# THE KNOWLEDGE COLLECTIVE: A MULTI-LAYER, MULTI-AGENT FRAMEWORK FOR INFORMATION MANAGEMENT IN AN INTELLIGENT KNOWLEDGE BASE

BY

JAY ALAN YUSKO

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
in the Graduate College of the
Illinois Institute of Technology

Approved _____
Adviser

Approved _____
Co-Adviser

Chicago, Illinois
July 2005

# ACKNOWLEDGMENT

I want to thank my advisor, Professor Martha Evens, and all the members of my thesis committee. I especially want to thank my loving wife, Carol, for all the things she has had to put up with while I was working on this thesis.

I would like to thank ILOG® for supplying their JRules™ System and IBM® for supplying their WebSphere QualityStage product for this work since both are commercial products.

This work was partially supported by the Cognitive Science Program, Office of Naval Research under Grants No. N00014-94-1-0338 and N00014-02-1-0442 to Illinois Institute of Technology. The content does not reflect the position or policy of the government and no official endorsement should be inferred.

J.A.Y.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

# LIST OF ABBREVIATIONS

| Abbreviation | Term |
|---|---|
| API | Application Programming Interface |
| BOM | Business Object Model |
| BRMS | Business Rule Management System |
| CBV | Central Blood Volume |
| CLOS | Common Lisp Object System |
| FIPA | Foundation for Intelligent Agents |
| IIT | Illinois Institute of Technology |
| ITS | Intelligent Tutoring System |
| JMS | Java Message Service |
| MAINTEX | Maintenance Expert |
| MAS | Multi-Agent System |
| ONTEX | Ontology Expert |
| ONTIE | Ontology Inference Engine |
| PIT | Mean Intrathoracic Pressure |
| QR | Qualitative Reasoning |
| RV | Venous Resistance |
| SOP | Standard Operating Procedure |
| TKC | The Knowledge Collective |
| TMS | Truth Maintenance System |
| UCK | Usable Corporate Knowledge |
| UML | Unified Modeling Language |
| XOM | Execution Object Model |

# ABSTRACT

The Knowledge Collective (TKC) is a multi-layer, multi-agent framework for information reuse in an intelligent knowledge base that supports a collection of agents called MicroDroids, which provide information management capabilities through a variety of interfaces for experts, human users, and software components. This information is stored in a variety of internal structures (e.g., Java objects, rules, database structures). The main concept is that information is stored in a format that is natural to the type of information being maintained as data, metadata, ontologies, concept maps, lexicons, rules. The Knowledge Collective will make ontology-based information accessible to many end users, maintainable by domain experts and reusable by many users across many applications without their needing to know how or where the information is stored. The Knowledge Collective's first use is in version 4 of CIRCSIM-Tutor, an Intelligent Tutoring System developed by Martha Evens and her group at the Illinois Institute of Technology in Chicago, IL.

What is unique about TKC is in the combination of technologies for problem solving and the use of a multi-agent system where each MicroDroid manages its own information and works with other MicroDroids to solve specific problems. Each MicroDroid manages its information using an Ontology Inference Engine (ONTIE) that is capable of combining all of its internal information structures for the purpose of access, maintenance and problem solving especially involving Qualitative Reasoning.

# CHAPTER 1

# INTRODUCTION

This dissertation investigates the feasibility of a new kind of knowledge base for CIRCSIM-Tutor and other Intelligent Tutoring Systems, capable of supporting Qualitative Reasoning and tutoring dialogue for a whole family of tutorials involving the same subject area. Currently CIRCSIM, CIRCSIM-Tutor, and the Concept Map tutorials all have separate handcrafted knowledge bases containing the same basic knowledge about the cardiovascular system. They all required a large-scale effort by the domain expert, Joel Michael, and a lot of programming. The new knowledge base, The Knowledge Collective (TKC), is a framework that supports a collection of agents called MicroDroids, which provide information management capabilities through a variety of interfaces for experts, human users, and software components. This information is stored in a variety of internal structures (e.g., Java objects, rules, database structures). The main idea is that information is stored in a format that is natural to the type of information being maintained (e.g., data, metadata, ontologies, concept maps, lexicons, rules).

The uniqueness of TKC is in the combination of technologies for problem solving and the use of a multi-agent system where each MicroDroid manages its own information and works with other MicroDroids to solve specific problems. Each MicroDroid manages its information using an Ontology Inference Engine (ONTIE) that is capable of combining all of its internal information structures for the purpose of access, maintenance and problem solving especially involving Qualitative Reasoning.

In the past, we separated data from programming logic in the form of relational databases for ease of maintenance and reuse. Today we are separating business logic

from programming logic in the form of business rules for the same purpose. Now it is time to separate information semantics from programming logic in the form of ontologies so that it can be easily maintained and reused. This is the main purpose of TKC.

One of the major concepts of TKC is that each MicroDroid is capable of solving problems in a small area of expertise like Baroreceptor Reflex physiology or curriculum planning. Together the whole community of MicroDroids in TKC can emulate the human intelligence needed to handle a complex situation like tutoring first year medical students as they solve problems about the Baroreceptor Reflex.

### 1.1 Motivational Background

An important consideration in developing an Intelligent Knowledge Base is to make the information usable in an industry application. All major corporations collect large amounts of data. Often the same data or at least data about the same topic areas is collected in many different systems. Let us start with one Information Technology system. When the system was initially built, the developers and the business people knew why the system was built and what data was needed to improve the business process. However, time goes on and people move. New applications need to use that data. The data is still there, but unless it was unusually well documented, no one really understands what the data means. The system can still produce the reports that people developed the system to produce, but times and markets change, while our data collection methods stay the same. This all comes down to Knowledge Management.

This is one of the major reasons why corporations build data warehouses. They are trying to recapture the information that the data represents. In addition, there is

probably a lot of hidden information in the data that could be used today if it were known. Over time, a lot of history is captured including important user trends. If the data warehouse is to be the arbiter of truth and the confidence of the end user is to be gained, then we have to provide an understanding of the data to give the end user that confidence. In this case, the term "data warehouse" is used generically. It does not matter whether or not we are talking about an operational data store, a historical data warehouse, or a data mart.

One of the major concepts of a data warehouse is to take data, process it into information and deliver it to the point of decision-making. By making this information usable for the end user, the information is then analyzed and turned into knowledge that then becomes Usable Corporate Knowledge (UCK) and a corporate asset. However, is it still a data warehouse or has it become a knowledge base? My definition of the difference between the two is intelligence. Intelligence is needed in a knowledge base because different types of data need different data structures (e.g., rules, cases, models, frames, logic, etc.). Standard relational technology cannot handle the complexity of a knowledge base. SQL by itself is not rich enough or flexible enough to deal with anything but the standard relational structure (Mundy, 2002). We need a way to access information without normal SQL, a common interface or API. The concept of MicroDroids in The Knowledge Collective makes this possible.

Up to this point, we have been talking about corporate data for applications. How about when we get beyond business information and reach the other end of the spectrum in areas like medical information processing? If we want to do intelligent decision support, diagnosis or medical tutoring, we need to go beyond the standard

relational data structures of a data warehouse. We are now into the realm of knowledge bases, but the knowledge base still has to contain intelligence so that it can function as an asset. It takes intelligence to have usable knowledge. This intelligence comes from the use of complex data structures, metadata and semantic understanding of the information.

The same problems can afflict academic projects over time. For example, the medical knowledge base underlying the CIRCSIM-Tutor project at the Illinois Institute of Illinois has changed slowly over time and now Martha Evens and her group want to use it to support rule-based inference instead of frame-based inference. We want to abstract that data, to define our own metadata, so that we can write new tutors in other areas of physiology. This project deals with tutoring first year medical students in the area of the Baroreceptor Reflex, which is the part of the cardiovascular system that keeps the blood pressure fairly constant (Cho, 2000; Freedman, 1996; Khuwaja, 1994; Mills, 2001). Currently Martha Evens and her group are starting to work on a new project called GASP-Tutor to help students understand the two interlocking negative feedback systems that control breathing in the human body. By using TKC methodology, knowledge from the CIRCSIM-Tutor project can be re-used in this project.

We need to take this a step further. When you have complex data structures (e.g., class models, concept maps, process flows, flow paths, lexicons, etc.) metadata is only the beginning. Understanding the many different kinds of metadata and information structures in a single knowledge base takes a great deal of diverse intelligence. An end user of the knowledge may spend more time figuring it out than using it. By adding intelligent agents to the knowledge base, you can lift the burden of understanding the information about the data internals from the end users giving them the freedom to use

the knowledge for solving their problems.

Each MicroDroid needs to understand a very small specific area of domain information. The MicroDroids understand the metadata and internal information structures, so that they can insert new information, retrieve information and use that information for reasoning and making inferences. The use of multiple agents as part of the knowledge base framework gives it the intelligence that it needs to help the end user do their needed tasks.

What this all boils down to is making computers more useful. Can we really make them understand human users and carry out a conversation in applications like Intelligent Tutoring Systems (ITS)? Can they teach students about concepts in a complex domain like medicine? Human tutors can teach another person about a specific domain because they understand the domain that they are teaching. A professor of physiology can teach a first year medical student about the Baroreceptor Reflex because the professor is an expert on the topic. That same professor would probably have a hard time tutoring students about finance or electrical engineering. It would take different professors, experts in these domains, to tutor a student about these topics.

So how do we expect a dumb computer to tutor medical students about different medical domains? In fact, how can a computer even understand how to be a tutor in the first place? The computer has to understand the domain it wants to tutor the student in and it has to understand how to be a tutor in general.

The computer can only understand a domain in terms of the model it is given (Yusko, 1994; Bredeweg and Forbus, 2003; Falkenhainer and Forbus, 1988). That model is an ontology. In fact, there are many models or ontologies involved in being a tutor

(Khuwaja and Patel, 1996). However, it is not that simple. Just as a university needs many professors to tutor students in different domains, this computer system will need many domain experts using many different kinds of models.

How can we build a system that meets the requirements to tutor a student in one domain area like the Baroreceptor Reflex? One large ontology will not meet the needs. It might work, but it would be very difficult to maintain because we are talking about many areas of expertise combined into one. This process is much more feasible with many smaller ontologies that cover the areas of expertise that are needed. This advantage has its own built-in cost: how do we make these ontologies work together and, more important, how do we maintain them in a consistent fashion?

What is needed is a university inside the computer system. All the members of this group of expert agents need to have their own ontologies and work together as a team to tutor students. The Knowledge Collective (TKC) is the realization of this vision. It is a multi-layered, multi-agent framework for developing and maintaining intelligent knowledge bases that can be used in areas like Intelligent Tutoring Systems (ITS) such as CIRCSIM-Tutor (Michael et al., 2003; Evens and Michael, in press). It is made up of many intelligent agents called MicroDroids that are capable of working together as a team to solve problems in an ITS. This is the team or virtual university that I am developing at the Illinois Institute of Technology to be used by the next generation of CIRCSIM-Tutor and by other medical tutoring systems in the future. TKC is the framework, or foundation, that will allow many smaller maintainable ontologies to be used together to solve problems in areas like Qualitative Reasoning across many domains in an ITS. In fact, TKC is developed around the concept of ontologies and is controlled

by ontologies and their inference engines.

Why pursue this line of research to develop TKC? What motivated it and what will be gained? The major idea is to develop a framework for developing intelligent knowledge bases with the following capabilities:

1. Deal with complex informational structures in a natural way
2. Reuse data, information, and knowledge across many applications
3. Allow domain experts to maintain their specific structural information in a way that makes sense to them

**1.1.1 Deal with Complex Informational Structures.** Intelligent knowledge bases need to support many information structures. The four major types are:

- Data
- Metadata
- Ontologies
- Reasoning Logic (i.e., Rules)

All of these structures need to be stored in a way that is natural and that enables them to be accessed by applications and users. All types of information need to be brought to bear to solve specific problems. Complex information should not be forced into a specific structure for ease of storage and retrieval. The way it is structured is part of the information itself. When you restructure information, you may actually lose some

of that information. Data and metadata should be stored in a normal relational database for ease of manipulation, maintenance, and access. Ontologies should be stored as objects with relationships like any model. Reasoning should be stored as rules in the system. All of these still have to work together and be maintained together.

No matter how information is stored, users, developers, domain experts and applications should not have to understand how or where the information is stored. There should not be a need for an Application Programming Interface (API). A mechanism such as an agent knows how the information is stored. This is what makes a knowledge base intelligent. This problem is a major reason for doing TKC research.

**1.1.2 Reuse of Data, Information, and Knowledge.** Data is the basic informational building block, containing the underlying facts. It is processed into usable information. This information is then analyzed into knowledge that can be used by the end user for decision-making. Throughout this paper, the word “information” will be used as a generic term denoting any of these three levels.

Information needs to be reusable by many applications and users. An example is the tutoring system CIRCSIM-Tutor developed at the Illinois Institute of Technology (IIT). Its knowledge base has been rebuilt 7 times in the past 15 years (Evens and Michael, in press, Chapter 15). Now Martha Evens and her group are developing a new tutoring system called GASP-Tutor that will use a good percentage of the same information from CIRCSIM-Tutor. Under the current circumstances, a new knowledge base will have to be built. This need for information reuse was the major reason for starting TKC research now.

To reuse information, many people are talking about how to develop new API standards, special protocols and structural standards so that applications and users can make use of information from many sources especially in the area of ontologies. Noy, et al. (2004, p. 78) describe the problem in biomedicine as follows:

> It is becoming impossible to contemplate successful biomedical research without canonical data structures. The biomedical computation community finds itself grappling with hundreds of different knowledge bases, metadata formats, and database schemas.

Even though this is only one discipline, the same problem is happening in many other domains. Noy et al. also talk about the solution being a "virtual ontology repository" with the following features (Noy et al., 2004, p. 78):

> A researcher faced with a task that requires a knowledge resource should be able to access a virtual repository, evaluate its content, understand if any of the resources are relevant to the task, and align the resources to his or her own resources and data.

This major problem needs to be solved if we are going to achieve reuse of information capabilities. I do not think that the solution is to develop yet another set of standards. I am looking at TKC to help solve this problem.

**1.1.3 Give Domain Experts the Ability to Maintain Their Information.** Maintainability is a major factor in any knowledge base. It is very difficult for a domain expert to maintain large ontologies or information in a combination of information

storage structures. The capabilities must exist for the domain experts to maintain their domain specific information without having to understand how the information is structured. The need to provide this type of maintainability is one of the major features of TKC.

## 1.2 Significance

The problem is devising a methodology for reuse of information among different systems that is maintainable and scalable. What makes this project unique is the combination of many technologies that are brought to bear on the problem. Each technology in and of itself is not unique, but the combination is unique. It is also unique to have many small ontologies working together instead of a single large ontology in a multi-agent system. Each MicroDroid is unique because of the types of information that they can bring to bear to solve a problem in a specific domain: data, metadata, ontologies and an Ontology Inference Engine. It is also unique that the ontologies are stored as Java objects with a production rule based inference engine used to reason about them.

The Knowledge Collective approach allows information, mainly ontological information, from many areas of expertise to be used together to solve problems without standardized ontology languages and APIs. The use of the multi-agent capability with the Ontology Inference Engine makes this possible.

### 1.3 Research Questions

This is a list of the specific research questions that have been answered in this research:

- Can TKC solve the information reuse problem?
- Should the Task Layer in TKC be used or eliminated, i.e., would this additional indirection make the system too slow and therefore not usable? If this layer stays, then is each MicroDroid truly independent and autonomous?
- Can a database actually be imbedded into a MicroDroid?
- What are MicroDroids really? Are they specialized agents or are they a community of nested agents? Do they get instantiated as needed and then go away after they are finished?
- What are the proper communication needs of the MicroDroids?
- What metadata will be needed and how will the MicroDroids use it?
- How will the individual MicroDroids and the communications between them use the individual ontologies? What are the types of ontologies that will be needed (e.g., artifact or object, process flows, flow diagrams, concept maps, etc.)? How will they work together? Will each MicroDroid have one or many ontologies?
- How will the ontologies work? Will an Ontology Inference Engine solve the Qualitative Reasoning (QR) problem?

- Can the needed parsing for the natural language interface be accomplished with a commercial generic parsing engine?

- Can TKC be implemented using existing tools, commercial or open source?

**1.4 Domain**

The ability of many users to reuse information across many applications is important for the success of an intelligent knowledge base. A good example is the work that is being done with Intelligent Tutoring Systems (ITS) developed by Martha Evens and her group at the Illinois Institute of Technology (IIT) in Chicago, IL. The knowledge base described in this paper is the seventh in a series of knowledge bases that have been built from scratch to support the CIRCSIM-Tutor system (Evens and Michael, in press). Each time Martha Evens and her group have interviewed their experts they have spent much time reprogramming the information in a form that the experts cannot read. They are now starting to build two new tutorials that cover much of the same material – one is a concept map tutorial for the Baroreceptor Reflex; the other is an intelligent tutoring system called GASP-Tutor. It is clearly time to stop tearing up knowledge bases and building new ones. Instead, it is time to plan in advance to make the knowledge reusable. Instead of starting the knowledge engineering process all over again, they want to design a knowledge base to serve all of these different systems and to create a tool that allows the expert to define and read what is in the knowledge base. I repeat this history of the dead knowledge bases developed in the past, in the hope of avoiding reliving it.

CIRCSIM-Tutor is an intelligent tutoring system that carries on a natural language dialogue with the goal of helping first-year medical students learn how to solve

problems involving the Baroreceptor Reflex, the negative reflex system that acts to maintain blood pressure in the human body. The CIRCSIM-Tutor project grew out of an earlier computer-aided instruction system called CIRCSIM. It presents students with a perturbation to the blood pressure, asks them for predictions about how this situation will affect seven important physiological parameters, analyzes the patterns of errors in those predictions, and reels out one of 243 paragraphs of canned remedial text stored in the system (Rovick and Michael, 1986). CIRCSIM was and is a big success, but its builders, who are Professors of Physiology at Rush Medical College, thought that their students could learn even more from a system that could ask the students to provide explanations, understand their answers, and comment on them. Martha Evens and her group set out to build a system capable of carrying on a natural dialogue with the users.

The first version calculated the predictions for the four problem situations correctly, but it did not solve those problems in the logical order that Michael and Rovick wanted their students to use. The second version changed some of the rules to produce the answers in a logical order, but as Yuemei Zhang (Zhang, 1991; Zhang et al., 1987, 1990), who was writing the Discourse Generation portion of the system, immediately pointed out, the knowledge base still contained nothing that the system could use to guide the students to solve the problem. The third version contained a solution tree for each of the four problems then implemented, which provided a trace of the ideal solution for that problem. Zhang agreed that this was a big improvement and she started to produce code to explicate the trace, but she argued that this version was still not enough, because it did not provide support for discussing the steps in the problem-solving algorithm with the student or for generating explanations.

Nakhoon Kim (Kim, 1989) built the fourth version of the CIRCSIM-Tutor problem solver using a knowledge base that consisted of a hierarchical set of Prolog rules that described the problem-solving algorithms and used them to solve problems. At this point, he declared victory and integrated the pieces of the prototype system so that it could request predictions, analyze the predictions entered by the student, build an overlay model of the student's knowledge of the Baroreceptor Reflex, determine a set of topics to be taught, and plan how to teach them. Kim's Prolog Prototype (Kim et al., 1989) did not attempt to carry out a natural language dialogue with the student, but it performed all the other steps in the tutoring process and a great deal was learned from building it, especially about knowledge representation (Kim, 1989).

In 1990-1991 Chong Woo (Woo, 1991; Woo et al., 1991) constructed a complete version of CIRCSIM-Tutor in Lisp and Zhang seized the chance to build the knowledge base of her dreams. The fifth version of the knowledge base is a collection of frames, which, with some additional problems added to the system, has powered the system for the last ten years. After being tested with large classes of students at Rush, it is now in routine use. Woo and Zhang built a frame for every phase, every parameter, every causal relationship, as well as for other concepts that the students need to learn in order to understand negative feedback systems like "neural variable" and "regulated parameter." At Zhang's insistence, some anatomy frames were added as well. The experts did not think that anatomical concepts belonged in a physiology knowledge base, but Zhang pointed out many places where anatomy was mentioned in the human tutoring transcripts. The experts agreed that it could include anatomical references so long as their use was limited to understanding them in student inputs and responding to those inputs. The code

for the problem-solving algorithms was added to the frames as well, so that the Discourse Generator could discuss the algorithms with the student. Fortunately, Lisp code can also serve as Lisp data.

Ramzan Ali Khuwaja (Khuwaja, 1994; Khuwaja et al., 1992, 1994), to support his multi-level model of the domain knowledge base, built the sixth version of the knowledge base in CLOS (the Common Lisp Object System). He carried out a detailed analysis of the domain knowledge in the transcripts and discovered that the experts organized it in three levels. The top level corresponds to the concept map that they hoped the students would internalize and utilize in problem solving. The middle level contains additional concepts that they used in devising hints and giving explanations. The bottom level contains many other concepts that were sometimes mentioned by students but not used by the expert tutors unless a student alluded to them first. Perhaps because CLOS was new and Khuwaja was a new CLOS user, his module suffered from performance problems. Woo and Zhang's frame knowledge base continues to be used but the other two levels have had frames added.

The current ideas about the knowledge base owes a great deal to the work of Reva Freedman (Freedman, 1996; Freedman and Evens, 1997; Freedman et al., 1998), who argued that much of the knowledge could and should be written in rule form. The result, she claimed, would be much easier for the experts to read and update. The experts never liked the frames. She demonstrated the feasibility of her approach by actually producing a large number of these rules in her dissertation.

Two new tutoring systems are now in the planning stage, another dialogue-based Intelligent Tutoring System called GASP-Tutor and a concept map tutorial. The focus of

GASP-Tutor centers around two interacting negative reflex systems that controls ventilation in the human body. Analysis of human tutoring transcripts shows a large overlap in language and reasoning between GASP-Tutor and CIRCSIM-Tutor. The concept map tutorial covers the same domain. If all three systems could use the new CIRCSIM-Tutor knowledge base, it would save a tremendous amount of time and effort both for the developers and for the experts.

Systems like the ones discussed above require models of the domain, models of tutoring, student models, and language models. These multiple models require the system to organize and store many kinds of knowledge. All these models need to be stored as individual ontologies so that:

- Many users and domain experts can access the information in the ontologies
- Domain (including Pedagogy) experts can maintain their specific ontologies
- The ontologies can be reused by many end users and across many applications
- The information from multiple ontologies can be used together to solve problems (e.g., using Qualitative Reasoning)

**1.5 Overview**

I have developed the chapters as building blocks. Chapter 2, Background, explains the existing literature that was researched. The basis of the information and its semantics are explained in Chapter 3, Ontologies. The core idea that makes the whole TKC concept work is discussed in Chapter 4, Ontology Inference Engine. How all the information is controlled by agents is explained in Chapter 5, MicroDroids. How the

MicroDroids work together as a team is described in Chapter 6, The Knowledge Collective. The research is summarized in Chapter 7, Conclusion.

# CHAPTER 2

# BACKGROUND

The Knowledge Collective (TKC) is a framework for an intelligent knowledge base that uses many different technologies. It is not any one technology that makes this project unique, but the combination of these technologies for the purpose of reuse that makes it unique. The current relevant technologies are divided into the following three areas of research: Multi-Agent Systems (MAS), ontologies and reasoning, which are explained in the following three sections. The focus of this thesis is on how these varied approaches to the knowledge and reasoning support the use and management of information. This research is not about how to develop a better Multi-Agent System, TKC just uses specialized agents called MicroDroids to manage information. A MicroDroid is just an intelligent container to manage specific domain information. Ontologies are a way of storing the information and Ontology Inference Engines are for reasoning about the information. These three technologies working together in combination make this work unique.

## 2.1 Multi-Agent Systems

As in any agent-based system, ontologies are very important to TKC. Agents must have an understanding of the environment in which they are working. If an agent is to be the keeper of specific knowledge, then the agent needs to know how the semantics of the knowledge is structured. The semantics of the knowledge is the ontology of the knowledge. This is the model that is given to the computer to understand the knowledge (Yusko, 1994). This model is not just for the agents to communicate with each other. In

addition, the model's information is used for reasoning about a specific problem in a specific domain.

The Foundation for Intelligent Agents (FIPA) has a specification for an ontology service (FIPA, 2002b). This specification assumes that the system has an ontology server. It talks about using ontology agents to make the ontology available to all agents in the system (FIPA, 2002b, pp. 1-8).

The concept of an ontology server with ontology agents does not properly fit The Knowledge Collective framework. Each time a user connects to TKC, a session is set up. This session will last until the end user has completed the desired tasks. An important feature of The Knowledge Collective is that each user session will dynamically build its own ontology. Each user session can have a different ontology depending on what the user is trying to accomplish.

Wooldridge (2002, p. 180) argues that ontologies are important because agents use ontologies to specify terms about a domain so that the agents can communicate with each other. The TKC approach differs in the fact that a MicroDroid manages information for a human end user, and does Qualitative Reasoning about the ontologies by using an Ontology Inference Engine. The MicroDroids allow the management of the information by a domain expert and the use of it by an end user.

For Ferber (1999, p. 31) the goal of using Multi-Agent Systems is problem solving. He defines problem solving as the ability to accomplish tasks that are useful to human beings. This is the main purpose of MicroDroids. However, the major theme of Ferber's book is how agents in a Multi-Agent System interact with each other. He does not consider my area of research, which involves information systems using ontologies

and Ontology Inference Engines. I am just using agents as managers of information.

There seems to be a lot of work in the area of software agents, both for internal programming use and for the internet. Two good texts on software agents are (Bradshaw, 1997) and (Murch and Johnson, 1999).

## 2.2 Ontologies

Many ontologies are stored physically in frames (FIPA, 2002b p.16). The thought was originally to store TKC ontologies in frames (Yusko, 1984). However, reasoning about frame-based ontologies in agent systems is usually done using predicate logic expressed in Prolog type rules. TKC will use production rules of various kinds as an Ontology Inference Engine to accomplish Qualitative Reasoning about specific domains. For this reason, ontologies will be maintained as UML (Unified Modeling Language) models, which are converted to Java classes. I consider these issues in more detail in the discussion of the Ontology Inference Engine in Chapter 4.

Table 2.1 compares some of the other major system ontology architectures and Ontology Inference Engines to the architecture of TKC. Two good references that have a good overview of ontologies are (Fensel, 2004) and (Gomez-Perez et al., 2004). This is a very small sample. There are many ontology languages and Ontology Inference Engines, which makes it very hard to mix and match ontology-based information from multiple sources. TKC helps solve this problem with its MicroDroid concept.

Table 2.1. Ontology System Comparison

| System | Ontology Architecture | Ontology Inference Engine | Reference |
|---|---|---|---|
| TKC | Object Oriented (Java Classes) | Production Rule Engine (ILOG JRules) | |
| OIL | Frame-Based | Description Logic | (Fensel et al., 2005) |
| Ontolingua | Frame-Based | First-Order Logic | Farquhar et al., 1997) |
| Protégé | Frame-Based | Algernon (Logic-Based) | (Knublauch, 2005) (Hewett, 2005) |

## 2.3 Reasoning

There are two main areas that were researched in the area of reasoning: Qualitative Reasoning and Truth Maintenance System (TMS). Qualitative Reasoning is the main area of reasoning for the MicroDroids. The Ontology Inference Engine does Qualitative Reasoning about the ontologies. The TMS is for dealing with goals, beliefs and learning of the MicroDroids.

**2.3.1 Qualitative Reasoning.** Qualitative Reasoning about physical objects such as the Circulatory System and the Baroreceptor Reflex is really one of the central themes of the CIRCSIM-Tutor domain that I am using for this research. Bobrow (1984, p. 1) talks about compositionality in dealing with Qualitative Reasoning:

> … the description of a system's behavior must be derivable from the structure of the system. The term 'structure' refers to the components of the analysis, component behaviors, and the connections between components. The term 'behavior' refers to the time course of observable changes of state of the components and the system as a whole.

Here, Bobrow really pinpoints the idea of using a group of ontologies and an Ontology Inference Engine. Another area that is important in Qualitative Reasoning is reasoning about function to understand the behavior of the system (Bobrow, 1984, p. 2). He defines function as "the relationship between a goal of a human user and the behavior of a system." This fits into the idea that in CIRCSIM-Tutor, students are doing problem solving in the area of the Baroreceptor Reflex. They reason about how it functionally keeps the blood pressure normal.

Forbus (1984, 1985) defines ontologies in terms of processes. His qualitative process theory allows for reasoning about process ontologies, "when they will occur, their effects, and when they will stop" (Forbus, 1984, p. 85). Process Ontologies are one of the four types of ontologies that MicroDroids deal with. The different types of ontologies are discussed in Chapter 3.

**2.3.2 Truth Maintenance System.** Truth Maintenance Systems are not used very widely today in industrial applications. The need is there, but the understanding seems to be lost. Even though ILOG® JRules™, which is a Business Rule Management System (BRMS), has truth maintenance capabilities included in the system, I was not able to find anyone in industry using ILOG's TMS capabilities. This is a shame because it is a very powerful system. Giorgio Ingargiola from Temple University (2005) has a good overview of a TMS. Forbus and deKleer (1993) discuss the TMS in depth.

A TMS is tied to an inference engine such as the inference engine in a BRMS or an expert system. It can also be tied to a Case Based Reasoning System such as in Yusko (1985). It decides whether facts are correct or not over time. Every time a fact is

justified, it becomes more believable. When another fact disproves a fact, that fact becomes less believable. There are upper and lower thresholds. When the score of a fact goes below the lower threshold, it is no longer considered true. Tracking many facts can become expensive, so if a fact goes above a certain threshold, the system can consider it true and will no longer track it. Yusko (1985) discusses a good example of dealing with a TMS in the area of generalizations and overgeneralizations. In this case, examples of known cases of shortness of breath were developed into a discrimination net. This formed generalizations about the facts in each of the medical cases. If an overgeneralization was found, it had to be retracted and the correct generalizations added.

A simple example of this involves the process of defining the concept of a bird. If you tell the system that a robin and an eagle can fly, the system will start to believe that all birds can fly. However, if you then give the system the facts about a penguin and an ostrich, which cannot fly, the system has to understand that there is a possibility that it has formed an overgeneralization. For the system to fix its information, it needs two categories of bird, birds that can fly and birds that cannot fly.

# CHAPTER 3

# ONTOLOGIES

Ontologies are the core concept that tie information about specific domains together and make them usable. Ontologies give the MicroDroids the semantics to communicate with each other. The MicroDroids give the ontologies the ability to be shared by multiple users and applications. An Ontology Inference Engine (ONTIE) controls the reasoning about the ontological information. For this discussion, it is important to understand what ontologies are in general terms and how they are used in TKC.

Don Hutcheson (2003, p. 45) defines an ontology as "a list with relationships to other lists." The Foundation for Intelligent Physical Agents (FIPA) defines an ontology in the following way (FIPA, 2002a, p. 34):

> An ontology provides a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary.

An ontology is a model of a specific domain that can be used for Qualitative Reasoning (Yusko and Evens, 2004) about either structural objects and their relationships or processes using the Qualitative Process Theory of Kenneth Forbus (1985). The ontologies enable agents to communicate with each other and with an end user in an intelligent manner.

This chapter discusses the types of ontologies defined in TKC in Section 3.1. Section 3.2 describes how the ontologies in TKC are maintained, accessed and reused.

## 3.1 Ontology Categories

Many ontologies are hierarchies of information typically based on the ISA relationship. If we use the FIPA definition stated above, then we can envision many types of ontologies with an unlimited number of relationships. We have to keep in mind that the purpose of using ontologies in TKC or any intelligent knowledge base is to store information and semantics about that information. I have classified some of the major types of ontologies used in TKC as follows: Artifact Based, Process Based, Concept Map Based, and Flow Based. Each of the categories defines specific parts of a domain. To do Qualitative Reasoning it takes a combination of ontologies to define a specific domain.

**3.1.1 Artifact Based Ontologies.** Artifact based ontologies are objects and their relationships. A UML Class Model can represent them. The class model for a MicroDroid in Figure 3.1 is a good example of this category of ontologies. In this case, it is really a part/whole model of a MicroDroid. Figure 3.2 shows another good example of an artifact ontology. It is the hierarchy model for MicroDroid classification. Its basic relationship is the ISA relationship.

**3.1.2 Process Based Ontologies.** Process based ontologies deal with process flows, which can be represented by UML Activity Models. An activity model represents how the flow of control works in a process. A good example can be seen in Figure 3.3: the process flow for CIRCSIM-Tutor (Evens and Michael, in press). The CIRCSIM-TUTOR

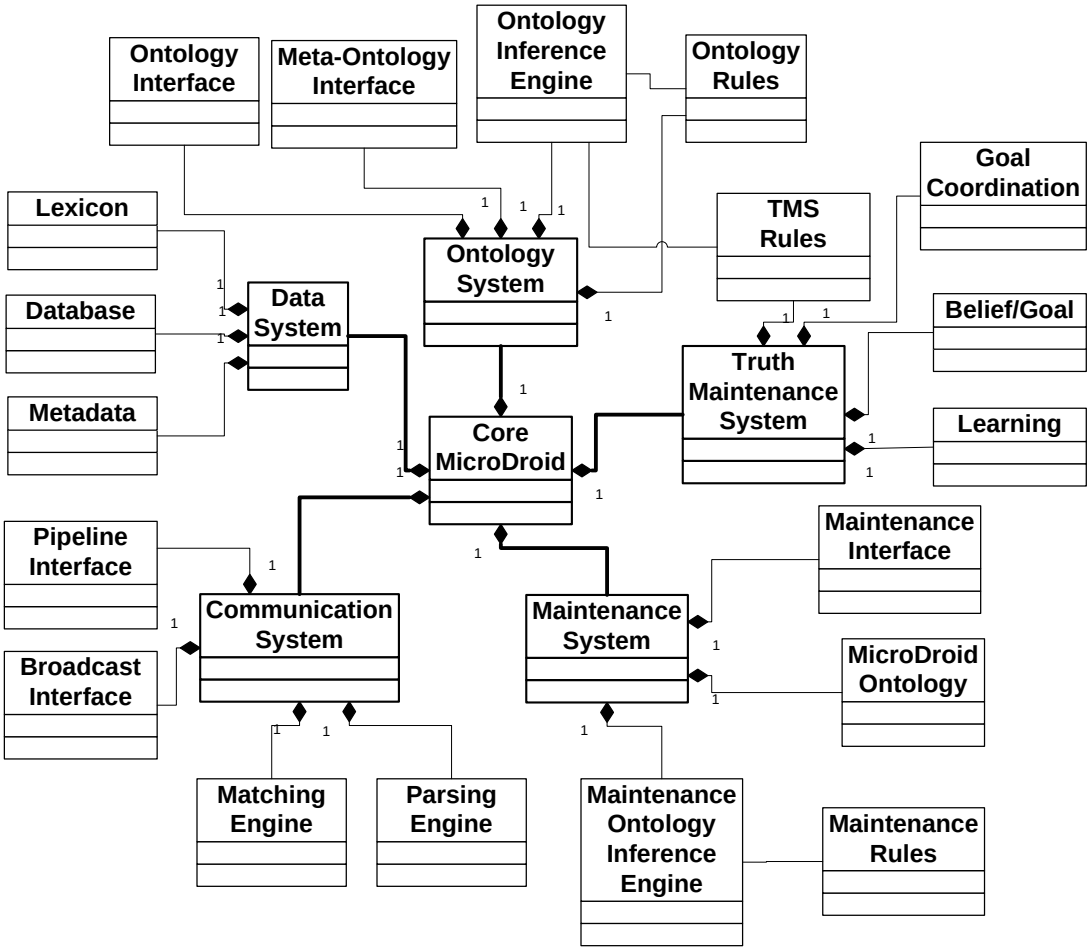


Figure 3.1. MicroDroid Part/Whole Ontology

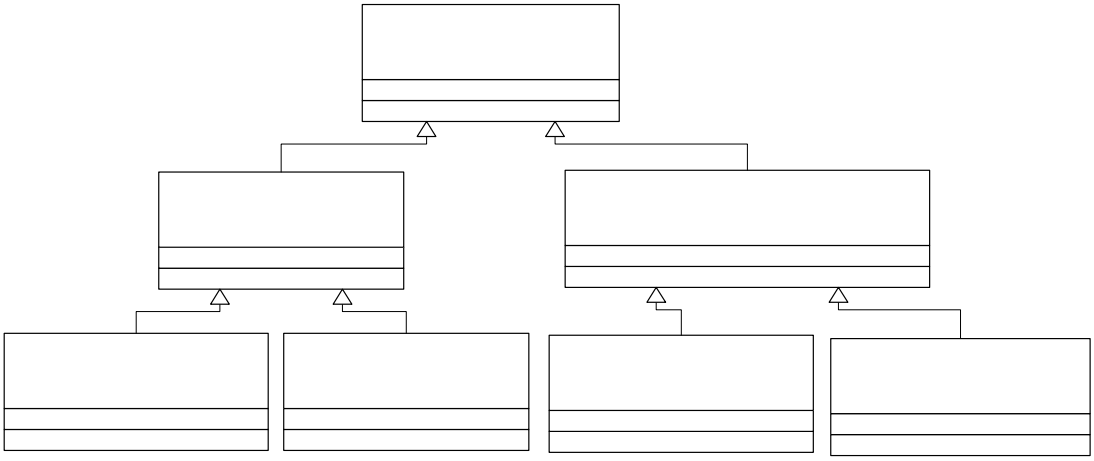

Figure 3.2. MicroDroid Hierarchy Ontology

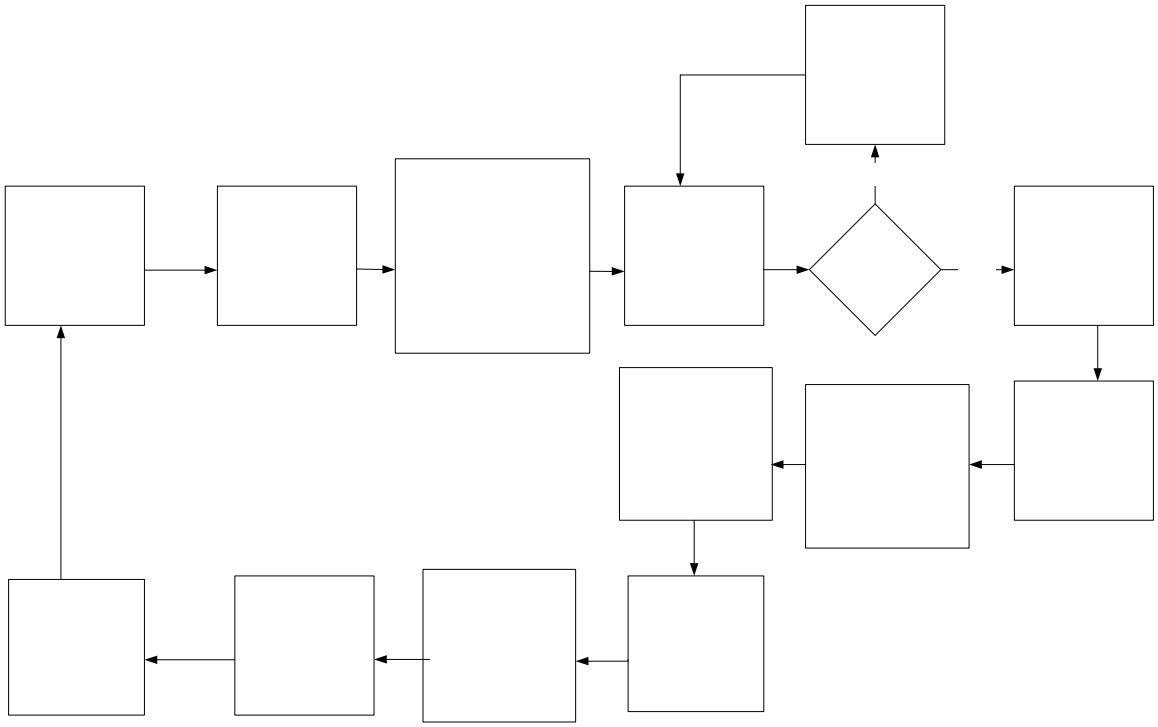

Figure 3.3. CIRCSIM-Tutor Process Ontology

MicroDroid in TKC manages this ontology. Since it is an Application MicroDroid, (see Chapter 5), it contains the model that describes the CIRCSIM-Tutor process and controls the process flow in CIRCSIM-Tutor. This process flow makes CIRCSIM-Tutor work.

System
Displays
Problem Menu

**3.1.3 Concept Map Based Ontologies.** Concept map based ontologies are concepts and their relationships. They represent how one concept affects another concept. Figure 3.4 shows a good example of the concept map for CIRCSIM-Tutor (Khuwaja, 1994, p. 73). This model is used to define the different concepts in CIRCSIM-Tutor and how they affect each other. This is vital to defining cause and effect relationships that are used in Qualitative Reasoning. The causal relationship between two concepts indicates how one concept affects another. The "+" on the arrow from SV to CO indicates a direct

relationship between the two. In other words, if SV increases, then CO increases. If SV decreases, then CO decreases. The "-" on the arrow from CO to CBV indicates an inverse relationship. In other words, if CO increases, then CBV decreases. If CO decreases, then CBV increases.

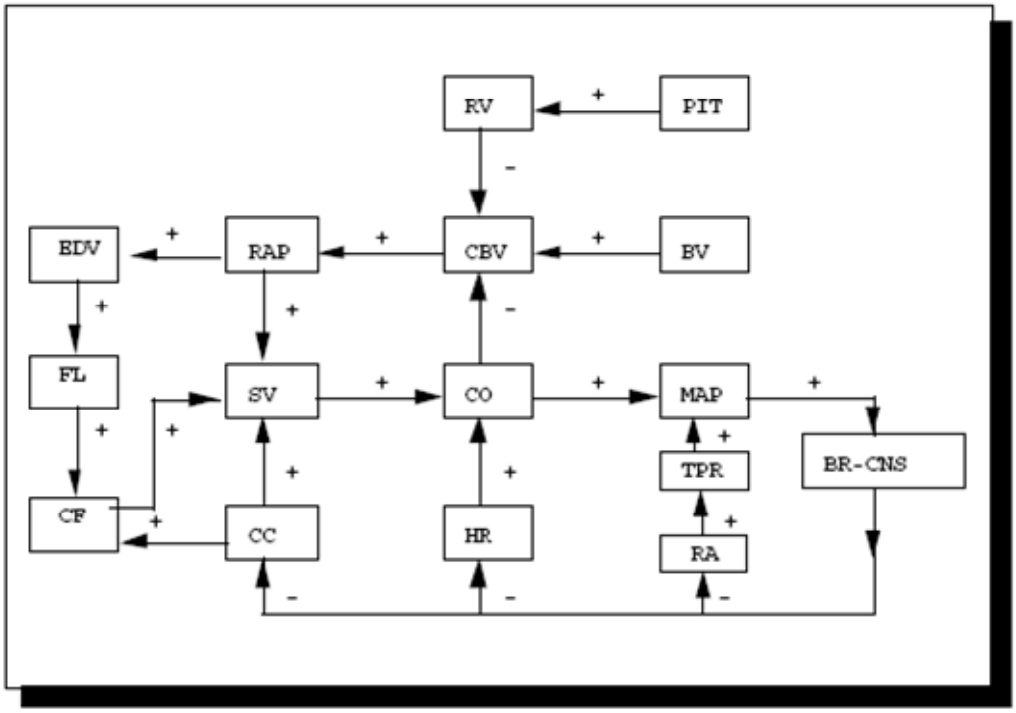


Figure 3.4. Baroreceptor Reflex Concept Map Ontology

**3.1.4 Flow Based Ontologies.** Flow based ontologies are very similar to process based ontologies. They model the flow of something physical instead of process control. Figure 3.5 shows a good example of the blood flow in the cardiovascular system (Zhang, 1991, p.68). (CIRCSIM-Tutor deliberately ignores the pulmonary circulation and the arterioles and capillaries.) In the case of this ontology, the flow of blood through the circulatory system is modeled. This allows Qualitative Reasoning about the blood flow.

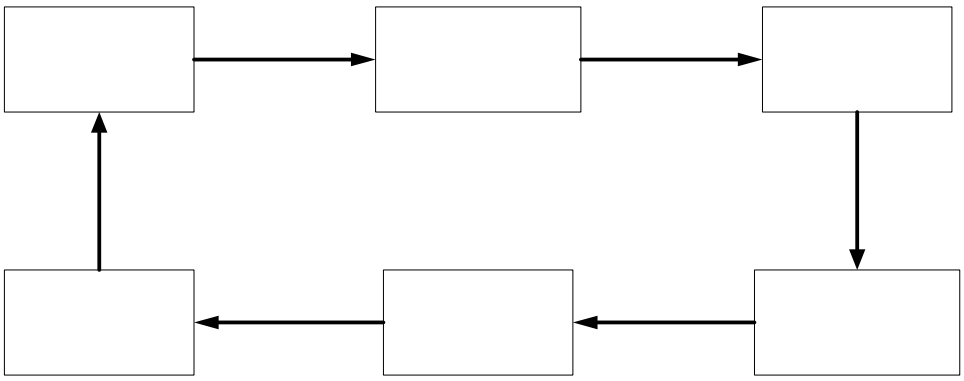

Figure 3.5. Circulatory Flow Ontology



**3.2 Maintainability, Accessibility, and Reusability Approach**

Many applications have ontologies buried in the actual code, like frames in LISP code. This is how it has been done in the past versions of CIRCSIM-Tutor. This makes it hard for domain experts to maintain the ontologies. Over the years, data has been pulled out of the applications and moved to relational databases for the purpose of maintainability. Business logic has been removed from applications and moved to rule repositories so the business logic can be maintained separate from the programming logic. Developers have also started to move ontologies out of the applications to ontology repositories so they can be maintained separately from the programming logic. Each of these separations allows domain experts to maintain information about their specific domains and end users to access and reuse it whether it is data, business logic, or ontologies. This is a good approach, unless the domain expert has to maintain all three information types at the same time.



In TKC, this tendency is taken a step further by allowing a single MicroDroid to manage and understand all three types of information specific domains so that the domain

expert only has to understand the domain, not how or where the information is stored. Chapter 6 defines the framework for The Knowledge Collective.

The Knowledge Collective has taken a different approach to managing ontologies. TKC is geared towards accessibility, maintainability by the domain expert and reusability by many end users and across many applications. All ontologies are maintained in small domain specific units and managed by a specific class of MicroDroids. The ontology itself is developed and maintained in a UML model using EclipseUML from Omondo® (Omondo, 2005), which is a plug-in to the Java environment called Eclipse (Gallardo, Burnett, and McGovern, 2003). This model is converted in the Eclipse environment to Java classes, which are in turn imported into the ILOG® JRules™ development environment as a Execution Object Model (XOM), which is really a set of Java classes. This is explained in Chapter 4.

# CHAPTER 4

# ONTOLOGY INFERENCE ENGINE

As stated earlier, most ontologies are frame based and use Prolog based logic rules to reason about the ontology. A good example of this is Protégé (Knublauch, 2005), which is a frame based ontology management system that uses an ontology repository. Algernon (Hewett, 2005) is a Prolog based logic rule inference engine that works with Protégé and is a Prolog based logic rule system. Many more examples can be found in (Gomez-Perez et al., 2004) and (Fensel, 2004).

The Knowledge Collective has taken a different approach to managing ontologies. TKC is geared towards accessibility, maintainability by the domain expert and reusability by many end users and across many applications. Each ontology is maintained as a small domains specific unit and managed by a specific class of MicroDroid. Production rules are used to do Qualitative Reasoning about the ontologies instead of logic-based rules. This will make it easier for the domain experts to maintain domain information.

It is important when looking at the understandability of a logic-based rule versus a production rule, that the production rule is easier to understand. A good example of this is by Bratko et al. (1989). They have the following logic-based rule (p.31):

[heart(art_focus: permanent(Rhythm, Rate))] →
[permanent(atr_focus: form(Origin, Rhythm, Rate))] &
Atr_focus(Origin, Rhythm, Rate).

They explain the logic-rule using a production rule (p. 29):

IF
the atrial focus discharges permanent impulses
at some rhythm Rhythm and rate Rate

THEN
there will be impulses at theatrical focus
characterized by Origin, Rhythm and Rate

WHERE
Origin, Rhythm and Rate must satisfy the
atr_focus relation

The TKC Ontology Inference Engine (ONTIE) is not only based on production rules like an Expert System, it is based on a Business Rule Management System (BRMS). BRMS differs from the basic Expert System technology. Both a BRMS and an Expert System can use production rules, be forward chaining, and use the Rete algorithm (Friedman-Hill, 2003, pp. 136-189) for conflict resolution. The real difference is the integration with an object model. An Expert System is not integrated with an object model. This integration with an object model is what makes a BRMS more useful as an Ontology Inference Engine than an Expert System. (Ross, 2003; Morgan, 2002; von Halle, 2002) are good references about Business Rule Management Systems.

Section 4.1 describes the ONTIE architecture. ONTIE rule examples are show in Section 4.2.

### 4.1 Architecture

The ONTIE architecture is show in Figure 4.1. The ontology itself is developed and maintained in a UML model using EclipseUML from Omondo (Omondo, 2004), which is a plug-in to the Java environment called Eclipse (Gallardo, Burnett and McGovern, 2003). This model is converted in the Eclipse environment to Java classes, using the Eclipse Modeling Framework (Budinsky et al., 2004). The Java classes are then imported into the ILOG® JRules™ development environment as an Execution Object Model (XOM). The XOM is then imported into JRules Rule Builder. The Java classes are also deployed to the MicroDroid. ILOG defines the XOM as (ILOG, 2005, p. 20):

> The Execution Object Model (XOM) is a model that references the various objects of your implementation, such as Java classes, XML Schemas, or Web services. In the XOM, XML Schemas or Web services become dynamic classes, and Java classes remain as native classes. All types of classes contained in the XOM are called execution classes. When you test the execution of a ruleset in the Rule Builder, the business rules that act on the business classes in the BOM are translated into execution rules that act on the execution classes in the XOM. Instances of these execution classes are sent to the rule engine.

JRules automatically builds a Business Object Model (BOM) from the XOM. The rules are then written against the BOM. ILOG defines the BOM as (ILOG, 2005, p. 22).

> The BOM defines the mapping between the classes it contains and the executable classes of the eXecution Object Model (XOM) used by the rule engine.

JRules is a Business Rule Management System (BRMS) that uses production rules that understand and are integrated with the BOM and are controlled by a rule

engine, which is a Java class. The rules are then deployed to the JRules rule engine in the MicroDroid. These rules are used to do Qualitative Reasoning about the ontology that the MicroDroid is maintaining.

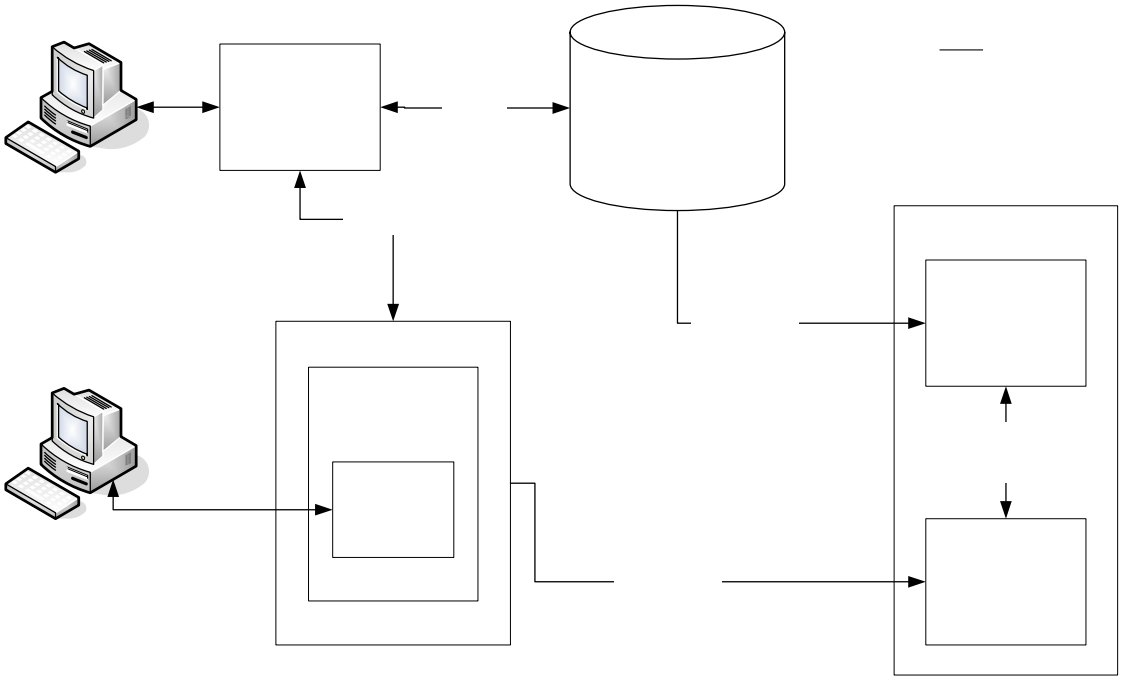

Figure 4.1. Ontology Inference Engine Architecture

The JRules Engine has three parts: a ruleset, working memory and an agenda. ILOG defines them as follows (ILOG, 2005, p. 38):

> A ruleset is a group of rules that is sent to the rule engine to be executed. The ruleset contains execution rules, namely business rules that have been translated into the ILOG Rule Language (IRL).
>
> The working memory is the place where the rule engine stores the objects with which it is currently working.

The agenda is the place where the rule engine stores the rules that are eligible to be fired. Rules are eligible to be fired when their conditions match the objects in the working memory.

**4.2 Rules**

As stated in Section 4.1, ONTIE uses production rules instead of logic-based rules. Production rules, including Business Rules, are forward-chaining, have a working memory and follow the following format:

IF condition THEN action

For a more in depth discussion of production rule systems, go to (Neches et al., 1987) and (Brachman and Levesque, 2004, pp. 117-134).

One important point is that the rules in ONTIE are designed to do Qualitative Reasoning and make inferences about the ontologies. Inferencing is the ability to produce new facts from existing facts. The ontology contains the existing facts. The Qualitative Reasoning capability produces new facts.

The rules can be used for either shallow reasoning or deep reasoning while doing Qualitative Reasoning. Shallow reasoning deals with how something functions. It is heuristic or "rule-of-thumb" reasoning. It can also be called surface or empirical reasoning. The actual information is stored as rules. There is always a need for this type of reasoning when dealing with relationship constraints and exceptions to an ontology. This type of rule is not really reasoning about the ontology. Since the rules are very specific, every change would mean writing a new rule or modifying an existing rule. Every time you have a new concept or relationship, you would have to add a new rule.

The rules are probably easy for the domain expert to understand, but the maintenance is very costly and difficult.

If you want to reason about an ontology, you need to use a deep reasoning approach. This can be called reasoning from first principles or model-based reasoning. The rules are more abstract or general since the actual information is not stored in the rules. The information is actually stored in the ontology. In this case, you are reasoning about objects and their relationships to other objects in the model. If you add a new concept to the ontology (i.e., a new object with relationships to existing objects), you do not need to add any additional rules to the ruleset. In some cases it might be a little more difficult for the domain expert to understand, but the cost of maintenance is greatly reduced. Figure 4.2 shows an example of shallow rules versus deep rules for a small Concept Map ontology.

This ontology has three nodes and two relationships. To represent it in a shallow reasoning ruleset would take four rules and you would not really need the ontology. To represent it with a deep reasoning set of rules, you would still have four rules, but they would actually reason about the ontology itself. Each rule reasons about the effects of the relationships of the objects. The big difference comes when you add more nodes and their relationships. As long as you are using one of the two existing relationships, you do not need to add any additional rules. With shallow reasoning, you would have to add two more rules for each new node. This does not seem like a lot until you add an additional 100 nodes with their relationships to the model.


Concept Map Ontology

**Shallow Reasoning Rules**

| | |
|---|---|
| **IF** | PIT increases |
| **THEN** | RV increases |
| **IF** | PIT decreases |
| **THEN** | RV decreases |
| **IF** | RV increases |
| **THEN** | CBV decreases |
| **IF** | RV decreases |
| **THEN** | CVB increases |

**Deep Reasoning Rules**

| | |
|---|---|
| **IF** | CONCEPT1 increases and RELATIONSHIP with CONCEPT2 is + |
| **THEN** | CONCEPT2 increases |
| **IF** | CONCEPT1 decreases and RELATIONSHIP with CONCEPT2 is + |
| **THEN** | CONCEPT2 decreases |
| **IF** | CONCEPT1 increases and RELATIONSHIP with CONCEPT2 is - |
| **THEN** | CONCEPT2 decreases |
| **IF** | CONCEPT1 decreases and RELATIONSHIP with CONCEPT2 is - |
| **THEN** | CONCEPT2 increases |

Figure 4.2. Shallow versus Deep Reasoning

The deep reasoning rules could be made even more abstract so that you would only have two rules. However, in my opinion based on my industrial experience, the more abstract that you make the rules, the harder it is for the domain expert to understand them. You need to find a happy medium between abstraction and understanding.

ONTIE uses a mixture of both shallow and deep reasoning rules for doing Qualitative Reasoning about ontologies. The shallow rules are used strictly to deal with constraints and exceptions. Since the deep reasoning rules are based on objects and their

relationship to other objects and ontology categories are based on objects and their relationships, there will be different rule category for each ontology type:

- Artifact Based Ontology Rules
- Process Based Ontology Rules
- Concept Map Based Ontology Rules
- Flow Based Ontology Rules

Within each rule category, there are rules for each type of relationship between the objects in that ontology category. As seen in Figure 4.2, there are deep reasoning rules for the "+" and "-" relationships. The same holds true for the relationships in each of the other ontology categories.

# CHAPTER 5

# MICRODROIDS

MicroDroids are a class of agents. Each MicroDroid is a pattern composed of objects that do a specific function or task. Each MicroDroid understands how it fits into the environment and knows what it can process and how to ask other MicroDroids for help. They are the navigators that translate the metadata into metaknowledge and help determine the truth about the knowledge. Each MicroDroid is an agent and a virtual object that controls its own smaller knowledge base, which is just a subset of the whole intelligent knowledge base. The MicroDroids are goal oriented and cooperate with the overall goal of a session. In simplest terms, a MicroDroid is:

Agent + Ontology + Ontology Inference Engine = MicroDroid

## 5.1 Hierarchy

If you look at Figure 5.1, the MicroDroids form a subclass of Task-Specific Agents using the terminology of Franklin and Graesser's Agent Taxonomy (1996, p. 23). Each box inside of TKC in Figure 6.1 represents a class of MicroDroids that do a very specific task.

What really makes a MicroDroid different from the Task-Specific Agents is the fact that MicroDroids do not depend on an ontology server or an ontology agent. Every MicroDroid has its own ontology and an Ontology Inference Engine built into it. This is vital since the ontology used during an end user session is built dynamically from the ontologies of each MicroDroid participating in the session. This also allows each

MicroDroid to do Qualitative Reasoning about its specific domain using an Ontology Inference Engine.

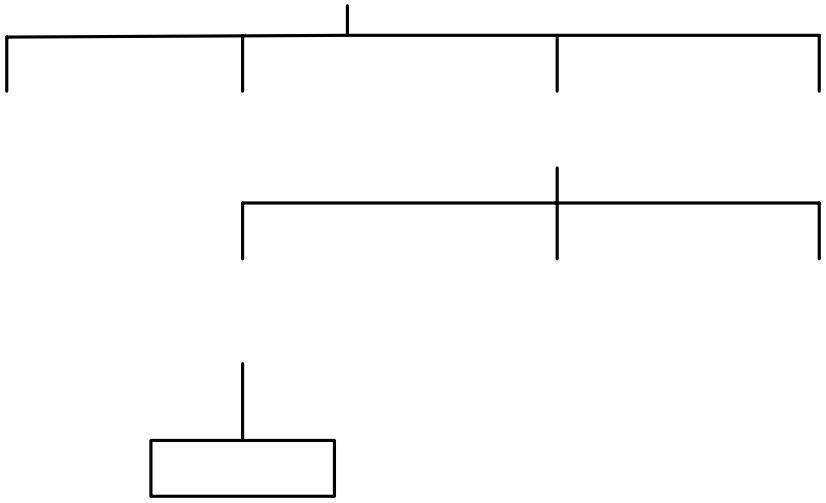

Figure 5.1. Franklin and Graesser's Agent Taxonomy

MicroDroids belong to a class hierarchy (see Figure 5.2). The root is MicroDroid. There are two main subclasses: Layer MicroDroid and Special Purpose MicroDroid. There are two Layer MicroDroid classes: Application MicroDroid and Solution MicroDroid. These classes have many subclasses. Each subclass can have many instances. There are two Special Purpose MicroDroid classes: Coordinator MicroDroid and User Profile MicroDroid. Each of these classes has no subclasses, but can have many instances.

Even though MicroDroids seem like objects in a hierarchy, they are not. Each MicroDroid is a pattern made up of many objects. This is discussed in Section 5.2, Architecture, and illustrated in Figure 5.3.

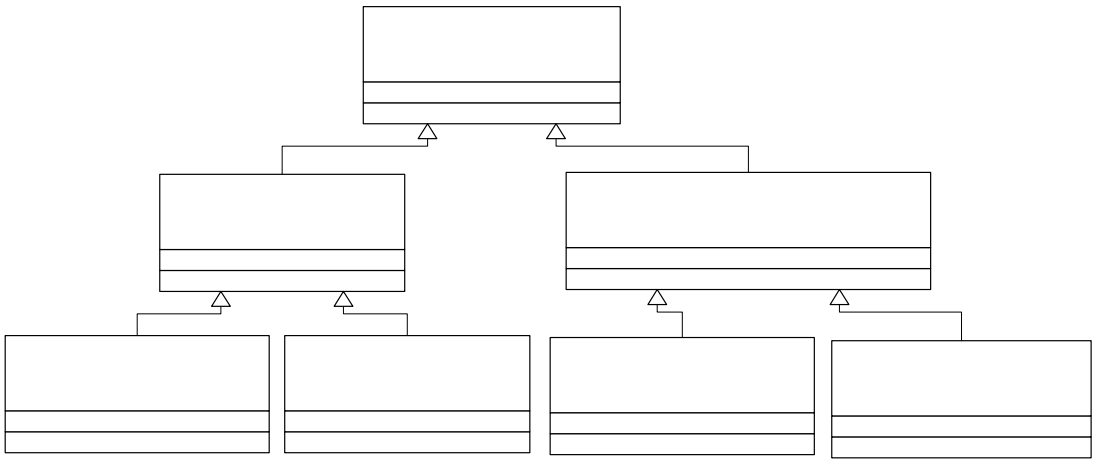

Figure 5.2. MicroDroid Hierarchy

**5.2 Architecture**

The MicroDroid architecture is really the ontology of the MicroDroid. This should not be confused with the domain information that it manages. There are five major systems and the core owner in the MicroDroid architecture, as diagrammed in Figure 5.3 with details in Figure 5.4:

1. Core MicroDroid
2. Maintenance System (MAINTEX)
3. Ontology System
4. Truth Maintenance System
5. Data System
6. Communication System

**Applicatior**
**MicroDroic**


Figure 5.3. MicroDroid Architecture

**5.2.1 Core MicroDroid.** The Core MicroDroid owns and controls all the other systems. It understand how to instantiate a new MicroDroid and retire one. There can be many instantiated for each class at any given time. When they have finished serving as part of a session, they go away. There is always one instance of a MicroDroid waiting to answer a call. The Core MicroDroid only has one object, the CORE MicroDroid object.

**5.2.2 Maintenance System.** This system is the Maintenance Expert (MAINTEX) that knows how to maintain all the information that a MicroDroid manages. There are three main objects in this system. The first object is the MicroDroid ONTOLOGY, which interfaces to the ontology that defines the MicroDroid. The second object is the Maintenance Ontology Inference Engine along with the Maintenance Rules, which are used to reason about the MicroDroid's own ontology. It can reason about any part of the MicroDroid. The last part of the MicroDroid is the Maintenance Interface object. This

object connects to the COORDINATOR MicroDroid and forms a portal from the Graphical User Interface Layer to MAINTEX.

**5.2.3 Ontology System.** This system contains the objects needed for dealing with domain specific ontologies that the MicroDroid has to manage. It has an Ontology Inference Engine (ONTIE) for doing Qualitative Reasoning about the ontology and other information that it manages. It also has a meta-ontology interface that deals with information about the ontology structure and how the ontology relates to other domain specific information within the MicroDroid.

**5.2.4 Truth Maintenance System.** The Truth Maintenance System (TMS) is for dealing with goals, beliefs and learning. MicroDroids have the ability to deal with Truth Maintenance. This Truth Maintenance System functionality comes with ILOG® JRules™ (ILOG, 2004, p. 115).

The MicroDroid learning system is part of this system. When a MicroDroid is instantiated, it has all the knowledge of the MicroDroid instances that came before it in this class. When the MicroDroid is finished with a session, it is destroyed, but all acquired knowledge is retained for use by future MicroDroids. This same type of learning can be used with student models since the student information is managed by a MicroDroid.

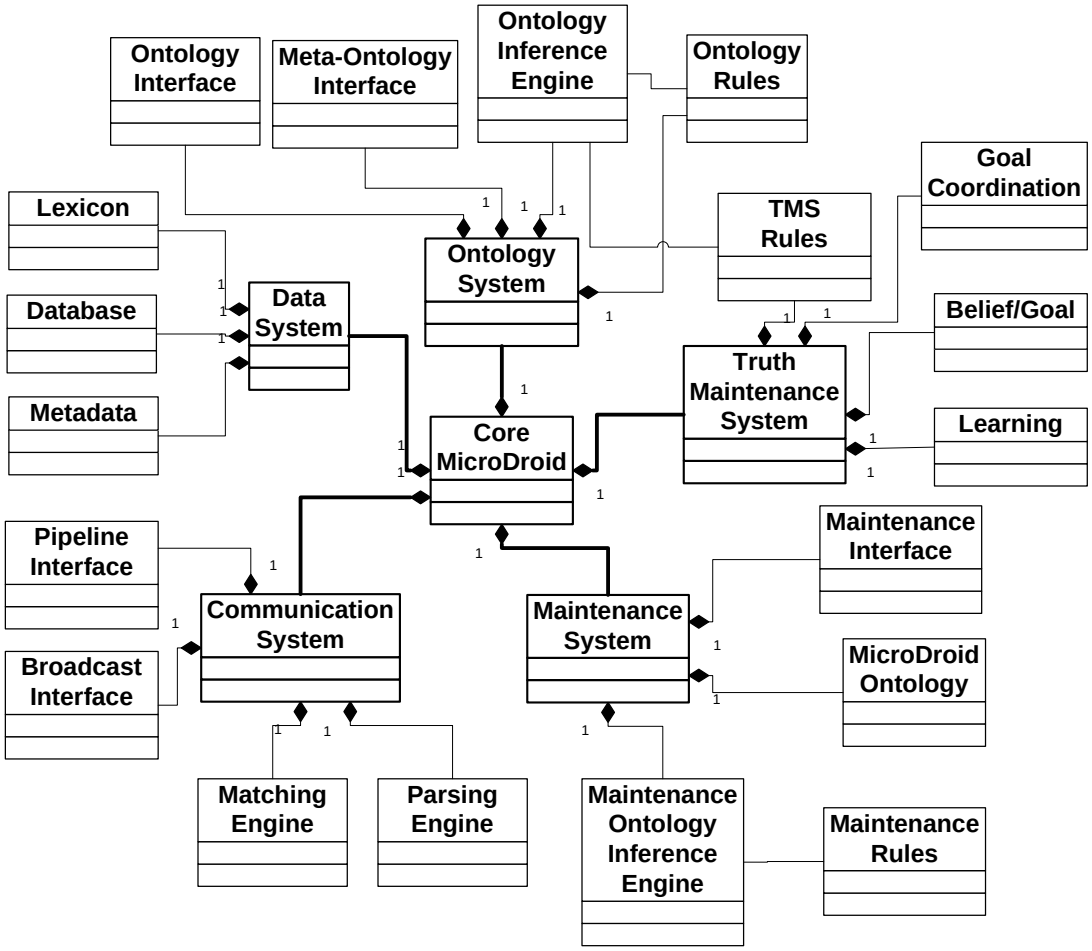


Figure 5.4. MicroDroid Ontology

**5.2.5 Data System.** The Data System stores all non-active ontology information such as facts, lexicons, metadata, ontology persistence information and rules. The information is stored in a database as standard relational tables and the Java objects are persisted to the database using a set of Java libraries called Hibernate (Iverson, 2005). In TKC, the Database System being used is Cloudscape® (Saunders and Anderson, 2004) an open source database system from IBM. Cloudscape® is being used because it was developed to be embedded in a Java application without the use of an independent database

management system. For a good article about embedded databases, see (Binstock, 2005).

The Data System has objects for dealing with different types of data: database, lexicon and metadata.

**5.2.6 Communication System.** The Communication System allows MicroDroids to communicate with each other in either a broadcast mode or peer-to-peer. This system takes care of both the end user and expert maintenance mode. Section 5.3 explains the communications between the MicroDroids in TKC.

### 5.3 Metadata

The standard definition of metadata is data about data. However, metadata also gives the data contextual understanding. Metadata is not useful without accompanying functionality capable of delivering it to the point of decision-making. There are really two types of metadata: static metadata that deals with structural information and dynamic metadata that deals with operational information (Giovinazzo, 2000, p. 307). We are talking about the static metadata here. This is the contextual understanding of the data structures themselves, not where it came from, but rather how it was transformed or how much of it was loaded into a database. We are also not dealing with the data dictionary information such as data types and sizes.

There are two categories of metadata exchange. Interchange is the concept where different processes share metadata and integration is the concept where a standard for both the structure and the content of the metadata is established (Giovinazzo, 2000, p. 308). This research is deals with the integration of metadata within a single system.

Metaknowledge is really metadata with functionality. This is what turns data into Usable Corporate Knowledge (UCK) and therefore turns the Data Warehouse into a Knowledge Base. MicroDroids are agents (Murch and Johnson, 1999) that give the Metaknowledge its functionality and change it from passive to active metadata. This group of MicroDroids along with their metadata is known as the Knowledge Collective. A Knowledge Collective is really a metadata repository (Marco, 2000) with functionality. This, in essence, turns passive metadata into active metadata.

MicroDroids are the metadata guardians, users and keepers. They understand the metadata so that the end user does not have to understand it. If you want to know something about the data, ask the MicroDroids. You do not even have to know which MicroDroid to ask for the specific information. The Knowledge Collective has a team of MicroDroids and a leader or coordinator that knows all the MicroDroids in the system. Ask it anything, it will find the answer.

Each MicroDroid has its own metadata that includes functionality on how to use its metadata. All of its functionality is designed to control system metadata, thereby controlling actual data. Even though a MicroDroid sounds very complex, it only has one major task to perform. If it needs additional information or has additional tasks to accomplish, it calls on other MicroDroids for help.

### 5.4 Communications

The multi-agent capabilities in TKC do not use any specific agent protocols as other multi-agent systems (FIPA, 2002a). The communications between the MicroDroids are handled using the Java Message Service (JMS) (Mahmoud, 2004). This is a loosely

coupled peer-to-peer communication system. The messages are asynchronous and the sender has no knowledge about the receiver. The MicroDroids are just clients and the controlling server for JMS is JBoss (Taylor et al., 2004).

Messages are sent out by the sending MicroDroid in a publish/subscribe methodology. This is a broadcast message. Once another MicroDroid answers, a pipeline is set up between the two MicroDroids using a point-to-point messaging methodology. IBM® WebSphere QualityStage handles the parsing, standardization, translation, and matching of the attributes and terms in the message.

Figure 5.5 is an example of what is happening during communications. Each box is an instance of the MicroDroid class that is represented by the label. The thick lines are active and the thin lines are waiting for use. The End User has asked the COORDINATOR MicroDroid for information. The COORDINATOR MicroDroid sends out a broadcast message. The CARDIOVASCULAR INFORMATION SYSTEM MicroDroid answers and a peer-to-peer connection is set up between them. The CARDIOVASCULAR INFORMATION SYSTEM MicroDroid then sends out a broadcast message. The CARDIOVASCULAR PHYSIOLOGY MicroDroid has not answered yet. When the CARDIOVASCULAR PHYSIOLOGY MicroDroid answers, a peer-to-peer with the CARDIOVASCULAR INFORMATION SYSTEM MicroDroid will be set up. The CIRCSIM-TUTOR MicroDroid is just waiting for a broadcast message that it is interested in answering.

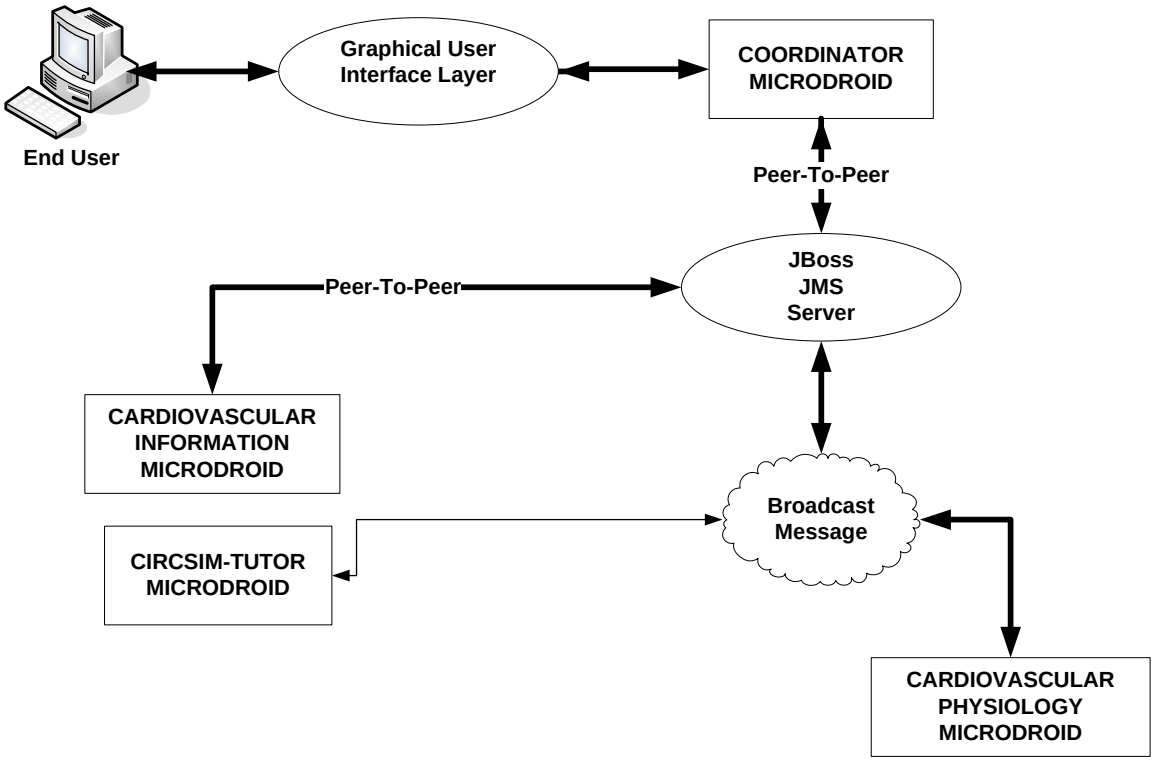


Figure 5.5. MicroDroid Architecture

# CHAPTER 6

# THE KNOWLEDGE COLLECTIVE

The Knowledge Collective (TKC) is the proposed solution for solving the knowledge base infrastructure problem. It is a multi-layer, multi-agent framework for dealing with reuse of information in an intelligent knowledge base. The data architecture for TKC was developed along with the supporting MicroDroids.

The idea is to show that this type of architecture will allow for information and data to be stored in a usable format (i.e., relational database tables, Java objects, rules, etc.) in a standard relational database. The end users will be able to store and retrieve the proper information by using a multi-layer, multi-agent infrastructure. This will also allow for the reusability of information across multiple applications (e.g., CIRCSIM-Tutor and GASP-Tutor) and allow disparate information from multiple applications to be stored in the same knowledge base.

TKC originally started out with a six-layer architecture as described in Section 6.1 and shown in Figure 6.1. This architecture would give great flexibility with the following drawbacks:

- Too much indirection would greatly slow down the process
- The MicroDroids would not be totally autonomous and self-contained

For these reasons, the architecture of TKC was changed to the four-layer architecture that is discussed in Section 6.2 and shown in Figure 6.3.

### 6.1 Original Architecture/Framework

The Knowledge Collective is a multi-agent system (Ferber, 1999; Weiss, 2000) composed originally of six layers as seen in Figure 6.1:

1. Graphical User Interface
2. Coordination
3. Application
4. Solution
5. Task
6. Database

These six layers make up the overall high-level ontology for TKC. This is an artifact ontology used for doing Qualitative Reasoning about The Knowledge Collective in general. The actual ontology model was discussed in the Chapter 3.

Each layer is composed of classes of MicroDroids. Therefore, there can be many instances of each MicroDroid class in Figure 6.3. The MicroDroids are described in Chapter 5.

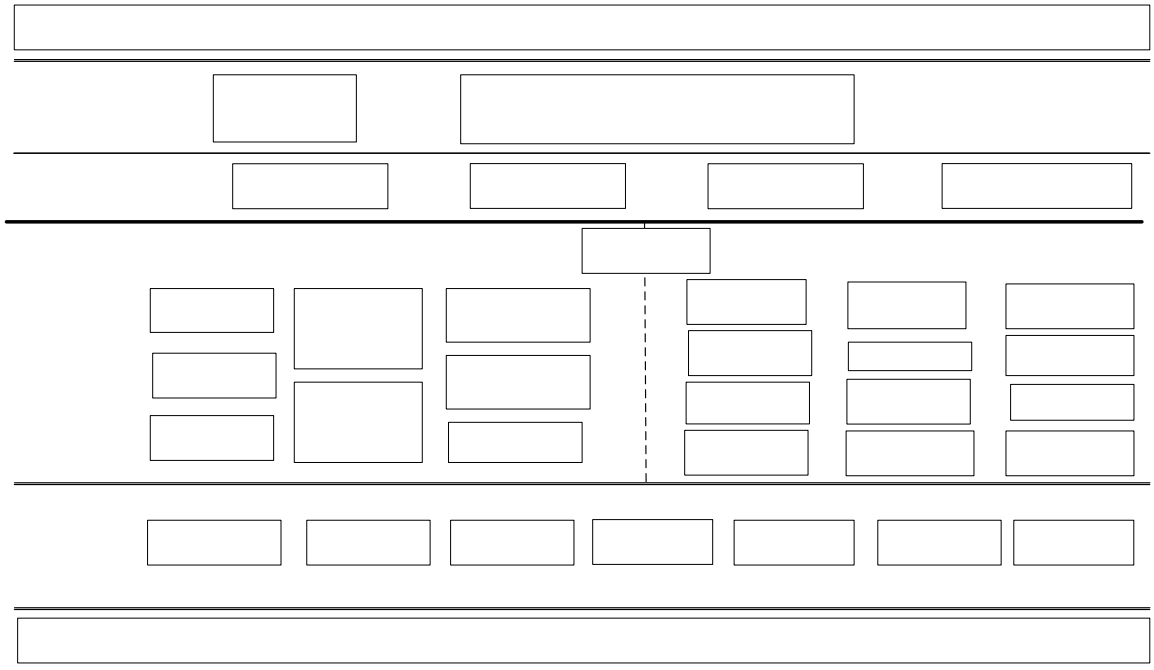

Figure 6.1. The New Knowledge Collective Architecture

**6.1.1 Graphical User Interface Layer.** The Graphical User Interface Layer is the portal into TKC and therefore, into the actual application information. It can be composed of many screens for the end user, the developer, and the expert to input and retrieve information, data and metadata from or to TKC. This is an application interface for the user. It gives the end user access to application information. It gives the developer the ability to add, delete, maintain or monitor the MicroDroids in each layer. It also allows the subject matter experts to view, add and update their subject areas. It is really the view into the COORDINATOR MicroDroid.

**6.1.2 Coordinator Layer.** The Coordinator Layer controls TKC. This layer always contains two specific purpose MicroDroids: COORDINATOR MicroDroid and USER PROFILE MicroDroid. The COORDINATOR MicroDroid works with the User

Interface to deal with all tasks from the Graphical User Interface Layer whether they come from the end user or a developer. The COORDINATOR MicroDroid also sets up a common goal that starts a session. The rest of the MicroDroids in TKC cooperate to satisfy this goal. The COORDINATOR MicroDroid can find out about all of the MicroDroids in TKC by asking them for information. If MicroDroids are added or deleted, or their functionality is changed, the COORDINATOR MicroDroid will know about these changes by inquiring about information by broadcasting a message. The information is never stored, it is always asked for by the COORDINATOR MicroDroid. It sends out orders along with the goals to find one or more MicroDroids in the Application Layer to solve end user problems. However, the COORDINATOR MicroDroid can also interface with any MicroDroid in TKC to solve various development and maintenance problems.

The COORDINATOR MicroDroid is the keeper of the information about TKC. It contains the TKC ontology (see Figure 6.2). It also controls the ONTIE ruleset for the ontology and any other needed information about TKC. If you have questions about TKC, the COORDINATOR MicroDroid is capable of answering those questions.

Another important role that the COORDINATOR MicroDroid plays is to help an expert (e.g., Joel Michael, Professor of Physiology at Rush Medical Center in Chicago, IL) maintain his Cardiovascular Physiology information. The COORDINATOR MicroDroid will send out a broadcast message looking for the MicroDroid that knows about the Cardiovascular Physiology System. When the CARDIOVASCULAR PHYSIOLOGY MicroDroid (see Figure 6.1) answers, a direct pipe between the COORDINATOR MicroDroid and the CARDIOVASCULAR PHYSIOLOGY

MicroDroid is set up along with the proper user interface for maintenance.

A USER PROFILE MicroDroid can represent an end user, a domain expert, or a developer. It maintains any known information about a specific user.

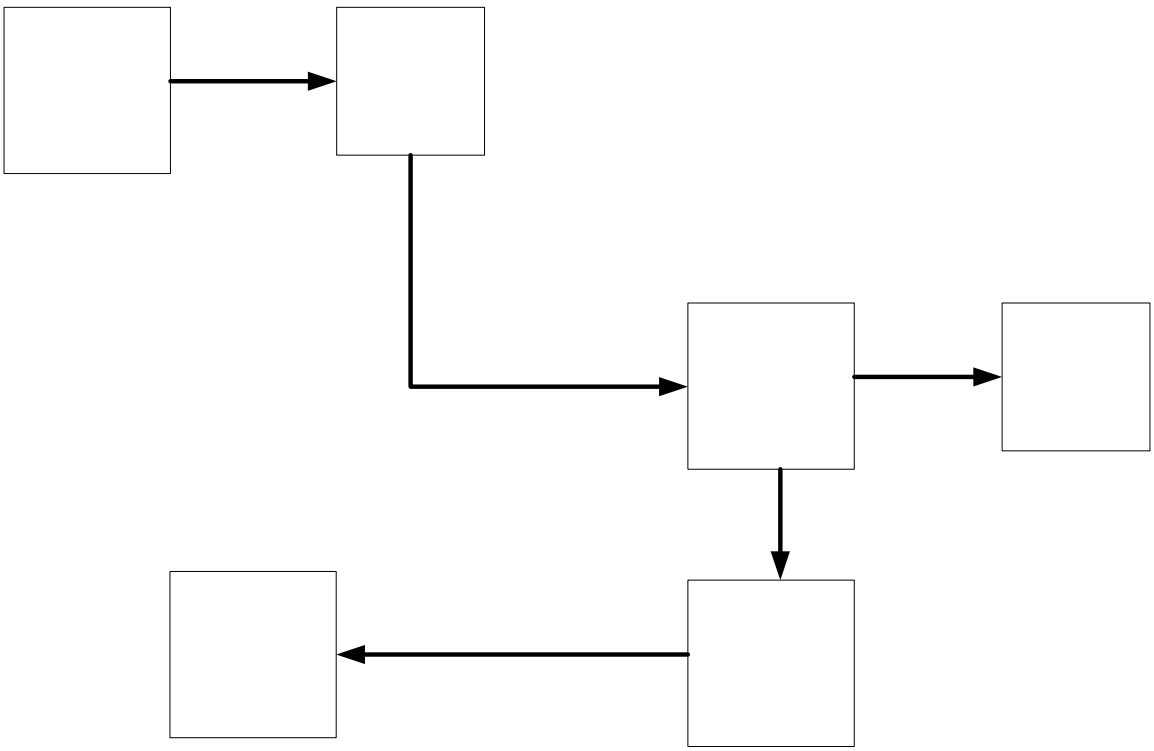

Figure 6.2. The Knowledge Collective Ontology

**6.1.3 Application Layer.** The Application Layer contains a MicroDroid for each application in TKC. These MicroDroids get their orders from the COORDINATOR MicroDroid. They work individually to solve application problems. They use from one to many MicroDroids in the Solution Layer to access needed information to solve the problem. This is the only layer that is application specific. The Knowledge Collective example in Figure 6.1 is designed to support three medical tutoring systems (CIRCSIM-Tutor, GASP-Tutor and the Concept Map Tutor for the Baroreceptor Reflex) and a

The Knowledge Collective Has

Cardiovascular Information System.

CIRCSIM-Tutor (Michael et al., 2003) deals with the Baroreceptor Reflex, the part of the Circulatory System that controls the blood pressure. GASP-Tutor is a new tutoring system (still being developed) that deals with the pulmonary system with a focus on the two interacting negative feedback systems that control breathing and gas exchange in the lungs. The Concept Map Tutor (Kim et al., 1989) helps the user manage concepts and their relationships in the Baroreceptor Reflex domain. The Cardiovascular Information System (development just started) answers questions about cardiovascular physiology.

**6.1.4 Solution Layer.** There are many classes of MicroDroids in the Solution Layer. Each one knows how to deal with a specific domain as seen in Figure 6.1. They get their marching orders from an Application MicroDroid. They can work individually or in teams to solve an application problem. They use from one to many MicroDroids in the Task Layer to access, delete, update, or insert application data in the Database Layer.

These MicroDroids are very information specific. If you add a new Application MicroDroid that will use the same solution information, the same Solution MicroDroid will respond. If changes are made to the application data structure, these MicroDroids will not have to be modified. There are specific MicroDroids that can deal with domain specific information and others that deal with application specific information. For instance, CIRCSIM-Tutor uses all the MicroDroids in this layer except the Chemistry and Respiratory Physiology MicroDroids, which are specific to GASP-Tutor. GASP-Tutor will use all the MicroDroids in the layer except the Baroreceptor Reflex Equations and

Baroreceptor Reflex Anatomy MicroDroids, as can be seen in Figure 6.1. This configuration really supports the knowledge and ontology reuse capabilities of TKC.

**6.1.5 Task Layer.** There are many MicroDroids in the Task Layer. Each one knows how to solve specific tasks. They get their marching orders from the Solution Layer. They work individually or in teams to deal with application data in the Database Layer in response to calls from the Solution Layer.

The MicroDroids in this layer are not application specific. They are data architecture specific. The only time you need to add a new one or modify an existing one is if there is a change to the actual data structure. An example would be the Frame MicroDroid. It uses the Frame Building Language (Yusko, 1984), which understands how to deal with frames and relationships like semantic networks. Even though the application data is stored in relational format in a relational database in the Database Layer, this MicroDroid knows how to interpret frames properly. However, there is one specific MicroDroid, DATABASE, which knows where to find all the information about the Database Layer.

**6.1.6 Database Layer.** The Database Layer can contain from one to many databases. There is a DATABASE MicroDroid in the Task Layer that knows about all the databases in this layer and how to access them. It knows about all the databases and what they contain. There can be connectivity to many databases, but there is only one centralized location to maintain the information about the databases. This layer is where the information used by the MicroDroids is stored. The database management system is an industry standard relational database.

## 6.2 Architectural Changes

The Knowledge Collective's new architecture is composed of four layers as seen in Figure 6.3, instead of six as seen in Figure 6.1:

1. Graphical User Interface
2. Coordination
3. Application
4. Solution

Layer 5 (Task Layer) and Layer 6 (Database Layer) have been eliminated (see Figure 6.3). This change speeds up processing, but more importantly, it makes the MicroDroids autonomous and self-contained. Each MicroDroid now must know how to access its own information and where that information lives. This also opens the door for distributing the MicroDroids across multiple platforms. This change has a major affect on the Solution Layer, since the MicroDroids in the Solution Layer control all the domain specific information.

**6.2.1 Graphical User Interface Layer.** The Graphical User Interface Layer has not changed from the original design. It is the portal into TKC and therefore, into the actual application information. It can be composed of many screens for the end user, the developer, and the expert to input and retrieve information, data and metadata from or to TKC. It gives the end user access to application information. It gives the developer the ability to add, delete, maintain or monitor the MicroDroids in each layer. It also allows the subject matter experts to view, add and update their subject areas.

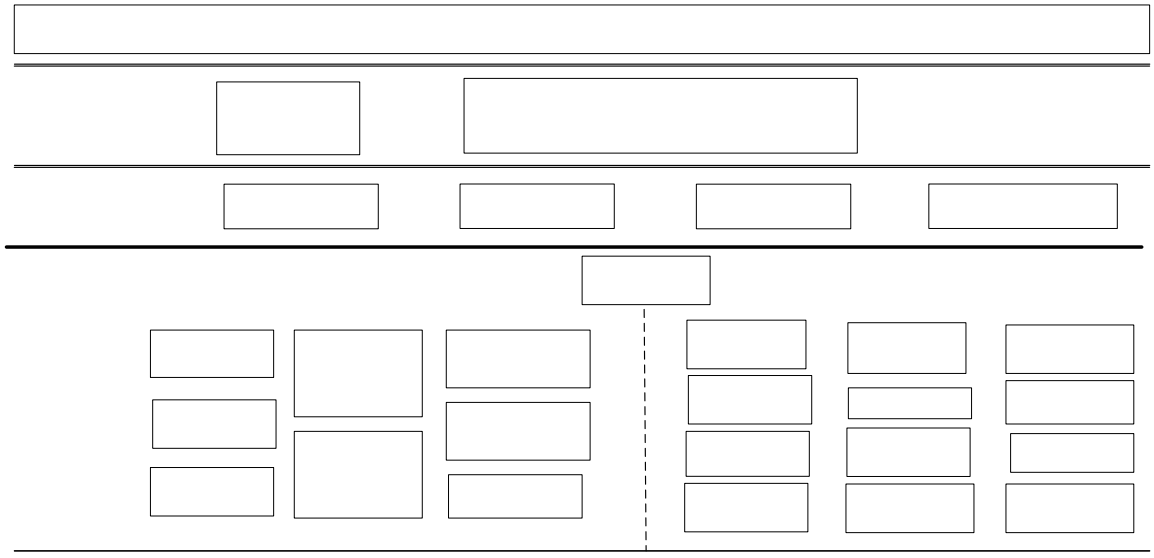

Figure 6.3. The New Knowledge Collective Architecture

**6.2.2 Coordinator Layer.** The Coordinator Layer controls TKC. It has not changed from the original design. This layer always contains two specific purpose MicroDroids: COORDINATOR MicroDroid and USER PROFILE MicroDroid. The COORDINATOR MicroDroid works with the Graphical User Interface Layer to communicate with end users, developers, and domain experts. The COORDINATOR MicroDroid also sets up a common goal that starts a session. The rest of the MicroDroids in TKC cooperate to satisfy this goal. The COORDINATOR MicroDroid can find out about all of the MicroDroids in TKC by querying for information. If MicroDroids are added or deleted, or their functionality is changed, the COORDINATOR MicroDroid will inquire about these changes by broadcasting a message. It sends out orders along with the goals to find one or more MicroDroids in the Application Layer to solve end user problems. However, the COORDINATOR MicroDroid can also interface with any MicroDroid in TKC to solve various development and maintenance problems.

Coordination Layer

Application Layer

Solution Layer

Concept Map

Predictive Information

The COORDINATOR MicroDroid is the keeper of the information about TKC. It has the TKC ontology (see Figure 6.2); it has the ONTIE ruleset for the ontology and any other needed information about TKC. If you want to ask questions about TKC, the COORDINATOR MicroDroid is capable of answering those questions.

Another important role that the COORDINATOR MicroDroid plays is to help an expert (e.g., Joel Michael, Professor of Physiology at Rush Medical Center in Chicago, IL) maintain his Cardiovascular Physiology information. The COORDINATOR MicroDroid will send out a broadcast message looking for the MicroDroid that knows about the Cardiovascular Physiology System. When the CARDIOVASCULAR PHYSIOLOGY MicroDroid (see Figure 6.3) answers, a direct pipe between the COORDINATOR MicroDroid and the CARDIOVASCULAR PHYSIOLOGY MicroDroid is set up along with the proper user interface for maintenance.

A USER PROFILE MicroDroid can represent an end user, a domain expert, or a developer. It controls the information about a specific user.

**6.2.3 Application Layer.** The Application Layer has not changed from the original design. It contains a MicroDroid for each application in TKC. These MicroDroids get their orders from the COORDINATOR MicroDroid. They work individually to solve application problems. They use from one to many MicroDroids in the Solution Layer to access, delete or insert application data in the Database Layer. This is the only layer that is application specific. The Knowledge Collective example in Figure 6.1 is designed to support three medical tutoring systems (CIRCSIM-Tutor, GASP-Tutor and Concept Map Tutor for the Baroreceptor Reflex) and a Cardiovascular Information System.

CIRCSIM-Tutor (Michael et al., 2003) deals with the Baroreceptor Reflex, the negative feedback component of the Circulatory System that controls the blood pressure. GASP-Tutor is a new tutoring system (still being implemented) that deals with the pulmonary system with a focus on the two interacting negative feedback systems that control breathing and gas exchange in the lungs. The Concept Map Tutor (Kim et al., 1989) helps the user manage concepts and their relationships in the Baroreceptor Reflex domain. The Cardiovascular Information System (development just started) answers questions about cardiovascular physiology.

**6.2.4 Solution Layer.** Since the Task Layer and Database Layer have been removed, the major changes to the architecture are in this layer. This layer controls most of the information in the knowledge base. There are many classes of MicroDroids in the Solution Layer. Each one knows how to deal with a specific domain as seen in Figure 6.3. They get their marching orders from an Application MicroDroid. They can work individually or in teams to solve an application problem.

These MicroDroids are very information specific. If you add a new Application MicroDroid that will use the same solution information, the same Solution MicroDroid will respond. If changes are made to the application data structure, these MicroDroids will not have to be modified. There are specific MicroDroids that can deal with domain specific information and others that deal with application specific information. For instance, CIRCSIM-Tutor uses all the MicroDroids in this layer except the Chemistry and Respiratory Physiology MicroDroids, which are specific to GASP-Tutor. GASP-Tutor will use all the MicroDroids in the layer except the Baroreceptor Reflex Equations and

Baroreceptor Reflex Anatomy MicroDroids, as can be seen in Figure 6.3. This configuration really supports the knowledge and ontology reuse capabilities of TKC.

### 6.3 TKC Ontology

All MicroDroids are experts in a specific domain. The COORDINATOR MicroDroid is the expert for TKC. Therefore, it contains the information about TKC, such as the TKC ontology in Figure 6.2 and the Ontology Inference Engine that controls TKC. This ontology is an Artifact Ontology (see Section 3.1.1) that contains all the parts and relationships of The Knowledge Collective. This also gives the user the ability to ask questions about TKC in general.

### 6.4 TKC Process Example

A good way of looking at TKC is that each MicroDroid has a story to tell. If an expert is a person that remembers the right story at the right time (Schank, 1990), then the right MicroDroid has to respond at the right time as an expert.

When dealing with a student, TKC is trying to tell a story based on the input from the student. A story is based on many parts. The COORDINATOR MicroDroid sets up the storyline. Based on the storyline and the ontology, different MicroDroids fill in different parts of the overall story with their stories. Each one is an expert and tells the right story at the right time. Therefore, like a human expert, TKC remembers the right story at the right time by combining the stories of all the MicroDroids that answer.

The system only understands the environment of the session with a user by the model of the environment that it is given. This model is an ontology. TKC is a

collection of ontologies. Each session with a user has its own ontology. When a user logs into TKC, the user is working through the User Interface Layer. This is the portal into the COORDINATOR MicroDroid and to a USER PROFILE MicroDroid. These two MicroDroids are a part of all sessions and supply the initial ontology for the session. Then, the session ontology grows as new MicroDroids are added to the session.

The COORDINATOR MicroDroid sets up a subject area ontology. The purpose of the COORDINATOR MicroDroid is to ascertain what the user wants to do. The USER PROFILE MicroDroid sets up the user ontology so that the system can understand the user. The COORDINATOR MicroDroid then ascertains what the user wants to do and sets up the subject. If the user wants to diagnose a circulation problem, the COORDINATOR MicroDroid broadcasts the request to the Application Layer. A CARDIOVASCULAR INFORMATION SYSTEM MicroDroid would answer and a session pipeline would be set up with the COORDINATOR MicroDroid for the user. The COORDINATOR MicroDroid communicates with the user via the User Interface Layer. If the user wants to learn about the Baroreceptor Reflex, the CIRCSIM-TUTOR MicroDroid would answer and a pipeline would be set up. If the user wants to understand circulatory chemistry issues, then the COORDINATOR MicroDroid broadcasts the request to the Application Layer. The GASP-TUTOR MicroDroid answers and a session pipeline is set up with the COORDINATOR MicroDroid for the user. The COORDINATOR MicroDroid communicates with the user via the User Interface Layer. If no MicroDroid answers, a list of possible applications is given to the user.

If the CIRCSIM-TUTOR MicroDroid answers, a session ontology is set up. In

this case, the CIRCSIM-TUTOR MicroDroid has a process ontology as shown in Figure 3.1 (Evens and Michael, in press). Every time a new MicroDroid answers, a new piece of the session ontology is produced. Each MicroDroid understands the environment it works in and adds a piece of the ontology to the session ontology. Therefore, the ontology for the session is developed dynamically as new MicroDroids become part of it. In other words, the ontology of the session grows as the session develops. A network of MicroDroids is developed with each bringing its own piece of the session ontology. As the session grows, the number of MicroDroids needed will increase. When a session is complete, the MicroDroids involved drop out and take their ontologies with them.

When a session is complete, the instantiated MicroDroids save what they have learned for future MicroDroids to use. Then they eliminate themselves from the system. At any given time, there is always a MicroDroid from each class waiting to serve the end user.

# CHAPTER 7

# CONCLUSION

The Knowledge Collective is like a whole team of very specialized experts that know how to retrieve, store and analyze data. This is what turns metadata into meta-knowledge and thereby turns data into information and allows it to be delivered to the point of decision-making. At that point, the information in The Knowledge Collective becomes UCK (Usable Corporate Knowledge) and a corporate asset. This will also give the novice end user the knowledge needed to make decisions like an expert. This also allows the best subject expert to make decisions faster and more consistently. The subject expert can then spend more time improving the decision support process.

If the intension is to share knowledge and the ontologies for that knowledge, the maintenance can become overwhelming when using general agents with an ontology server or ontology agent concept. If the goal is to share knowledge between systems, the agents need to be more fine-tuned and task-specific. The ontologies for these task-specific agents need to be part of the agent with their own Ontology Inference Engine that performs Qualitative Reasoning. This is the concept behind MicroDroids. With this concept, as the end user session grows, the number of MicroDroids used also grows. The session ontology then grows dynamically as the number of MicroDroids that are needed increases. This means that the Qualitative Reasoning capabilities must also be expanded as new MicroDroids are added to solve specific problems.

The concept of embedding models (ontologies) into tutoring systems is not new (Bredeweg and Forbus, 2003; Glass, 1999; Khuwaja, 1994). Better standards are needed and using UML models that can generate Java objects is a good start.

TKC is not only a collection of knowledge managed by the MicroDroids; it is also a collection of ontologies modeling the knowledge. This makes TKC an intelligent reusable knowledge base for doing Qualitative Reasoning.

### 7.1 Summary

I have designed and developed The Knowledge Collective to make information accessible, maintainable, and reusable by dividing information into small domain specific units. A MicroDroid manages each unit. All users, whether they are using the information or are experts maintaining their specific domain information, must be able to perform the needed tasks without any knowledge of structure or storage of the information. This allows the information to be stored in the most efficient way possible for any particular type of information: data, metadata, ontologies, or rules.

One of the major reasons for information is to answer questions and solve problems. Applications like CIRCSIM-Tutor need both Baroreceptor Reflex information and tutoring information. Therefore, the information from many different domain specific units need to be combined to solve the total problem.

This thesis describes the goals and the implementation of The Knowledge Collective framework. The implementation is unique because of the combination and use of many general commercial and open source products rather than developing vertical modules specific to the framework. The main emphasis is to design and implement a framework that can handle multiple smaller, more manageable ontologies. In addition, avoid developing yet another ontology language, application programming interface, or struggling with how to best map or merge different existing ontologies.

The Knowledge Collective framework evolved because of a challenge to devise a methodology for reuse of information among different systems that is maintainable and scalable. What makes this thesis unique is the combination of many technologies that are brought to bear on the problem. Each technology in and of itself is not unique, but the combination is unique. It is also unique to have many small ontologies working together instead of a single large ontology in a multi-agent system. Each MicroDroid is unique because of the types of information that they can bring to bear to solve a problem in a specific domain: data, metadata, ontologies and an Ontology Inference Engine. Another unique feature is that the ontologies are stored as Java objects with a production rule based inference engine used for Qualitative Reasoning.

The Knowledge Collective approach allows information, mainly ontological information, from many areas of expertise to be used together to solve problems without standardized ontology languages and APIs. The use of the multi-agent capability with the Ontology Inference Engine makes this possible.

### 7.2 Tasks Implemented to Answer Research Questions

The purpose of this thesis is to answer some very specific questions about intelligent knowledge bases. The central question answered is: Does TKC provide a good framework for developing intelligent knowledge bases with reuse capabilities? The answer is yes. The following was learned in the process of answering the research questions in Section 1.3:

- The Task Layer and the Database Layer were eliminated from the TKC leaving 4 layers instead of 6 layers

- IBM® Cloudscape™ (Saunders and Anderson, 2004) has been imbedded into the MicroDroids to replace the Task Layer and the Database Layer

- Each MicroDroid is self contained and autonomous

- MicroDroids are specialized agents, not a community of nested agents that get instantiated as needed and then go away after they are finished

- The four types of ontologies used in TKC were sufficient to define the domain information needed: Artifact Based, Process Based, Concept Map Based, and Flow Based.

- Qualitative Reasoning was accomplished using an ONTIE (Ontology Inference Engine)

- Multiple applications were able to reuse the same information.

- Multiple commercial tools were used to implement TKC, which demonstrates that off-the-shelf tools can be combined to build a functioning knowledge base.

### 7.3 Future Research

The research described in this thesis is just the beginning. It lays the foundation for The Knowledge Collective framework. It sets up the core functionality of a MicroDroid. The items below describe opportunities for future research and

development. Even though they have been mentioned in this thesis they have not yet been implemented:

- The Java Message Service (JMS) worked well as a concept and has been used in many applications. I plan to use it in the future to maintain communications between the MicroDroids.

- The Truth Maintenance System and learning capabilities need more research; I hope to implement this in the future.

- Natural language parsing with IBM® WebSphere QualityStage has been used many times as a parser in industrial applications and should work here. The parsers for CIRCSIM-Tutor were developed specifically for the application. More research on using a generic commercial parser like QualityStage needs to be accomplished in the future.

- JESS is a rule-based system developed in Java (Friedman-Hill, 2003) that is open source. Since it is an Open Source System, it should be investigated and considered as a replacement to the commercial system ILOG JRules for ONTIE. This will allow ONTIE to have unlimited expense free use at universities.

- The tutoring information (see Figure 6.3) needs to be completely implemented in the future.

# BIBLIOGRAPHY

Alberts, L. K., Wognum, R. M. and Mars, N. J. (1992). Structuring Design Knowledge on the Basis of Generic Components. In: Gero, J. S. (Ed.). *Artificial Intelligence in Design '92.* Boston, MA: Kluwer Academic Publishers. pp. 639-656.

Alonso, E. (2002). AI and Agents. In: *AI Magazine.* Vol. 23, No. 3. Fall 2002. pp. 25-29.

Barsalow, L. W., and Bower, G. H. (1984). Discrimination Nets as Psychological Models. In: *Cognitive Science,* Vol. 8, No. 1. pp. 1-26.

Binstock, A. (2005). Why Use an Embedded Database? http://www.devx.com/IBMCloudscape/Article/27622. 6/01/2005.

Birnbaum, L., and Collins, G. (1989). Reminding and Engineering Design Themes: A Case Study in Indexing Vocabulary. In *Proceedings of Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann Publishers. pp. 47-51.

Bobrow, D. G. (1984). Qualitative Reasoning about Physical Systems: An Introduction. In *Qualitative Reasoning about Physical Systems.*. Cambridge, MA: MIT Press. pp. 1-6.

Brachman, R. J., and Levesque, H. J. (2004). *Knowledge Representation and Reasoning.* New York, NY: Morgan Kaufmann Publishers.

Bradshaw, J. M. (Ed.) (1997). *Software Agents.* Menlo Park, CA: The AAAI Press.

Bratko, I., Mozetic, I., and Lavrac, N. (1989). *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems.* Cambridge, MA: The MIT Press.

Bredeweg, B. and Forbus, K. D. (2003). Qualitative Modeling in Education. *AI Magazine*, Vol. 24, No. 4, pages 35-46.

Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., and Grose, T. (2004). *Eclipse Modeling Framework.* Reading, MA: Addison-Wesley.

Charniak, E., Riesbeck, C. K., and McDermott, D. V. (1980). *Artificial Intelligence Programming.* Hillsdale, NJ: Lawrence Erlbaum.

Cho, B. (2000). Dynamic Planning Models to Support Curriculum Planning and Multiple Tutoring Protocols in Intelligent Tutoring Systems. Ph.D. Dissertation. Chicago, IL: Illinois Institute of Technology.

Evens, M. W., and Michael, J. A. (in press). *One on One Tutoring by Humans and Computers.* Mahwah, NJ: Lawrence Erlbaum.

Evens, M. W., Litowitz, B. E., Markowitz, J. A., Smith, R. N. and Wemer, 0. (1983). *Lexical Semantic Relations: A Comparative Survey.* Edmonton, Alberta, Canada: Linguistic Research, Inc.

Falkenhainer, B., and Forbus, K. D. (1988). Setting up Large-Scale Qualitative Models. In: *Proceedings of the American Association for Artificial Intelligence* (AAAI-90). St. Paul, MN. pp. 301-301.

Farquhar, A., Fikes, R., and Rice J. (1997). The Ontolingua Server: A Tool for Collaborative Ontology. In: *International Journal of Human-Computer Studies.* Vol. 46. pp. 707-728.

Fellbaum, C. (Ed.) (1999). *WordNet: An Electronic Lexical Database.* Cambridge, MA: The MIT Press.

Fensel, D. (2004). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce.* Second Edition. New York, NY: Springer.

Fensel, D., Horrocks, I., Van Harmelen, I., Decker, S., Erdmann, M., and Klein, M. (2005). OIL in a Nutshell. http://www.cs.vu.nl/~ontoknow/oil/downl/oilnutshell.pdf. 6/12/2005.

Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Reading, MA: Addison-Wesley.

FIPA (2002a). FIPA Abstract Architecture Specification. Foundation for Intelligent Physical Agents. Geneva, Switzerland.

FIPA (2002b). FIPA Ontology Service Specification. Foundation for Intelligent Physical Agents. Geneva, Switzerland.

Forbus, K. D. (1984). Qualitative Process Theory, In: *Artificial Intelligence* Vol. 24 No. 1. pp. 85-168.

Forbus, K. D. (1985). Qualitative Process Theory. In: D. Bobrow (Ed.) *Qualitative Reasoning about Physical Systems.* Cambridge, MA: The MIT Press. pp. 85–168.

Forbus, K. D. (1996). Qualitative Reasoning. In: A. Tucker (Ed.), *CRC Handbook of Computer Science*, Boca Raton, FL: CRC Press. 817-833.

Forbus, K. D., and deKleer, J. (1993). *Building Problem Solvers.* Cambridge, MA: The MIT Press.

Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing.* Indianapolis, IN: Wiley Publishing, Inc.

Franklin, S., and Graesser, A. (1996). Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents. In: *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.* New York, NY: Springer-Verlag. pp. 21-35.

Freedman, R. (1996). Interaction of Discourse Planning, Instructional Planning and Dialogue Management in an Interactive Tutoring System. Ph.D. Dissertation. Evanston, IL: Northwestern University.

Freedman, R., and Evens, M. W. (1997). The Use of Multiple Knowledge Types in an Intelligent Tutoring System. In: *Proceedings of the Cognitive Science Conference.* Stanford, CA. p. 20.

Freedman, R., Zhou, Y., Glass, M. S., Kim, J. H., and Evens. M. W. (1998). Using Rule Induction to Assist in Rule Construction for a Natural Language-based Intelligent Tutoring System. In: *Proceedings of 20th Annual Cognitive Science Conference.* Madison, WI, August. pp. 362-367.

Friedman-Hill, E. (2003). *JESS In Action: Rule-Based Systems in Java.* Greenwich, CT: Manning Publications Co.

Gallardo, D., Burnett, E., and McGovern, R. (2003). *Eclipse in Action: A Guide for Java Developers.* Greenwich, CT: Manning Publishing Co.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F. and Tu, S. W. (2005). The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. http://www.smi.stanford.edu/pubs/SMI_Reports/SMI-2002-0943.pdf. 4/25/2005.

Giovinazzo, W. (2000). *Object-Oriented Data Warehouse Design: Building a Star Schema.* Upper Saddle River, NJ: Prentice Hall PTR.

Glass, M.S. (1999). Broadening input understanding in a language-based intelligent tutoring system. Ph.D. Dissertation, Computer Science Department. Chicago, IL: Illinois Institute of Technology.

Gomez-Perez, A., Fernandez-Lopez, M., and Corcho, O. (2004). *Ontological Engineering.* New York, NY: Springer.

Haase, K. (2002). *Java Message Service API Tutorial.* Palo Alto: Sun Microsystems, Inc.

Hewett, M. (2005). Algernon Overview. http://algernonj.sourceforge.net/doc/overview.html. 1/4/2005.

Hutcheson, D. S. (2003). Architecture Comes Alive for IBM. In *Enterprise Architect.* Fawcett Technical Publications Inc. Palo Alto, CA. Vol. 1 No. 2. pp. 41-45.

ILOG (2004). ILOG JRules 4.6.2 Rule Engine User's Manual. Mountain View, CA: ILOG Inc.

ILOG (2005). ILOG JRules 5.1 Rule Builder Tutorial. Mountain View, CA: ILOG Inc.

Ingargiola, G. (2005). Problem Solving and Truth Maintenance Systems. http://www.cis.temple.edu/~ingargio/cis587/readings/tms.html. 6/1/2005.

Iverson, W. (2005). *Hibernate: A J2EE™ Developer's Guide.* Reading, MA: Addison-Wesley.

Khuwaja, R. A. (1994). A Model of Tutoring: Facilitating Knowledge Integration Using Multiple Models of the Domain. Ph.D. Dissertation. Chicago, IL: Illinois Institute of Technology.

Khuwaja, R. A., and Patel, V. (1996). A Model of Tutoring Based on the Behavior of Effective Human Yutors. In: *Proceedings of the Third International Conference on Intelligent Tutoring Systems (ITS '96),* Montreal, Canada. pp. 130-138.

Khuwaja, R A., Rovick, A. A., Michael, J. A., and Evens, M. W. (1992). Knowledge Representation for an Intelligent Tutoring System Based on a Multilevel Causal Model. In: *Proceedings of ITS '92,* Berlin: Springer. pp. 217-224.

Khuwaja, R. A., Evens, M. W., Rovick, A. A., and Michael, J. A. (1994). Architecture of CIRCSIM-TUTOR (v.3): A Smart Cardiovascular Physiology Tutor. *Proc. CBMS94*. Winston-Salem, NC, June 10-11. pp. 158-163.

Kim, N. (1989). An Intelligent Tutoring System for Physiology, Ph.D. Dissertation, Chicago, IL: Illinois Institute of Technology,

Kim, N., Evens, M. W., Michael, J. A., and Rovick, A. A. (1989). An intelligent tutoring system for circulatory physiology. In: H. Maurer, (ed.), *Computer Assisted Learning.* Berlin: Springer-Verlag. pp. 254-266.

Kim, Y. C., Evens, M. W. and Trace D. (2001). Building Concept Maps from Medical Index Terms. In: *Proceedings of the Twelfth Midwest Artificial Intelligence and Cognitive Science Conference.* Chicago, IL. pp. 75-80.

Kim, Y. C., Evens, M. W., Michael, J. and Trace D. (2002). Physiology Tutorials Using Concept Mapping. In: *Proceedings of the Thirteenth Midwest Artificial Intelligence and Cognitive Science Conference.* Chicago, IL. pp. 61-64.

Knublauch, H. (2005). An AI Tool for the Real World: Knowledge Modeling with Protégé. http://www.javaworld.com/javaworld/jw-06-2003/jw-0620-protege.html. 1/28/2005.

Kuipers, B. (1984). Commonsense Reasoning about Causality: Deriving Behavior from Structure. In: *Qualitative Reasoning About Physical Systems.*. Cambridge, MA: MIT Press. pp. 169-204.

Lee, C. H., and Evens, M. W. (2003). Interleaved Syntactic and Semantic Processing for CIRCSIM-Tutor Dialogues. In *Proceedings of the Fourteenth Midwest Artificial Intelligence and Cognitive Science Conference.* Cincinnati, OH. pp. 69-73.

Lee, C. H., Seu, J. H., and Evens, M. W. (2002). Building an Ontology for CIRCSIM-Tutor. In: *Proceedings of the Thirteenth Midwest Artificial Intelligence and Cognitive Science Conference.* Chicago, IL. pp. 161-168.

Lee, Y. H. (1990). Handling Ill-Formed Natural Language Input for an Intelligent Tutoring System. Ph.D. Dissertation, Chicago, IL: Illinois Institute of Technology.

Lee, Y. H., and Evens, M. W. (1998). Natural Language Interface for an Expert System. *In: Expert Systems (International Journal of Knowledge Engineering).* November, Vol. 15, No.4, pp. 233-239.

Mahmoud, Q. H. (2004). Getting Started with Java Message Service (JMS). http://java.sun.com/developer/technicalArticles/Ecommerce/jms/. 4/23/2005.

Marco, D. (2000). *Building and Managing the Meta Data Repository: A Full Lifecycle Guide.* New York, NY: John Wiley & Sons, Inc.

Martin, C. (1989). Indexing Using Complex Features. In: *Proceedings of Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann Publishers. pp. 26-30.

May, D., and Taylor, P. (2003). Knowledge Management with Patterns. *Communications of the ACM.* Vol. 46, No. 7. pp. 94-99.

Michael, J. A., Rovick, A. A., Glass, M.S., Zhou, Y., and Evens, M. (2003). Learning from a Computer Tutor with Natural Language Capabilities. *Interactive Learning Environments,* Vol. 11, No.3, pp. 233-262.

Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach.* pp. 83-134, Palo Alto, CA: Tioga Publishing Co.

Mills, B. (2001). Using the Atlas Planning Engine to Drive an Intelligent Tutoring System. *FLAIRS 2001*, Menlo Park, CA: AAAI Press. pp. 211-215.

Milzner, K., and Harbecke, A. (1992). Incremental Learning for Improved Decision Support in Knowledge Based Design Systems. In: Gero, J. S. (Ed.). *Artificial Intelligence in Design '92.* Boston, MA: Kluwer Academic Publishers. pp. 719-738.

Morgan, T. (2002). *Business Rules and Information Systems: Aligning IT with Business Goals.* Boston, MA: Addison-Wesley.

Mundy, J. (2002). Smarter Data Warehouses: We Need a Better Analytic Language Than SQL. http://www.intelligententerprice.com/010216/print/webhouse.htm. 2/03/2002.

Murch, R., and Johnson, T. (1999). *Intelligent Software Agents.* Upper Saddle River, NJ: Prentice Hall PTR.

Neches, R., Langley, P. and Klahr, D. (1987). Learning, Development, and Production Systems. In: Klahr, D., Langley, P. and Neches, R. (Ed.). *Production System Models of Learning and Development.* Cambridge, MA: The MIT Press. pp. 1-53.

Noy, N. F., Rubin, D. L., and Musen, M. A. (2004) Making Biomedical Ontologies and Ontology Repositories Work. In: *IEEE Intelligent Systems, Vol.* 19 No. 6. pp. 78-80.

Omondo (2005). *EclipseUML Studio Product Documentation: EclipseUML & Eclipse Database (Version 1.1.0).* http://www.download-omondo.com/documentation/EclipseUMLStudioDocumentation.pdf. 4/23/2005.

Porter, B. W. (1989). Similarity Assessment: Computation vs. Representation. In: *Proceedings of Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann Publishers. pp. 82-84.

Ross, R. G. (2003). *Principles of the Business Rule Approach.* Boston, MA: Addison-Wesley.

Rovick, A. A., and Michael, J. A. (1986). CIRCSIM: An IBM PC Computer Teaching Exercise on Blood Pressure Regulation. Paper presented at the XXX IUPS Congress, Vancouver, Canada.

Rychener, M. D. (1988). Research in Expert Systems for Engineering Design. In: Rychener, M. D. (Ed.). *Expert Systems for Engineering Design.* Boston, MA: Academic Press. pp. 1-34.

Saunders, K., and Anderson, J. (2004). Cloudscape Version 10: A Technical Overview. http://www-106.ibm.com/developerworks/db2/library/techarticle/dm-0408anderson/index.html. 12/17/2004.

Schank, R. C. (1990). *Tell Me A Story: A New Look at Real and Artificial Memory.* New York, NY: Charles Scribner's Sons.

Sturges, R. H. (1992). A Computational Model for Conceptual Design Based on Function Logic. In: Gero, J. S. (Ed.). *Artificial Intelligence in Design '92.* Boston, MA: Kluwer Academic Publishers. pp. 757-72.

Taylor, L., and The JBoss Group (2004). Getting Started with JBoss: J2ee applications on the JBoss 3.2.x Server. http://www.jboss.org/index.html?module=downloads&op=displayCategory&authid=b589041aeedbb907344975f1756201ac&categoryId=8. 4/23/2005.

von Halle, B. (2002). *Business Rules Applied.* New York, NY: John Wiley & Sons, Inc.

Waltz, D. L. (1989) Is Indexing Used for Retrieval? In: *Proceedings of Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann Publishers. pp. 41-44.

Waterman, D. A. (1976). *An Introduction to Production Systems.* Rand Paper Series, Santa Monica, CA: The Rand Corp.

Weiss, G. (Ed.) (2000). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: The MIT Press.

Woo, C. W. (1991). Instructional Planning in an Intelligent Tutoring System: Combining Global Lesson Plans with Local Discourse Control. Ph.D. Dissertation, Computer Science Department, Illinois Institute of Technology.

Woo, C. W., Evens, M. W., Michael, J. A., and Rovick, A. A. (1991b). Dynamic Planning in an Intelligent Cardiovascular Tutoring System. *Proceedings of the Fourth Annual IEEE Symposium on Computer Based Medical Systems,* Baltimore, May. pp. 226-233.

Wooldridge, M. (2002). *An Introduction to Multiagent Systems.* Chichester, West Sussex, England: John Wiley & Sons, LTD.

Yusko, J. A. (1984). FBL: Frame Building Language. Final Project CSC580, Dept. of Computer Science. Chicago, IL: DePaul University.

Yusko, J. A. (1985). An Automatic Indexing and Generalization Module for Knowledge Acquisition in a MOP-Based Medical Expert System. M.S. Thesis, Dept. of Computer Science, Chicago, IL: DePaul University.

Yusko, J. A. (1994). The Reality of Change. Internal white paper. Unlimited Solutions, Inc. Lombard, IL.

Yusko, J. A., and Evens, M. W. (2002). The Knowledge Collective: Using MicroDroids to Turn Meta Data into Meta Knowledge. In: *Proceedings of the Thirteenth Midwest Artificial Intelligence and Cognitive Science Conference.* Chicago, IL. pp. 56-60.

Yusko, J. A., and Evens, M. W. (2004). Dynamic Ontological Support for Qualitative Reasoning in The Knowledge Collective (TKC). In *Workshop on Qualitative Reasoning,* Evanston, IL: Northwestern University, pp. 187-193.

Yusko, J. A., and Evens, M. W. (in press). The Knowledge Collective Framework Makes Ontology Based Information Accessible, Maintainable, and Reusable. In: Kishore, R., Ramesh, R. and Sharman, R. (Ed.). *Ontologies in the Context of Information Systems.* Berlin: Springer-Verlag.

Zhang, Y. (1991). Knowledge-Based Discourse Generation for an Intelligent Tutoring System. Ph.D. Dissertation, Computer Science Department. Chicago, IL: Illinois Institute of Technology.

Zhang, Y., Evens, M. W., Michael, J. A., and Rovick, A. A. (1987). Knowledge Compiler for an Expert Physiology Tutor, In: *Proceedings ESD/SMI Conference on Expert Systems,* Dearborn, June, 1987, pp. 153-169.

Zhang, Y., Evens, M. W., Michael, J. A. & Rovick, A. A. (1990). Extending a Knowledge Base to Support Explanations." In: *Proceedings of the Third IEEE Conference on Computer-Based Medical Systems,* Chapel Hill, NC, June 4-6. pp. 259-266.