The whiteboard contains a list of requests posted by connected components. Each request specifies an instruction and its associated parameters appropriate to that instruction. For example, an information manager can post a request to the whiteboard for sending a message to another agent. The “sending message” request contains information such as message type, content, and the recipients.
After a message or request is posted on the whiteboard, it can be read and responded to by other components. Since the requestor component does not need to specify which components should respond to the request, any other components can read or respond according to their capabilities. If a component knows how to respond to the request, then the responder component will update the request status from “new” to “responded.” In the above example, a communication manager has the capability required for sending a message. Therefore, it will respond to the message by creating a message object, looking up the address of the recipients first, and then sending the message as instructed, and finally marking the request as “finished.”

From each component’s perspective, it doesn’t need to maintain any interaction with other components. It only needs a list of pre-defined request types that it should respond to. This flexible structure allows designers to select only the necessary components to construct an agent model. For example, a specific agent model may exclude the communication manager because the agent has very little need for communication. Without the whiteboard infrastructure, a component needs to maintain connections to many other components. As a result, components depend on each other or are tightly-coupled, losing the flexibility to exclude unnecessary components or to add new components.

4.1.2 Realizing Decision-making Process Model in R-CAST

In this research, cognition can be analyzed from a decision-making perspective: how to make and implement decisions. In order to focus on high-level behaviors, many of
low-level behaviors are simplified. For example, R-CAST does not have fully implemented physical or psychological mechanisms [51]. Cognitive processes are realized with four components: a reasoning engine, an RPD decision model, a task manager, and a process manager. Figure 4-4 shows the cognition of an R-CAST agent, which is composed of a sequence of basic decisions made by several key components.

Figure 4-4: R-CAST cognition.

1) A reasoning engine interprets perceived information according to a set of rules. It makes conditional decisions such as “what is the current situation.” The result can form input to the follow-up decisions.

2) An RPD decision model decides “what to do” by comparing the current situation and the agent experiences: if a solution worked for a similar situation in the past, it is likely to work in the current situation [20]. The result is a course of actions (COA).

Most component diagrams are color coded. Shapes with the same color usually belong to a common component.
3) Once the agent decides “what to do,” it will pass the decision to a task manager. The task manager will assign the task to appropriate agents based on agents’ availabilities, capabilities, and other situational constraints. The task manager also schedules when to carry out the assignments according to task dependencies and agents’ availabilities. In other words, the task manager makes resource decisions: “Who should do what?” and “At what time?”

4) Last, when the scheduled time arrives, a process manager will execute the assigned tasks according to plans. A plan is a formal procedure for performing a task. Executing a plan requires conditional evaluations and action decisions. The process manager makes procedural decisions: “How to do it?”

Along the decision process there can be multiple decision points. Each decision point requires information, which may remain unknown when it is needed. The goal of this research is to find out how information is used in task decisions and how to satisfy information requirements.

### 4.1.3 Anticipate Information Needs in R-CAST

Figure 4-5 illustrates relations between components for cognition (top layer) and components for information management (bottom layer). Information management involves two steps: demand management and supply management.

Demand management identifies information needs along with its needers, information type, and time. The task manager can provide answers to “Who needs the
information?" and “When is it needed?” An information need is linked to a task instance so that when the task context is changed, the need will be updated accordingly.

Supply management organizes information requirements and information seeking operations. First, the information manager (IM) will consolidate the information needs into a set of non-redundant information requirements. Then, the IM creates an information seeking plan. Next, the IM launches the seeking actions according to the plan. Once the information is available, the IM will pass the information to the needer.

Table 4-1 summarizes the information needs in R-CAST. Information needs must specify what is needed, who need it, and when it is needed. Who need it and when it is needed can be determined according to task assignments and schedules. What is needed or the purposes of needed information can be categorized according to different types of decision-making. The RPD decision-maker needs information for recognition and
expectancy monitor. Plan execution and process management need to use information to evaluate conditions such as preconditions, preference conditions, termination conditions, and failure conditions. Task assignment decisions need information to evaluate run-time constrains. Partial or indirect information needs can be identified by the knowledge base. Finally, the information manager needs meta-information, which defines “information about information,” such as duration, credibility, and source, etc.

Table 4-1: Anticipate Information Needs in R-CAST.

<table>
<thead>
<tr>
<th>Components</th>
<th>What is needed?</th>
<th>Who needs it?</th>
<th>When it is needed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPD decision-maker</td>
<td>Expectancy</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Anomaly</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Cue</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Process manager</td>
<td>Termination condition</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Failure condition</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Preference condition</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Precondition</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Task manager</td>
<td>Constraint*</td>
<td>According to assignment</td>
<td>According to schedule</td>
</tr>
<tr>
<td>Knowledge base</td>
<td>Partial information need</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Information manager</td>
<td>Meta information*</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4.1.4 An Integration Perspective

From the framework perspective, components appear to be independent of each other. From the function perspective, however, components are connected in one way or another. Figure 4-6 shows an overview of how components are integrated with others. A link between two components only indicates a functional integration, and it does not mean the link is essential in the system. For example, the information manger integrates
with the task manager. However, one can build an agent with only one of the two components.

The domain adapter is a special component that connects an agent to its environment. Being autonomous is the most critical property of an agent, and the agent must connect to its environment, sense the environment, and act upon it. The domain adapter provides sensing and acting capabilities. Because agents are built to work in vastly different domain environments, the design of an agent architecture must be independent of any specific domain environment. For example, playing chess requires different sensing and acting functions from those required by playing poker. By extending the basic domain adapter functions, designers can apply R-CAST to many problem domains, some of which have been realized as combat simulations [179-181] or anti-terrorism analysis [18, 19].
4.1.5 Control and Interface

The R-CAST architecture provides a configurable interface for agent monitor and control. The basic graphic user interface (GUI) contains three parts: cycle speed controller, command shell window, and component monitor. Figure 4-7 shows a typical R-CAST interface.
The cycle speed controller can start, stop, or adjust the speed of an agent. The shell command window allows human users to issue commands to an agent. An agent has a set of system level commands. Each component within an agent architecture has its own command set. When a user enters a command, all components will try to handle the command in turn. If a component understands the command, it will execute the command and return the result. For example, a user can assign a new task by the command “assign.” There are over 30 commands defined in R-CAST, which are summarized in Appendix C.

A large portion of the window is used to monitor a component. Because each component is unique in knowledge representation, functionalities, and behavior, a unique monitor is designed for each component. Users can monitor the agent status which is updated dynamically. Each monitor window provides visualization of an agent’s mental
status in the form of tables, trees, or graphs. A user can also decide to display or hide a component monitor by setting the agent configurations.

4.2 The R-CAST Components

R-CAST components fulfill two purposes: cognitive modeling and information management. Components for cognitive modeling include 1) a knowledge-base for beliefs and logical reasoning, 2) a naturalistic decision-making model, 3) a process manager for procedural operation, and 4) a task manager for task assignment and scheduling. Components for information sharing include 1) an information requirement manager for information requirement planning and information seeking, 2) a communication manager for message exchange and conversation management, and 3) an auctioneer for efficient information provider discovery.

This section introduces the features, design, knowledge syntax, interface functions, and main configuration options of each component. Some components may have a great deal of features and functions that are common in similar software systems. For example, the knowledge base has many functions similar to a typical knowledge base. However, there are also functions that are unique in R-CAST. The introduction focuses on these unique functions and briefly explains typical ones.
4.2.1 Active Knowledge base

Knowledge base provides basic knowledge repository and reasoning capabilities for other components such as evaluating conditions for process manager, test constraints for task manager, and appraising cues for RPD decision module. In addition, the knowledge base contains a key capability to fuse information and to diagnose any missing information.

4.2.1.1 AKB Key Features

Active knowledge base (AKB) is a forward chaining rule-based system. Like a typical forward chain system, AKB uses inference rules to generate new facts from available ones. In addition, AKB uses the rules to construct the dependency relationship among the information types (IDR). At runtime, the rules will remain the same, whereas facts are changing to reflect the agent’s changing beliefs about its environment and about itself. AKB groups facts in three categories: 1) constant facts: facts that will not change over time, 2) volatile facts: facts that have limited validation time, and 3) implied facts: facts that have been implied from other facts by using the inference rules.

AKB has three unique features. First, facts can be time sensitive because they can become obsolete as time proceeds. AKB has a built-in clock that keeps track of volatile facts so that when the time expires, the volatile fact will be retracted. This feature can also be realized by adding second order rules that handle the time value for each fact or by linking the facts to an external time tracking system.
Second, implied facts are linked to their evidences or sets of facts that match all the antecedents of the rule that generate the facts. An implied fact may contain multiple sets of evidence. When a fact in an evidence set is retracted, all evident sets that contain that fact will be retracted. When all evidences of an implied fact are retracted, the implied fact will be retracted. Keeping the links between implied facts and their evidences is important in information link analysis. This may be viewed as realization of information dependency relations (IDR) at the instance level. If the number of facts is high, however, the memory cost for keeping the evidence links will be high. This problem can be solved by implementing more efficient inference algorithms such as the Rete algorithm [182].

The third feature is that AKB has fundamental information management functions including diagnosing missing information and maintaining information partners. ABK uses a backward chaining mechanism to diagnose missing information. Partners (providers or needers) are defined as names and value pairs. The name indicates the partner’s name. In order to communicate with partners, an agent has to include a communication manager and a directory that specifies the address and protocol of the partner. The value part, a numerical value ranging from 0 to 1, indicates the degree of relationship. A low value can suggest the provider is not desirable, e.g., less reliable or more expensive, and a high value indicates that the provider is a better choice. The value for a needer can be used to represent any property that the needer has. For example, it can be used to represent the likelihood that a piece of information is needed by the needer.

Figure 4-8 shows the AKB interface in R-CAST which allows an agent designer to monitor and interact with the agent’s knowledge base. For example, one can select a fact type and lists the all the facts and their current status such as where they are obtained.
and when they will expire. It visualizes the information dependency relationships at both fact type and fact instance levels. In addition, a designer can check and update the agent knowledge using basic AKB commands such as query, assert, and retract.

4.2.1.2 AKB Syntax

Syntax in R-CAST adopts an informal notation: <must> is a required element; [optional] are optional; multiple* can appear zero or more times; and x|y indicates x or y. Appendix B summarizes all the syntaxes used by R-CAST agents. Table 4-2 gives a list of syntaxes for knowledge definition in AKB.
Similar to other forward chaining inference systems, AKB rules have two parts: antecedents or left handed side (LHS) and consequence or right handed side (RHS). It can be read as a “if … then…” statement. LHS can contain match patterns, logical comparisons, and arithmetic functions. Match patterns will match with known facts. For example (money ?person ?money) will match a fact (money Alice 201.0): ?person = Alice and ?money =201.0. All antecedents in a rule have conjunctive connections. Disjunctives must be realized by adding an additional rule. The consequence or RHS contains a single assertion. This differs from popular production systems such as Soar [49, 50] or Jess [183], which also allows operators in RHS. An AKB makes decisions about situation
assessment not operational decisions. In R-CAST, operational decisions are made by the process manager.

Rules are used to represent relations between fact types. Once fired, a rule can generate an implied fact on the basis of a set of facts called “evidences,” which are preserved for each implied fact. This is very useful when information link-analysis is needed — the more evidences, the more credible the information. In a rule, the information dependency relationship (IDR) between the consequence predicates and the rest (antecedent predicates) forms a dependency relationship for the information type. An agent can interpret each piece of new information with rules that contain predicates that match the information.

Fact types can be specified by a frame structure and associated properties include a moving-average time duration, a template for converting a fact to natural language, a source that specifies actions that can observe or measure the information, and information partners. The time of a volatile fact is set in the fact definition. The time value in fact type only suggests a moving average validation time of the type. When a new volatile fact is asserted, the time value of the type will be updated accordingly. Keeping an average validation time value is useful for information requirement planning, because the information has to be valid at the time when it is used. To simplify the model implementations, AKB uses integers to represent time values.

 Templating is a simple way to map the knowledge represented in the AKB to a human understandable sentence in natural languages: e.g., a fact (rich Alice) can be translated into, “Alice has a lot of money.” With the template mechanism, R-CAST agents can share information with human users in a predefined sentence structure. A
sentence may be translated and also be pronounced with a speech synthesizer [184] that can produce voice for human users. However, sentence translation and pronunciation belong to the discipline of natural language processing, and they are not the focus of this work.

4.2.1.3 AKB Interface Functions

AKB has four basic functions that are essential for an inference system: parse, assert, retract, and query. Table 4-3 shows the list of interface functions.

<table>
<thead>
<tr>
<th>Function/command</th>
<th>Parameters</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Condition set</td>
<td>Bindings</td>
<td>Query (&quot;((rich ?person))&quot;)</td>
</tr>
<tr>
<td>queryif</td>
<td>Condition set</td>
<td>Boolean</td>
<td>Queryif (&quot;((rich Alice))&quot;)</td>
</tr>
<tr>
<td>assert</td>
<td>fact</td>
<td>Null</td>
<td>Assert(fact1)</td>
</tr>
<tr>
<td>retract</td>
<td>fact</td>
<td>Null</td>
<td>Retract(fact1)</td>
</tr>
<tr>
<td>diagnose</td>
<td>hypothesis</td>
<td>Condition set</td>
<td>Diagnose(&quot;((rich Alice))&quot;)</td>
</tr>
<tr>
<td>naturalAssert</td>
<td>sentence</td>
<td>Null</td>
<td>natrualAssert(&quot;Alice has a lot of money&quot;)</td>
</tr>
<tr>
<td>naturalQuery</td>
<td>Condition set</td>
<td>Sentence</td>
<td>natrualQuery(&quot;((rich Alice))&quot;)</td>
</tr>
<tr>
<td>parse</td>
<td>Knowledge definition</td>
<td>Null</td>
<td>parse(&quot;(Rule &quot;if a person is rich&quot; (money ?person ?money) (&gt;= ?money 100) -&gt; (rich ?person))&quot;)</td>
</tr>
<tr>
<td>clone</td>
<td>none</td>
<td>AKB</td>
<td>Clone()</td>
</tr>
</tbody>
</table>

AKB has a unique function for information management: it can use backward chaining to diagnose the missing information according to an appropriate IDR. Diagnose starts with a list of goals (or a hypothesis) and works backwards to see if there are facts available that will support any of these goals. This feature is important for the situation when high-level information is not easy to observe but possible to fuse by collecting
sufficient evidences. AKB also provides two functions for interfacing with natural languages: “naturalAssert” will parse a sentence into a fact according to the template definition; “naturalQuery” will respond to a query in a sentence. This simple design provides information sharing agents with a basic capability to handle information that is used by human users. Currently, due to the complexity of forming a question, AKB has no function that can accept a query in natural language.

4.2.2 Process Manager

Processes or predefined plans are used to model procedures for operation. The process manager stores the templates of plans and executes a plan after instantiating it.

4.2.2.1 Process Manager Key Features

A plan contains preconditions, effects, termination conditions, fail conditions, a contingency plan, and a process body. Upon being requested (by a decision-making module, a task manager, or an information manager) the process manager can generate plan instances from appropriate templates. An agent may run multiple plan instances simultaneously or in a sequence. Each process instance can be in one of five states: active, suspended, waiting, failed, or terminated. Figure 4-9 shows the state transitions.
The execution of plan instances takes into account the constraints associated with the instances and the current KB state. Table 4-4 illustrates an example process that specifies how to count numbers from an initial value to a target value.

Table 4-4: An Example of Process That Count Numbers.

```
(plan plan_count_from_to(?from ?to)
 (precondition (current_number ?number))
 (failcondition (> ?number ?to))
 (contingency (plan_count_back_to ?number ?to))
 (termcondition (current_number ?to))
 (process
   (plan_count)
   (choice big_or_small
     ((precondition (> ?number 7.0))(print "big"))
     ((precondition (< ?number 3.0))(print "small"))
     (default)(print "medium"))
   )
 )
```

Figure 4-9: Process state transitions.
A process manager has two roles: to model procedure activities and the needed information for the activities. Procedure activities can be modeled as plans, which may be either for performing tasks or for seeking information. For example, moving from one place to another, pursuing and attacking a target, and controlling a system are task performing activities. Measurements, scout operations, and inquiries to a search engine or a database system are information seeking activities.

A task performer often finds that a provided information is irrelevant and not actionable [185]. One can overcome this issue by directly capturing an information requirement as a part of an operational procedure. Information that is required for operational decision points is modeled as preconditions, termination conditions, fail conditions, or preference conditions. Thus, information useful for evaluating these conditions is relevant to the operation. In order to be relevant and actionable, agents should only share information that fits these requirements with the right type and at the right time.

Figure 4-10 shows the PM interface. Each instantiated process will be added to the monitor as a new tab, captioned by the process identification. A process monitor window shows the dynamics of a process and its sub-processes with a tree-table structure. The first column displays the process identification and process relations. The second column shows the plan name. The third one gives the current state. The last column keeps records of applied operators.
4.2.2.2 Process Knowledge Syntax and Characteristic

Process knowledge is specified by operators and plans. Table 4-5 lists the syntax of process knowledge specification. The term “plan” or process knowledge is used to describe the prototype of a process. A process is executed once in a decision cycle. Other possible actions on a process are controlled by the current state of the process and the state transition diagram shown in Figure 4-11.
An operator is the most basic action that an agent can perform in a domain. When an operator is applied, the domain status and an agent’s belief can be updated. The agent can update its belief according to the specified effects of the operator. For example, the new position is an effect of a move operator. In contrast to an operator, a process consists of a sequence of operators. For complex plans, a process can also contain sub-processes. This allows a designer, using a hierarchical structure, to represent complex processes. A

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition set</td>
<td>(&lt;condition&gt;)*</td>
<td>(next_number ?number ?next)</td>
</tr>
<tr>
<td>Precondition</td>
<td>(precondition &lt;condition set&gt;)</td>
<td>(precondition (next_number ?number ?next))</td>
</tr>
<tr>
<td>Effect</td>
<td>(effect &lt;condition set&gt;)</td>
<td>(effect (current_number ?next))</td>
</tr>
<tr>
<td>Fail condition</td>
<td>(failcondition &lt;condition set&gt;)</td>
<td>(failcondition (&lt; ?to ?from))</td>
</tr>
<tr>
<td>Termination condition</td>
<td>(termcondition &lt;condition set&gt;)</td>
<td>(termcondition (current_number ?to))</td>
</tr>
<tr>
<td>Preference condition</td>
<td>(prefcondition &lt;condition set&gt;)</td>
<td>(prefcondition (&gt; ?number 7))</td>
</tr>
<tr>
<td>Choice</td>
<td>(choice &lt;choice name&gt; (&lt;Preference condition&gt; &lt;step&gt;)* )</td>
<td>(choice big_small ((precondition (&gt; ?number 7)) (print big))</td>
</tr>
<tr>
<td></td>
<td>((default) &lt;step&gt;)</td>
<td>((precondition (&lt; ?number 3)) (print small))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>((default) (print normal))</td>
</tr>
<tr>
<td>Step</td>
<td>(&lt;operator</td>
<td>plan name&gt; [&lt;argument&gt;]* )</td>
</tr>
<tr>
<td>Process</td>
<td>(process &lt;step&gt;*)</td>
<td>(process (plan_count_current))</td>
</tr>
<tr>
<td>Contingency plan</td>
<td>(contingency (&lt;plan name&gt;))</td>
<td>(contingency (plan_countback))</td>
</tr>
<tr>
<td>Operator</td>
<td>(operator &lt;operator name&gt; [argument]* [Precondition] [Effect] )</td>
<td>(operator count (?number)</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>(precondition (next_number ?number ?next))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>(effect (not (current_number ?number)))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>(current_number ?next)</td>
</tr>
<tr>
<td>Plan</td>
<td>(plan &lt;plan name&gt; [(&lt;argument&gt;*)] [Precondition] [Effect]</td>
<td>(plan plan_count_from_to (?from ?to)</td>
</tr>
<tr>
<td></td>
<td>[Termination condition] [Fail condition]</td>
<td>(failure (&lt; ?to ?from))</td>
</tr>
<tr>
<td></td>
<td>[Contingency plan]</td>
<td>(contingency (plan_countback))</td>
</tr>
<tr>
<td></td>
<td>&lt;Process&gt;</td>
<td>(termcondition (current_number ?to))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>(process (plan_count_current))</td>
</tr>
</tbody>
</table>

| Table 4-5: Process Knowledge Syntax. |
process can also have effects on agents’ beliefs. In a process, operators or sub-processes are called steps. Which step an agent should execute is determined by conditions.

There are four types of conditions in a process definition: preconditions, termination conditions, fail conditions, and preference conditions. At execution time, a process manager requests an inference engine to check these conditions, and uses the results to step forward a plan. The four types have different meanings and interpretations.

A precondition is a set of conjunctive conditions that are used to guard the execution of a process or an operator. It is checked before the attempt to execute an action. If the precondition is satisfied, the agent can carry out the action. Otherwise, the agent must wait. For example, a fire engine must be in position before it can start to extinguish a fire. If the agent lacks sufficient information to evaluate the preconditions, the agent can generate an information need and actively seek the needed information.

Other types of conditions, such as termination conditions, fail conditions, and preference conditions, can also become informational needs.

A termination condition is a set of disjunctive conditions that represents the situations when the goal is achieved or becomes irrelevant [71, 72]. Whenever a termination condition is satisfied, the process that contains it will terminate immediately. “A fire is extinguished” is a condition that tells firefighters to terminate the current extinguishing operation. Additionally, a termination condition can represent a condition that tells the current goal is irrelevant such as when the firefighters learned that the fire is set on purpose and does not need to be put out. If the termination conditions are not satisfied, the process will continue until the final step. Then, depending on configuration, an agent can repeat the process or terminate it.
Fail conditions are used to detect and handle operational failures. A fail condition is a set of disjunctive conditions that represent the conditions when the goal is not achievable. When a failure is detected, an agent will abandon the current goal and instantiate a contingency plan. An operational failure may require remedial actions to fix the errors caused either by a wrong action or by unfavorable conditions. Such remedial actions are called contingencies, which are based on contingency plans, a part of R-CAST plans. For example, a fire is out of control. If so, a contingency plan, such as “to evacuate,” should be instantiated.

Failure conditions and contingencies are the main avenues of failure detection and handling mechanisms in R-CAST. If the contingency plan defined for a process sees a fail-condition, the contingency will be instantiated. Otherwise, the fail status will be thrown to its parent process, and, recursively, the parent process should handle the fail status. If the parent process has a contingency, it should execute the contingency, otherwise it will throw the error further. In Figure 4-11, the failure condition that is detected in Process 3 will trigger the contingency plan that is defined in Process 1.

Figure 4-11: Enacting contingencies.

Preference conditions are used to make decisions in a choice node. Each alternative has a preference condition except the default action, which will be selected if
none of the preference conditions are satisfied. The preference conditions are non-exclusive. When multiple preference conditions are satisfied, depending on configurations, an agent will randomly pick one among them or pick the first one.

Termination conditions and fail conditions will be checked in every time cycle. The precondition will be checked just before the operator or the process is executed. Once the process is started, its precondition will not be checked any more. Preference conditions will only be checked at the time of making a choice. Any situation changes after it will not affect the choice anymore. This differs from RPD decisions, in which an agent will constantly monitor anomalies and adjust its decisions if necessary.

Information that is relevant to evaluate preconditions, termination conditions, fail conditions, and preference conditions is essential for successful missions. Thus, anticipating the requirements and proactively sending the relevant information to the mission operators are always desirable.

4.2.2.3 Process Manager Interface Functions

A process manager provides a process execution function that can be used by other components such as a task manager, an information manager, or an RPD decision-making model. The function parameters include a plan name and plan arguments. Upon requests, the process manager will append the execution request to its execution queue. In general, the requested processes are executed in turn unless the agent is configured to allow parallel process execution. When the turn arrives, the process manager will create a
process instance according to the plan prototype and the calling arguments. The instance will be executed by setting the state to activate.

In addition to execution, a process manager can also simulate a process under the current situation for decision support. This is seen in human decision-making models. When the simulation is complete, the agent will compare the result with the desired goal to decide whether or not the plan was workable.

4.2.3 RPD Decision-maker

RPD is one of naturalistic decision models that are motivated to explain expert decision-making [20]. Unlike classical decision model theories, RPD focuses on decision process and situation assessment rather than evaluating options.

Based on the RPD model, a computational RPD model is designed. The design reflects the main steps and philosophy of RPD but simplifies the functions that are difficult to model. For example, feature-matching and story-building are two diagnostic strategies for situation awareness. In case feature-matching cannot provide an adequate picture due to lack of information or experience, story-building is adopted to construct a story (i.e., a causal sequence of events) that links pieces of observed and available information into a coherent form. The story provides an explanation of how the current situation might have been emerging. However, due to the complexity of story building, the computational RPD model simply relies on the feature matching for assessing the situation and realizing recognition. For the same reason, the modification of COA is not
included in the current design. Figure 4-12 shows the process of computational RPD model in the R-CAST architecture.

![Diagram of Computational RPD model]

Figure 4-12: Computational RPD model.

The model starts from a situation analysis based on the agent’s knowledge base. The agent compares the results with experiences that are captured in an experience base. Then the agent evaluates the COA that was used in the past by simulating the effects in the process manager, which will 1) make a copy of current knowledge base, 2) execute the course of actions by checking the conditions and applying the effects in the new knowledge base copy, and 3) evaluate if the desired goals are achieved. The mental simulation provides a simple way to improve soundness of a decision. If the COA works (the desired goal is achieved), the agent creates a new task to implement it. Meanwhile,
after recognition, the agent starts to monitor the expectancies. If anomalies are detected, the agent will investigate more information and launch another recognition cycle. Finally, the computational model retains the new experience after the execution of the COA is complete.

4.2.3.1 RPD Model design

In RPD component, a decision instance encompasses five stages: recognition, investigation, evaluation, implementation, and experience retaining. According to the computational model’s decision process, the state transition of the decision instance follows Figure 4-13.

![Figure 4-13: RPD state transitions.](image)

The decision-making is a process of finding an experience that is similar to the current situation. In dynamic situations, agents use hierarchical experiences to recognize experiences and to suggest actions. As shown in Figure 4-14, lower level experiences refine higher level experiences. According to the evaluation of relevant cues, agents make recognition by switching current attention from a higher-level experience space to a lower lever one. Agents find a concrete experience, when the attention moves to the bottom. When new events are perceived as expected, the recognition is enforced; when
anomalies are detected, the previous recognition is questioned and even revoked. This mechanism can help an agent realize its decision mistakes and make immediate corrections. This process can also be used to suggest what information to pass to teammates based on an understanding of what the teammates are doing.

Figure 4-14: Identifying experiences in R-CAST through RPD.

The computational RPD model is also a collaborative model. An RPD-agent teams up with other RPD-agents and interacts with them. As far as a specific decision is concerned, if one RPD-agent who has the required expertise is designated as the final decision-maker, all other RPD agents will be supporters (Of course, a supporter may be the final decision-maker in other decisions). Each agent can project its experience spaces to a decision space. The decision space defines the agent’s expertise at a very abstract level without including any concrete experiences. Then the decision space is shared with other agents. As a team decision space, the shared decision space covers more situations. Figure 4-15 shows a collaborative decision space that is formed by four RPD agents.
Agents collaborate with each other in making decisions by exploring the shared decision space and in assessing the situation by information sharing. The team will select the final decision-maker based on the current situation and (or) agents’ expertise. As the situation changes or as the missing information is made available, the team may change the final decision-maker from one agent to another. Effective change of the commander-in-charge is needed for the best decision-making quality.

Figure 4-16 shows an RPD interface, which includes two sections. The top section shows a state monitor, which indicates the current state for decision-making, i.e., one of the following five states: investigation, recognition, evaluation, implementation, or new experience retaining. The bottom section is a decision space monitor, which uses a Radial Graph [186, 187] to show the structure of the experience space and the current attention.
In R-CAST, recognition primed decision-making has also been organically integrated with other types of cognitive processes. First, RPD is integrated with a process manager. The process manager can use a “mental simulation” to evaluate a course of actions (COA) that are executed after the decision-maker recognizes the current situation. Then if the simulation result can “prove” the COA to be workable, the process manager will implement it. Second, RPD is integrated with a task manager. Complex decision-making tasks often require a team of experts. A task manager can determine who should do what after the decision-maker decided what to do. Third, R-CAST provides a rich set of functions for information management. It can be used at the investigation phase in the RPD process. Finally, RPD is integrated with a knowledge base, which offers logical

Figure 4-16: RPD interface.
rules and reasoning capabilities for flexible representation of conditional cues and expectancies.

4.2.3.2 Experience Knowledge Syntax

Experience knowledge is captured in a tree structure, which has branches as experience space and leaf nodes as concrete experiences. Figure 4-14 gives an example of a typical tree. The structure of the experience tree can be flat or hierarchical. In a flat tree structure, the depth of the tree spaces is small and breadth is large. It represents unorganized experiences, and recognition relies on retrieving, which is typically seen in case-based reasoning [38]. In contrast, a hierarchical tree is more structured. The depth of the spaces is large, and the breadth is small. It represents well organized experiences. The recognition relies on the index of experiences, and is similar to making decisions with a decision tree. Table 4-6 lists the syntax of the experience knowledge specification language.
Cues, expectancies, anomalies are “fuzzy” conditions. The evaluation of these conditions sets results in a degree of match value, which equals the ratio between the number of the matched cues and the number of the conditions in the set. The more conditions are satisfied, the higher degree the match value has. In current implementation, the conditions are equally weighed. A more refined design should allow conditions to be weighed non-uniformly.

Goals are disjunctive conditions. When a decision-maker evaluates a plan with mental simulation, it will replace the plan’s termination conditions. If the goals are satisfied during or at the end of simulation, the candidate plan is considered to be workable.

Results of the experience, success or failure, are equally important for making a decision. If COA in the experience was a success, it can be used and applied to the current situation. By contrast, the decision-maker should avoid choosing COAs that
failed in similar situations. Sometimes, a COA can be both successful and a failure in two matched experiences. This indicates an ambiguity in experiences, and can be caused by over generalization when adding experiences. Such experiences should be resolved by making the cues more specific.

Cues, expectancies, anomalies, and plausible goals are conditions that require information. If a condition cannot be evaluated due to insufficient information, it will become an information need. Information for feature matching at the recognition stage is defined in cues. Information for expectancy monitoring is defined as expectancies and anomalies. In a collaborative decision-making process, team members can anticipate others’ information needs based on the current situations and a shared mental model that is established, based on the shared decision space [180].

4.2.4 Task manager

A task manager is responsible for assigning and scheduling tasks according to the tasks’ resource and capability requirements. There are three types of tasks: process execution, (RPD) decision-making, and information seeking. Task descriptions are abstract in that they don’t have any specific knowledge on how to do the task. The details are handled by other components according to their roles. A process execution task is detailed as process knowledge in a process manager; a decision-making task is detailed as experiences captured in RPD module, and information seeking task is handled by an information manager. The task manager is not responsible for executing the tasks, but for coordinating, assigning, and scheduling the task. It makes decisions about who will do
what and at what time. When the scheduled time arrives, the task manager will notify the proper component such as a process manager or a decision-making module to launch the task.

Figure 4-17 shows the interfaces for a task manager. The window on the left shows a task view, which lists a task in a tree-table. It illustrates the task structure and visualizes the assignment state such as scheduling, in process, or over-due. The window on the right show all the resources, their capabilities, and current assignments. By selecting a resource in a row, users can monitor all the assignments and their real-time status.

A task model specifies an agent’s capabilities, groups, roles, and task decompositions. The goal is to provide sufficient knowledge to assign and schedule tasks.
Table 4-7 lists the syntax for task specifications, which can be grouped into resource specification and task requirement specification.

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition set</td>
<td>(&lt;condition&gt;)*</td>
<td>(close ?person ?place)</td>
</tr>
<tr>
<td>Capability</td>
<td>(capability (&lt;capability&gt;*))</td>
<td>(capability (plan_send_gift))</td>
</tr>
<tr>
<td>Agent</td>
<td>(agent &lt;agent name&gt; &lt;capability&gt;</td>
<td>(agent agent1 (capability (plan_send_gift))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>Group/team</td>
<td>(group</td>
<td>team &lt;group</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>(role &lt;role name&gt; &lt;capability&gt;</td>
<td>(role purchaser (capability (plan_buy_gift))</td>
</tr>
<tr>
<td></td>
<td>[([constraint &lt;Condition set&gt;]</td>
<td></td>
</tr>
<tr>
<td>Simple task</td>
<td>(task &lt;task name&gt; (type &lt;process</td>
<td>decision</td>
</tr>
<tr>
<td>Sub task</td>
<td>(subtasks &lt;(&lt;task name&gt; [(predecessors &lt;predecessor name&gt;)]&gt;)*</td>
<td>(subtasks (decision_choose_gift) (plan_buy_gift (predecessors decision_choose_gift))</td>
</tr>
<tr>
<td>Complex task</td>
<td>(task &lt;task name&gt; &lt;Sub task&gt;)</td>
<td>(task sending_gift (subtasks (decision_choose_gift) (plan_buy_gift (predecessors decision_choose_gift) (plan_send_gift (predecessors plan_buy_gift))</td>
</tr>
</tbody>
</table>

The resource specifications define the team compositions, team capabilities, and individual agent’s capabilities. A team can contain several agents or sub-teams. A team has certain capabilities, which can differ from the aggregated capabilities of individual
members. For example, two players can form a pair and play a tennis game, but an individual player can not form a game even if the player knows how to play tennis.

A task defines capability requirements by roles. Each role requires certain capabilities and constraints. For example, an attacking role may require certain fighting power and needs to be close to the target area. Thus, the task can be assigned according to the capability and constraint requirements.

In addition to assigning tasks, the task manager can also schedule the assignments. Each basic task is characterized with a time, which is an estimated duration that the task requires. A task is scheduled to a time frame when the agent is free. An agent is free when it has nothing to do or if both the new task and the current task are not dedicated jobs. Two non-dedicated jobs can be carried out simultaneously.

### 4.2.5 Information Manager

An information manager (IM) is responsible for managing demand and supply of information. The design of the IM is based on the information supply chain (ISC) framework. An IM contains a) a demand manager, which anticipates information needs, consolidates open requirements, and coordinates information requirement planning, and b) a supply manager, which carries out and monitors investigation actions according to the strategies and plans.

An IM coordinates information requirements, launches investigations, and fulfills the requirements. Figure 4-18 shows the steps of this process as numbered labels.
First, initial information requirements are collected by a demand manager, which either anticipates others’ requirements (1a) or creates requirements upon request (1b). Next, the requirements are consolidated and prioritized by the IRP algorithm (step 2). Then, the IRP investigates each requirement following an investigation strategy, which specifies an order of different investigation methods. An agent has three methods to investigate: taking investigative action (3a), diagnosing a requirement and seeking information for dependent requirements (3b), querying others who might know or can obtain the required information (3c), and opening an auction to let everyone bid the information seeking job and awarding the job to the agent who can do it with the lowest cost or the highest quality (3d). Lastly, a supply manager monitors the investigation status and fulfills the requirements when information is available.

Figure 4-19 shows an IM interface. It contains two lists, one for anticipated or demanded information requirements and one for consolidated information requirements.
Each lists information requirement contents such as what is needed, who needs it and who requested it, and when it is needed. In addition, there are three columns that display the identification, status, and a history of sought actions.

![Figure 4-19: IM interface.](image)

**4.2.6 Communications Manager**

A communications manager governs inter-agent communication. An agent may either initiate a new conversation or simply follow existing ones. The communications manager organizes related messages into a conversation session, and monitors the development of the on-going conversation according to a conversation protocol. An “inquiry” or “inform” creates a new conversation. An “answer,” “acknowledgement,” or “reject” may end the current conversation. The communications manager serves as a
channel to coordinate decisions, request and bid tasks, and inquire and fulfill information requirements among agents.

The design and implementation of the agent communications is compatible with the FIPA interaction protocol [81]. The communications management includes a directory, an inbox, an outbox, and a conversation manager. Figure 4-20 shows the CM interface.

The design of the communication manager follows a typical email system. Each agent has an inbox and an outbox for receiving and sending messages. Each message contains a unique message identification, a conversation identification, the sender name, a list of recipients, a message type, and message content. Messages are exchanged through low level message transportation protocols, which are independent to
communications management, i.e., an agent can choose different message transportation protocols. For example, an agent can use J2EE JMS [188] and other agents can send the agent a message by following this protocol. Meantime, the agent can use Web service [189] to send a message to another agent. This flexibility of using different protocols gives R-CAST agents greater portability for various domain applications.

A conversation identification equals the message identification of the first message that initiates the conversation. All messages in the same conversation shares a common conversation identification. The conversation identification is critical for agents to handle multiple conversations at the same time and be able to keep track of the conversation history. Figure 4-21 shows a conversation between two agents. It starts from message M1 and ending with message M4. The dashed line between two messages indicates a function that identifies the appropriate conversation for an incoming message. In other words, related messages belonging to the same conversation are grouped together when an agent receives a message.

![Conversation management](image-url)
Message types define performatives of an agent’s speech. All components have a chance to access a message. However, only those that can understand the message can read and response to the message. Table 4-8 shows the implemented performatives and their content, initiating and responding components. IM, RM, CM, PM, TM stand for information manager, resource manager, communications manager, process manager, and task manager, respectively.

A message type can be potentially read by multiple components. Each component that read the message will check the content of the message to decide if it should handle the message. This design requires components (not just the communications manager) to be responsible for implementing a functional message reader. The design allows easy extension of agent communication because adding a new performative does not require modifying the communications manager.

---

8 The resource manager was mainly developed by Rui Wang. It is not included in this thesis.
4.2.7 Auctioneer

An auctioneer is an agent component that is designed for conducting auctions. In R-CAST, procurement auctions are used to determine the best service providers for information, tasks, and resources. The market mechanism provides a simple way to obtain approximate, optimum solutions for problems that require distributed reasoning due to the distributed nature of knowledge and information [190]. The goal of each provider is not to maximize its own utility, but to offer a way to provide simple feedbacks to the auctioneer to make a decision. Without an auction, a decision-maker has to collect detailed information such as each bidder’s expertise, availabilities, and costs, etc., and compile all the information to make a final decision. In contrast, using an auction allows

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Content</th>
<th>Initiating Components</th>
<th>Responding Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>Acknowledgement</td>
<td>IM, RM, CM, PM, TM, Auctioneer</td>
<td>CM</td>
</tr>
<tr>
<td>Answer</td>
<td>Results of an inquiry</td>
<td>IM</td>
<td>IM</td>
</tr>
<tr>
<td>AuctionBid</td>
<td>A bid for an auction</td>
<td>IM, RM, PM, TM</td>
<td>Auctioneer</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel an request or inquiry</td>
<td>IM, RM, TM</td>
<td>IM, RM, TM</td>
</tr>
<tr>
<td>Count</td>
<td>Current and target numbers</td>
<td>CM</td>
<td>CM</td>
</tr>
<tr>
<td>Inform</td>
<td>Facts</td>
<td>IM</td>
<td>IM</td>
</tr>
<tr>
<td>Inquiry</td>
<td>An information need</td>
<td>IM</td>
<td>IM</td>
</tr>
<tr>
<td>ListAuction</td>
<td>An auctionable (task, resource, query)</td>
<td>Auctioneer</td>
<td>IM, RM, PM, TM</td>
</tr>
<tr>
<td>NotUnderstand</td>
<td>Unknown fact type, plan, or task</td>
<td>IM, PM, RM, TM</td>
<td>IM, PM, RM, TM</td>
</tr>
<tr>
<td>Quit</td>
<td>Remove the sender from recipients’ address books</td>
<td>Agent command shell</td>
<td>CM</td>
</tr>
<tr>
<td>Refuse</td>
<td>Deny an inquiry or request due to unavailability</td>
<td>IM, RM, PM, TM</td>
<td>IM, PM, RM, TM</td>
</tr>
<tr>
<td>Request</td>
<td>Request a task at certain time</td>
<td>RM, PM, TM</td>
<td>PM, RM, TM</td>
</tr>
<tr>
<td>RequestDone</td>
<td>Notify the requestor that the task is done</td>
<td>RM, PM, TM</td>
<td>PM, RM, TM</td>
</tr>
<tr>
<td>Subscription</td>
<td>An information need within a duration</td>
<td>IM</td>
<td>IM</td>
</tr>
<tr>
<td>TimeInquiry</td>
<td>An information need with a deadline</td>
<td>IM</td>
<td>IM</td>
</tr>
</tbody>
</table>
the decision-maker to make the same decision by sorting the bids. R-CAST assumes that all the bidders are honest and bid according to their true costs.

An auctioneer can hold an auction for a task assignment, a resource requirement, or an information need. However, the auction mechanism and protocols are independent to these object types. A typical auction process has six steps: list RFQ, offer, determine winner, order, provide service, and terminate. Figure 4-22 shows the basic process of an auction.

---

**Figure 4-22**: Auction process.

---

Figure 4-23 shows an auctioneer interface. The table on the top displays auctions (RFQs). Each RFQ that the agent posted is displayed as a row, which shows the identification, the content specification, the due time, a reserved price, and the current state, which can be listed, bid, cleared, ordered, and terminated. Once an RFQ is selected,
the bids will be listed in the window at the bottom. The bidder, price, and results of win or not are displayed in a table.

---

**Figure 4-23**: Auctioneer interface.

---

**4.3 Lessons Learned**

R-CAST is a result achieved through multiple projects. It can be traced back to 2000, when Dr. John Yen and his colleagues at Texas A&M University created CAST [27] and originated the ideas of proactive information delivery and simulating teamwork using agent technologies. Since 2003, CAST has been redesigned to incorporate ideas of computational RPD decision-making model and information supply chain. The first implementation finished in late 2004.

R-CAST has been designed to achieve a high degree of flexibility through configurable system parameters. The new system offers a more comprehensive and richer
set of knowledge representation that can cover a wide range of agents’ cognitions and behaviors. R-CAST design abandons strong procedure control statements such as, if, while, and let. Instead, it adopts descriptive knowledge representations that can be easily understood by human users. R-CAST also emphasizes its usability by applying comprehensive visualization solutions, which can simplify the model design and testing tasks. The interface design follows the autonomous principle of agent technologies and requires no input from human users.

4.3.1 Configurability Leads to Flexibility

Each component can be adjusted and tuned through configuring a set of parameters. Setting of configuration parameters will affect knowledge interpretation and result in different cognitive behaviors. For example, one can start or stop certain components, allow or disallow applying parallel operators, and speed-up or slow-down decision cycles. With over eighty designed configuration parameters (from agent components, clock speed, to icon images), R-CAST architecture and components can be tailored according to a model designer’s specific needs. For example, one can configure the process manager to terminate a process when the final step is finished. Alternatively, one can choose to repeat the process until the termination conditions or the fail conditions are satisfied. The latter better reflects goal driven agent theories. However, the former configuration turns out to be useful for simple models. Making the architecture configurable can give designers flexibilities for different modeling needs. To some degree, making critical parameters configurable also enables psychological constrains
modeling. For example, a designer can choose to allow or disallow parallelism in executing a process. Depending on the type of cognition, both settings comply with psychological theories. Appendix A gives a configuration example.

Some configuration functionalities have been achieved through programming or knowledge engineering. However, they are different in many ways from the approach used here. First, configuration differs from programming. Program code has to be recompiled after it is modified. In contrast, configurations can be changed at run time. Modification of a program may contain bugs. Whereas, setting configurations should not cause any errors.

Configuration also differs from knowledge. Knowledge is domain dependent. Configuration is domain independent. At run time, knowledge is dynamically updated, while configuration is relatively static and updated infrequently. An agent can change knowledge autonomously by itself. Configurations can only be changed by a human designer. In other words, an agent should adjust or learn its knowledge (not configurations) to adapt to its environment.

4.3.2 Component-based Design Leads to Robustness

Component-based design is the key enabler for complex agent architectures and models. It makes implementation easy and systems robust. A component should be independent and be able to run as a stand-alone instance so that it will not require complex interaction with other components [191]. Being independent of other components reduces the chances of errors in design and implementation. In addition,
component-based design helps knowledge engineering efforts because the knowledge representations of a model are also distributed, which encourages a designer to work on a small set of knowledge at a time. A designer can only focus on designing and testing specialized knowledge. By contrast, representing complex knowledge with general knowledge representation methods, such as production rules, can be much more difficult [192].

An agent component must be intelligent for solving complex problems. A low level software function such as a set of database functionalities should not be qualified as an agent’s component because it does not contribute directly to the agent’s cognitive abilities. Furthermore, an agent component must integrate with other components. A stand alone component that does not fit in the whole family of components is useless for a complex modeling task. All R-CAST components interact with others in one way or another.

4.3.3 Two Perspectives to Knowledge Engineering

Agent specification language is designed for coding knowledge. Some process specifications are extended from MALLET language that is used by CAST agent architecture [27, 97]. Some keywords used by rules are adapted from the Jess system. Because the language is not for building system functions, R-CAST knowledge syntax abandons some popular programming controls, such as if, then, while, etc. Instead, it relies on more descriptive representations such as rules and logic conditions. Knowledge engineering should not become a programming job. Realizing some system functions or
even certain behaviors is not the goal of creating a model. A model should closely reflect a human mental model. Therefore, general programming languages (e.g. Java or C++) are not suitable for specifying agent knowledge even though a model may end up with the same machine instructions as these programming languages do.

A model serves two purposes: to predict or automate behaviors (forward thinking) or to explain behaviors (backward thinking). For the first purpose, the modeling effort is focused on knowledge elicitation. Then the creation of the model is based on knowledge. The process follows a simple equation, Eq. 4.1:

\[ \text{knowledge} + \text{architecture} = \text{behavior} \]

This modeling is called knowledge oriented, which is a forward thinking methodology. This method is suitable for building predictable models. One of the main goals of this research is to build models that can predict the information needs. The R-CAST architecture has the basic mechanisms to anticipate information needs from a cognitive model. Realizing a cognitive model requires engineering the domain knowledge for the agent. In R-CAST, knowledge design can be grouped into four categories: 1) declarative and descriptive knowledge, which can be represented as rules and facts, 2) procedure knowledge, which can be represented as plans that define how to do a task, 3) experience knowledge, which is used to make naturalistic decisions on what to do, and 4) social knowledge, which explains collaborations and who should do it.

In addition to predicting or automating behaviors, a model can also be used to study behaviors. This modeling philosophy is called, “behavior oriented.” Behavior oriented modeling follows backward thinking. Agent mental model or knowledge must
be able to explain the observed behaviors. The method follows a different equation, Eq. 4.2:

\[ \text{behavior} - \text{architecture} = \text{knowledge}. \]  

The modeling effort is focused on comparing the human behavior and the model’s. The modeler tunes the model until the model’s behavior is closely matched to the human’s. Knowledge oriented modeling cares more about functionalities and performance than matching the behaviors between a human and a model. Two philosophies require different implementation methodologies.

The two methods serve two different purposes. In forward thinking method, a designer is often concerned with agent performance. The final model may even have better performance than an human expert. In contrast, in a backward thinking method, the designer may be concerned with how well the model fits the desired behavior. If necessary, the designer may have to intentionally add some limitations or constraints to reflect similar constrains embodied by humans. Therefore, when building models that have performance in mind, one should use forward thinking, which can best leverage the human experts’ knowledge. One should use backward thinking to design models for studying and explaining behaviors.

**4.3.4 General Implementation Guidelines**

Building an agent model is a software engineering process which includes six activities: requirement analysis, specification, architecture, coding, testing, documentation, and maintenance. However, implementing, deploying, and refining an
agent model require more considerations. Several implementations of R-CAST agents suggest that the Living Lab method is a very good guideline to follow. This method used an ecological approach for integration of theory and practice, continuous process improvement, and tool development in multi-operator systems [193]. The approach consists of four steps: ethnography study, knowledge elicitation, scaled worlds (as experimental test bed), and prototyping. Figure 4-24 [194] illustrates the general outline of the Living Lab process.

![Living Laboratory Framework](image)

**Figure 4-24**: The Living Lab framework (McNeese et al. 2005).

In the first step, a designer must learn the basics about the users, their tasks, cognitive challenges, and working environments. This study is called ethnographic study, which employs qualitative methods such as informal interviews, case study, and focused group observations.

Next, the designer should capture and document the user’s knowledge structure. From the R-CAST architecture perspective, knowledge can be grouped into four categories: declarative, procedure, experiences, and social. Depending on the specific
problem domain, a designer can capture knowledge into relevant categories. As a result, the designer should be clear about how to encode all the needed knowledge.

After knowledge elicitation, the designer should create a world model, which serves as a test bed for an agent. The world model should simulate the major problem and environment that the agent will act upon. It should also include functions that interact with the agent. In particular, it must simulate agent’s perception such as sensing and observation capabilities as well as application of agent operators. When the scaled world is ready, the designer can implement the agent model staring from each individual component in a dependency order:

1. Implement and test AKB (declarative)
2. Implement and test process (procedure)
3. Implement and test RPD (experiences)
4. Implement and test agent collaborations (task assignment, resource allocation, auction, communication)
5. Integrate all components and test

Finally, when the agent model is ready, it can be tested in a real-world problem domain. The whole process can be iterative until the agent model is refined to the desired quality.
Chapter 5

Experiments and Results

Previous chapters introduced the task-oriented information supply chain framework and R-CAST, an agent architecture that realizes the framework. This chapter introduces experiments designed for evaluating the ideas and methods proposed in the framework and for testing the specific solutions implemented for R-CAST.

The framework is proposed to study a wide range of interrelated research problems including cognitive modeling for decision-making processes, anticipation of information needs, information requirement planning, and information seeking. Before its deployment for real world problems such as emergency response and rescue, battle field command and control, and supply chain collaboration and forecast, the framework must undergo testing via lab experiments in simulated environments.

The simulation is designed so that it is appropriate for testing two basic features. First, the tasks in the simulation should require cognitive abilities as well as decision-making. The tasks must be complex and dynamic enough to reproduce situations where decision processes and information requirements change and information must be obtained from distributed sources. Second, the simulation should involve a large volume of available information, which requires efficient means of information seeking and communications. Only does such an information rich environment reveal the information overload problem, and provide an environment for evaluating corresponding strategies such as information requirement planning.
Many proposals exist for simulators and problem domains study the agent collaborations. RoboCup provides a standard yet challenging problem — a soccer game for promoting research in AI and robotics [104]. The problem requires a team of robots or agents to collaborate and compete with their opponents. RoboCup Rescue has a generic urban disaster simulation environment, which allows heterogeneous intelligent agents such as fire fighters, commanders, victims, and volunteers to conduct search and rescue [195]. Although these simulators offer complex task problems and require significant amount of communication, the complexity of problems required great detail for achieving desirable behaviors. Furthermore, simulation scenarios are difficult to modify to tailor agent collaboration behaviors.

Besides RoboCup games, trading agent competition (TAC) offers problem domain in economics, where agents make business decisions in auctions [117, 118, 120, 121, 125, 148, 149, 196]. The TAC game also involves a large volume of information. However, even it is a challenging problem for data processing and decision-making. The task of TAC games represents a very simple decision process such that the decision points and information requirements are easy to predict. For this reason, it does not offer enough dynamics for agent collaboration.

This chapter introduces two specifically designed experiments. The design of first experiment tests decision support under time pressure in a combat simulator. The second experiment evaluates the efficiency and monitors the formation of an information supply chain in a simulated “information market.” These two experiments provide ideal situations that require both dynamic information exchange and a large volume of information.
5.1 Experiment 1: Using R-CAST Agents to Model and Assist Decision-making Tasks

R-CAST offers a novel method in anticipating information needs: it creates a comprehensive modeling technique by combining the research ideas in naturalistic decision-making, teamwork, and cognitive modeling. With this method, R-CAST agents can accurately anticipate a task performer’s information requirements according to a collaborative RPD model. This unique feature allows R-CAST agents to assist human decision-makers by collecting useful information before the time when it is needed. The experiment evaluates R-CAST agents in a real-time simulation environment, feeding teams with frequent decision-making tasks under different time pressures. The result\(^9\) suggests that the R-CAST architecture can model complex decision-making process and that R-CAST agents can improve human team performance in high time pressure situations.

5.1.1 Introduction

In a complex and dynamic task environment, a team must develop shared situation awareness in order to make sound and efficient decisions. This requires sufficient information exchange so that a decision-maker has needed information at a decision point. In addition, the communication must not overload the decision-maker

---

with irrelevant information [57, 135]. Research suggests that this requires teammates to establish a shared mental model [16, 26, 28] which refers to an overlapping understanding among members of the team regarding their objectives, their structure, and their process. Because of the cognitive constraints, natural communication among humans is often time consuming, an information link can easily breakdown in high time pressure situations. Thus, humans must leverage intelligent technologies to facilitate their information sharing tasks.

Understanding a human decision-maker’s needs and autonomously fulfill the needs is a challenging problem. To be able to work with human users, software agents must understand humans’ objectives, organizational structures, and decision processes. R-CAST is focused on the integration of high level decision models and information management. The architecture includes a set of cognitive process models such as rules, processes, communication, and decision-making.

This section describes a real-time experiment for evaluating the modeling capabilities of the R-CAST architecture in collaborative decision-making and for studying R-CAST agents’ abilities for supporting humans by anticipating and providing needed information under time pressure.

5.1.2 Scenario Design

This experiment chooses DDD [197] as the test bed. DDD is a distributed multi-person simulation and software tool for understanding command and control issues in a dynamic environment. It allows an experimental designer to specify desired collaboration
scenarios. A DDD scenario contains a predefined chain of events and situations as well as simulation conditions and rules. The scenarios in this experiment involve a blue team that defends a supply route and a red team that attacks it. Figure 5-1 shows a screen shot of the simulation map.

Figure 5-1: A screen shot of S2 interface.

5.1.2.1 The Blue Team

The blue team consists of three types of members, also known as battle functional areas (BFAs): the intelligence cell (S2), the operation cell (S3), and the logistics cell (S4).
The supply route connects an airport (the bold red square) and a target area (the bold red ring) inside a safety zone (the green rectangle).

The overall goal of the blue team is to protect the airport and the target area, and to ensure as many rounds of supplies as possible be delivered by S4 between the two points. Table 5-1 lists roles, assets, functions, and ranges of each BFA.

- S2 has an unmanned aerial vehicle (UAV) under its control, is able to collect information regarding the approaching red team objects\(^2\), and is able to identify whether an object is a neutral force or an enemy unit.
- S3 has a tank under its control, is able to destroy enemies and protect the supply route. The tank cannot attack an unidentified object (heavily penalized).
- S4 has one truck under its control, is able to deliver supplies from airport to target area along the supply route. If within the attack range (realized as be-attacked range) of an approaching enemy, the truck will be destroyed.

<table>
<thead>
<tr>
<th>Code</th>
<th>Role</th>
<th>Function</th>
<th>Asset</th>
<th>Asset Speed</th>
<th>Detection range</th>
<th>Identification range</th>
<th>Attack range</th>
<th>Be-attacked range</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>intelligence cell</td>
<td>identify whether a task is a neutral force or enemy unit</td>
<td>unmanned aerial vehicle (UAV)</td>
<td>0.12</td>
<td>1.60</td>
<td>0.30</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S3</td>
<td>operations cell</td>
<td>destroy enemies</td>
<td>tank</td>
<td>0.05</td>
<td>1.60</td>
<td>0.01</td>
<td>0.30</td>
<td>0.00</td>
</tr>
<tr>
<td>S4</td>
<td>logistics cell</td>
<td>deliver supplies</td>
<td>truck</td>
<td>0.05</td>
<td>0.05</td>
<td>0.01</td>
<td>0.01</td>
<td>0.30</td>
</tr>
</tbody>
</table>

The simulation realizes detection, identification, and attacking capacities as four ranges. A BFA can detect a (moving) object in its detection range. But it can only
identify friendly or foe when the object is within its identification range. A BFA allows attacking an object inside its attack range. A BFA’s attack range may differ from its identification range. A BFA will be destroyed if an enemy gets into its be-attacked range.

Clearly, the BFA decision-makers have to collaborate with each other to have a better team performance. Both S3 and S4 rely on S2 to share information regarding the approaching objects. Otherwise, S3 would not be able to protect the airport or the target area from being attacked, and S4 would have to either take risk (not moving truck away) or be overly cautious (always moving truck away). S4 relies on S3 to protect the truck so that faster delivery can be accomplished.

5.1.2.2 The Red Team

The red team is composed of simulated members or objects \(^{10}\), which are specified as a part of scenario definition. DDD allows different classes of objects with different characteristics. A scenario in an experiment defines four classes of objects (two classes of neutral force and two classes of enemy units) with different strength values (the minimum resource required to attack a member of the blue team) and different moving speeds. Each class of object can generate many instances. Two instances from any class can combine to form a blue team. In every two second interval, a member is generated by the above combination method. Scenarios in a simulation refer to the moving patterns of the red team members: there are six such patterns.

\(^{10}\) DDD software uses a special term called “task” to refer these objects. However, here it is replaced with the word “object” to avoid confusion to the word “task” used in its normal sense, e.g., “decision-making task”. A DDD object is generated in-situ and is not played by a human or an R-CAST agent.
• H: one object moves toward the airport, the other toward the target area;
• V: both objects move toward the airport;
• K: moves like V initially, then one object moves away;
• IL: moves like H initially, then the object heading toward the airport moves away;
• JI: moves like H initially, then the object heading toward the target area moves away;
• JL: moves like H initially, and then both objects move away.

When an object moves away, even if it belongs to an enemy unit, it is no threat to the blue team, who can therefore save resources for the next round of attack. Since H pattern may change to IL, JI, or JL pattern, and V pattern may change to K pattern, the decision-makers in the blue team must be alert to adapt their decisions in a timely manner.

5.1.3 Agent Models

Members of the blue team are modeled with R-CAST agents. S2 and S4 agents are composed with four components: 1) an active knowledge base (AKB) for conditional reasoning, 2) a process manager (PM) for coordinates the “physical” actions (e.g., movement and identification), 3) an information manager for anticipating information needs, and 4) a communication manager for information exchange. In addition to the above four components, S3 agents also include a RPD decision-maker for situational decision-making.
The AKB, PM and RPD decision-maker are used to model the high-level decision-making processes that are required for blue teams. This experiment does not include a task manager because the roles and responsibilities among the three agents are well-defined. In R-CAST architecture, a task manager is usually used for assigning and scheduling more complex tasks that involve a large number of decision-makers. The AKB use rules to represent knowledge and provide basic logical reasoning capability. In this experiment AKB is responsible for maintaining current beliefs and assessing current situations. Table 5-2 gives a rule example that determines whether a target is in the identification range of an (UAV) asset.

Table 5-2: A Rule Example.

```plaintext
(Rule "in_range"
  (unidentified_task ?task)
  (task ?task ?ex ?ey)
  (asset ?asset ?ax ?ay)
  (id_range ?asset ?range)
  (< ?dis ?range)
  ->
  (target_in_id_range ?task ?asset))
```

The PM uses plans to represent knowledge of how an agent performs a procedural task. It uses AKB to evaluate the conditions that are defined in processes. For example, a S2 agent is responsible for pursuing and identifying the incoming objects (Table 5-3). The pre-condition for this process is that the agent believes there is an unidentified object, and the termination-condition (S2 will abort the identifying object task) can be one of the following (1) the object becomes invisible (out of detection range), (2) the object moves away, or (3) the object has been successfully identified.
The RPD decision-maker uses experiences to represent knowledge on what an agent should do in a particular situation and how an agent should respond to an abnormal situation. It uses AKB to evaluate the cues and expectancies, and anomalies that are defined in experiences. In this experiment, only the S3 agent is designed with RPD decision capability. Table 5-4 shows an experience example.

### Table 5-4: An Experience Example.

<table>
<thead>
<tr>
<th>Experience</th>
<th>e-hxe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cue</td>
<td>(attack_pattern hxe)</td>
</tr>
<tr>
<td>Expectancy</td>
<td>(attack_pattern hxe)</td>
</tr>
<tr>
<td>Anomaly</td>
<td>(moving_pattern null)</td>
</tr>
<tr>
<td>Goal</td>
<td>(attacked_task ?task)</td>
</tr>
<tr>
<td>Action</td>
<td>(plan_move_to_airport_attack)</td>
</tr>
<tr>
<td>Result</td>
<td>success</td>
</tr>
</tbody>
</table>

5.1.3 Procedure

5.1.3.1 The Blue Team Configuration

This experiment has two purposes. First, it tests the R-CAST capabilities for modeling team decision-making processes. Therefore, the experiment compares the R-
CAST model with human teams. Second, it evaluates whether R-CAST agents can support human decision-making by anticipating and providing relevant information, especially under time pressure. Therefore, the experiment creates hybrid human-agent teams and compares them with agent-only and human-only teams. In summary, there are three types of teams in the experiment: agent teams, human teams, and human-agent hybrid teams (Figure 5-2).

Figure 5-2: Team configuration.

Figure 5-2(a) shows the structure of the human team, where each experiment participant has a DDD screen like Figure 5-1. A human can control the movement and actions of UAV, tank, or truck by using a mouse. DDD software has been developed for over 15 years [197, 198]. With a short training session, a typical human volunteer who uses computers on a daily basis can learn to play the role of a BFA member quickly. Six human participants were selected among the Pennsylvania State University staff and graduate students. The participants were asked to play the roles of BFAs using some

11 Using human subjects in this experiment has been approved by the Office for Research Protections (ORP) at Pennsylvania State University.
testing scenarios, which allow them to be comfortable with basic simulation operations, scenario cases, and communications needed for decision-making tasks.

S2 is responsible for identifying enemy units for the whole team. S2 can (1) move the UAV to an approaching object, (2) identify an object by opening the Identify-Task window, and (3) transfer the enemy information to S3 and S4 through a Transfer-Info popup window. On S3’s and S4’s screens, the icon of an object is changed to either “neutral” or “enemy” only after S2 has transferred the identity of the object to them. S3 is responsible for attacking enemy units. S3 can (1) move the tank to an attacking point, (2) attack an enemy unit, and (3) move the tank to an optimal waiting point. S4 is responsible for delivering supplies. S4 can (1) move the truck to the target and (2) move the truck away from threats and wait.

Figure 5-2(c) shows the structure of an agent team, in which all roles (S2, S3, and S4) are played by R-CAST agents. Useful knowledge and experience were elicited from preliminary testing experiments with human participants. These were used to encode R-CAST agent’s knowledge (see Appendix E). R-CAST agents are connected to the DDD simulator through a network communication socket [199]. The agents can perceive the “same” types and “same” amount of visual inputs as human participants can. In addition, agent can issue the same command set as human players can, e.g. launching an UAV from the owner’s base or pursuing an unfriendly object. The agent team autonomously runs without any interaction with humans.

12 It is impossible to model the human perception exactly and with the same details. Here, an agent has a regular update of key information such as object location, speed, and moving direction according to human observable ranges.
Finally, Figure 5-2(b) shows the structure of a human-agent hybrid team, where two R-CAST agents play the roles of S2 and S4, respectively. An R-CAST agent, together with a human partner, plays the role of S3. The human partner of S3 only informs the R-CAST agent of the object moving pattern by clicking the corresponding button in the human-agent interface (Figure 5-3).

![Figure 5-3: Interface for moving pattern inputs.](image)

S2 agents in all three teams need to transfer enemy information to S3 and S4 agents. Clearly, a human team requires human collaboration; an agent team requires agent communication, and a hybrid team requires both human-agent and agent-agent collaborations.
5.1.3.2 The Red Team Configuration and Scenario Settings

All three teams are working on the same set of scenarios in a random sequence. In each scenario, objects of the red team move at a fixed speed, which range from 0.2 to 1 with 0.1 increment step. Thus, each scenario has a total of nine speed configurations. Speed is selected as an independent variable because it allows testing the team performance under controlled time pressure. The faster the enemy objects of the red team move, the more stressful a blue team member (either an agent or a human) would feel. Depending on the speed selection, it can take four to twenty-three minutes to run through a scenario. Each scenario contains 48 objects, with the attack pattern changing randomly. This is to ensure that human participants are not mentally prepared for the next task so that their performance would improve over time, which might complicate comparison with the performance of an agent which does not have memory and learning built into its cognition repertoire. However, it will be seen in later discussions that it is nearly impossible to remove all cognitive advantages of humans. Each scenario is tested with six runs conducted either with six human-agent teams or six human teams. Roles of each team are played by different human participants. Because agents have no variation in personality or other traits, a scenario is tested with only one agent team.

5.1.3.3 Equipments

The experimental environment consists of four Pentium 4 computers:

- one server that runs the DDD simulator,
• three client computers that run clients S2, S3, and S4 respectively. In addition, the S3 client has dual displays: one for DDD simulation interface window (Figure 5-1) and one for the agent-human interface window (Figure 5-3).

5.1.4 Results

<table>
<thead>
<tr>
<th>Speed</th>
<th>Human</th>
<th>Human-Agent</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>1.000</td>
<td>1.000</td>
<td>0.882</td>
</tr>
<tr>
<td>0.3</td>
<td>0.980</td>
<td>1.000</td>
<td>0.882</td>
</tr>
<tr>
<td>0.4</td>
<td>0.784</td>
<td>0.941</td>
<td>0.882</td>
</tr>
<tr>
<td>0.5</td>
<td>0.725</td>
<td>0.961</td>
<td>0.882</td>
</tr>
<tr>
<td>0.6</td>
<td>0.539</td>
<td>0.794</td>
<td>0.765</td>
</tr>
<tr>
<td>0.7</td>
<td>0.500</td>
<td>0.745</td>
<td>0.882</td>
</tr>
<tr>
<td>0.8</td>
<td>0.245</td>
<td>0.706</td>
<td>0.588</td>
</tr>
<tr>
<td>0.9</td>
<td>0.235</td>
<td>0.706</td>
<td>0.471</td>
</tr>
<tr>
<td>1.0</td>
<td>0.029</td>
<td>0.304</td>
<td>0.118</td>
</tr>
</tbody>
</table>

Table 5-5: Comparing S3 Performance.

Table 5-5 and Figure 5-4(a) list and plot average performances evaluated by the percentage of enemy units that S3 destroyed. Figure 5-4(b) plots average performance in terms of how many rounds of supplies were successfully delivered.
In Figure 5-4(a), performance of S3 in the agent team, in human teams, and in human-agent hybrid teams are compared side-by-side. For human and human-agent teams, the variance on each point is plotted along with the average value obtained from the six test runs. The performance of the agent team reflects one test run on each scenario so no variance is labeled. Figure 5-4(b) compares the performance of S4 in the three types of teams. Both figures confirm that performances of the agent team and the human team follow roughly the same trend (i.e., closely correlated). Thus, R-CAST agents are effective in modeling the collaborative and decision-making behaviors of humans for the three roles tested (S2, S3, and S4).

In addition, paired t-tests show that human-agent hybrid teams performed better than pure human teams (P-Value = 0.001) and pure agent team (P-Value = 0.017). When time pressure (incoming enemy speed) is low, all teams could identify and destroy most enemies. As time pressure increases, the performance difference of the hybrid teams, the agent team, and human teams also increases. The performance of human teams decreased dramatically when enemies attacked at high speeds, whereas the performance decreased dramatically when enemies attacked at high speeds, whereas the performance decreased dramatically when enemies attacked at high speeds, whereas the performance...
of hybrid teams and that of agent teams decreased at slower rates. This result reinforces the findings that people are extremely sensitive to time pressure [6]. More importantly, it indicates that as a cognitive aid, R-CAST agents can alleviate stress to humans caused by time pressure.

By comparing the agent teams and hybrid teams, one can find that on average hybrid teams performed better than the agent team. This is because hybrid teams leverage the human intelligence especially humans’ spatial reasoning capability, which is superior to that of an agent. In easy scenarios, humans can make better decisions in recognizing the objects’ moving patterns. In more difficult scenarios, when enemies attacked at high speeds, humans can also adjust their decision tempo to the situation. In comparison, the decision cycle time of an agent is fixed and cannot be adjusted. Agent models lack basic capabilities for meta-level cognitions, which make decisions of how to make decisions [52]. Certainly, this is a limitation in R-CAST architecture and deserves future research work.

Figure 5-4(a) also shows the variance of each data point. It reveals that the performance of six hybrid teams is stable and that the performance of the six human teams varies from run-to-run dramatically. To have a better insight into how human personalities may affect the performance, Figure 5-5(a) plots the performance indicators of the S4 member in six human teams, and Figure 5-5(b) those of S4 agents in six hybrid teams.

---

13 Since there is only one agent team, the data set for the agent team has no variance.
The results shown in Figure 5-5 and Table 5-6 clearly reveal that the performance of S4 agents in a hybrid team is more stable than that of S4 humans in a purely human team (P-Value = 0.065). In addition, the S4 agent performance is independent of who plays the hybrid team’s S3 human partner. The performance of S4 human in the pure human teams does have some correlation with human personalities: as time pressure changes, overcautious persons tend to perform poorly but smoothly, whereas impulsive persons tend to perform inconsistently. However, the last three points in Figure 5-5(a) suggest that such differences in personality would not affect the performance much when the time pressure is extremely high.
5.1.5 Summary

This experiment evaluated R-CAST agents in a real-time simulation environment using scenarios with frequent decision-making tasks. The results suggest that R-CAST architecture is effective in modeling collaborative decision-making in a team environment. More importantly, it suggests that when supported by R-CAST agents, humans can perform better than either R-CAST or unaided humans, especially under time pressure. This result is largely due to the ability of R-CAST agents to accurately anticipate the information needs of human users so that decisions can be made quickly. Finally, the finding of this experiment may also indicate a need for incorporating metacognition theories in R-CAST architecture.

5.2 Experiment 2: Forming Information Supply Chains (ISC)

Effective information sharing requires both accurate anticipation of information needs and efficient information seeking. The previous experiment tested the R-CAST
capabilities in cognitive modeling and decision support. However, it cannot evaluate the collaboration for information requirement planning. Such evaluation requires not only a dynamic environment with distributed information sources, but also a large amount of information exchange. In other words, the previous experiment does not contain information needs or supplies as major characteristics of the simulating environment. Therefore, a new experimental domain is needed to evaluate the information planning capability of R-CAST. It is instructive to realize that agents for such an experiment do not need to consider how to obtain the needed information, that is, they can neglect some tasks (task modeling, information needs anticipation, and information retrieval) and focus only on regulating information demands and supplies intelligently so that appropriate demands and supplies can be brought together efficiently. The information supply chain (ISC) concept has been incorporated into the R-CAST architecture to address the problem of information requirement planning; therefore, ISC is the main subject of discussions in the experiments reported in this section.

5.2.1 Introduction

This experiment tested the ISC model by comparing it with two other models. Since in an information supply chain, agents can seek, and share information directly with the information provider, the ISC model is called “asking provider” model. The other two models which have existed before this research are “asking everyone” and “asking broker” models. The results of the experiment suggest that ISC can reduce the chance of information overload by efficient communication. Another goal of the
experiment is to evaluate auction algorithms that help to form an information supply chain by discovering efficient information providers. Finally, the experiment sheds light on understanding the relations between cognitive capacity and information overload in a collaborative environment.

5.2.2 Color Block Game Settings

5.2.2.1 Game Design

The Color Block Game (CBG) is a specially designed test bed for evaluating the ISC concept: each block represents either a demand (empty block) or a supply (filled block) of information. The color of a block represents a particular type; thus, the need for a piece of information is considered to have been met when there is a supply block with the same color has been obtained by fusing existing supplies. These blocks are generated randomly, and an agent tries to coordinate them so that demands can be fulfilled as quickly as possible. The game is designed to test two processes: information requirement planning and supply chain formation. The goal of an agent in ISC is to fulfill the information demands in a dynamic information market by fusing the supplied information. The performance is evaluated by the fill-rate or the percentage of demands that are fulfilled.

Structurally, information needs can be viewed as high-level information types, and information supplies as low-level information types. Figure 5-6 shows a typical information dependency relation (IDR). Type 1, 2, 3, and 4 are low-level information,
which are being supplied in the market. Type 5, 6, 7, and 8 are high-level information types. Type 5 depends on Type 1 and 2 while Type 7 on Type 5 and 3.

---

![Diagram](image)

**Figure 5-6:** An IDR in CBG game.

In CBG game experiment, decision on which information type to generate follows a random pattern with a uniform probability. The probability for information demand may be different from that for information supply. However, with a particular category (either demand or supply), all types of information (block colors) have an equal chance for being generated. In addition, each demand or supply has a validation time, which varies, but statistically its length follows a Gaussian distribution. Table 5-8 specifies the experimental settings of the Color Block games. An agent will be penalized if it attempts to fulfill an expired demand or use an expired supply.

Since a particular type of information is generated randomly and with a fixed average probability, all color blocks associated with this type will follow the Poisson distribution. In this work, a Poisson generator is used for simulating information demands and supplies, and the rate of appearance for demand or for supply is determined by one adjustable parameter, namely, the average number of demand or supply (blocks) produced per cycle time. To be more specific, the number of new demand/supply blocks generated in each time cycle is determined by a Poisson function (Eq. 5.1).
The average number is bounded by a certain range, and it slowly drifts according to several trend patterns. (Eq. 5.2) shows how the average number-per-cycle time is calculated.

\[ \bar{R} = \min(R_{\max}, \max(R_{\min}, \bar{R} \times \tau)) \]
\[ \tau \text{ is a trend.} \]
\[ [R_{\min}, R_{\max}] \text{ is a range for } \bar{R}. \]

ISC is inspired by supply chains in a normal economic world of material goods. Thus, it is clear that, in a typical market scenario, the trend is controlled by a random walk function: that is, it can go up or down with equal probability (Eq. 5.3). Alternatively, one can choose a linear trend function to simulate a growing market. For example, the upper right window in Figure 5-7 shows a linear trend curve for steady growing demands, and the bottom right window illustrates a random walk trend used to generate the supply of information.

\[ \tau = \max(\tau_{\min}, \min(\tau_{\max}, \tau + \text{random}(-0.005, +0.005))) \]
\[ [\tau_{\min}, \tau_{\max}] \text{ is a range for } \tau. \]
5.2.2.2 Game Monitor

Figure 5-7 shows a screen shot of the CBG monitor. As briefly mentioned previously, in order to visualize the demands and supplies, different colors are used to represent information types. In addition, a hollow block represents an information demand, and a solid color block represents a supply. The top-middle window of Figure 5-7 shows the demands of market, and the bottom-middle window the supplies. The top-left window shows the current overall number of demands and the number of demands that have been fulfilled. At run time, a demand (hollow block) will drop down from the top of its window (top-middle window) to simulate passing of time. When a demand expires, the hollow block will disappear. If there is fulfillment, the hollow block will become a filled block momentarily for a short period of time so that a human designer has a chance to know that the demand has been fulfilled. The top-right window illustrates the mean and actual numbers of demands that are generated in a time cycle.

The monitor can also be used for testing the performance of people in a CBG.
Similarly, the windows on the bottom show the number of supplies, the current available supplies (solid blocks), and the mean and actual numbers of supplies that are generated in a time cycle.

### 5.2.3 Agent Models

Since each agent in an ISC can only have limited knowledge and observing capabilities, they must collaborate with others for obtaining the needed information. Another reason for collaboration is that, sometimes, several pieces of information must be fused together in order to fulfill one demand, but an agent may only possess the capability of accessing those supplies of information but not the capability to fuse them. In this case, the agent should forward the information to the agents who can fuse information. Table 5-7 shows various agents’ observation and fusion capabilities. It is
worth noting that agents may not know each others’ capabilities. For example, agent 5 may not know who can provide the dependent information types 1 and 2, and hence has to obtain them indirectly.

<table>
<thead>
<tr>
<th>Agent name</th>
<th>Supply types that can observe</th>
<th>Demand types that can observe</th>
<th>Fusion rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner</td>
<td><img src="image1" alt="Supply types" /></td>
<td><img src="image2" alt="Demand types" /></td>
<td></td>
</tr>
<tr>
<td>Broker</td>
<td><img src="image3" alt="Supply types" /></td>
<td><img src="image4" alt="Demand types" /></td>
<td></td>
</tr>
<tr>
<td>Inter5</td>
<td><img src="image5" alt="Supply types" /></td>
<td><img src="image6" alt="Demand types" /></td>
<td></td>
</tr>
<tr>
<td>Inter6</td>
<td><img src="image7" alt="Supply types" /></td>
<td><img src="image8" alt="Demand types" /></td>
<td></td>
</tr>
<tr>
<td>Inter7</td>
<td><img src="image9" alt="Supply types" /></td>
<td><img src="image10" alt="Demand types" /></td>
<td></td>
</tr>
</tbody>
</table>

To simulate human cognitive constrains, agents are configured with fixed amounts of communication and information processing capacity at each time cycle. Each communication or information processing action will consume certain cognitive capacities. When all capacity is consumed, the agent is (cognitively) overloaded. An agent can be overloaded by a large number of incoming messages or by too many inquiries. The overall performance is evaluated by the fill-rate, the percentage of
information demands that are satisfied. When an agent is overloaded, it can not receive new messages, send messages to others, or plan for an existing requirement.

Figure 5-8 illustrates an information supply chain in a CBG game. Each rectangle node represents an agent and its current utilization rate (capacity that is consumed in current cycle over the total capacity). The red link represents an on-going communication.

Figure 5-8: An ISC in a CBG.

The goal of agents is to fulfill the demands of the market as quickly as possible. Agents used in this test experiment are composed of three components: 1) an active knowledge base for decomposing an information requirement and fusing the required information, 2) an information manager for information requirement planning, and 3) a
communication manager for information exchange. Compared with agents that are used in the previous experiments these agents do not include the high-level decision processes (RPD decision-maker and process manager).

The game can be configured to represent a broad range of information sharing problems. It is challenging because agents have limited capacities for observation and for obtaining and processing information. To succeed, agents will have to collaborate with others and determine the best strategy to react to variations in demand and availability of supplies.

5.2.3 Procedure

This experiment involves two tests. The first test compares three information sharing models: asking everyone, asking a broker, and asking providers. Figure 5-9 show the model structures.

![Figure 5-9: Three supply models.](image)
In the *asking every one* model, no agent knows who can provide what types of information. So an agent broadcasts its inquiries to every other agent. Of course, duplicated requirements from different agents will be consolidated. In addition, an agent will not ask an information needer. In the *asking a broker* model, only the broker knows agents capabilities. Everyone else has to ask the broker to forward their queries. Upon receiving a request, the broker will identify an appropriate provider and let the provider inform the needer about the needed information. In the *asking providers* model, every agent knows who can provide what. A needer can directly query the providers. Three teams of agents will be created according to each of the supply models. The teams were tested in the same set of game scenarios.

The second test studies formation of the information supply chain using auction mechanisms. Initially agents have no knowledge of who can provide what. They use auction protocols to determine the most efficient providers and maintain a partnership value according to the providers’ performances. Some providers’ partnership values increase gradually after they win and complete enough information seeking tasks. Then, after the partnership values increase to a certain point, an information needer will no long hold auctions any more. It will directly ask the best provider. Agents used in this experiment have an additional auctioneer for handling the information auctions.

Table 5-8 lists the major parameter settings for the experiment.
CBGs can be run on a stand alone computer. However, one can choose to run a game over three computers: one for CBG server, one for agents and one for database server that is used to record the experiment data.

Table 5-8: Parameters Used in a Typical CBG.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit of measure</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market cycle speed</td>
<td>millisecond</td>
<td>100</td>
</tr>
<tr>
<td>Session time</td>
<td>cycles</td>
<td>1200</td>
</tr>
<tr>
<td>Average new demand number (min)</td>
<td>number</td>
<td>0.1</td>
</tr>
<tr>
<td>Average new demand number (max)</td>
<td>number</td>
<td>5.0</td>
</tr>
<tr>
<td>Demand trend (min)</td>
<td>number</td>
<td>0.9</td>
</tr>
<tr>
<td>Demand trend (max)</td>
<td>number</td>
<td>1.1</td>
</tr>
<tr>
<td>Demand timeout mean</td>
<td>cycle</td>
<td>100</td>
</tr>
<tr>
<td>Demand timeout variance</td>
<td>cycle</td>
<td>20</td>
</tr>
<tr>
<td>Average new supply number (min)</td>
<td>number</td>
<td>0.1</td>
</tr>
<tr>
<td>Average new supply number (max)</td>
<td>number</td>
<td>0.2</td>
</tr>
<tr>
<td>Supply trend (min)</td>
<td>number</td>
<td>0.9</td>
</tr>
<tr>
<td>Supply trend (max)</td>
<td>number</td>
<td>1.1</td>
</tr>
<tr>
<td>Supply timeout mean</td>
<td>cycle</td>
<td>100</td>
</tr>
<tr>
<td>Supply timeout variance</td>
<td>cycle</td>
<td>20</td>
</tr>
<tr>
<td>Agent information manager cycle speed</td>
<td>millisecond</td>
<td>100</td>
</tr>
<tr>
<td>Agent communication manager cycle speed</td>
<td>millisecond</td>
<td>100</td>
</tr>
<tr>
<td>Information processing capacity</td>
<td>number</td>
<td>[1,3]</td>
</tr>
<tr>
<td>Capacity consumed for reading a message</td>
<td>number</td>
<td>0.2</td>
</tr>
<tr>
<td>Capacity consumed for sending a message</td>
<td>number</td>
<td>0.2</td>
</tr>
<tr>
<td>Capacity consumed for a fusion</td>
<td>number</td>
<td>0.2</td>
</tr>
<tr>
<td>Capacity consumed for a planning</td>
<td>number</td>
<td>0.2</td>
</tr>
</tbody>
</table>
5.2.4 Results

5.2.4.1 Result 1: Comparison of Three Information Sharing Models

Figure 5-10 shows the performance comparison among the three sharing models. Each game continually ran for 1200 cycles\(^\text{15}\), during which the overall fill-rate was recorded. In addition, within each sharing model, three runs were conducted with three agents of different cognitive capacities, labeled as low (1), medium (2), to high (3).

The results suggest that cognitive capacity has significant impact on an ISC’s overall performance. Cognitive capacity affects the asking everyone model most. In asking everyone, an agent broadcasts its inquiries to all other agents, even to the one who cannot provide the needed information. As a result, the broadcasting agent wastes cognitive capacity in sending extra information, and the agents being asked waste their capacities in reading the inquiries that they can not fulfill. This model has the lowest performance when the cognitive capacity is low.

By contrast, cognitive capacity has the least impact on the asking providers model. An information requester only needs to send one message for an information requirement. From the requestor’s perspective, this model is more efficient than asking everyone. Meanwhile, the recipients can avoid wasting their cognitive capacities on irrelevant incoming inquiries.

---

\(^{15}\)The data of the first 200 cycles are removed because at the early stage of a game the fill-rate is unstable. At the early stage, the number of demands or the denominator of the fill-rate is small. In this case, the fill-rate can change dramatically when the satisfied demand changes.
The *asking a broker* model also suffered when the cognitive capacity is low because the broker becomes a bottle-neck in the whole information supply chain. In other words, the broker can experience overloaded easily. This can be further explained in Figure 5-11, which illustrate the average utilization of each agent. In the figures, each point represents the average utilization of an agent at a certain cognitive capacity. A line indicates the trend of how the utilization changes when the cognitive capacity changes from low to high. An agent is likely to have been overloaded if the average utilization is high.

Figure 5-11(a) shows that *asking everyone* will overload everyone. Figure 5-11(b) shows that *asking a broker* will overload the broker. Figure 5-11(c) shows that *asking providers* results in a balanced utilization.

It is worth noting that sometime increasing everyone’s cognitive capacity may not result in reduction of the overall utilization rate or reduction of cognitive overload. On the contrary, increasing cognitive capacity may sometimes increase the chance of cognitive or information overload. Figure 5-11(a) shows that, as the cognitive capacity increases, the utilization rate increases, too. This finding appears to be counterintuitive. However one can imagine that as the cognitive capacity increases, the number of overall communication increases, too. This can cause further information overload.

In conclusion, increasing the information processing capacity is not always a valid solution to overcome the information overload problem. Instead, one should adopt a better information supply chain model to achieve efficient information management.
Figure 5-10: Comparing performances of three supply models.
Figure 5-11: Comparing average utilizations of three supply models.

a) asking everyone

b) asking a broker

c) ISC (asking providers)
5.2.4.2 Result 2: Forming Information Supply Chains

Figure 5-12 shows the testing result of forming information supply chains: the fill-rate increases steadily as partnerships or stable information supply chains are formed. In this experiment, agents use an auction mechanism to determine efficient agents as their information providers. As time goes by, when an agent establishes a strong relationship with a provider, the agent will no longer need to hold an auction for every information requirement. The agent will ask the provider directly and save the cognitive capacity that was consumed by auctions.

The result confirmed the idea of using an auction to discover the efficient information providers for an agent. This is also a learning process for the agent in acquiring knowledge on others’ capabilities.
5.2.5 Summary

Information supply chain is proposed for efficient information management. This experiment shows that when cognitive capacity is limited, an ISC model (asking providers) is more efficient than broadcasting queries to everyone (asking everyone model) or sending queries to a designated broker (asking a broker model). The experiment also tested the idea of using an auction to form information supply chains. The result proves that the method is effective: an ISC slowly formed over time, and the overall performance increased. The experiment result also suggests that increases in cognitive capacity may worsen the information overload problem. Instead, an effective
strategy should focus on reducing the number of contacts and establishing information supply chains by building tight partnerships.

5.3 Conclusion

The task-oriented information supply chain framework can be tailored to study a wide range of problems from task modeling, information anticipating, to information seeking. Evaluations of the ideas, architectures, and algorithms in the framework require different experiments, each focusing only on one of the above aspects.

This chapter reports two experiments. The first experiment evaluates R-CAST agent’s collaboration and decision-making abilities in a DDD simulator using scenarios involving combat, logistics, and operational units. It is an important step towards research in human-agent collaboration and has provided rich lessons in agent-based decision support.

The second experiment evaluates the information supply chain concept in a color block game, which uses an information market to simulate a large volume of information demands and supplies. The result clearly shows that an ISC can achieve high efficiency in information management and avoid information overload especially when cognitive capacity is limited. The experiment also confirms the idea of using an auction to form information supply chains. It also has a counterintuitive finding that increasing cognitive capacity may not alleviate the information overload problem, but very often worsens the problem.
In summary, these experiments offered a rich set of findings and lessons of how to effectively assist team tasks through efficient collaboration in information sharing. The R-CAST architecture and methods in the information supply chain framework are verified and proved to be effective in team modeling, decision support, and combating the information overload problem.
Chapter 6

Conclusions and Future Research

This thesis focuses on a task-oriented information supply chain (ISC) framework and how to use it to solve problems commonly encountered in information sharing. The ISC framework is built first with creation a high-level cognitive model that can predict what information is needed for a task performer and when it is needed. Next, ISC framework is given the capability of proposing solutions for planning and fulfilling the information requirements, with high responsiveness and efficiency as the aims here. Methods for checking potential information leakage and for evaluating the credibility of information and its sources are only touched upon briefly, although these topics should become an important part of the ISC framework.

The ISC framework is formalized with existing agent theories and is implemented in R-CAST, an agent architecture composed of multiple intelligent components, including knowledge base, a process manager, an RPD decision-maker, a task manager, an information manager, a communications manager, and an auctioneer. The ISC concept and R-CAST architecture were tested in three simulation experiments. The first experiment is aimed to compare the architecture and behaviors of R-CAST to those of Soar architecture. The second one is designed to test decision support under time pressure in a combat simulator. The third experiment is to evaluate the performance and formation of information supply chain in a simulated “information market.” The results
of the above experiments suggest that R-CAST architecture can be used to model and support complex decision-making in a team environment.

In summary, the research in the ISC framework and R-CAST agent architecture represents a fruitful attempt in solving two common problems in information sharing: namely, information deficiency and information overload, especially within the context of dynamic and complex tasks. The remainder of this chapter discusses in detail contributions and future research.

6.1 Contributions

Chapter 1 set out two research questions: 1) How to accurately anticipate the information needs of decision-making teams in a dynamic environment? and 2) How to effectively coordinate and improve the efficiency of the information sharing activities?

This research has made two major contributions in addressing these two challenges of information sharing among decision-making teams. First, more accurate information needs can be anticipated using a high-level cognitive model of decision-makers. This avoids “pushing” irrelevant information to a decision-maker, which often leads to information overload. Second, the cost associated with information seeking and distributing activities can be greatly reduced because these activities can now be well-coordinated within the ISC framework.
6.1.1 Proposed a Cognitive Model for Information Push

A common information sharing problem is *information overload* that often results from pushing irrelevant information onto a decision-maker [135]. Implementation of R-CAST agent architecture covered this problem. R-CAST contains a cognitive model for high-level collaboration and decision-making. The model allows *accurate* anticipation of information needs from the decision-maker. This represents a new approach that improves the current information push technologies [11, 12]. Current push is usually based on personalization [200], which cannot effectively address the *dynamic* information requirements for a complex decision-making process.

In contrast, the high-level cognitive model in R-CAST is designed to closely reflect the dynamics of decision processes and information need at each decision point. The cognition for decision-making is realized through integration of three models: a naturalistic RPD model [20], a collaborative task allocation model, and a plan-based process model. Compared with other computational RPD models, the design of R-CAST does not emphasize a particular function such as cue matching. Instead, R-CAST focuses on the cognitive aspect of a decision *process*: e.g., how to identify missing information, which is critical for an agent to “recognize” a particular situation or an anomaly. Furthermore, R-CAST extends the RPD model to incorporate collaborative decision-making: a team of decision-makers can reason through the current situation based on a shared decision space. During a reasoning process, they can anticipate each other’s information needs, seek missing information, and proactively exchange relevant information.
R-CAST is capable of representing and interpreting high-level knowledge, which is similar to some existing cognitive architectures such as Soar [49, 50] and ACT-R [51]. However, R-CAST has the unique feature of anticipating information that is needed in a decision process. As the decision process moves forward, anticipated information needs are updated dynamically according to current situations. This feature greatly improves the visibility of information needed by decision-makers, which is useful in designing information systems for supporting complex decision-making teams such as first responders, military command and control, and intensive care professionals. Such a system will provide decision-makers with relevant information without overloading them with irrelevances.

In summary, this research implemented R-CAST, an extension of the RPD model for team collaborations, and integrated into the RPD model, other intelligent components for logical, procedural, and communicative processes. R-CAST can be used to anticipate information needs for supporting complex, decision-making processes.

6.1.2 Improved Coordination for Information Sharing Activities

The best of all, this research represents the first attempt at applying the ideas in supply chain management to coordinate information sharing activities. The information supply chain (ISC) framework invokes a metaphor useful for identifying and understanding problems, proposing better information sharing methods, and setting clear evaluation criteria.
Guided by the ISC metaphor, the thesis’ research led to a novel information requirement planning (IRP) algorithm that can anticipate, consolidate, and identify the best supply strategies for informational needs. IRP is inspired by material requirement planning (MRP), a method commonly used in supply chain management. Generally, a piece of needed information can be obtained in three ways: by direct observation, by inquiry to others, or by fusing dependent information according to information dependency relations. The goal of IRP is to identify the best supply strategy for information needs.

The ISC framework also includes an auction for identifying and selecting information providers. The auction protocol is based on the contract net protocol [144], which has been used to identify the best service providers in an multi-agent system. ISC extends the contract net protocol so that a chain of information providers can be identified through auction. By collaborating with each other, information providers can efficiently seek and fuse the needed information. However, a practical problem would arise if an auction is used to identify providers for every information need. This problem is largely the result of frequent communications required in an auction, which can push an agent’s “cognitive capacity” for processing information and communication beyond its limited capacity. To avoid this problem, the ISC framework uses long-term information partnerships to fulfill information needs, and this method has proved to be effective in establishing information supply chains, although initially these chains have to be established through multiple rounds of auction processes.

R-CAST is an agent embodiment of the ISC framework concept. R-CAST supports decision-making tasks by planning, seeking, fusing, and sharing the needed
information proactively. Experimental results show that R-CAST agents can be used to assist human teams in developing shared situation awareness while balancing information requirements against the dynamic and time sensitive decision-making processes. In summary, this research helps improve information systems that allow the right information to be shared with the right people at the right time.

6.2 Future Research

This research has taken a step forward to tackle the information sharing problem. However, many problems still remain unaddressed. This section will discuss these problems as two groups: 1) long-term research problems that are important but solutions cannot be found easily without proposing and implementing many detailed strategies, and 2) three research directions that are specific enough so that they may be pursued in the near future.

6.2.1 Long-term Research Problems

There are two major problems that must be solved before agents for information sharing can be deployed in real-world systems.

First, cognitive modeling for identifying information needs is still in its infancy; real-world applications require more sophisticated cognitive capabilities. In addition, rigorous experiments are also needed to see if agents can identify information needs more accurately than those based on the best personalization methods. To improve the
cognitive capabilities of our agents, it needs to begin with finding efficient ways that synchronize the decision processes between an agent and a human decision-maker.

Second, this research uses a logic-based representation for information and information relations. However, real-world information can be highly unstructured and in multi-media formats. Therefore, it is desirable to integrate the ISC concept with currently available information retrieval systems, such as Google, an internet search engine.

These problems can require a great deal of research effort, hence demanding long-term and continuous experimentation of new ideas or new methodologies. In the following section, there will be a list of some problems that are practically feasible and deserve to be considered as short-term research projects.

6.2.2 Develop Combinatorial Auctions for Dependent Information Needs

Chapter 3 introduced a method of using an auction to identify a chain of information providers. However, auctioning requires a large volume of information and hence often overloads the capacity of a given communication scheme. This is especially true when multiple pieces of dependent information are needed in order to generated a single piece of fused information. One possible method is to hold multiple auctions: each is for identifying a specific piece of information. This method may cause inefficiency in communication if not implemented properly. For example, an agent needs X, Y, and Z to fuse a piece of information, and three auctions are held to find X, Y, and Z. If the agent has already awarded bidders for X and Y before realizing that Z has no bid, then the agent needs to revoke the contracts to the existing bidders and abort the plan for information
fusing (it lacks \(Z\)). In this example, all the efforts (auction, contract, and cancellation) in finding information \(X\) and \(Y\) are wasted. A better method is to recognize the problem (\(Z\) is not available) before awarding the bids for \(X\) and \(Y\).

Research in auction design suggests that this problem can be handled by a combinatorial auction approach [152-154]. In this approach, an agent can specify multiple items in a bundled auction, and bidders can bid for one or multiple items. The auctioneer will determine a winner of an auction based on the specified combinational criteria. In the above example, the agent will hold just one auction for \(X\), \(Y\), and \(Z\), and each provider can bid for one or more of the needed information packets. Consequently, the agent can determine quickly if fused information can be generated, and will not award partial bids prematurely. This avoids wasting the agent’s resources and other agents’ (\(X\) and \(Y\)) resources because they don’t have to prepare the needed information if \(Z\) is not available.

Therefore, combinatorial auction may improve the overall communication efficiency of an ISC framework, especially when bundled information needs have to be met by fusing information from multiple sources. A potential problem along this research direction is that winner determination and agent valuation of bundled information needs can be computationally complex (\(NP\)-complete) [154].

### 6.2.3 Research in Meta Cognition

Chapter 5 mentioned that R-CAST architecture lacks Meta cognition. To review briefly, Meta cognition refers to making decisions on how to make decisions, and it
would improve the fidelity of a cognitive model [52]. For example, determining which
decision model (procedure, logic, or experience-based) to use for a particular situation is
a Meta decision. Such a decision requires extra knowledge about the characteristics of a
problem, its situation, and associated decision models. Obviously, procedural decision is
suited to tasks that follow a particular method or recipe; conditional decision calls for
logical reasoning; and experience-based decision should be used if information related to
a past experience is needed. However, it is not easy to implement the above intelligence
in an agent because making such a decision can be subjective. In addition, sometimes
more than one type of decisions could be used to obtain the same information, and it is
needed to implement methods for the agent to evaluate and select the best type. The
subjective side of a Meta decision is perhaps the biggest challenge. Human decision-
making is affected by many factors other than knowledge or formal methods. These
factors can be psychological conditions, emotion, fatigue, etc. A decision that is made
when a person is happy can differ from the one that is made when the person is sad. To
understand how a decision process can be affected and include the results in a model can
make human behavior modeling more accurate.

It should be emphasized here, that there exists a trade-off between the cognitive
capabilities of an agent and the cost or the efficiency of the agent’s architecture. An agent
designer must not implement cognitive capabilities or behaviors beyond what a given
architecture can handle. A good design practice is to avoid building complex architecture
that contains multiple components tightly coupled together: i.e., avoid interdependency
and direct interactions among multiple components. Otherwise, development cost will
skyrocket because it is time consuming to develop an architectural structure, and complex
architectures would require frequent and extensive revisions. Therefore, unless the goal of an agent is to understand human cognitive behaviors, a designer should always keep the cognitive modeling requirements realistic and simple. This will make agent architecture more efficient yet functional and more likely to be deployed in real-world applications. It is instructive to realize that present technologies are still far away from duplicating all cognitive abilities of a human task performer. The key to making an agent application successful is to leverage the agent architecture’s strength (by being simple) and avoiding implementation of cognitive functionalities for unintended usage.

### 6.2.4 Realize Learning in R-CAST

Currently, R-CAST has limited learning capabilities, partially because 1) R-CAST uses a complex knowledge representation system, which is difficult for incorporation of learning; and 2) learning requires a large data set for training, which is expensive if involving people.

Even though R-CAST architecture contains flexible and indirectly coupled components, it is still difficult to incorporate leaning. When a designer encodes knowledge for an agent, the designer has to be aware of which components are involved. Then, interactions among components must be carefully considered and planned. This requires a great deal of intelligence and exigencies. Current machine learning capabilities are still inadequate to handle complexities required for decision-making tasks. This is also an important reason for why knowledge engineering, which studies how to obtain humans’ knowledge, is still more practical in creating complex models.
The main issue that this research concerns is how to anticipate information needs from cognitive models and how to fulfill the information needs. A cognitive model can be used as a decision support system. R-CAST, however, is not designed to model all decision points exactly like a human. In other words, the information provided by an agent to a human may seem to be needed according to the agent’s “mental” state but not be always relevant from the standpoint of the human. One way to solve this problem is to synchronize the decision processes of the human and the agent. One can allow the model to monitor human decisions and update the agent’s decision process accordingly. A potential pitfall for this method is that an extra burden for the human may arise because constant feedback that requires the human’s attention could be overwhelming. Alternatively, the agent should continuously monitor the human, learn new knowledge, and adapt itself to provide better support. Many types of learning are possible: e.g., learning new rules, new plans, new experiences, or new communication strategies.
Bibliography


[125] S. Sun, V. Avasarala, T. Mullen, and J. Yen, "PSUTAC: A Trading Agent Designed from Heuristics to Knowledge," in *The Workshop on Trading Agent Design and Analysis, colocated with the Trading Agent Competition at the*


Appendix A

Agent Configuration Example

# Agent configuration file for agent "combat" file name: s3.conf
#

# about the agent overall configuration
# the agent name, must be unique for each agent
agentName = S3

agentIcon = ./images/tank.gif

# the components of the agents
agentComponents = kbImpl processImpl decisionImpl comImpl imImpl domainImpl

# to show gui or not, if false no gui will be shown,
# even you decide to use a component gui
useGUI = true

cycleSpeed = 0.5

isStoped = false

displayedMessage = Count Inform

# the thread cycle time for the whiteboard, in 1/1000 sec
whiteboardClock = 1000

# use whiteboard gui or not,
whiteboardGUI = true

domainImpl = edu.psu.domainadapter.ddd.HumanDDDAdapter

domainGUI = true

kbImpl = edu.psu.activeknowledgebase.AKB
# the kb file for this agent
kbFile = ./ddd/S3.kb

# the thread cycle time for the kb, in 1/1000 sec
kbClock = 1000

# show the kb gui or not
kbGUI = true

# if you want to hear a voice when query for natural reply
kbSpeakNaTrualReply = false

############################ about the process manager ####################################
# the implementation of the process manager
processImpl = edu.psu.process.ProcessManager

# the process specification file
processFile = ./ddd/S3.process

# the initial process that the agent should execute, null for none
processInitialProcess = plan_combat

# the clock cycle for process manager, in 1/1000 sec
processClock = 1000

# if or not to show the process monitor
processGUI = true

# if you want to enable voice for speak operator
operatorSpeak = false

# true, multiple processes can be live and executing in parallel
multipleLiveRootProcess = false

# true, process will terminate at end; false, the process will repeat at the end
# false, then you should define termination conditions to terminate processes
processTerminateIfEnd = false

# true, a terminated process will be removed from the process set
# false, if you want to keep dead process (cost more memory)
processRemoveIfInactive = true

# if the simulation should be in a new (cloned) KB, or in the agent's current kb
# agent should simulate in a new KB
simulationNewKB = true

# if precondition is tested, will the result affect the execution of
# the simulation? true, the result will not affect; false, the failed
# precondition will stop the simulation
simulationRelaxPrecond = true

# if the precondition is going to be tested in simulation
simulationTestPrecond = true
```plaintext
# how many levels will simulation decompose a process
simulationDepth = 1

# the recognition primed decision-making
# the rpd implementation
decisionImpl = edu.psu.rpd.RecognitionPrimedDecision

decisionClock = 500

decisionGUI = true

ebFile = ./ddd/S3.eb

rpdRoot = combat_experience

rpdAttention = combat_experience

rpdRetainState = false

rpdEvaluateState = false

rpdRepeatRecognition = true

rpdRecognitionEvidence = false

recognitionThreshold = 0.4

weightFailExperience = 0.5

weightAnormaly = 1.0

anomalyTimeout = 3

# the implementation of the communication manager
comImpl = edu.psu.communication.CommunicationManager

dirImpl = edu.psu.communication.SimpleDirectory

# a directory implementation
dirImpl = edu.psu.communication.SimpleDirectory

# the directory specification, dirImpl must be able to read this file
```
dirFile = ./ddd/ddd.dir

# how many times the count message should be send for a ping function
pingNumber = 15

# the cycle time for communication, in 1/1000 sec
comClock = 1000

# if to display the communication manager
comGUI = true

# The Information Manager
# the implementation of the information manager
imImpl = edu.psu.irp.InformationManager

defaultStrategyImpl = edu.psu.irp.strategy.AuctionOrientedInvestigationStrategy

# the cycle time for im, in 1/1000 sec
imClock = 1000

# to display the im gui or not
imGUI = true

# the default supply mode
defaultSupplyMode = edu.psu.irp.mode.AskReply

# the default time condition, when a informaiton is needed
defaultRequirementTime = 500

da a value indicate the value of information, this value should be determined by the task that use the information
defaultInformationValue = 800

# if to anticipate precondition
anticipateProcessPrecondition = true

# if to anticipate termination condition
anticipateProcessTermcondition = true

# if to anticipate fail condition
anticipateProcessFailcondition = true

# if to anticipate cues
anticipateRPDCue = true

# if to anticipate expectancy
anticipateRPDExpectancy = true

# if to anticipate anomaly
anticipateRPDAnormaly = true
## Appendix B

### Knowledge Specification Syntax

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>?Variable Name</td>
<td>?position, ?color</td>
</tr>
<tr>
<td>Argument</td>
<td>&lt;String</td>
<td>number&gt;</td>
</tr>
<tr>
<td>Comments</td>
<td># comments</td>
<td># rule for direction</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>(+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+ 3 ?remain 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(- 17 4 ?remain)</td>
</tr>
<tr>
<td>Comparison</td>
<td>(&lt;</td>
<td>&lt;=</td>
</tr>
<tr>
<td>String comparison</td>
<td>(eq number</td>
<td>variable number</td>
</tr>
<tr>
<td>Negation</td>
<td>(not &lt;proposition&gt;)</td>
<td>(not (&lt; 16 ?x))</td>
</tr>
<tr>
<td>Condition</td>
<td>([not]&lt;predicate&gt; &lt;arguments</td>
<td>Variables&gt;)</td>
</tr>
<tr>
<td>Condition set</td>
<td>(&lt;</td>
<td>condition</td>
</tr>
<tr>
<td>Fact type</td>
<td>(FactType &lt;type name&gt; (&lt;variable&gt;*))</td>
<td>(FactType rich (?person)</td>
</tr>
<tr>
<td></td>
<td>[[time &lt;time value&gt;]]</td>
<td>(template &quot;?person has a lot of money&quot;) (provider bank 1.0) (time 20)</td>
</tr>
<tr>
<td></td>
<td>[[template &quot;&lt;translations into sentence&gt;&quot;]]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[[sources &lt;seeking plan&gt;*]]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[[needer &lt;name&gt; &lt;value&gt;]<em>)</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[[provider &lt;name&gt; &lt;value&gt;]]<em>)</em></td>
<td></td>
</tr>
<tr>
<td>Rule</td>
<td>(Rule &quot;&lt;rule name&gt;&quot; &lt;Condition set&gt; -) &lt;fact&gt;</td>
<td>(Rule &quot;if a person is rich&quot; (money ?person ?money) (= ?money 100) -) (rich ?person)</td>
</tr>
<tr>
<td>Fact</td>
<td>(Fact &lt;type name&gt; (&lt;value&gt;*))</td>
<td>(Fact money (Alice 201.0) (source (bank)) (time 50)</td>
</tr>
<tr>
<td>Precondition</td>
<td>(precondition &lt;condition set&gt;)</td>
<td>(precondition (next_number ?number ?next))</td>
</tr>
<tr>
<td>Effect</td>
<td>(effect &lt;condition set&gt;)</td>
<td>(effect (current_number ?next))</td>
</tr>
<tr>
<td>Fail condition</td>
<td>(failcondition &lt;condition set&gt;)</td>
<td>(failcondition (&lt; ?to ?from))</td>
</tr>
<tr>
<td>Termination condition</td>
<td>(termcondition &lt;condition set&gt;)</td>
<td>(termcondition (current_number ?to))</td>
</tr>
<tr>
<td>Preference condition</td>
<td>(prefcondition &lt;condition set&gt;)</td>
<td>(prefcondition (&gt; ?number 7))</td>
</tr>
<tr>
<td>Choice</td>
<td>(choice &lt;choice name&gt; (&lt;Preference condition&gt; &lt;step&gt;* (default) &lt;step&gt;)</td>
<td>(choice big_small (prefcondition (&gt; ?number 7)) (print big)) (prefcondition (&lt; ?number 3)) (print small) (default) (print normal))</td>
</tr>
<tr>
<td>Step</td>
<td>(&lt;operator</td>
<td>plan name&gt; (&lt;argument&gt;*))</td>
</tr>
<tr>
<td>Type</td>
<td>Syntax</td>
<td>Example</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Process</td>
<td>(process &lt;step&gt;*)</td>
<td>(process (plan_count_current))</td>
</tr>
<tr>
<td>Contingency plan</td>
<td>(contingency (&lt;plan name&gt;))</td>
<td>(contingency (plan_countback))</td>
</tr>
<tr>
<td>Operator</td>
<td>(operator &lt;operator name&gt; [argument]*</td>
<td>(operator count (?number)</td>
</tr>
<tr>
<td></td>
<td>[Precondition]</td>
<td>(precondition</td>
</tr>
<tr>
<td></td>
<td>[Effect]</td>
<td>(next_number ?number ?next))</td>
</tr>
<tr>
<td>Plan</td>
<td>(plan &lt;plan name&gt; [([argument]*)]</td>
<td>(plan_plan_from_to (?from ?to)</td>
</tr>
<tr>
<td></td>
<td>[Precondition]</td>
<td>(failcondition (?from ?to))</td>
</tr>
<tr>
<td></td>
<td>[Effect]</td>
<td>(contingency (plan_countback))</td>
</tr>
<tr>
<td></td>
<td>[Termination condition]</td>
<td>(termcondition (current_number ?to))</td>
</tr>
<tr>
<td></td>
<td>[Fail condition]</td>
<td>(process</td>
</tr>
<tr>
<td></td>
<td>[Contingency plan]</td>
<td>(plan_count_current)</td>
</tr>
<tr>
<td></td>
<td>&lt;Process&gt;</td>
<td></td>
</tr>
<tr>
<td>Experience</td>
<td>(Experience &lt;experience name&gt;</td>
<td>(Experience e1</td>
</tr>
<tr>
<td></td>
<td>(Cue &lt;Condition set&gt;)</td>
<td>(Cue (near today valentine))</td>
</tr>
<tr>
<td></td>
<td>(Expectancy &lt;Condition set&gt;)</td>
<td>(Expectancy (love ?person me))</td>
</tr>
<tr>
<td></td>
<td>(Anomaly &lt;Condition set&gt;)</td>
<td>(Anomaly (dislike ?person me))</td>
</tr>
<tr>
<td></td>
<td>(Goal &lt;Condition set&gt;)</td>
<td>(Goal (love ?person me))</td>
</tr>
<tr>
<td></td>
<td>(Action &lt;Condition set&gt;)</td>
<td>(Action (plan_send_rose))</td>
</tr>
<tr>
<td></td>
<td>(Result &lt;success</td>
<td>failure&gt;)</td>
</tr>
<tr>
<td>Experience Space</td>
<td>(ExperienceSpace &lt;experience space name&gt;</td>
<td>(ExperienceSpace choose_gift</td>
</tr>
<tr>
<td></td>
<td>(Cue &lt;Condition set&gt;)</td>
<td>(Cue (today ?date) (relation_type ?person))</td>
</tr>
<tr>
<td></td>
<td>(Expectancy &lt;Condition set&gt;)</td>
<td>(Expectancy (happy ?person))</td>
</tr>
<tr>
<td></td>
<td>(Anomaly &lt;Condition set&gt;)</td>
<td>(Anomaly (unhappy ?person))</td>
</tr>
<tr>
<td></td>
<td>(Experience &lt;experience&gt;*</td>
<td>(Experience (gift_holiday)</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>(gift_occasion)</td>
</tr>
<tr>
<td>Capability</td>
<td>(capability (&lt;capability&gt;*))</td>
<td>(capability (plan_send_gift))</td>
</tr>
<tr>
<td>Agent</td>
<td>(agent &lt;agent name&gt;</td>
<td>(agent agent1</td>
</tr>
<tr>
<td></td>
<td>&lt;capability&gt;</td>
<td>(capability (plan_send_gift))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>team</td>
<td>(group</td>
</tr>
<tr>
<td></td>
<td>&lt;capability&gt;</td>
<td>(capability (sending_gift))</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>(role &lt;role name&gt;</td>
<td>(role purchaser</td>
</tr>
<tr>
<td></td>
<td>&lt;capability&gt;</td>
<td>(capability (plan_buy_gift))</td>
</tr>
<tr>
<td></td>
<td>[[constraint &lt;Condition set&gt;]]</td>
<td>(isDedicated true)</td>
</tr>
<tr>
<td></td>
<td>[[isDedicated &lt;true</td>
<td>false&gt;]]</td>
</tr>
<tr>
<td>Simple task</td>
<td>(task &lt;task name&gt;</td>
<td>(task plan_send_gift</td>
</tr>
<tr>
<td></td>
<td>(type &lt;process</td>
<td>decision</td>
</tr>
<tr>
<td></td>
<td>(role (&lt;role name&gt;)*</td>
<td>(role (sender))</td>
</tr>
<tr>
<td></td>
<td>(time &lt;time&gt;)</td>
<td>(time 10)</td>
</tr>
<tr>
<td>Type</td>
<td>Syntax</td>
<td>Example</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Sub task</td>
<td>(subtasks {{&lt;task name&gt; [[predecessors &lt;predecessor name&gt;]]}*})</td>
<td>(subtasks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(decision_choose_gift)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(plan_buy_gift</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(predecessors decision_choose_gift))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Complex task</td>
<td>(task &lt;task name&gt; &lt;Sub task&gt;)</td>
<td>(task sending_gift</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(subtasks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(decision_choose_gift)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(plan_buy_gift</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(predecessors decision_choose_gift))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(plan_send_gift</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(predecessors plan_buy_gift))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
## Appendix C

### R-CAST Commands

Table C-1: R-CAST Commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
<th>Returns</th>
<th>Function</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Predicate</td>
<td>Matched facts</td>
<td>Query the knowledge base</td>
<td>AKB</td>
</tr>
<tr>
<td>assert</td>
<td>Fact</td>
<td>None</td>
<td>Asserting a fact to the knowledge base</td>
<td>AKB</td>
</tr>
<tr>
<td>retract</td>
<td>Predicate</td>
<td>None</td>
<td>Retract all facts that match the predicate</td>
<td>AKB</td>
</tr>
<tr>
<td>naturalQuery</td>
<td>Predicate</td>
<td>Results in sentence</td>
<td>Query and return results in sentence</td>
<td>AKB</td>
</tr>
<tr>
<td>naturalAssert</td>
<td>A sentence</td>
<td>None</td>
<td>Assert according to a template</td>
<td>AKB</td>
</tr>
<tr>
<td>diagnose</td>
<td>Predicate (set)</td>
<td>Predicate (set)</td>
<td>Check the missing information</td>
<td>AKB</td>
</tr>
<tr>
<td>printKB</td>
<td>None</td>
<td>KB Content</td>
<td>Dump the content in the knowledge base</td>
<td>AKB</td>
</tr>
<tr>
<td>parseKB</td>
<td>KB syntax</td>
<td>Depends</td>
<td>Parse the input according to the syntax</td>
<td>AKB</td>
</tr>
<tr>
<td>schedule</td>
<td>Plan + arguments</td>
<td>Process id</td>
<td>Add the plan instance to the process queue for execution</td>
<td>PM</td>
</tr>
<tr>
<td>simulate</td>
<td>Plan + arguments</td>
<td>Process id</td>
<td>Add the plan instance to the process queue for simulation</td>
<td>PM</td>
</tr>
<tr>
<td>terminate</td>
<td>Process id</td>
<td>None</td>
<td>Terminate a on-going process</td>
<td>PM</td>
</tr>
<tr>
<td>assign</td>
<td>Task name</td>
<td>Task id</td>
<td>Assign and schedule a task</td>
<td>TM</td>
</tr>
<tr>
<td>attend</td>
<td>Experience space</td>
<td>None</td>
<td>Set current recognition to a particular experience space</td>
<td>RPD Decision-maker</td>
</tr>
<tr>
<td>ping</td>
<td>Agent name</td>
<td>None</td>
<td>Check the communication link between two agents</td>
<td>CM</td>
</tr>
<tr>
<td>investigate</td>
<td>Predicate</td>
<td>Information requirement</td>
<td>Add a new requirement</td>
<td>IM</td>
</tr>
<tr>
<td>anticipate</td>
<td>Plan or experience</td>
<td>Information requirement</td>
<td>Add requirements according to a process/decision model</td>
<td>IM</td>
</tr>
<tr>
<td>help</td>
<td>None</td>
<td>Command list</td>
<td>List all the commands</td>
<td>Agent</td>
</tr>
<tr>
<td>clean</td>
<td>None</td>
<td>None</td>
<td>Remove history data</td>
<td>Agent</td>
</tr>
<tr>
<td>step</td>
<td>None</td>
<td>None</td>
<td>Step forward one decision cycle</td>
<td>Agent</td>
</tr>
<tr>
<td>start</td>
<td>None</td>
<td>None</td>
<td>Start all components</td>
<td>Agent</td>
</tr>
<tr>
<td>stop</td>
<td>None</td>
<td>None</td>
<td>Stop all components</td>
<td>Agent</td>
</tr>
<tr>
<td>hide</td>
<td>None</td>
<td>None</td>
<td>Hide the agent interface</td>
<td>Agent</td>
</tr>
<tr>
<td>show</td>
<td>None</td>
<td>None</td>
<td>Display the agent interface window</td>
<td>Agent</td>
</tr>
<tr>
<td>quit/exit</td>
<td>None</td>
<td>None</td>
<td>Kill the agent instance</td>
<td>Agent</td>
</tr>
<tr>
<td>list/print</td>
<td>None</td>
<td>None</td>
<td>Show all configuration settings</td>
<td>Agent</td>
</tr>
<tr>
<td>set</td>
<td>Property + value</td>
<td>None</td>
<td>Set the configuration property value</td>
<td>Agent</td>
</tr>
<tr>
<td>get</td>
<td>Property name</td>
<td>Property value</td>
<td>Show the property value</td>
<td>Agent</td>
</tr>
</tbody>
</table>
## Appendix D

### R-CAST UML Design Diagrams

Table **D-1**: R-CAST Design UML Overview.

<table>
<thead>
<tr>
<th>Figure</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure D-1</td>
<td>Whiteboard class diagram</td>
</tr>
<tr>
<td>Figure D-2</td>
<td>AKB class diagram</td>
</tr>
<tr>
<td>Figure D-3</td>
<td>AKB sequential diagram</td>
</tr>
<tr>
<td>Figure D-4</td>
<td>PM class diagram</td>
</tr>
<tr>
<td>Figure D-5</td>
<td>PM sequential diagram</td>
</tr>
<tr>
<td>Figure D-6</td>
<td>RDP class diagram</td>
</tr>
<tr>
<td>Figure D-7</td>
<td>RDP sequential diagram</td>
</tr>
<tr>
<td>Figure D-8</td>
<td>TM class diagram</td>
</tr>
<tr>
<td>Figure D-9</td>
<td>CM class diagram</td>
</tr>
<tr>
<td>Figure D-10</td>
<td>CM sequential diagram</td>
</tr>
<tr>
<td>Figure D-11</td>
<td>IM class diagram</td>
</tr>
<tr>
<td>Figure D-12</td>
<td>IM sequential diagram</td>
</tr>
</tbody>
</table>
Figure D-1: Whiteboard class diagram.
Figure D-2: AKB class diagram.
Figure D-3: AKB sequential diagram.
Figure D-4: PM class diagram.
Figure D-5: PM sequential diagram.
Figure D-6: RPD class diagram.
Figure D-7: RPD sequential diagram.
Figure D-8: TM class diagram.
Figure D-9: CM class diagram.
Figure D-10: CM sequential diagram.
Figure D-11: IM class diagram.
Figure D-12: IM sequential diagram.
Figure D-13: Auctioneer class diagram.
Appendix E

DDD Domain Agent Models

Table E-1: Overview of the File Names and Purposes.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2.kb</td>
<td>S2 inference knowledge</td>
</tr>
<tr>
<td>S2.process</td>
<td>S2 plan knowledge</td>
</tr>
<tr>
<td>S4.kb</td>
<td>S4 inference knowledge</td>
</tr>
<tr>
<td>S4.process</td>
<td>S4 plan knowledge</td>
</tr>
<tr>
<td>S3.kb</td>
<td>S3 inference knowledge</td>
</tr>
<tr>
<td>S3.process</td>
<td>S3 plan knowledge</td>
</tr>
<tr>
<td>S3.eb</td>
<td>S3 experience knowledge</td>
</tr>
</tbody>
</table>

S2.kb

;************************** common domain fact types
(FactType stop_game (?ifTure))
(FactType enemy_location (?enemy ?x ?y))
(FactType task (?id ?x ?y))
(FactType task_class (?id ?class)
  (needer (S3 0.6)))
)
(FactType task_disappeared (?true))
(FactType task_speed (?id ?vx ?vy))
(FactType task_direction (?id ?direction))
(FactType task_strength (?id ?strength))
(FactType asset (?id ?x ?y))
(FactType asset_status (?id ?status))
(FactType waypoint (?id ?x ?y))
(FactType at_asset_waypoint (?asset ?point))
(FactType is_enemy (?task))
(FactType is_neutral (?task))
;************************** S2 domain fact types
(FactType pursue_position (?task ?x ?y))
(FactType unidentified_task (?task))
(FactType identifiable_task (?task ?asset))
(FactType id_range (?asset ?range))
(FactType target_in_id_range (?task ?asset))
(FactType is_identified (?if_true))
(FactType fused_task (?target))

;************************** Rules
(Rule "at_asset_wp"
  (asset ?asset ?ax ?ay)
  (waypoint ?point ?px ?py)
  (< ?dis 0.05)
  ->
  (at_asset_waypoint ?asset ?point))
(Rule "enemy_location"
(task ?task ?x ?y)
(is_enemy ?task)
->
(enemy_location ?task ?x ?y))

(Rule "pursue_position"
(task ?task ?x ?y)
(+ ?y 0.09 ?py)
->
(pursue_position ?task ?x ?py))

;************************************************ direction
(Rule "task_away"
(task_speed ?task ?vx ?vy)
(< ?vy 0.001)
->
(task_direction ?task away))

(Rule "task_airport1"
(task_speed ?task ?vx ?vy)
(> ?vx 0)
->
(task_direction ?task airport))

(Rule "task_obj"
(task ?task ?x ?y)
(task_speed ?task ?vx ?vy)
(< ?x 0.65)
(= ?vx 0)
->
(task_direction ?task obj))

(Rule "task_airport2"
(task ?task ?x ?y)
(task_speed ?task ?vx ?vy)
(> ?x 0.65)
(= ?vx 0)
->
(task_direction ?task airport))

;************************************************ class or types
(Rule "is_enemy1"
(task_class ?task ?class)
(eq ?class GE0)
->
(is_enemy ?task))

(Rule "is_enemy2"
(task_class ?task ?class)
(eq ?class GE1)
->
(is_enemy ?task))

(Rule "is_neutral1"
(task_class ?task ?class)
(eq ?class GN0)
->
(is_neutral ?task))

(Rule "is_neutral2"
(task_class ?task ?class)
(eq ?class GN1)
->
(is_neutral ?task))

(Rule "is_invisible"
(task ?task ?x ?y)
(> ?y 0.75)
->
(invisible_task ?task))

;************************************************ S2 rules
(Rule "ided1"
(is_enemy ?task)
->
(is_identified true))

(Rule "ided2"
(is_neutral ?task)
->
(is_identified true))

(Rule "unid_task"
(task_class ?task unknown)
(not (task_direction ?task away))
(task ?task ?x ?y)
(< ?y 0.75)
->
(unidentified_task ?task))

(Rule "in_range2"
(unidentified_task ?task)
(task ?task ?ex ?ey)
(asset ?asset ?ax ?ay)
(id_range ?asset ?range)
(< ?dis ?range)
->
(target_in_id_range ?task ?asset))

(Rule "idable"
(unidentifiable_task ?task)
(target_in_id_range ?task ?asset)
->
(identifiable_task ?task ?asset))

;************************************************ facts about the ddd domain
(Fact waypoint (obj 0.1500 0.8750))
(Fact waypoint (airport 0.8500 0.8750))
(Fact stop_game (false))

(Fact id_range (UA 0.3))
(Fact waypoint (wp_s2 0.7000 0.2500))
**S2.process**

- **operators**
  - (operator print)
  - (operator launch (?asset ?base)
    - (precondition (asset_status ?asset hangar)))
  - (operator attack (?target ?asset)
    - (precondition (target_in_range ?target ?asset))
    - (effect (attacked_task ?target)))
  - (operator identify (?target))
  - (operator fusion (?target ?dm)
    - (effect (fused_task ?target)))
  - (operator move (?asset ?x ?y ?throttle)
    - (precondition (not (asset_status ?asset hangar))))
  - (operator stop (?asset))

- **basic plans**
  - (plan plan_move_to_waypoint (?asset ?destination ?speed)
    - (precondition (waypoint ?destination ?x ?y))
    - (termcondition (at_asset_waypoint ?asset ?destination))
    - (process
      - (move ?asset ?x ?y ?speed)))
  - (plan plan_move_from_to_waypoint (?asset ?origin ?destination ?speed)
    - (precondition (at_asset_waypoint ?asset ?origin))
    - (termcondition (at_asset_waypoint ?asset ?destination))
    - (process
      - (plan_move_to_waypoint ?asset ?destination ?speed)))
  - (plan plan_launch (?asset ?base)
    - (precondition (asset_status ?asset hangar))
    - (termcondition (asset_status ?asset station))
    - (process
      - (launch ?asset ?base)))

- **patrol around the center, identify task if any one found**
  - (plan plan_patrol (?asset ?base ?speed ?wp)
    - (process
      - (choice patrol_or_scout
        - ((precondition (unidentified_task ?task))
          (plan_identify_task ?asset))
        - (default) (plan_move_to_center ?asset ?speed ?wp))))
  - (plan plan_move_to_center (?asset ?speed ?wp)
    - (termcondition (unidentified_task ?task))
    - (process
      - (plan_move_to_waypoint ?asset ?wp ?speed)))
  - (plan plan_identify_task (?asset)
    - (precondition (unidentified_task ?task))
    - (termcondition (invisible_task ?task) (task_direction task away) (fused_task ?task))
    - (process
      - (plan_pursue_a_task ?task)
      - (plan_identify_a_task ?asset ALL)))
  - (plan plan_pursue_a_task (?task ?asset)
    - (precondition (unidentified_task ?task)
      (pursue_position ?task ?x ?py))
    - (termcondition (identifiable_task ?asset))
    - (process
      - (move ?asset ?x ?py 1)))

- **identify and transfer/fuse information to dm**
  - (plan plan_identify_a_task (?asset ?dm)
    - (precondition (identifiable_task ?asset))
    - (termcondition (fused_task ?asset))
    - (process
      - (identify ?task)
      - (fusion ?task ?dm)))
  - (plan plan_move_to_target (?asset ?x ?py)
    - (precondition (unidentified_task ?asset)
      (pursue_position ?asset ?x ?py))
    - (termcondition (identifiable_task ?task1 ?asset))
    - (process
      - (move ?asset ?x ?py 1)))

; don't have forecast capabilities
(plan plan_launch_asset
    (process
        (plan_launch UA H2)
    )
)

(plan plan_scout
    (process
        (plan_launch UA H2)
        (plan_patrol UA H2 1 wp_s2)
    )
)
S4.kb

；************************** common domain fact types
(FactType stop_game (?ifTure))
(FactType enemy_location (?enemy ?x ?y))
(FactType task (?id ?x ?y))
(FactType task_class (?id ?class))
(FactType task_disappeared (?true))
(FactType task_speed (?id ?vx ?vy))
(FactType task_direction (?id ?direction))
(FactType task_strength (?id ?strength))
(FactType asset (?id ?x ?y))
(FactType asset_status (?id ?status))
(FactType waypoint (?id ?x ?y))
(FactType at_asset_waypoint (?asset ?point))
(FactType is_enemy (?task))
(FactType is_neutral (?task))
(FactType invisible_task (?task))

；************************** S4 domain fact types
(FactType is_threat (?task))
(FactType escape_direction (?task ?dir))
(FactType threatened_area (?task ?point))
(FactType be_fire_range (?asset ?range))
(FactType in_enemy_fire_range (?task ?asset))

；************************** Rules *
(Rule "enemy_location"
  (task ?task ?x ?y)
  (is_enemy ?task)
  ->
  (enemy_location ?task ?x ?y))

；************************** direction
(Rule "task_away"
  (task_speed ?task ?vx ?vy)
  (< ?vy 0.001)
  ->
  (task_direction ?task away))

(Rule "task_airport1"
  (task_speed ?task ?vx ?vy)
  (> ?vx 0)
  ->
  (task_direction ?task airport))

(Rule "task_obj"
  (task ?task ?x ?y)
  (task_speed ?task ?vx ?vy)
  (< ?x 0.65)
  (= ?vx 0)
  ->
  (task_direction ?task obj))

(Rule "task_airport2"
  (task ?task ?x ?y)
  (task_speed ?task ?vx ?vy)
  (> ?x 0.65)
  (= ?vx 0)
  ->
  (task_direction ?task airport))

；************************** class or types
(Rule "is_enemy1"
  (task_class ?task ?class)
  (eq ?class GE0)
  ->
  (is_enemy ?task))

(Rule "is_enemy2"
  (task_class ?task ?class)
  (eq ?class GE1)
  ->
  (is_enemy ?task))

(Rule "is_neutral1"
  (task_class ?task ?class)
  (eq ?class GN0)
  ->
  (is_neutral ?task))
(Rule "is_neutral2"
  (task_class ?task ?class)
  (eq ?class GN1)
  ->
  (is_neutral ?task))

(Rule "is_invisible"
  (task ?task ?x ?y)
  (> ?y 0.75)
  ->
  (invisible_task ?task))

;**************************
S4 domain rules

(Rule "is_threat1"
  (task_class ?task ?class)
  (eq ?class unknown)
  (not (task_direction ?task away))
  ->
  (is_threat ?task))

(Rule "is_threat2"
  (is_enemy ?task)
  (not (task_direction ?task away))
  ->
  (is_threat ?task))

(Rule "threatened_area1"
  (is_threat ?task)
  (task_direction ?task airport)
  ->
  (threatened_area ?task airport))

(Rule "threatened_area2"
  (is_threat ?task)
  (task_direction ?task obj)
  ->
  (threatened_area ?task obj))

(Rule "escape_obj"
  (threatened_area ?task1 airport)
  ->
  (escape_direction ?task1 obj))

(Rule "escape_air"
  (threatened_area ?task1 obj)
  ->
  (escape_direction ?task1 airport))

(Rule "in_range3"
  (task ?task ?ex ?ey)
  (is_threat ?task)
  (asset ?asset ?ax ?ay)
  (be_fire_range ?asset ?range)
  (task_speed ?task ?vx ?vy)
  (> ?vx 0.001)
  (> ?vy 0.001)
  (< ?dis ?range)
  ->
  (in_enemy_fire_range ?task ?asset))

(Rule "in_range4"
  (task ?task ?ex ?ey)
  (is_threat ?task)
  (task_direction ?task airport)
  (task_speed ?task ?vx ?vy)
  (> ?vx 0.001)
  (> ?vy 0.001)
  (asset ?asset ?ax ?ay)
  (< ?dis 0.8)
  ->
  (in_enemy_fire_range ?task ?asset))

;**********************
facts about the ddd domain

(Fact waypoint (obj 0.1500 0.8750))
(Fact waypoint (airport 0.8500 0.8750))
(Fact stop_game (false))

(Fact be_fire_range (TU 0.63))
(Fact waypoint (wp_mid_o 0.3000 0.8750))
(Fact waypoint (wp_mid 0.5000 0.8750))
S4.process

;*******************************operators***************
(operator print)

(operator launch (?asset ?base)
  (precondition (asset_status ?asset hangar)))
)

(operator attack (?target ?asset)
  (precondition (target_in_range ?target ?asset))
  (effect (attacked_task ?target)))
)

(operator identify (?target)
)

(operator fusion (?target ?dm)
  (effect (fused_task ?target)))
)

(operator move (?asset ?x ?y ?throttle)
  (precondition (not (asset_status ?asset hangar))))
)

(operator stop (?asset)
)

;*******************************basic plans***
(plan plan_move_to_waypoint (?asset ?destination ?speed)
  (precondition (waypoint ?destination ?x ?y))
  (termcondition (at_asset_waypoint ?asset ?destination))
  (process
    (move ?asset ?x ?y ?speed)
  ))
)

(plan plan_move_from_to_waypoint (?asset ?origin ?destination ?speed)
  (precondition (at_asset_waypoint ?asset ?origin))
  (termcondition (at_asset_waypoint ?asset ?destination))
  (process
    (move_to_waypoint ?asset ?destination ?speed)
  ))
)

(plan plan_launch (?asset ?base)
  (precondition (asset_status ?asset hangar))
  (termcondition (asset_status ?asset station)))
)

(plan plan_delivery
  (process
    (plan_launch TU H4)
    (plan_move_between_waypoint TU 1 airport obj)
  ))
)

(plan plan_move_between_waypoint (?asset ?speed ?wp1 ?wp2)
  (termcondition (stop_game ture))
  (process
    (plan_deliver_safly_to_waypoint ?asset ?wp1 ?speed)
    (plan_deliver_safly_to_waypoint ?asset ?wp2 ?speed)
  ))
)

(plan plan_deliver_safly_to_waypoint (?asset ?wp ?speed)
  (termcondition (at_asset_waypoint ?asset ?wp))
  (process
    (choice delivery_or_escape
      ((precondition (in_enemy_fire_range ?task ?asset))
        (plan_escape ?asset ?wp))
      ((default) (plan_move_safly_to_waypoint ?asset ?wp ?speed)))
  ))
)

(plan plan_move_safly_to_waypoint (?asset ?wp ?speed)
  (termcondition (in_enemy_fire_range ?task ?asset))
  (process
    (plan_move_to_waypoint ?asset ?wp ?speed)
  ))
)

(plan plan_escape (?asset ?wp)
  (precondition (in_enemy_fire_range ?task ?asset))
  (termcondition (task_disappeared true)
    (not (in_enemy_fire_range ?task ?asset)))
  (process
(plan_escape_from_enemy ?asset ?task ?wp)
  (stop ?asset)
)
)

(plan plan_escape_from_enemy (?asset ?task ?wp)
  (termcondition (task_direction ?task away))
  (process
    (choice wp_or_mid
      ((prefcondition (escape_direction ?task ?wp)
        (not (threatened_area ?task2 ?wp)))
        (plan_move_to_waypoint ?asset ?wp 1))
      ((default) (plan_move_to.waypoint ?asset wp_mid 1)))))
)
)
S3.kb

************************** common domain fact types
(FactType stop_game (?ifTure))
(FactType enemy_location (?enemy ?x ?y))
(FactType task (?id ?x ?y))
(FactType task_class (?id ?class))
(FactType task_disappeared (?true))
(FactType task_speed (?id ?vx ?vy))
(FactType task_direction (?id ?direction))
(FactType task_strength (?id ?strength))
(FactType asset (?id ?x ?y))
(FactType asset_status (?id ?status))
(FactType waypoint (?id ?x ?y))
(FactType at_asset_waypoint (?asset ?point))
(FactType is_enemy (?task))
(FactType is_neutral (?task))
(FactType invisible_task (?task))

************************** S3 domain fact types
(FactType fire_range (?asset ?range))
(FactType target_in_range (?task ?asset))
(FactType attacked_task (?task))
(FactType moving_pattern (?pattern))
(FactType attackable_task (?task ?asset))
(FactType attack_pattern (?pattern))

************************** Rules

(Rule "at_asset_wp"
 (asset ?asset ?ax ?ay)
 (waypoint ?point ?px ?py)
 (< ?dis 0.05)
 ->
 (at_asset_waypoint ?asset ?point))

(Rule "enemy_location"
 (task ?task ?x ?y)
 (is_enemy ?task)
 ->
 (enemy_location ?task ?x ?y))

************************** direction

(Rule "task_away"
 (task_speed ?task ?vx ?vy)
 (< ?vy 0.001)
 ->
 (task_direction ?task away))

(Rule "task_airport1"
 (task_speed ?task ?vx ?vy)
 (> ?vx 0)
 ->
 (task_direction ?task airport))

(Rule "task_obj"
 (task ?task ?x ?y)
 (task_speed ?task ?vx ?vy)
 (< ?x 0.19)
 (= ?vx 0)
 ->
 (task_direction ?task obj))

(Rule "task_airport2"
 (task ?task ?x ?y)
 (task_speed ?task ?vx ?vy)
 (> ?x 0.21)
 (= ?vx 0)
 ->
 (task_direction ?task airport))

(Rule "moving_dir_h"
 (task_direction ?task1 airport)
 (task_direction ?task2 obj)
 ->
 (moving_pattern h))

(Rule "moving_dir_v"
 (task_direction ?task1 airport)
 (task_direction ?task2 airport)
 (not (eq ?task2 ?task1))
 ->
 (moving_pattern v))

************************** class or types
(Rule "is_enemy1"
 (task_class ?task ?class)
 (eq ?class GE0)
 ->
 (is_enemy ?task))

(Rule "is_enemy2"
 (task_class ?task ?class)
 (eq ?class GE1)
 ->
 (is_enemy ?task))

(Rule "is_neutral1"
 (task_class ?task ?class)
 (eq ?class GN0)
 ->
 (is_neutral ?task))

(Rule "is_neutral2"
 (task_class ?task ?class)
 (eq ?class GN1)
 ->
 (is_neutral ?task))

(Rule "is_invisible"
 (task ?task ?x ?y)
 (> ?y 0.75)
 ->
 (invisible_task ?task))

;************************** S3 domain rules

(Rule "in_range1"
 (enemy_location ?task ?ex ?ey)
 (asset ?asset ?ax ?ay)
 (fire_range ?asset ?range)
 (> ?ey 0.5)
 (< ?dis ?range)
 ->
 (target_in_range ?task ?asset))

(Rule "attackable"
 (target_in_range ?task ?asset)
 (not (attacked_task ?task))
 ->
 (attackable_task ?task ?asset))

(Rule "disappeared"
 (task_disappeared true)
 ->
 (moving_pattern null))

(Rule "disappeared"
 (invisible_task ?task)
 ->
 (moving_pattern null))

(Rule "pattern1"
 (moving_pattern v)
 ->
 (attack_pattern vxx))

(Rule "pattern2"
 (moving_pattern k)
 ->
 (attack_pattern vxx))

(Rule "pattern3"
 (moving_pattern h)
 (is_enemy ?task)
 (task_direction ?task airport)
 ->
 (attack_pattern hxe))

(Rule "pattern4"
 (moving_pattern h)
 (is_enemy ?task)
 (task_direction ?task obj)
 ->
 (attack_pattern hex))

(Rule "pattern5"
 (moving_pattern h)
 (is_neutral ?task)
 (task_direction ?task airport)
 ->
 (attack_pattern hxn))

(Rule "pattern6"
 (moving_pattern ii)
 ->
 (attack_pattern hxn))

(Rule "pattern7"
 (moving_pattern h)
 (is_neutral ?task)
 (task_direction ?task obj)
 ->
 (attack_pattern hnx))

(Rule "pattern8"
 (moving_pattern ji)
 ->
 (attack_pattern hnx))

(Rule "pattern9"
(task_class ?task1 ?class1)
(task_class ?task2 ?class2)
(not (eq ?task1 ?task2))
->
(attack_pattern xxx))

************************* facts about the ddd domain
(Fact waypoint (obj 0.1500 0.8750))
(Fact waypoint (airport 0.8500 0.8750))
(Fact stop_game (false))
(Fact fire_range (TK 0.3))
(Fact waypoint (wp_s3 0.5000 0.6500))
(Fact waypoint (wp_s3_a 0.6000 0.6500))
(Fact waypoint (wp_s3_r 0.4000 0.6500))
(Fact moving_pattern (null))
S3.process

;***************************************************************operators***************
(operator print)

(operator launch (?asset ?base)
  (precondition (asset_status ?asset hangar)))
)

(operator attack (?target ?asset)
  (precondition (attackable_task ?task ?asset))
  (effect (attacked_task ?target)))
)

(operator identify (?target)
)

(operator fusion (?target ?dm)
  (effect (fused_task ?target)))
)

(operator move (?asset ?x ?y ?throttle)
  (precondition (not (asset_status ?asset hangar))))
)

(operator stop (?asset)
)

;***************************************************************basic plans*
(plan plan_move_to_waypoint (?asset ?destination ?speed)
  (precondition (waypoint ?destination ?x ?y))
  (termcondition (at_asset_waypoint ?asset ?destination))
  (process
    (move ?asset ?x ?y ?speed)
  )
)

(plan plan_move_from_to_waypoint (?asset ?origin ?destination ?speed)
  (precondition (at_asset_waypoint ?asset ?origin))
  (termcondition (at_asset_waypoint ?asset ?destination))
  (process
    (plan_move_to_waypoint ?asset ?destination ?speed)
  )
)

(plan plan_launch (?asset ?base)
  (precondition (asset_status ?asset hangar))
  (termcondition (asset_status ?asset station))
  (process
    (launch ?asset ?base)
  )
)

;processes for S3 agent
;******************************************************************************
; plans needed from the experience KB
(plan plan_move_to_mid
  (termcondition (attack_pattern ?pattern))
  (process
    (plan_move_to_waypoint TK wp_s3 1)
  )
)

(plan plan_move_to_A_attack
  (termcondition (moving_pattern null))
  (process
    (plan_move_to_waypoint TK wp_s3_a 1)
    (plan_attack_enemy)
  )
)

(plan plan_move_to_A_attack_then_R_attack
  (termcondition (moving_pattern null))
  (process
    (plan_move_to_waypoint TK wp_s3_a 1)
    (plan_attack_enemy)
    (plan_move_to_waypoint TK wp_s3_r 1)
    (plan_attack_enemy)
  )
)

(plan plan_move_to_R_attack_then_A_attack
  (termcondition (moving_pattern null))
  (process
    (plan_move_to_waypoint TK wp_s3_r 1)
    (plan_attack_enemy)
    (plan_move_to_waypoint TK wp_s3_a 1)
    (plan_attack_enemy)
  )
)

(plan plan_attack_enemy
  (precondition (attackable_task ?task ?asset))
  (termcondition (attacked_task ?task))
  (effect (not (target_in_range ?task TK)))
  (process
    (attack ?task TK)
  )
)
(plan plan_combat
  (termcondition (asset_status TK station))
  (process
    (plan_launch TK H3)
    (plan_escort TK)
    (plan_escort (?asset)
      (termcondition (stop_game ture))
      (process
        (choice patrol_or_attack
          ((prefcondition (moving_pattern null))
            (plan_move_to_waypoint TK wp_s3 1))
          ((default) (plan_attack_mode TK))
        )
      )
    )
  )
)

(plan plan_attack_mode (?asset)
  (termcondition (moving_pattern null))
  (process
    (choice attack_strategy
      ((prefcondition (attack_pattern vxx))
        (plan_move_to_A_attack))
      ((prefcondition (attack_pattern hxe))
        (plan_move_to_A_attack_then_R_attack))
      ((prefcondition (attack_pattern hex))
        (plan_move_to_R_attack_then_A_attack))
      ((prefcondition (attack_pattern hxn))
        (plan_move_to_R_attack_then_A_attack))
      ((prefcondition (attack_pattern hnx))
        (plan_move_to_A_attack_then_R_attack))
      ((default) (plan_move_to_waypoint TK wp_s3 1))
    )
  )
)

(ExperienceSpace combat_experience
(Cue (stop_game false))
(Expectancy (stop_game false))
(Anomaly (stop_game true))
(Experience (es-escort) (es-combat))
)

(ExperienceSpace es-combat
(Cue (attack_pattern ?pattern))
(Expectancy (attack_pattern ?pattern))
(Anomaly (moving_pattern null))
(Experience (e-vxx) (e-hxe) (e-hxn) (e-hnx) (e-xxx))
)

(ExperienceSpace es-escort
(Cue (moving_pattern null))
(Expectancy (moving_pattern null))
(Anomaly (attack_pattern ?pattern))
(Experience (e-mid))
)

############################ Experiences

(Experience e-mid
(Cue (moving_pattern null))
(Expectancy (task_disappeared true))
(Anomaly (attack_pattern ?pattern))
(Goal (attacked_task ?task))
(Action (plan_move_to_mid))
(Result success)
)

(Experience e-vxx
(Cue (attack_pattern vxx))
(Expectancy (attack_pattern vxx))
(Anomaly (moving_pattern null))
(Goal (attacked_task ?task))
(Action (plan_move_to_A_attack))
(Result success)
)

(Experience e-hex
(Cue (attack_pattern hex))
(Expectancy (attack_pattern hex))
(Anomaly (moving_pattern null))
(Goal (attacked_task ?task))
(Action (plan_move_to_R_attack_then_A_attack))
(Result success)
)

(Experience e-hxn
(Cue (move_pattern hnx))
(Expectancy (move_pattern hnx))
(Anomaly (move_pattern null))
(Goal (attacked_task ?task))
(Action (plan_move_to_R_attack_then_A_attack))
(Result success)
)

(Experience e-hxe
(Cue (move_pattern hxe))
(Expectancy (move_pattern hxe))
(Anomaly (move_pattern null))
(Goal (attacked_task ?task))
(Action (plan_move_to_A_attack_then_R_attack))
(Result success)
)

(Experience e-hxn
(Cue (move_pattern hnx))
(Expectancy (move_pattern hnx))
(Anomaly (move_pattern null))
(Goal (attacked_task ?task))
(Action (plan_move_to_A_attack_then_R_attack))
(Result success)
)

(Experience e-xxx
(Cue (attack_pattern xxx) (stop_game true))
(Expectancy (attack_pattern xxx))
(Anomaly (attack_pattern hnx))
(Goal (attacked_task ?task))
(Action (plan_move_to_mid))
(Result success)
)
# Appendix F

## Acronym Index

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABLE</td>
<td>Agent Building And Learning Environment</td>
</tr>
<tr>
<td>ACL</td>
<td>Agent Communication Languages</td>
</tr>
<tr>
<td>ACT-R&lt;sup&gt;16&lt;/sup&gt;</td>
<td>Atomic Components of Thought - Rational</td>
</tr>
<tr>
<td>KBB&lt;sup&gt;17&lt;/sup&gt;</td>
<td>Active Knowledge Base</td>
</tr>
<tr>
<td>AMBR</td>
<td>Agent-Based Modeling And Behavior Representation</td>
</tr>
<tr>
<td>AUML</td>
<td>Agent Unified Modeling Language</td>
</tr>
<tr>
<td>BDI</td>
<td>Belief-Desire-Intention</td>
</tr>
<tr>
<td>BFA</td>
<td>Battle Functional Area</td>
</tr>
<tr>
<td>BOM</td>
<td>Bill Of Materials</td>
</tr>
<tr>
<td>CAST</td>
<td>Collaborative Agents for Simulating Teamwork</td>
</tr>
<tr>
<td>CBG</td>
<td>Color Block Game</td>
</tr>
<tr>
<td>CM</td>
<td>Communication Manager</td>
</tr>
<tr>
<td>COA</td>
<td>Course Of Actions</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DARPA</td>
<td>Defense Advanced Research Projects Agency</td>
</tr>
<tr>
<td>DDD</td>
<td>Distributed Dynamic Decision-Making</td>
</tr>
<tr>
<td>EPIC</td>
<td>Executive Process-Interactive Control</td>
</tr>
<tr>
<td>FBI</td>
<td>Federal Bureau of Investigation</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation For Intelligent Physical Agents</td>
</tr>
<tr>
<td>GRATE</td>
<td>Generic Rules and Agent Testbed Environment</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IDR</td>
<td>Information Dependency Relation</td>
</tr>
<tr>
<td>IDR-L</td>
<td>Information Dependency Relation - Leakage</td>
</tr>
<tr>
<td>IM</td>
<td>Information Manager</td>
</tr>
<tr>
<td>IRP</td>
<td>Information Requirement Planning</td>
</tr>
<tr>
<td>ISC</td>
<td>Information Supply Chain</td>
</tr>
<tr>
<td>J2EE JMS</td>
<td>Java 2 Enterprise Edition Java Message Service</td>
</tr>
<tr>
<td>JADE</td>
<td>Java Agent Development Environment</td>
</tr>
</tbody>
</table>

<sup>16</sup> Some acronyms such as ACT-R, Soar, and Jess are become more popular than what they stand for.

<sup>17</sup> An acronym in a **bold** font indicates being created in this thesis.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAVA RMI</td>
<td>Java Remote Method Invocation</td>
</tr>
<tr>
<td>Jess</td>
<td>Java Expert System Shell</td>
</tr>
<tr>
<td>JIT</td>
<td>Joint Intention Theory</td>
</tr>
<tr>
<td>JIT</td>
<td>Just-In-Time</td>
</tr>
<tr>
<td>KB</td>
<td>Knowledge Base</td>
</tr>
<tr>
<td>KQML</td>
<td>Knowledge Query and Manipulation Language</td>
</tr>
<tr>
<td>LARK</td>
<td>Language for Advertisement and Request for Knowledge Sharing</td>
</tr>
<tr>
<td>LHS</td>
<td>Left Handed Side</td>
</tr>
<tr>
<td>MALLEET</td>
<td>Multi-Agent Logic Language for Encoding Teamwork</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>MaSE</td>
<td>Multi-Agent Systems Engineering</td>
</tr>
<tr>
<td>OAA</td>
<td>Open Agent Architecture</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PIP</td>
<td>Public Interface Process</td>
</tr>
<tr>
<td>PM</td>
<td>Process Manager</td>
</tr>
<tr>
<td>R-CAST</td>
<td>RPD-enabled Collaborative Agents Simulating Teamwork</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RESINTA</td>
<td>Reusable Environment for Structured Intelligent Networked Task Agents</td>
</tr>
<tr>
<td>RFQ</td>
<td>Request For Quote</td>
</tr>
<tr>
<td>RHS</td>
<td>Right Handed Side</td>
</tr>
<tr>
<td>RM</td>
<td>Resource Manager</td>
</tr>
<tr>
<td>RPD</td>
<td>Recognition Primed Decision</td>
</tr>
<tr>
<td>SCM</td>
<td>Supply Chain Management</td>
</tr>
<tr>
<td>SHARE</td>
<td>System-Wide Homeland Analysis and Resource Exchange</td>
</tr>
<tr>
<td>SHOE</td>
<td>Simple Html Ontology Extensions</td>
</tr>
<tr>
<td>SMM</td>
<td>Shared Mental Model</td>
</tr>
<tr>
<td>Soar</td>
<td>State, Operator, And Result</td>
</tr>
<tr>
<td>SPT</td>
<td>Shared Plans Theory</td>
</tr>
<tr>
<td>STEAM</td>
<td>Shell for TEAMwork</td>
</tr>
<tr>
<td>TAC</td>
<td>Trading Agent Competition</td>
</tr>
<tr>
<td>TIS</td>
<td>Task Information Subscribe</td>
</tr>
<tr>
<td>TM</td>
<td>Task Manager</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>UTC</td>
<td>Unified Theory of Cognition</td>
</tr>
<tr>
<td>VMI</td>
<td>Vendor Managed Inventory</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
VITA

Shuang Sun

College of Information Sciences and Technology
The Pennsylvania State University

Education

Ph.D. Information Sciences and Technology  Aug. 2006
The Pennsylvania State University, University Park

B.S. Electrical Engineering  Aug. 1994
Dalian University of Technology

Experience


Research & Teaching Assistant


Project Manager & SAP Consultant


Consultant


Assistant Engineer

Affiliations

Member of ACM

Member of IEEE