# Agent-Based Mixed-Initiative Collaboration: The ABMIC project final report

**Michael T. Cox, Thomas Hartrum, Scott DeLoach,[1] and S. Narayanan**

**WSU-CS-02-01**

[1] Kansas State University, Department of Computing and Information Sciences, Manhattan, KS

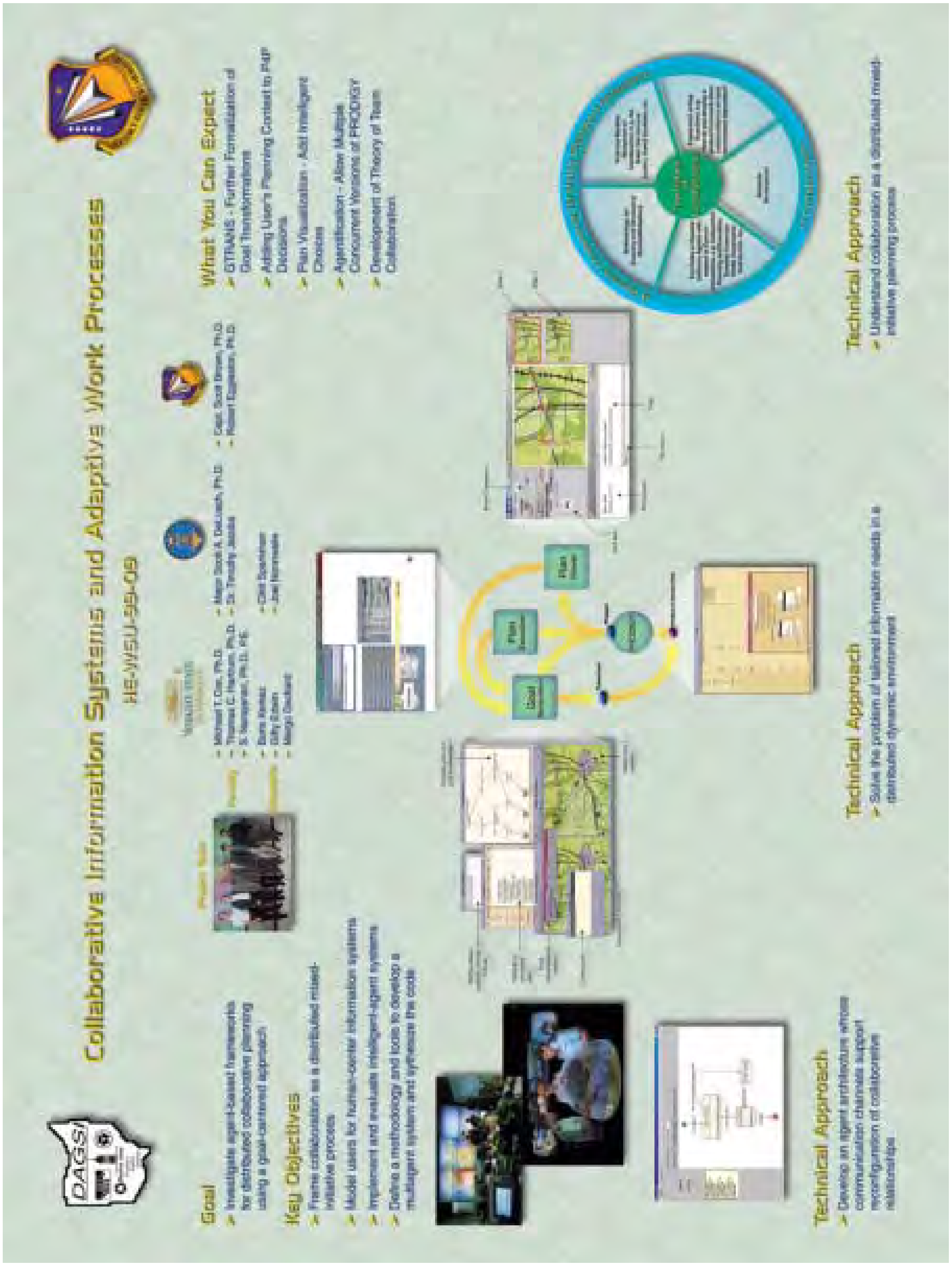
Collaboration & Cognition Lab

Artificial Intelligence with humans in mind

# Agent-Based Mixed-Initiative Collaboration: *The ABMIC Project*

Final Report

Computer Science Dept
College of Engineering
Wright State University
Dayton, OH 45435-0001



# Final Report
# Collaborative Information Systems and Adaptive Workflow

Project No.: HE-WSU-99-09

Project Title: Collaborative Information Systems and Adaptive Workflow

Internal Title: Agent-Based Mixed-Initiative Collaboration (ABMIC)

Team Leader: Dr. Michael T. Cox

Lead Institution: WSU

Partner: AFIT

## Executive Summary

This report summarizes the research and progress of the DAGSI/AFRL Research Program under grant #HE-WSU-99-09 entitled Collaborative Information Systems and Adaptive Work Processes. We overview the project, the results, and the points to where the remaining research is directed. The research looks at collaboration in general and planning in particular. Our project goal was to investigate agent-based frameworks for distributed collaborative planning using a goal-centered approach. One major thrust of the project was to examine intelligent agent and multiagent system architectures to model distributed planning (i.e., teamwork). In a second thrust we looked at mixed-initiative systems in which a symbiosis between human and machine planners is the main objective. To demonstrate the problems and solutions we have tackled, we have built a number of prototypes that solve basic collaborative problems in both a small air campaign scenario and in a small business logistics scenario. Many of these programs are available on the Web.

# TABLE OF CONTENTS

# LIST OF FIGURES

# I The Promise of Research on Agent-Based Mixed-Initiative Collaboration

The following statements defining project goals and deliverables are cited from the original proposal for grant #HE-WSU-99-09. In this document, we examine the results of the project and the extent to which we have achieved these goals and produced these deliverables.

**Goals and Objectives**: The overall project goal is to investigate agent-based frameworks for distributed collaborative planning using a goal-centered approach. The focus of this proposal is on three main research objectives associated with the main project goal:

1. Framing collaboration as a distributed mixed-initiative planning process;
2. Modeling users for human-centered information systems development;
3. Designing, implementing, and evaluating intelligent-agent systems that support collaborative planning in typical problems of interest to the Air Force and the commercial sector.

**Deliverable Objectives.** The proposed project is a basic research thrust that will deliver the following tangible results:

1. Progress toward and completed graduate degrees;
2. Scientific publications;
3. A documented prototype planning-system and associated interface that implements a novel solution to the problems of distributed mixed-initiative collaboration;
4. An internet home established to organize the dissemination of research results;
5. A graduate research course on mixed-initiative planning.

## II Introduction

Our research in this project involves three main areas of investigation. First, we seek to understand collaboration as a *distributed mixed-initiative planning* process. That is, we see collaborative planning as a process involving humans and computers in which results are potentially better than either the human or the machine can generate alone. Secondly with respect to user-modelling, we are interested in the problem of *tailored information needs* in a distributed dynamic environment. In complex planning problems, it is crucial that human (or artificial) agents receive relevant information in a way that facilitates the collaboration. And because a dynamic situation will change the planning initiative, we finally must develop an *agent architecture* whose communication channels support reconfiguration of collaborative relationships. As a unifying concept, we hypothesize that the data presentation and visualization problem itself can be viewed as a distributed planning problem from the perspective of artificial agents in an information space.

These three areas of interest have driven our research over the duration of the project. To focus the research, we chose planning as the collaborative task in which to study collaboration. Many problems associated with the interactions and coordinations of actions, decisions, informations, and goals in collaborative environments are readily apparent in distributed planning. Moreover, we are interested in both human, artificial, and mixed-teams of planning agents. Although we are not interested in competitive or adversarial interactions *per se*, our research does not assume that these characteristics cannot exist. Because we include the case where both human and artificial agents must collaborate, we also have a mixed-initiative perspective. Indeed, much of our expertise is in this area.

We have also chosen to examine the problem of tailored information presentation as part of our activity, because addressing it is so important in addressing other central issues associated with planning in large information-intensive collaborative tasks with many agents. To make progress toward the goal of modeling users within human-centered information systems, acknowledgement of the information presentation problem is crucial. We have only begun to examine fully the models of human users, however, so our results are principally relevant to single-user models.

Many real-world planning problems of interest both to the Air Force and to industry involve teams of humans and computational systems that together must solve complex planning problems. This research investigates agent-based architectures for distributed collaborative planning and the use of formal methods to automate the construction of well-formed agent organizations. We have shown this to be important by implementing and evaluating a number of multiagent systems that perform collaborative planning between humans, groups of artificial agents, and combinations of both.

While we review below the research progress the project has made in the last two years, we will also illustrate to what extent our research goals and objectives have been achieved. Furthermore, our deliverables will be described and enumerated along with the discussion.

## III Framing Collaboration

Collaboration by its very definition is a distributed problem-solving process that brings together more than one participant. In general a problem is decomposed into parts that each participant can solve. The individual solutions are then fused into a larger product that acts as an overall solution to the original problem. However given that many types of problems and various configurations of participants exist, it is not clear how best to organize and frame the collaborative process, especially in highly complex and technical problems involving large amount of information and differing expertise.

Mixed-initiative settings are the environment where humans and computers are tasked together to accomplish a set of goals. For these goals to be successfully executed, humans and software agents must work together. We have examined a number of natural systems such as the brain, the immune system, population ecology, and economics for potential metaphors of facilitating collaboration (Deckard, Narayanan, & Cox, 2000). Each of these natural systems has salient common characteristics including the ability to represent collaboration as well as competition in distributed networks. However, our specific research has been framed as *agent-based mixed-initiative collaboration*. Hence the view that we assume is one in which a system of agents (both human and artificial) must cooperate and in which significant interaction is mixed between both humans and machines. This section first takes a look at the metaphor of planning a mixed-initiative system should encompass. Subsequently the section will briefly look at some of the issues in multiagent collaboration.

## Planning Metaphors in a Mixed-Initiative System

Many mixed-initiative planning systems attempt to integrate humans and (semi)autonomous planning systems so that a synthesis results in high quality plans. In the artificial intelligence community, however, the dominant model of planning (and virtually all problem-solving) treats cognition as a *search* processes. Given an initial state, I, and a goal state, G, the task is to search for a sequence of steps that transform I into state S such that S contains G. State-space planners, such as the PRODIGY planning and learning architecture (Carbonell *et al.* 1992; Veloso *et al*. 1995), perform linear or nonlinear search through a space of action sequences where actions are represented as operators. An operator can be applied when its preconditions are met and its variables bound thereby changing the state through its effects. However, humans (especially those that are unfamiliar with AI planning theory) often have difficulty understanding planning in these terms. Pre- and post-conditions and variable bindings are not the types of choices natural to all users. We offer an alternative that represents a mixed-initiative approach to planning. Instead of search, the metaphor our systems support is that of *planning as a goal manipulation process* (Cox, 2000). The user's task is to define the goals or objectives of a system, to assign resources to the goals, and to assign relative importance between goals over time. That is, goals are not some static state provided by the user to the system as input. Instead goals are dynamically changing states that the user steers through the planning process. Classes of transformations exist that the user applies to change the goals from one state to another.

### *The Problems of Search as a Metaphor*

Search has been a dominant model for planning for a number of reasons. Importantly the model is formal enough to construct proofs and informal enough to implement and obtain empirical results. Computationally search captures important aspects of planning with a simple uniform mechanism and with a domain-independent representation. From any state of the world, new states can be generated by considering what happens if an action occurs. An action makes changes that are reflected in new states becoming true that previously were not and in other states becoming false that previously were true. For example if one block (e.g., BlockA) is placed on another (e.g., BlockB) by the action stack, then the world is different. BlockA is now on BlockB, and BlockB, which was formerly "clear" of other blocks, is no longer clear.

More formally the stack operator is shown in Figure 1. The operator has two variables, <ob> and <underob>, representing the top and bottom objects respectively. These variables are bound to particular tokens in the blocksworld such as BlockA and BlockB. The preconditions for the operator to be applied are that the bottom object be clear and the top be held (by a virtual robot arm). The effects of performing this action is that the top object is no longer being held and the bottom object is no longer clear (the delete-list) and that the top object is now clear, the arm is empty, and the top object is on the lower one (the add-list).

**stack**
<ob>: type OBJECT
<underob>: type OBJECT
*Pre*: (clear <underob>)
(holding <ob>)
*Del*: (holding <ob>)
(clear <underob>)
*Add*: (clear <ob>)
 (arm-empty)
 (on <ob> <underob>)

Figure 1. Stack Operator (Notation adapted from Veloso *et al*., 1995)

A set of such operators along with an object type hierarchy define a domain suitable for search. A set of goals along with a collection of objects and their relationships define a search problem. Naive users (i.e., ones not familiar with the technology, rather than the domain) often have problems with this arrangement. For example users understand forward search from the initial state better than backchaining from the goals. This backwards search finds an operator that can achieve a goal, and then searches for operators that can achieve unmet preconditions.

Furthermore, associated planning choices such as variable binding assignments are also user-unfriendly. For example if the stack operator from Figure 1 is chosen for some reason to achieve the condition that BlockA is clear given the state of the block being held by the robot, then the variable <ob> is determined (i.e., it is bound to the object BlockA). However, the variable <underob> is not determined and must be bound to some other object for the operator to be instantiated and applied. Random choices could be any object of type OBJECT in the current problem. A user would have to realize that a relevant candidate object to use in the binding should be an object different from BlockA, that already is on the table, and is clear. Many objects may exists that suit these constraints, but the constraints may not be obvious from the operator definition. Given N objects in the problem, the user selects candidates from the listing. However, if N objects of type OBJECT exist and the goal state has two arguments of this type, then the user must select from $N^2$ choices (each object could be identical). In general, for a predicate of with Q arguments, each type of which has amount(arg-i) instances, the number of choices is

$$\prod \text{amount(arg-i)}$$

The PRODIGY automated planning and learning system was modified to allow the user to make any decision that the system itself could (Cox & Veloso, 1997b), but in terms of variable binding choices, the user is often presented with such a large number of choices that they will not fit on a single screen and must be scrolled through. Our current research has departed from the classical planning metaphor and substitutes an alternative for the user. In the next section, we examine the modeling of planning as a goal manipulation task and show how this metaphor is realized in a mixed-initiative system called GTrans.

## Mixed-Initiative Planning as Goal Manipulation

Our research has mainly concentrated on two avenues with which to include the user as an active participant in mixed-initiative systems. First one can change the algorithms, representations, and parameters of the planner with which the user interacts. Second one can change the interface through which the user interacts with the planner. We briefly examine both in turn.

### *Changing the Planner*

In earlier research, we modified a standard planning system to enable more reasonable goals. As a result this effort has allowed us to broadly address choices of goal representation within a mixed-initiative planning system. A state-space planner takes as input a set of ground literals states that compose the top-level goals of the user. A ground literal is a predicate whose arguments are instances in the problem. However, when specifying a goal, users tend to make three departures from this framework (Cox and Veloso 1997a).

- Users specify goals as actions rather than states

- Users mix abstract and grounded goals
- Users include subgoals along with top-level goals

Using an expanded goal vocabulary, we modified the PRODIGY state-space planner to handle these three cases. We then experimented with various ways the user and planning system can interact. One way was embodied by the JADE system (Veloso, Mulvehill, & Cox, 1997; Mulvehill & Cox, 1999). It integrates a version of PRODIGY called Prodigy/CBMIP with a plan construction tool called ForMAT2. Prodigy/CBMIP sends the ForMAT2 user suggested actions rather than mandatory actions. The user can reject, accept, or ignore each suggestion. Conversely the user provides to the planner constraint information that makes planning more efficient. The most relevant details are available in Cox and Veloso (1997a).

*Changing the Interface*

To directly support the goal manipulation metaphor, this project implemented an interface through which the user manipulates both the goals and the problem. In such an approach, the interface hides many of the planning algorithms and representations from the user and instead emphasizes the goal manipulation process. The interface, called GTrans, presents a direct manipulation interface to the user that consists of a graphical map with drag and drop capability for objects superimposed upon the map surface. GTrans helps the user create a problem file that is internally represented as described in the previous section. That is, a planning problem consists of a set of objects, a set of relations between these objects, and a set of goals to achieve.

At initialization time an exchange of information occurs between GTrans and PRODIGY. An extension to PRODIGY called Prodigy/Agent (see subsection on distributed planning starting on page 15 for further details) allows the system to read and analyze the current domain file. The systems then sends to GTrans information about object types and about goal choices. To assemble a list of object types from which the user may choose to include in a given problem, Prodigy/Agent outputs a set of lowest-level object types from the semantic hierarchy.

Figure 2 shows a user selecting another F15 object to add to the current planning problem. A small icon is placed on the map with the pointing device to represent an object of the selected type. Looking ahead and closely at Figure 3, the reader can detect two infantry and three F15 icons near the lower left part of the map.

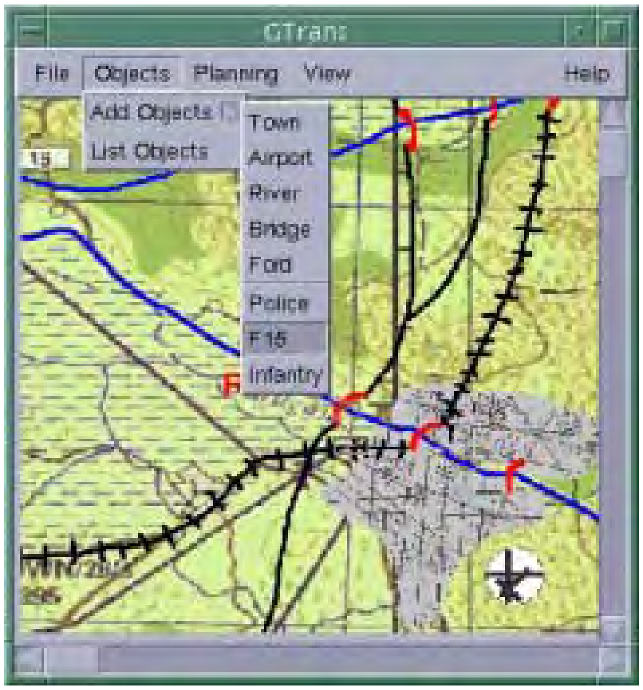


Figure 2. Adding objects to the problem with GTrans

To support goal classification information, Prodigy/Agent reads each operator from its domain file and sends to GTrans a list of all primary effects from each operator. The effects are represented as (predicate arg1-type arg2-type ... argN-type). GTrans then can associate objects defined on the map with goals relevant to each**.**

For example the **damage** operator has the following state specification within its add-list as its sole effect.

```
(is-damaged  <air-force>  <crossing>)
```

In the current domain file, two types of crossings exist (fords and bridges) and one type of air-force unit exists (F15). Therefore the potential goals associated with this effect is as follows.

```
(is-damaged  <F15>  <ford>)
(is-damaged  <F15>  <bridge>)
```

Goals can be created in two ways. The user can drag a mobile object from its location on the map to another object. If a goal relationship exists between these two types of objects, the system will present a pop-up menu with any relevant goal states that can be created. Depending the objects of each type, GTrans will create cascaded pop-up menus appropriately. For example if a particular F15 unit is moved over a ford the former goal choice is appropriate, whereas the latter is true when moving it over a bridge.

Alternatively a user can press Control-LeftMouse upon an object on the map. If goals are associated with the object's type, the system will provide a cascaded choice of goals with possible resources to be assigned to the goals. Figure 3 illustrates this method.

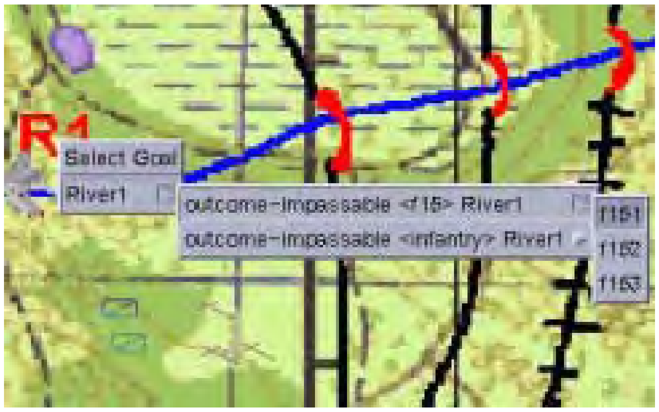

R1
Select Goal
River1
outcome-impassable <f15> River1
outcome-impassable <infantry> River1
f151
f152
f153

Figure 3. Creating a goal in GTrans

The user has selected the river designated at the symbol R1 with the left mouse button while holding down the control key. This pops up a menu displaying the types of goals relevant to a river. In this case the goal of achieving the state of the river being impassable is relevant. The user can choose either air units or infantry resources to associate with the goal. Here the user is selecting from among three available F15 air units.

GTrans is not simply a problem creation tool, however. With respect to goal manipulation, GTrans provides a unique capability to transform or *steer* the goals in response to Prodigy/Agent responses. When the user receives feedback from the underlying planner (either a successful plan that may not be acceptable by the user, a failure to generate any plan, or an apparent problem due to Prodigy/Agent taking too long), the user can asynchronously modify the goals and send them back to the planner for another round of planing. The user does this by selecting a *goal transformation* that moves the goal within a space of possible goals (Cox & Veloso, 1998).

Figure 4 shows a dialogue box that allows the user to change the goal of having the river be impassable (perhaps not enough resources exists to carry this out). By clicking on goal arguments or the goal type itself the user can modify parts of the goal. The figure illustrates a user's decision to "erode" the goal by changing it from making a river impassable to reducing transportation across it. Cox, Kerkez, Srinivas, Edwin, and Archer (2000) and Cox, Edwin, Balasubramanian, & Elahi (2001) provide further details concerning this feature.

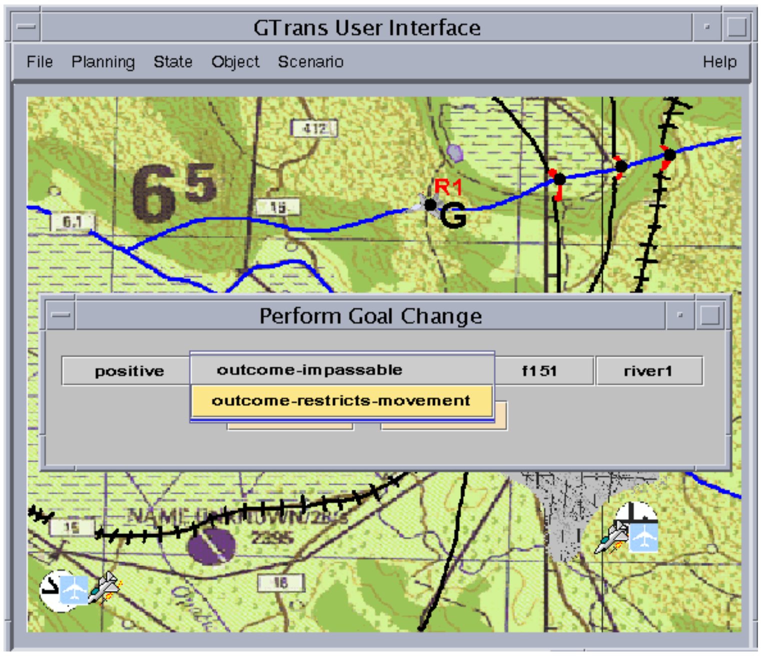


Figure 4. Performing an *erosion* goal transformation in GTrans

Although the arguments presented here may appear reasonable, they do not sufficiently support the hypothesis that users perform better when provided an appropriate planning metaphor at the interface of interaction. This section has contrasted two models of planning as a conflict of metaphors, but, the models are not mutually exclusive. Search is still an important AI technique that can be used in mixed-initiative systems as illustrated in this section. However, we argue that search is not an appropriate metaphor to present to the user. Although this is still a hypothesis, preliminary evaluation of this claim suggests that the difference is a real one.

We have started a pilot study in which we present to human subjects a series of randomized problem in the air campaign domain. The set includes 18 easy, medium, and hard problems. One group of subjects performs problem-solving with the Prodigy 4.0 Graphical User-Interface 2.0 in which planning is presented as a search problem. Another group uses GTrans. As shown in Figure 5, subjects perform better using an interface that presents planning as a goal manipulation problem than in one that presents it as search. Moreover the improvement in performance increases with the difficulty of the problems. Figure 5 shows the benefit of GTrans over PRODIGY's public graphical user-interface. The latter interface presents planning as a search task.

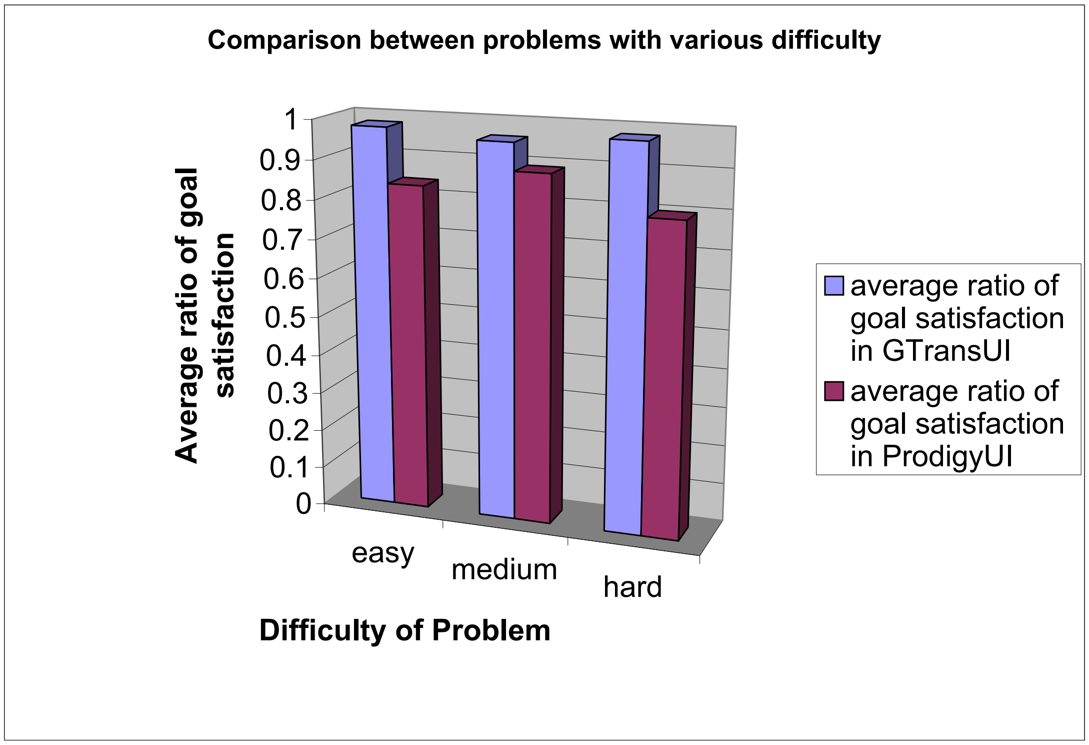


Figure 5. Pilot human subjects study with planning metaphors

In addition to a mixed-initiative approach, we also frame collaboration as a distributed agent-based phenomenon. The next section examines this approach taken by our project in this research.

## Distributed Planning

In the artificial intelligence (AI) community, mixed-initiative planning often involves collaboration only between a single human user and a single machine agent where the division of planning responsibility and initiative is divided between each, and such characteristics may shift over time. By the term *agent,* we simply mean some process that can perceive the environmental state within which it exists, can act upon its environment to produce a change in state, can posses goals to achieve particular states, and can adapt to its environment to improve its ability to achieve goals. To project results in the AI literature to problems of collaboration, we must generalize to a distributed (i.e., multi-user/multiagent) planning framework. The task is a complex one, however, because in realistic collaborative situations, the assumptions of classical planning (e.g., knowledge is complete and certain, the world is static and does not change unless acted upon, and plans can be generated fully and then executed) do not hold. Instead, realistic collaboration involves much uncertainty, as well as collaborators and exogenous events that dynamically change the state of the world independently of the decision maker.

To understand collaboration we have experimented with artificial multiagent systems in which several autonomous agents interact to solve complex tasks. Each agent has its own goal. These goals must be coordinated to achieve better and more efficient overall plans. Another important characteristic of multiagent systems is that the environment is no longer static. As opposed to single agent systems, the dynamics of the environment are affected by the actions of other agents in that environment. An example domain where multiagent systems are used is the simple air campaign planning domain for military scenarios shown in the previous subsection. Various agents plan for different goals such as deploying units, destroying bridges, and securing rivers. A number of reasons exist for using multiagent systems. The first and foremost reason for using multiagent systems is that it helps us understand teamwork and social behavior in a human organization. Secondly, a planner with multiple goals needs to know the resources available to it and the states associated with each of its goals. Because there is a deluge of information, collaboration becomes difficult. The third reason is that the time required for a multiagent system to generate a solution is often less than the time required by a single planner with the same goal. This is because of the inherent parallelism in multiagent systems. Scalability, robustness and nature of the domains also factor into reasons for using multiagent systems.

In the most general of terms however, we have examined how agents can be *flexible* in the decisions, both within a single agent system and also within a multiagent environment. Given that we view planning, and indeed any problem-solving process, as a goal and resource manipulation task in a dynamic (i.e., changing) environment, flexibility demands a number of capabilities. Three capabilities for both types of arrangements are enumerated below.

Single Agent and Multiagent Systems

- Agent can change a goal
- Agent can reassign a resource
- Agent can react to environment change

When the world changes a planner must replan if the change results in a situation that violates a planning assumption. For example, if an air unit becomes grounded for repairs, an agent must be able to modify the existing plan in order to substitute an alternative air unit that is flight worthy. However if the same unit becomes unavailable and no substitute exists, or if the agent is given a goal to achieve and insufficient resources exist with which to achieve it, the agent should not simply fail to generate a plan. Instead, the agent should be able to modify its own goals in order to achieve something close to the original goal it was given. We have refined Cox and Veloso's (1998) theory of goal transformation in order to allow for such contingencies.

A goal transformation is defined as a minimal movement of goals in a goal hyper-space and implements the concept of goal movement. Cox and Veloso (1997a) show that goals can exist in an abstraction hierarchy so that some goals specify desired states that are more general than others. The goal arguments and predicates may be moved along an abstraction hierarchy, an enumerated set, a number line, or a component partonomy. Furthermore goals have a measure associated with it that represents the amount of goal satisfaction achieved. For example the relationship between a jar and its lid is an enumerated set. The jar lid may be removed, loosely-on, on, and tightly-on. Although an agent may desire a lid to a container to be tightly-on for secure storage, if the lid is on (or even loosely-on) the condition may still be acceptable given time limitations or other constraints. Given more important goals and a lack of resources (e.g., in this case time), an agent may satisfy it goal concerning a container only partially.

Even if sufficient resources do exist, the reassignment of resources may be too costly to justify a solution. In such instances, an agent may wish to evaluate the benefit of resource change with respect to the cost of goal change. Edwin and Cox (2001a; 2001b, Edwin, 2001) report a cost-benefit analysis that is used to determine at what point an agent should perform a resource re-assignment (i.e., reallocate a resource used in the service of one goal so that it can be used to achieve an alternative goal) and at what point it should perform a goal change instead.

Another set of flexibility capabilities arise strictly in multiagent environments. We list these below.

*Multiagent*

- Agent can subcontract with other agents
- Agent can request a resource
- Agent can react to other agent actions

Unlike single agent systems, an agent in a multiagent environment may request another agent to cooperate with it when resources are limited or when some tasks cannot be completed alone. When an agent is unable to perform some task that achieves a goal state or some precondition of another action that achieves a goal state, the agent may subcontract with another agent to perform the service. Unlike other AI formalisms, we have framed this situation more generally as an example of a *split-agent goal transformation* (Cox, Edwin, Balasubramanian, & Elahi, 2001). That is, an agent can change a goal to achieve some state G into two (perhaps empty) sets of goals, one of which it can achieve and the other it cannot. The solution is then to achieve the first set and also find collaborators who are able to assume the responsibility of achieving the latter subset. By doing both it can consider the goal G achieved with respect to its own needs.

Just as an agent in a single agent environment can reallocate resources from one of its goals to another, so too can an agent in a multiagent scenario request additional resource be allocated to a goal that is currently suffering from a lack of resources. As with the single agent framework, however, the agent must evaluate the costs and benefits associated with obtaining a resource from other agents with respect to the costs and benefits of simply changing the failing goal (Edwin, 2001; Edwin & Cox, 2001b).

Finally, like an exogenous change in the environment for a single agent system, multiagent systems have positive and negative interactions that result from the actions of collaborators and adversaries. We have modified the mechanism of rationale-based planning monitors (Veloso, Pollack, & Cox, 1998) to signal to agents those changes in the environment that effect the planning decisions already made by the agent in its own task. We call the execution of this behavior goal-interaction coordination (Edwin & Cox, 2001a).

In a general cooperative distributed planning framework, some of the key questions one must address include:

- How are the overall planning problems decomposed and allocated to the agents
- How do agents communicate with one another during planning?
- How do agents coordinate their actions and decisions?
- How do the sub-plans of individual agents compose to produce the overall plan that can be executed coherently and effectively?

The research in this project has examined these questions in detail and many others. This report summarizes just some of the progress along these lines.

## IV User modelling and Tailored Information Presentation

We are keenly interested in the problem of tailored information presentation. That is, we wish to understand how a system can present the right information to the a user at the right time and in the right format and modality when the user is in a collaborative environment (Brown & Cox, 1999). Any complex system has an information presentation problem, because large amounts of information poses a high cognitive load upon the user. However in a complex collaborative task, the problem is not simply one of information filtering (i.e., remove unnecessary information to remove unnecessary cognitive load) and mode selection. Rather it also includes a decision of what information to provide to a user with respect to what information is presented to other users, given the trade-off to personalize and minimize the information while at the same time assuring that the user has the appropriate information to understand how individual decisions support shared goals (Cox, in press). In such a situation, user-modeling is crucial.

For the most part, we have examined how and in what detail to model the single user and the user's problem-solving context. However, the long term goal is to generalize the research to team modeling and group behavior. To begin to approach these problems we have examined a subset of the larger problem of tailored information presentation. This subproblem is to dynamically manage the user interface in support of user tasks. In particular we have built preliminary systems to solve simple screen clutter problems of multiple overlapping windows on a desktop environment. When the number of windows in the user-interface passes a given threshold, the task is simply to determine which window should be either moved, iconified, or killed. However, to perform such a task effectively, a system must know more than just screen dimensions and low-level window information. The system must be able to base a decision to change the user interface upon the user's problem-solving context. Especially in collaborative environments where multiple users interact, determining individual problem-solving contexts is difficult, even when individual user goals are known. Although many user goals may be explicit in a sub-context, many are implicit and must be inferred. Moreover for both implicit and explicit goals, actions to achieve such goals may interact negatively between users. This section first discusses two approaches to determining these goals and interactions. We look at a collaborative system that encourages user-specification of goals so that the system can track them, and then we look at a keyhole plan recognition approach that infers user intent from individual user action. Secondly we examine two other approaches to user-modeling in our project.

**Tracking User Intent**

As described in Section III, GTrans is a mixed-initiative planning system that presents planning tasks to the user as goal manipulation problems rather than a problems of search. In the system a user performs direct drag and drop manipulation of objects on a graphical map. The user can set domain states and goals that are then sent to an automatic planner for results. When the planner produces poor plans (from the user's perspective) or when it cannot produce a plan at all, the user can manipulate goals in the system by performing various goal transformations (Cox & Veloso, 1998) upon them. For example in the logistics domain, the goal of having all packages in a region delivered may not be possible due to a delivery truck being repaired. The goal may be "reduced" to delivering packages in cities located in a subset of the region instead.

A supporting system called P4P (Kerkez, Cox, & Srinivas, 2000) is a planner that plans for its own user-interface. As a human uses a planner such as GTrans, P4P "watches" the user interactions with the planning interface. After each user action, the current state of the interface represents a new initial state from which P4P will plan to reduce screen clutter. If no screen clutter exists, then no plan is needed. Otherwise P4P creates a sequence of interface actions such as (restore window1) (iconify window3) that can achieve the state of screen clarity. A simple planning domain called the micro-window domain has been created to model these behaviors. We have identified three types of knowledge that can help a system reason about user interactions with user interfaces. First and most obvious is knowledge about window characteristics such as screen geometry and recency of window use (Kerkez & Cox, 2000). This knowledge represents a kind of window syntax. Second is knowledge about window content and function (Kerkez, Cox, & Srinivas, 2000). Such knowledge represents a kind of interface semantics. Third is knowledge about the user problem-solving context and represents conceptual knowledge about the reasons the user requires information transmitted through the interface. The context information is represented as the set of user planning goals. In the simplest form, the problem consists of matching the user goals to functions of objects in the user-interface.

## Multiuser Collaboration

The GTrans system also has a collaboration mode whereby multiple users can each plan on a common map representation. All users will view the same map and each may manipulate objects and goals independently. The results of individual interactions, however, are viewable to all users. For example a map may contain any number of regions in the logistics domain for which each user is responsible. As long as each user has independent goals that pertain to their region only, then few planning problem exist. However, interactions occur when the actions of one user interferes with the plans of another. This may happen in the logistics domain when packages must be routed between regions and limited resources exist. Aircraft are shared by both users, so planning must be coordinated. In particular conflicting goals must be negotiated so that they “move closer” to each other to avoid conflict in a team collaboration.

A system called COMAS (Edwin & Cox, 2001) allows multiple planning agents to interact and create plans to achieve shared goals. Each agent is represented by an automatic planning system called Prodigy/Agent (Cox, Edwin, Balasubramanian, & Elahi, 2001). COMAS uses planning monitors (Veloso, Pollack, & Cox, 1998) to identify negative and positive interactions between committed planning decisions as each pursues their respective planning goals. The goal of this effort is to provide an underlying goal tracking and management system that advises users on optional goal changes to avoid planning failure. As these goals change, however, the choice of user-interface (window) management decisions become more difficult.

**Inferring User Intent**

Forcing users to make all goals explicit and tracking goals with common software (e.g., GTrans) that users share are not always realistic strategies. In addition to planning with a collaborative system such as GTrans, users will also have supporting applications that comprise the larger planning context. For example external text editors and graphical displays may be involved to change plan sequences and visualize plan graphs. Geographically separated users may also use heterogeneous email applications to communicate. In such cases, the overall problem solving context may be arbitrarily complex.

Plan recognition is an alternative approach to inferring user intent when a system has only observations of user actions. The concept is to have a library of common actions in some domain and to match the current sequence of observation to the histories of known plans. Each plan then has associated goals which represent the intent. We have developed a novel method of plan recognition that combine intermediate states with action sequences to produce annotated plans. The plan recognition system then uses the current state as well as the latest set of actions to find previous plans matching isomorphic states. The problem is that such states may be quite large so the combinatorics are prohibitive. The solution we use is to represent states in an abstract representation that generalizes token to types, leaving a much smaller search space (Kerkez, 2001; Kerkez & Cox, 2001).

The plan library contains instances of plans an observed agent may pursue. Planning episodes in the plan library can be viewed as cases, and the recognition process can utilize these past cases to generate predictions of planner's intentions. The case-based nature of the recognizer enables it to incrementally acquire planning instances, eliminating the need for hand-coded or automatic library construction methods. A pure case-based approach to plan recognition allows the recognizer to partially match cases (plans) from the library with its observations. The case-based plan recognizer can therefore recognize plans that are not exact matches to the plans from the library, but instead are similar to the description of a current situation, based on the observed planning steps.

At the current time, our plan recognition system does not observe user actions at the interface. Instead it operates in the blocksworld and logistics domains using a case library of plans generated by a state-space planner. Given a new set of observations (another instance of the planner's behavior), the system retrieves old plans to make predictions of current behavior. We are currently building another system to trap user-interface events in arbitrary Unix desktop environments. The system will possess a “conduit” that monitors these events and passes it on to a sentinel. In turn the sentinel will be able to interact with and control the interface itself.

The support software to this system uses an extensible window manager called Sawfish (`http://sawmill.sourceforge.net/`) that runs on X11 platforms. It resides on top of a GNOME (GNU Network Object Model Environment) integrated desktop environment and application suite to allow multi-machine interactions. The goal of the effort is to be able to extract interface events (e.g., window movements, button clicks, and scroll bar manipulations) and to coordinate them across multiple machines in a multi-user environment.

### Other User-Modelling Approaches

Work in Dr. Narayanan’s laboratory has also examined user-models in information-intensive activities. Information analysis involves discovering information, determining contextual relevance, and synthesizing useful knowledge from distributed, heterogeneous information sources. For example, a logistician may potentially have to review a large quantity of data from many relevant sources in order to make a resource decision. During the review process, logisticians may be able to provide a list of keywords related to the specific information that they need. The search process then begins using this list of keywords. The logistician must then manually filter the results of the search in order to synthesize useful information. A search process based solely on keywords, may result in missing important sources of information, finding too many unrelated sources of data, or a combination of both situations due to the problems of synonymy or polysemy. Synonymy is when several different terms have the same meaning and polysemy is when a single word has more than one meaning. Information about the content of the information sources is needed for integrating sources and for overcoming the synonymy and polysemy problems associated with key word search techniques. Higher-level understanding of terms in a domain, i.e., semantics, can help in overcoming these problems. This higher-level information can then be used for supporting information search and retrieval techniques. Developing an understanding of information sources in the form of semantic representation is fundamentally a complex reasoning process. Manual generation of semantic concepts in a domain is inefficient. Complete automation of the process could lead to inaccuracies and therefore be ineffective.

In this research, we have developed an approach that integrates intelligent agent technology with human reasoning for generating semantic concept spaces to support information search and retrieval from distributed information sources (Prasad, 2001). Our approach is based on a frame-based metadata representation, which contains knowledge about interesting concepts in a domain and their location in the information sources. We generate metadata associated with a collection of distributed information sources through a combination of computerized reasoning processes and a human reasoner. The metadata is used by a novel information search and retrieval system, which supports keyword and semantic search methods and filters results, based on user profiles. The user profiles are adapted through unsupervised and supervised learning techniques. This research included the design, implementation, and evaluation of a computational system called NARAD that embodies our approach (Prasad. 2001).

Further work in Dr. Narayanan's laboratory has examined a machine learning approach to modelling and predicting user actions (Moss, 2001). Among the many applications of machine learning are information search and retrieval, user modeling, and failure prediction. The present study examines the application of machine learning using the k-nearest neighbor and naïve Bayesian algorithms in order to make real-time predictions of user behavior in a game of chance. This study looks at several research questions, including the use of group data for modeling individual users, the question of how well these algorithms perform with access to extensive system state parameters, and a direct comparison of the two machine learning algorithms under investigation. Results show that the k-nearest neighbor classifier outperformed the naïve Bayesian classifier, but learning for either algorithm did not improve over time. Additionally, it was found that group data did not have a significant impact by means of improving individual user models.

## Summary

The purpose of this section is not to present a finished research effort nor to report results that directly pertain to team intent inference. Rather it is an attempt to begin to associate a set of research tools that can examine the issues surrounding an incredibly complex problem. The problem is to support multiagent teams (whether human, artificial, or mixed-agent teams) that together must collaborate to achieve some set of goals by providing the team members with the proper kind of information to enable them to solve their part of an overall problem without interfering with each other. Although we believe that determining the goals and intent of the team is a crucial piece of knowledge that must be determined if this support is effective, at this point, we are simply beginning the process of forming some of the important questions that surround the issue.

By recognizing the plans and the goals the operators are trying to achieve, a mixed-initiative system can infer the problem-solving context of human operators in terms of tasks they are pursuing. Once the problem-solving context of the planner is identified, the automated component can better assist the planners by tailoring the information presented in the user interface to the specific task the planner is currently pursuing, while hiding the information in the interface that does not pertain to the current task.

# V Agent System Design, Implementation, and Evaluation

A major objective of our research is to develop formal methods for multiagent design and specification to allow more principled implementation and evaluation of such systems. In general we wish to develop a methodology using formal methods to allow the specification of a multi-agent system independent of any specific underlying protocol (e.g. agentMom (DeLoach, 1999), Carolina (Sabe & Santos, 2000)), and then to automatically synthesize the integration with one specific protocol. This allows the system analyst to concentrate on the application-specific aspects of the multi-agent system requirements without having to worry about low level protocol details.

The approach taken is to develop a formal specification of the multi-agent system as an object-oriented system using the AWL specification language (Hartrum & Graham, 2000). In the object-oriented model, the objects, agents in this case, communicate via events in the dynamic (finite state machine) model. This *abstract* system representation ignores the issue of how the event communications will be realized in a specific agent protocol, shielding the specifier from such details. There also exists a library of formal protocol models, also written in AWL. After the designer selects one of these protocols, formal methods are used to integrate the multi-agent specification with the protocol model following the approach of Nonnweiler (1999). Two synthesis systems, AWSOME (Hartrum & Graham, 2000) and agentTool (DeLoach, 1999), are then used to synthesize the multi-agent system over the desired protocol.

The approach was demonstrated with a multi-agent system for routing shipments via trucks and airplanes using the PRODIGY planner and synthesized in Java over the agentMom protocol.

## Formal Multi-Agent Specification

Basically agents are modeled as objects that communicate via events within their state models. Figure 6 shows a typical Event Flow Diagram for three communicating agents.

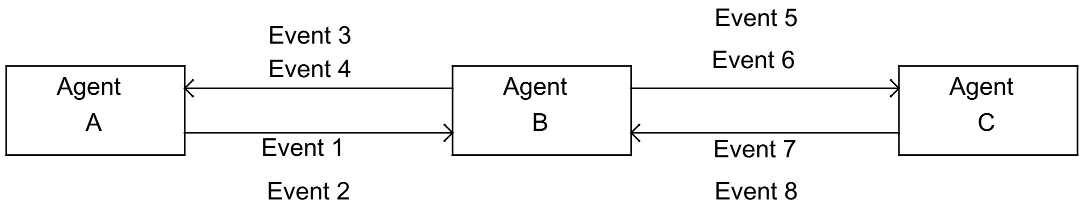


Figure 6. Agent Event Flow Diagram.

The approach follows the UML object model. Objects react to events (from other objects) by executing local actions, changing state, and sending events to other objects. Events can carry information as parameters, each defined over a specific type. This is formalized in the AWL language.

## Formal Agent Protocol Specification

The intent of the methodology is to have available a library of formal specifications for different multi-agent protocols. The designer could then select a specific protocol over which to implement the abstractly specified multi-agent application. In this section we examine the definition of a single such protocol, agentMom (DeLoach, 1999).

Actually agentMom is a framework for implementing multi-agent systems in Java. As shown in Figure 7, an agentMom system consists of a number of Application Agents, each a subclass of an existing Agent class. Communication between agents is done through *conversations*. Application Conversations are defined in pairs, and each is a subclass of an existing Conversation class. Each Application Conversation communicates only with its matching Application Conversation and with its associated Application Agent. Communication between conversations is done by passing instances of a predefined Message class, a variant of the KQML syntax (Finin, McKay, & Fritzson, 1992), using send and receive methods inherited from the existing Conversation class. Communication between a conversation and its agent is limited. The agent instantiates a conversation dynamically when it is needed, and passes all “input” parameters as constructor arguments, along with a pointer to itself. The conversation communicates with its agent by calling methods based on this pointer (callback). Each agent has a predefined Message Handler which facilitates listening on a port for incoming messages. The predefined Agent, Conversation, Message Handler, and Message classes are implemented in Java. A specific multi-agent application system must provide a set of Application Agents and Application Conversations in Java to complete the implementation.

Note that agentMom is a Java implementation. Our methodology is based on a formal object-oriented specification using communicating state models. Thus some work is needed to derive a necessary model from the existing Java classes. This not only involves an equivalent AWL specification of the Java code, but an architectural transformation to provide a state-based event-passing interface instead of an inheritance-based interface.

We start by abstracting the agentMom architecture shown in Figure 7 into the architecture shown in Figure 8. The system becomes a set of Application Agents that communicate via UML events. Which agents actually communicate with each other is captured by the event mapping association. The underlying antonym communication protocol is modeled by representing the underlying communication channel used by the conversation pairs to communicate as illustrated in Figure 9 and Figure 10. This is a simple representation wherein both the ClientChannel and the ServerChannel react to incoming events (carrying an agentMom Message) by relaying the Message in an

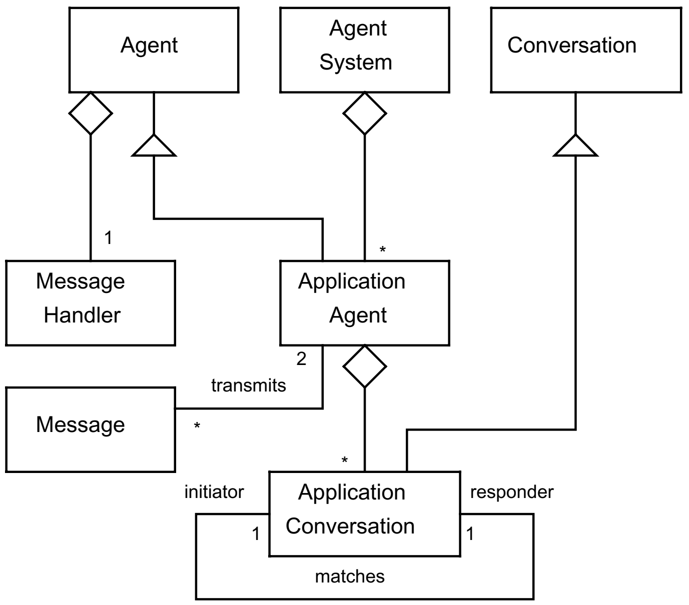


Figure 7. The agentMom Java framework architecture.

outgoing event. Thus the overall interface to the Channel System is to react to WriteObjectC events by generating output ObjectIsS events, and to react to WriteObjectS events by generating output ObjectIsC events. This interface easily supports our integration methodology described in the next section.

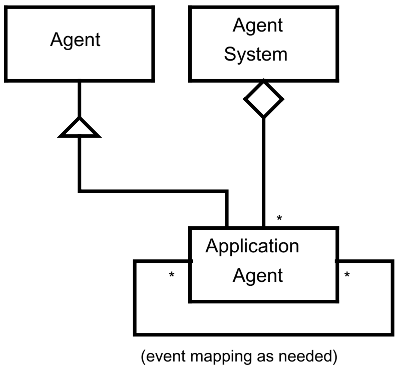
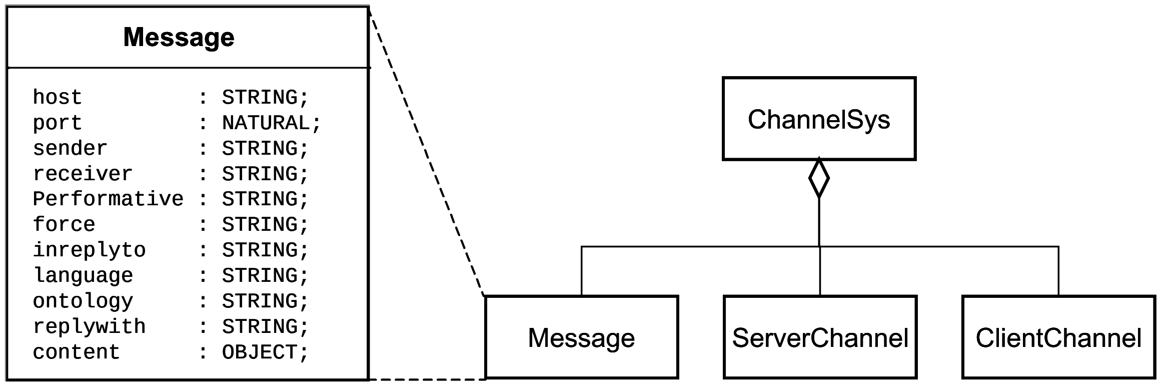


Figure 8. The equivalent application agent architecture.



```
host        : STRING;
port        : NATURAL;
sender      : STRING;
receiver    : STRING;
Performative : STRING;
force       : STRING;
inreplyto   : STRING;
language    : STRING;
ontology    : STRING;
replywith   : STRING;
content     : OBJECT;
```

Figure 9. The equivalent agentMom channel architecture.

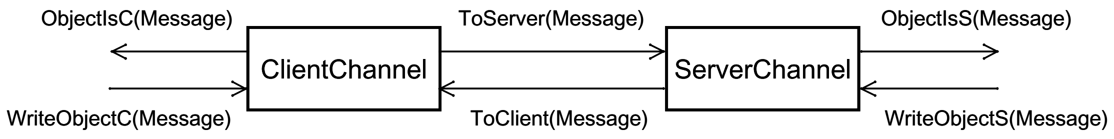


Figure 10. The channel architecture Event Flow Diagram.

*Integration of Specifications*

Nonnweiler (1999) developed several approaches to integrating formal specification models, each based on defining a formal interface contract for each model (Szyperski, 1999). This contract consists of the events to which the model reacts, and of events that the model generates. Two models are integrated by defining a set of Model Interface Conversions (MICs). Each MIC reacts to a subset of one model's generated events and generates a subset of the other model's input events and vice-versa. Each MIC, itself a formal object-oriented specification model with a state-based behavior, has several responsibilities. These include mapping event and parameter names, type conversions of event parameters, and control of the triggering of its outgoing events. The two models, along with the MICs, are then integrated into a new aggregate class.

Nonnweiler's methodology allows a variation on the interface contract that is useful in the agent situation. Consider the specification shown in Figure A, but assume Agent A and Agent B are part of Model 1, while Agent C is in another model. Events 1, 2, 3, and 4 are considered *intra-model* events, as they are generated and reacted to totally within Model 1. Events 5, 6, 7, and 8 are considered *inter-model* events, as they are generated or reacted to external to Model 1. However, it is possible to *sever* an intra-model connection and make it into an inter-model connection. This effectively interjects another model between two communicating objects in the first model. For example, the connections between Agent A and Agent B could be severed, with the result shown in Figure C.

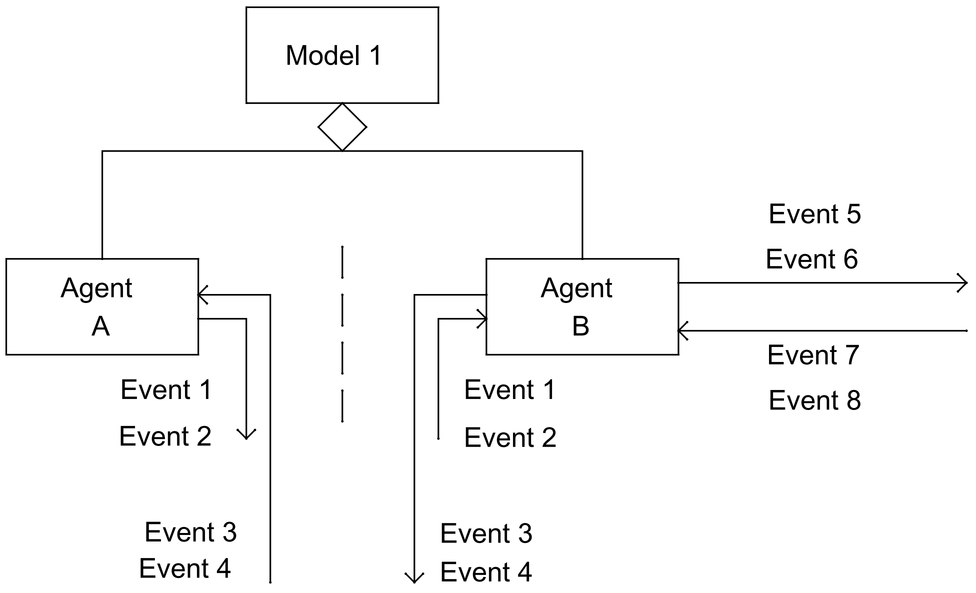


Figure 11. Agent Event Flow Diagram.

Nonnweiler's full methodology consists of the following steps.

*Pre-Integration Step p-1: Input Model Selection.* For our purpose the first model to be integrated is the application multi-agent system, and the second model is the agent protocol of choice.

*Pre-Integration Step p-2: Communication Pattern Identification.* Nonnweiler defines five communication patterns between the models: *transfer, split, merge, mixed*, and *combined*. The event flows between the two models are partitioned into a set of these patterns. For the multi-agent system, this is discussed in more detail in the next section (Evolving Conversations).

*Pre-Integration Step p-3: Communication Pattern Analysis.* Once the communication patterns are identified, the necessary type conversions and trigger strategies need to be defined. This is also discussed in more detail in the next section (Evolving Conversations).

*Step 1: Deconflict the Input Models.* Before integrating them, names of global scope that are the same in both models (conflict) must be changed in one of the models. For the AWL language, this includes types, constants, global functions, classes, aggregations, associations, and events.

*Step 2: Create the New Integrated Model.* This step consists of defining a new top-level class and aggregating the two input models as its components.

*Step 3: Create Each MAC's Structural Component.* For each MIC, based on the Communication Patterns identified, a new class with appropriate attributes is added as a component of the new top-level class.

*Step 4: Create Each MAC's Functional Component.* Here the methods for processing received events, doing type conversions, and generating outgoing events are added to each MIC.

*Step 5: Create Each MIC's Dynamic Component.* Here the state model for each MIC is defined. This must react to each incoming (to the MIC) event, invoke the appropriate conversion methods, and send the appropriate outgoing events.

*Evolving Conversations*

In Pre-Integration Step p-2, communication patterns are identified. One of the design challenges is to select the right combination of communication patterns that will lead to a "good" set of agentMom conversations.

UML Event Sequence Diagrams provide a graphical tool for examining scenarios that can be grouped as Conversations. For example, if the events in Figure C work together to form a single "transaction," one approach would be to combine the two events in each direction into a single Message, resulting in a single pair of MICs, as shown in Figure 12.

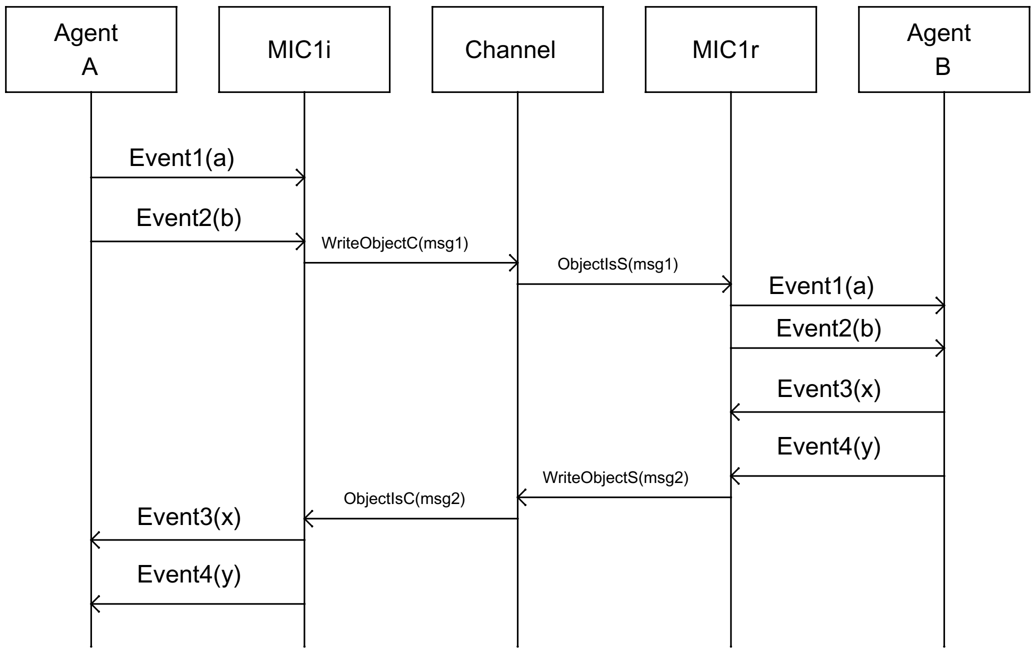


Figure 12. A pair of MICs as Conversations.

*Formal Transformations*

Applying the integration methodology defined by Nonnweiler leads to the general structure shown in Figure 13. Next, protocol-specific (for *agentMom*) formal transforms are applied, transforming the OO UML model into a OOP model in AWL.

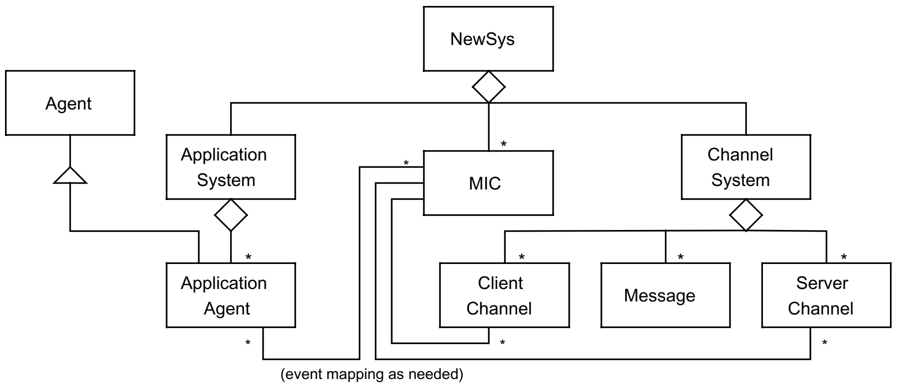


Figure 13. Result of Nonnweiler Integration.

First, the *Application System* aggregate is deleted, and the agents are made direct components of *NewSys*. Second, the *Conversation* is added as a superclass of the *MIC*s. Third, each *MIC* becomes an aggregate component of the agent with which it communicates. Finally, the *Channel System* and its components become implicit in the *send* and *receive* methods inherited from *Conversation*, and disappear from explicit representation, leaving the integrated system shown in Figure 14.

*Code Synthesis*

At this point, AWSOME and *agentTool* can be used to synthesize Java. Each MIC represents a conversation (half), and the description can be entered into *agentTool* to synthesize the corre-

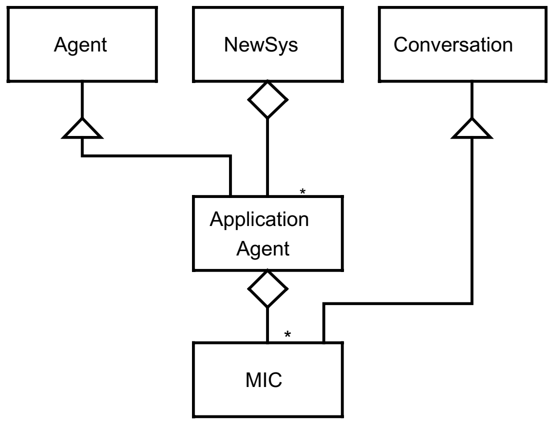


Figure 14. Result of Formal Transforms.

sponding Java conversation classes. The Java agents can be synthesized by the Java synthesizing transforms in AWSOME.

*Summary*

The object of this research was to develop a formal-based methodology to allow the specification of a multi-agent system independent of underlying agent protocol, and to automatically synthesize the integration with a specific protocol. The feasibility of that approach has been demonstrated. A methodology was defined for integrating two (or more) system models, represented in the formal language AWL, into a single AWL specification, and a tool was developed to assist in this integration (Nonnweiler, 1999). An AWL formal model of the channel communication system underlying *agentMom* was developed and its integration with a multi-agent application system demonstrated (Nonnweiler, 1999). Additional transforms to convert the result into an *agentMom* architecture are described in this paper. The resulting formal specification is then divided into two sections: *conversations* and *agents*. The conversations are then input manually into agentTool, which is capable of synthesizing the corresponding Java source code. The agents are parsed into the AWSOME tool to synthesize the Java agent code.

All of this system has been demonstrated in one form or another. However, the pieces still need to be fully integrated into a single tool to provide seamless flow from the specification of a multi-agent system, independent of agent protocol, into a working Java system over the *agentMom* protocol. Following that, extending the demonstration to other protocols, such as *Carolina*, needs to be done.

## Design with Multiagent Systems Engineering

In this research, we developed both a complete-lifecycle methodology and a complimentary environment for analyzing, designing, and developing heterogeneous multiagent systems. The methodology we are developing is called Multiagent Systems Engineering (MaSE) (DeLoach, Wood & Sparkman 2001) while the tool we are building to support that methodology is called agentTool. The general flow of MaSE follows the seven steps shown in Figure 1. The rounded rectangles on the left side denote the models used in each step. The goal of MaSE is to guide a system developer from an initial system specification to a multiagent system implementation. This is accomplished by directing the designer through this set of inter-related system models. Although the majority of the MaSE models are graphical, the underlying semantics clearly and unambiguously defines specific relationships between the various model components.

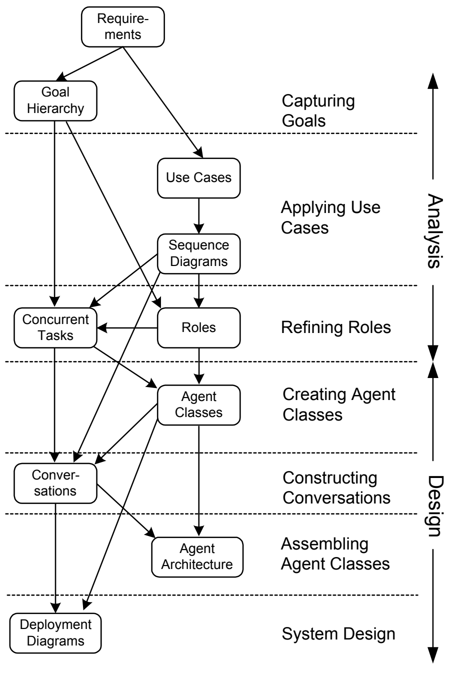


Figure 1. MaSE Methodology

MaSE is designed to be applied iteratively. Under normal circumstances, we would expect a designer to move through each step multiple times, moving back and forth between models to ensure each model is complete and consistent. While this is common practice using most design methodologies, MaSE was specifically designed to support this process by formally capturing the relationships between the models. By automating the MaSE models in our agentTool environment, these relationships are captured and enforced thus supporting the designer's ability to freely move between steps. The result is consistency between the various MaSE models and a system design that satisfies the original system goals.

MaSE, as well as agentTool, is independent of a particular multiagent system architecture, agent architecture, programming language, or communication framework. Systems designed using MaSE can be implemented in a variety ways. For example, a system could be designed and implemented that included a heterogeneous mix of agent architectures and used any one of a number of existing agent communication frameworks. The ultimate goal of MaSE and agentTool is the automatic generation of code that is correct with respect to the original system specification.

### *Capturing Goals*

The first step in the MaSE methodology is *Capturing Goals*, which takes the initial system specification and transforms it into a structured set of system goals, depicted in a *Goal Hierarchy Diagram*, as shown in Figure 2. In MaSE, a *goal* is always defined as a system-level objective. Lower-level constructs may inherit or be responsible for goals, but goals always have a system-level context.

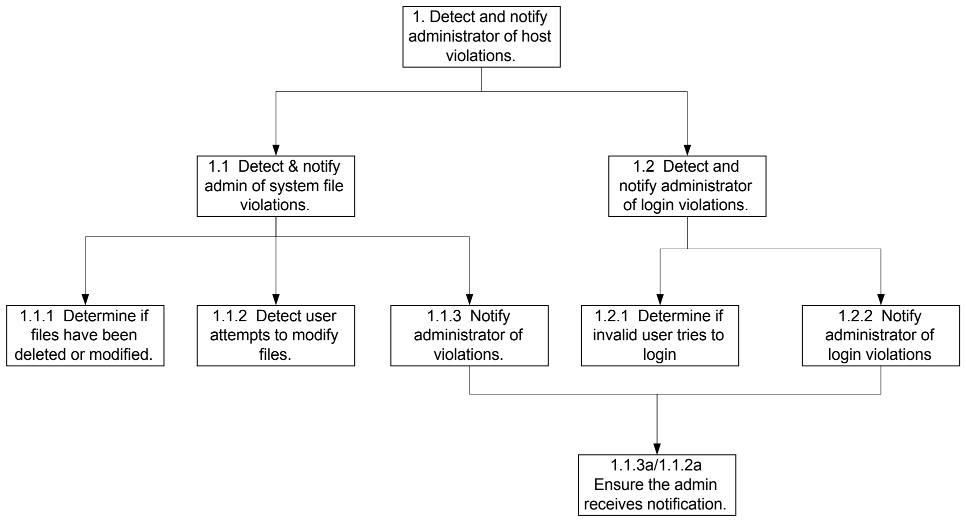

1. Detect and notify administrator of host violations.
1.1 Detect & notify admin of system file violations.
1.2 Detect and notify administrator of login violations.
1.1.1 Determine if files have been deleted or modified.
1.1.2 Detect user attempts to modify files.
1.1.3 Notify administrator of violations.
1.2.1 Determine if invalid user tries to login
1.2.2 Notify administrator of login violations
1.1.3a/1.1.2a Ensure the admin receives notification.

Figure 2. Goal Hierarchy Diagram

There are two steps to *Capturing Goals*: identifying the goals and structuring goals. Goals are identified by distilling the essence of the set of requirements. These requirements may include detailed technical documents, user stories, or formalized specifications. Once these goals have been captured and explicitly stated, they are less likely to change than the detailed steps and activities involved in accomplishing them (Kendall, Palanivelan, & Kalikivayi 1998). Next, the identified goals are analyzed and structured into a Goal Hierarchy Diagram. In a Goal Hierarchy Diagram, goals are organized by importance. Each level of the hierarchy contains goals that are roughly equal in scope and sub-goals are necessary to satisfy parent goals. Eventually, each goal will be associated with roles and agent classes that are responsible for satisfying that goal.

### *Applying Use Cases*

The *Applying Uses Cases* step is a crucial step in translating goals into roles and associated tasks. U*se cases* are drawn from the system requirements and are narrative descriptions of a sequence of events that define desired system behavior. They are examples of how the system should behave in a given case. To help determine the actual communications required within a multiagent system, the use cases are restructured as Sequence Diagrams, as shown in Figure 3. A *Sequence Diagram* depicts a sequence of events between multiple roles and, as a result, defines the minimum communication that must take place between roles. The roles identified in this step form the initial set of roles used to fully define the system roles in the next step. The events identified here are also used later to help define tasks and conversations since all events between roles will require a conversation between the agent classes if the roles are played by different agent classes.

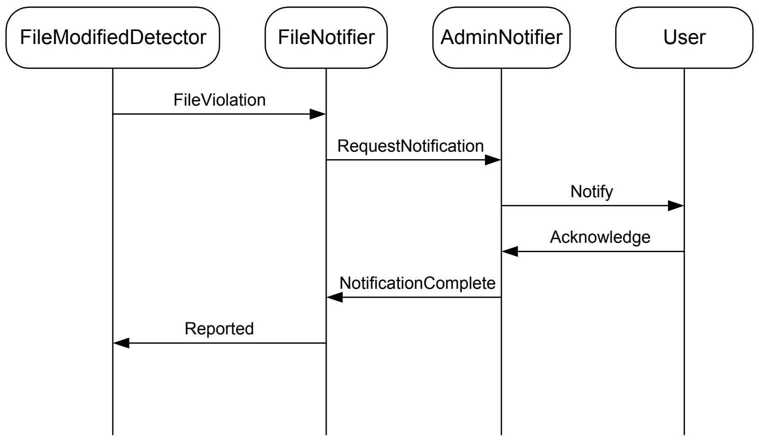


**Figure 3.** Sequence Diagram

*Refining Roles*

The third step in MaSE is to ensure we have identified all the necessary roles and to develop the tasks that define role behavior and communication patterns. Roles are identified from the Sequence Diagrams developed during the Applying Use Cases step as well as the system goals defined in Capturing Goals. We ensure all system goals are accounted for by associating each goal with a specific role that is eventually played by at least one agent in the final design. A *role* is an abstract description of an entity's expected function and is similar to the notion of an actor in a play or an office within an organization (Kendall 1998). Each goal is usually mapped to a single role. However, there are many situations where it is useful to combine multiple goals in a single role for convenience or efficiency. We base these decisions on standard software engineering concepts such as functional, communicational, procedural, or temporal cohesion. Other factors include the natural distribution of resources or special interfacing issues. Roles are captured in a Role Model as shown in Figure 4.

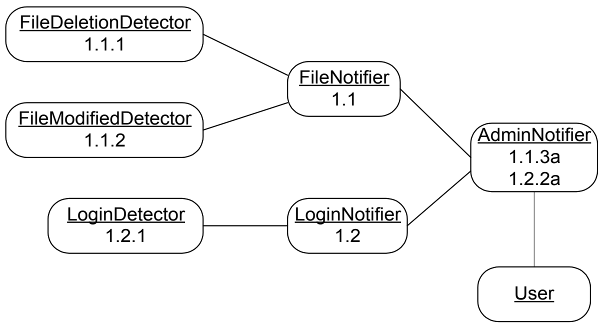


Figure 4. Role Model

Once roles have been defined, tasks are created. A set of concurrent tasks provide a high-level description of what a role must do to satisfy its goals including how it interacts with other roles. An example of a MaSE *Concurrent Task Diagram*, which defines the Notify User task of the AdminNotifier role, is shown in Figure 5. The syntax of a transition follows the notation shown below (DeLoach 2001).

```
trigger(args1) [ guard ] / transmission(args2)
```

The statement is interpreted to say that if an event *trigger* is received with a number of arguments *args*1 and the condition *guard* holds, then the message *transmission* is sent with the set of arguments *args2.* All items are optional. For example, a transition with just a guard condition, *[guard]*, is allowed, as well as one with a received message and a transmission, *trigger / transmission*. Multiple transmission events are also allowed and are separated by semi-colons (;). Actions may be performed in a state and are written as functions.

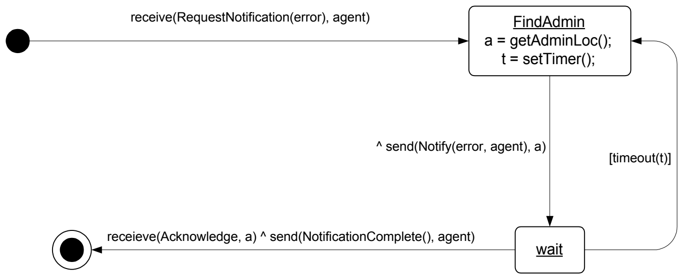

receive(RequestNotification(error), agent)
FindAdmin
a = getAdminLoc();
t = setTimer();
^ send(Notify(error, agent), a)
[timeout(t)]
receieve(Acknowledge, a) ^ send(NotificationComplete(), agent)
wait

**Figure 5.** MaSE Task

*Creating Agent Classes*

In *Creating Agent Classes*, agent classes are identified from roles and documented in an Agent Class Diagram, as shown in Figure 6. Agent Class Diagrams depict agent classes as boxes and the conversations between them as lines connecting the agent classes. As with goals and roles, we generally define a one-to-one mapping between roles, which are listed under the agent class name, and agent classes. However, the designer may combine multiple roles in a single agent class or map a single role to multiple agent classes. Since agents inherit the communication paths between roles, any paths between two roles become conversations between their respective classes. Thus, as the designer assigns roles to agent classes, the overall organization of the system is defined. To make the organization more efficient, it is often desirable to combine two roles that share a high volume of message traffic. When determining which roles to combine, concepts such as cohesion and the volume of message traffic are important considerations.

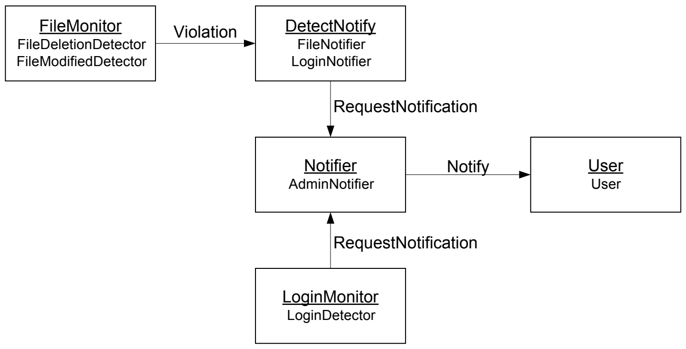


Figure 6. Agent Class Diagram

*Constructing Conversations*

*Constructing Conversations* is the next step of MaSE, which is often performed almost in parallel with the succeeding step of Assembling Agents. The two steps are closely linked, as the agent architecture defined in Assembling Agents must implement the conversations and methods defined in Constructing Conversations. A MaSE conversation defines a coordination protocol between two agents. Specifically, a conversation consists of two Communication Class Diagrams, one each for the initiator and responder. A *Communication Class Diagram* is a pair of finite state machines that define a conversation between two participant agent classes. One side of a conversation is shown in Figure 7. The initiator always begins the conversation by sending the first message. The syntax for Communication Class Diagrams is very similar to that of Concurrent Task Diagrams. The main difference between conversations and concurrent tasks is that concurrent tasks may include multiple conversations between many different roles and tasks whereas conversations are binary exchanges between individual agents.

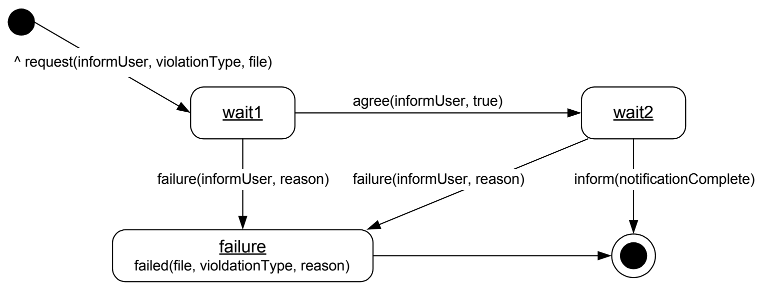

^ request(informUser, violationType, file)
wait1
agree(informUser, true)
wait2
failure(informUser, reason)
failure(informUser, reason)
inform(notificationComplete)
failure
failed(file, violdationType, reason)

Figure 7. Communication Class Diagram

*Assembling Agents*

In this step of MaSE, the internals of agent classes are created. Robinson (Robinson 2000) describes the details of assembling agents from a set of standard or user-defined architectures. This process is simplified by using an architectural modeling language that combines the abstract nature of traditional architectural description languages with the Object Constraint Language, which allows the designer to specify low-level details. The actions specified in the tasks and conversations must be mapped to internal functions of the agent architecture.

*System Deployment*

The final step of MaSE defines the configuration of the actual system to be implemented. To date, we have only looked at static, non-mobile systems although we are currently investigating the specification and design of dynamic and mobile agent systems. In MaSE, we define the overall system architecture using Deployment Diagrams to show the numbers, types, and locations of agents within a system as shown in Figure 8. The three dimensional boxes denote individual agents while the lines connecting them represent actual conversations.  A dashed-line box encompasses agents that are located on the same physical platform.

The agents in a Deployment Diagram are actual instances of agent classes from the Agent Class Diagram.  Since the lines between agents indicate communications paths, they are derived from the conversations defined in the Agent Class Diagram as well.  However, just because an agent type or conversation is defined in the Agent Class Diagram, it does not necessarily have to appear in a Deployment Diagram.

System Deployment is also where all previously undefined implementation decisions, such as programming language or communication framework, must be made.  While in a pure software engineering sense, we want to put off these decisions until this step, there will obviously be times when the decision are made early, perhaps even as part of the requirements.

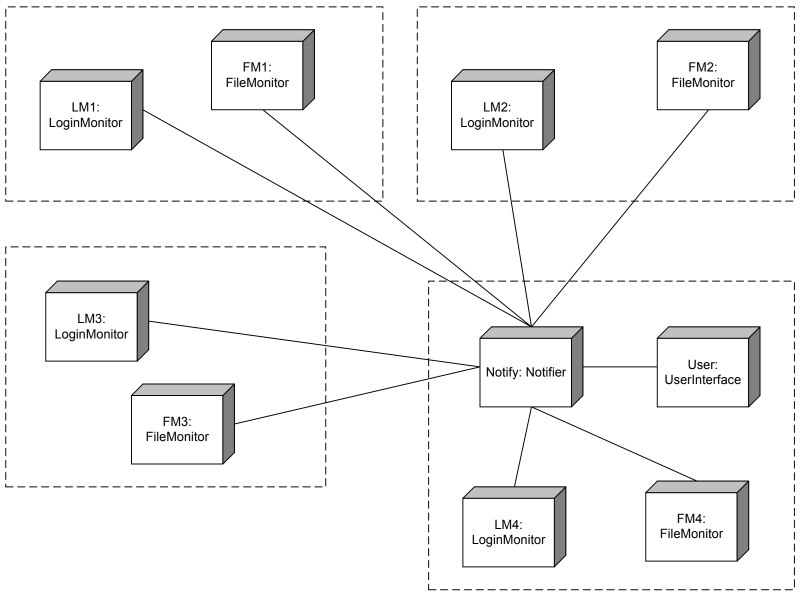


Figure 8.  Deployment Diagram

MaSE, along with agentTool, has been used to develop over 20 small to medium sized multiagent systems ranging from information systems (Kern, Cox & Talbert 2000) (McDonald, Talbert, & DeLoach 2000) and mixed-initiative distributed planners (Cox, Kerkez, Srinivas, Edwin, & Archer 2000) to biologically based immune systems (Harmer & Lamont 2000).  From our research on MaSE and agentTool, we have learned many lessons. First, it is clear that developing a methodology with an eye towards automation and formally defined relationships between the various models simplifies the semantics and makes implementation much easier. Secondly, using object-oriented principles as a basis for our methodology was the right choice. We consider MaSE a domain specific instance of the more general object-oriented paradigm. This also simplifies the underlying formal model as well as code generation. Instead of dealing with a general association, we have just one – a conversation between agents. While agents are not equivalent to an object, they are a specialization. Once again, we can focus our methodology and tool thus making the entire process less complex. Finally, we have shown that you can develop a methodology and tool to support multiple types of agent architectures, languages, and communications frameworks.

## System Implementations

In this section we briefly describe the major implementations that arose from the research we conducted. Along with each description, we also provide a synopsis of evaluation performed upon the system (if any).

### *agentTool*

`http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm`

agentTool is a Java-based graphical development environment to help users analyze, design, and implement multiagent systems. It is designed to support the Multiagent Systems Engineering (MaSE) methodology. The system designer defines high-level system behavior graphically using the Multiagent Systems Engineering methodology.  The system design defines the types of agents in the system as well as the possible communications that may take place between agents. This system-level specification is then refined for each type of agent in the system. To refine an agent, the designer either selects or creates an agent architecture and then provides detailed behavioral specification for each component in the agent architecture.

Once the system has been completely specified, the designer generates the code for the agent system. It is at this point that the designer actually defines the underlying framework and any implementation specific communication and security protocols.  These are automatically built into the code during system synthesis. To take advantage of pre-existing software components, agentTool also uses a library of AI components. Each component has a formal definition which allows an agentTool designer to determine if the component meets the requirements of the specified agent. In this way agentTool only has to synthesize code for components that are not already implementable via component reuse.

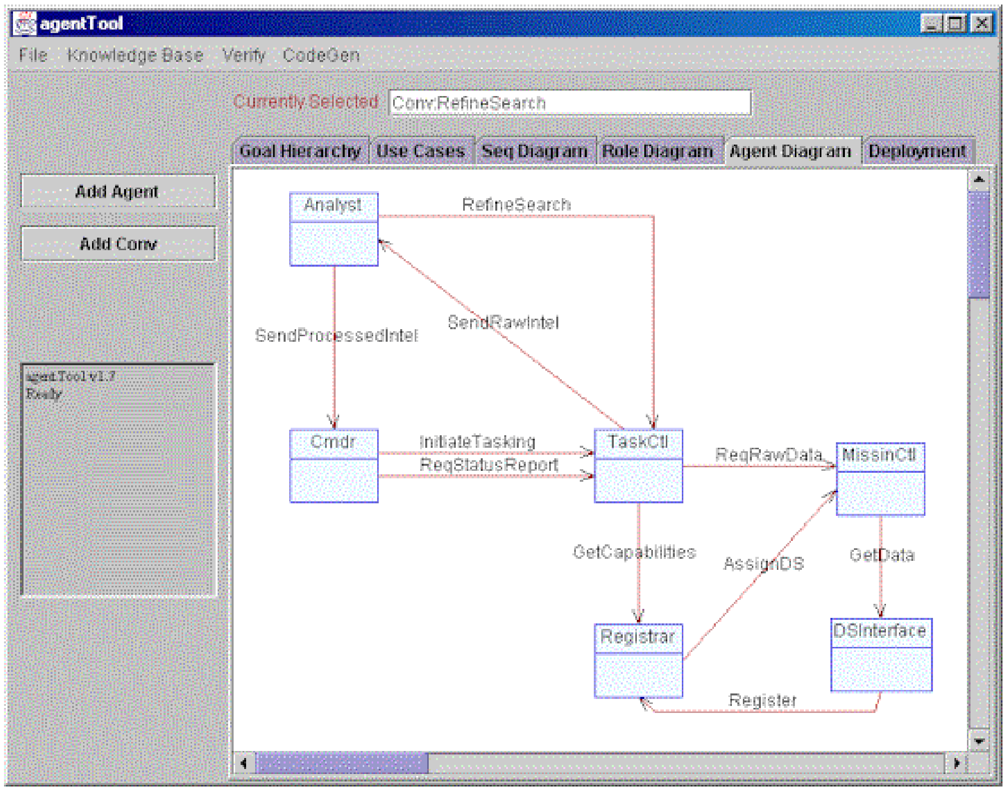


Figure 15. Defining an Agent Communication Protocol in agentTool

The agentTool system was developed using Java 1.3 and runs best on Windows platform, although it has been run successfully on other Java compatible platforms.

*AWSOME*

The AFIT Wide-Spectrum Object-Oriented Modeling Environment (AWSOME) is an environment that transforms knowledge of a domain model over a problem space into correct software solutions (Hartrum & Graham, 2000). The process involves several major transforming steps, but uses the same language (the AFIT Wide spectrum Language AWL) throughout to abstractly represent the system until executable code is produced.

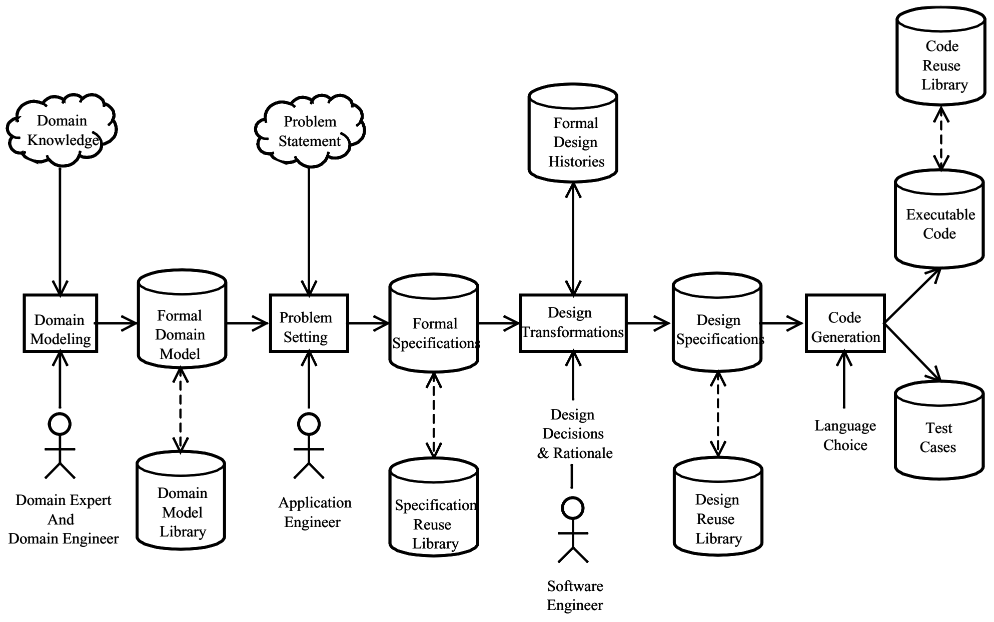


Figure 16. Formal Approach to the Creation of Correct Domain Specific Software

The first stage, Domain Modeling, involves domain experts and engineers using their extensive knowledge of their respective domains to create model representations of the systems. The models, which are represented by a formal surface syntax language such as AWL, are parsed into abstract syntax trees (Acts). As parsers are not available for all model representation languages, other methods, such as interactive editors, can also be used. AWSOME uses this abstract representation of the knowledge throughout the remaining transforms. The output of domain modeling is the Formal Domain Model, denoted DOM, which can be saved in libraries for future utilization in both the surface syntax or AST formats, and is the starting point for the next stage in the process. Application engineers use the Problem Setting process to create system Formal Specifications by satisfying the Problem Statement over a given domain model. The specification that is the result of this stage is then subjected to Design Transforms that convert the system from an analysis to a design representation. Here software engineers make design decisions based on knowledge of software architectures, problem specifications, and other concerns. The formal design histories can be captured and stored for future use if design regeneration is required. The final stage, Code Generation, converts the transformed AST into some chosen language. Currently Java source code is supported.

*COMAS*

We have constructed a multiagent system called COMAS that implements resource coordination as well as goal interaction coordination. COMAS integrates multiple planning agents. We used a version of Prodigy/Agent to represent multiple planning agents. Each agent can make a resource request to another if it needs a resource to achieve resource coordination. Additionally, each agent passes the state change information to the other agent after it has performed an action to achieve goal interaction coordination.

There is a resource coordinator that resolves the resource conflicts using cost-benefit analyses. Goal interaction coordination is implemented in COMAS by broadcasting the state changes to all the other agents in the environment. We used a simple air campaign-planning (ACP) domain (Kent, & Simons, 1994; Thaler, & Shlapak, 1995) for the implementation of resource and goal interaction coordination. This domain required some modifications to meet our desired needs. First, in the initial ACP domain the resources were not assigned to individual rivers. It had a predicate that indicated the state of a resource for example, ‘is-ready f-15A’. This f-15A could be used by any of the goals that needed it. We changed the domain so that it accommodated the assignment of resources explicitly to a goal for example ‘is-available f-15A river1.’

Second, we had to rewrite the control rule that was used to determine if any of the goals needed extra resources. We wrote this control rule so that we were able to identify if a particular goal needed more resources.

Third, we introduced a function in the control rule that performed the cost-benefit analysis.[1] This function identified the number of resources required and then sent the resource requests. The function then receives the cost of obtaining the resources from other agents. The utility value is calculated which is the difference of benefit and the minimum cost obtained. We wrote this control rule such that the function returned the utility value. If the returned utility value is positive, then the originally intended goal is pursued. If the returned utility value is negative then the goal is transformed. Figure 8 shows the control rule that was used to identify if extra resources were required to achieve the intended. The control rule states that if the current sub goal is to destroy a bridge and all the F-15s are no longer available then sends out the resources requests to the resource coordinator. The cost-benefit analysis is performed in the function named “call”. The “call” function returns a positive value if the resource coordinator determines that new resources can be assigned to this goal. A negative value is returned if the resource coordinator determines that the intended goal has to be changed.

*GTrans*

`http://www.cs.wright.edu/~mcox/GTrans/`

As discussed in Section III, GTrans is a mixed-initiative planning system that presents to the user the metaphor as planning as a goal manipulation problem.

1. [5] See Edwin (2001) for the utility calculation details.

In addition to the air campaign domain, GTrans also includes problem in the logistics, or package delivery domain (see Figure 17).

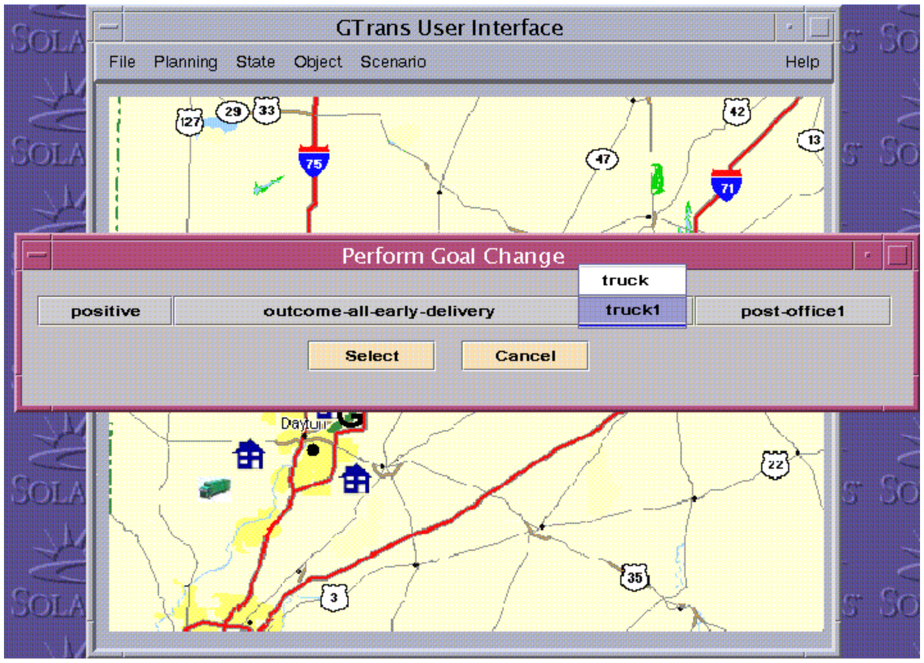


Figure 17. Logistics domain in GTrans

*NARAD*

The search and retrieval of electronic information based purely on keywords that are devoid of context may result in missing potentially important sources or could result in finding too many unrelated sources. Approaches that augment keyword search with contextual profiles of users facilitates tailoring the information search process to the needs of the user. The NARAD distributed agent architecture profiles users by nonintrusively studying the browsing patterns of the user. When participants used a commercial search engine with the NARAD architecture, results showed significant improvement with the search and retrieval process.

NARAD is a collaborative information agent architecture that personalizes the browsing experience of the user by unobtrusively observing with the user's information seeking behavior. The system studies documents that the user visits to create a contextually rich user profile. This profile is dynamic, and it changes with the shift in user interests. The agent operates on the assumption that not all the documents being visited are relevant to the users context. This allows the system to fine tune the framework to accurately reflect the users interest. The architecture identifies, and works with the framework on two levels. At the first level, the frameworks generated are personalized for the individual user. This level is "real-time" in nature; it helps to profile the users current interests. At the second level the concepts are generalized to include other facets that could be related to the current framework, as a result of collaboration between all the agents in the first level. Essentially it distills the expertise of many individual users on a particular topic.

*Prodigy/Agent*

`http://www.cs.wright.edu/~mcox/Prodigy-Agent/`

Prodigy/Agent is both a standalone multiagent planner based on the PRODIGY (Carbonell, et al., 1992) planning and learning architecture and the mixed-initiative planning component in the GTrans system (see above). Rather than rewriting major portions of the PRODIGY planning system so that it can cooperate with other systems (human or otherwise), we have implemented a wrapper in Allegro Common LISP 6.1 to "agentify" PRODIGY. In the resulting Prodigy/Agent system, a simple client-server relationship exists for interaction. Prodigy/Agent is the server and the human or another system is the client. Our conversation protocol is implemented in KQML with sockets. The conversation, as defined with the AFIT agentTool (DeLoach, 1999) is shown in Figure 18.[2]

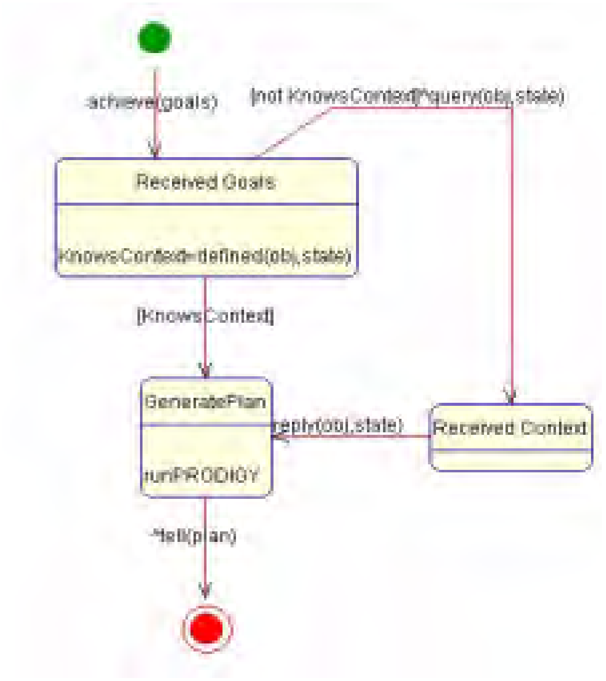


Figure 18. Wrapper conversation protocol
from Prodigy/Agent perspective

Upon receipt of an achieve(goals) message in KQML,[3] the PRODIGY wrapper tests to see if the context of the problem (i.e., the list of objects and state relationships between objects) is known. If not, the wrapper transmits a query(obj,state) message to the client that sent the achieve message. Once the state and object information is known (via a reply(obj,state) message), PRODIGY executes the problem defined by the goals, objects, and state in order to generate a plan. The wrapper then sends the list of actions (operators plus variable bindings) to the client with a tell(plan) message. In this manner PRODIGY is functionally an intelligent agent from the perspective of all other agents communicating with it.

2. Messages marked by the carrot (^) are transmitted by Prodigy/Agent, whereas all other messages are received. Unlike the conventional semantics of achieve that require an agent to actually achieve the request, Prodigy/Agent only plans for the achievement. Note also that both this conversation and the next do not show the acknowledgment messages sent between agents.
3. In these messages, the KQML *language* field is defined to be the PRODIGY Definition Language BNF (Carbonell et a., 1992) and the *ontology* field specifies a URL to the domain.lisp text file. This file defines the semantic hierarchy and the operator specifications of actions.

*P4P*

The P4P system implements a planner that uses knowledge about the characteristics and the content of windows to achieve a state of a reduced screen-clutter for its own user interface. The architecture consists of three major components (see Figure 19). The first component is an underlying state-space nonlinear planner called PRODIGY. PRODIGY produces plans for the user in a user-selected domain, and concurrently, it produces plans for the interface to reduce screen clutter. The second component is the User Interface for PRODIGY (UIP). UIP allows the user to control the PRODIGY planner and to view the plan as it unfolds. It also performs actions on the window layout as suggested by PRODIGY. The third component is the meta level communication layer that consists of a transducer, a wrapper, and a Java User Interface Agent (UIA) application. The meta level communication layer performs I/O between the UIP and the PRODIGY planner.

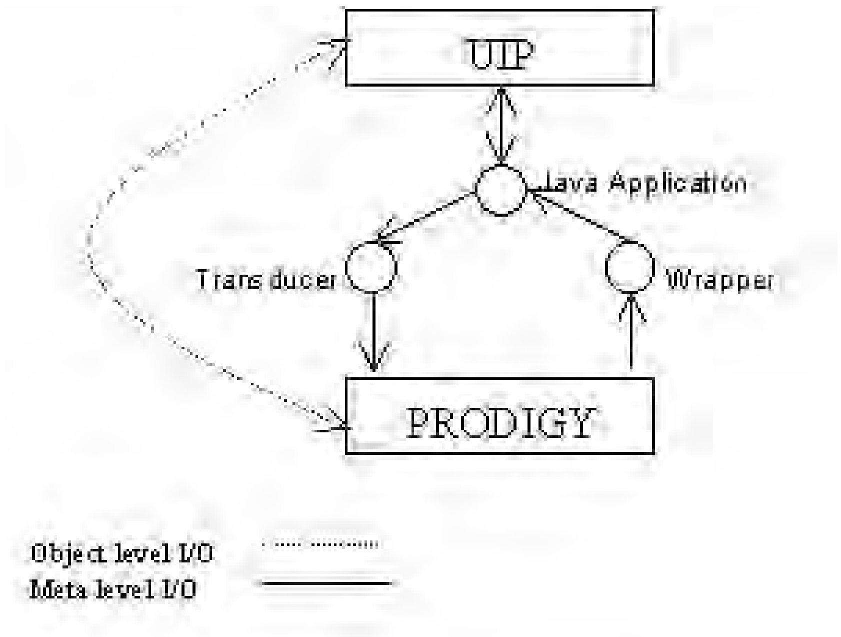


Figure 19. P4P System Architecture

## VI Deliverables

In this section we enumerate the contributions the research has made toward the promised deliverables first outlined in the original research proposal. Although the research is focussed upon basic research, we have also provided a proof of concept by implementing a number of computational systems. In terms of basic research, we first list the graduate degrees and proposals completed by DAGSI graduate students under our supervision. Next we list the publications generated by the project. Subsequently we briefly describe the implemented systems and interfaces created and significantly modified by members of the project. Finally we describe the efforts at scientific dissemination provided by the project Web pages and the graduate course in multiagent systems and mixed-initiative planning that we put together during the course of this research.

### Graduate Degrees and Proposals

Edwin, G. (2001). *Coordination in multi-agent systems*. Masters dissertation, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Kerkez, B. (2001). *Incremental case-based keyhole plan recognition* (Tech. Report No. WSU-CS-01-01). Doctoral proposal, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Kern, S. (2000). *A human-centered approach for designing decision support systems*. Masters dissertation, Air Force Institute of Technology, Computer Science Department, Dayton Ohio.

Lacey, Timothy H., Captain, USAF. A Formal Methodology and Technique for Verifying Communication Protocols in a Multi-agent Environment. MS thesis, AFIT/GCS/ENG/00M-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 2000.

Lee, P. (2002). *Dimensional indexing for case-based retrieval*. Masters dissertation, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Moss, P. (2001). *Modeling human performance in a game of chance using machine learning*. Masters dissertation, Wright State University, Department of Biomedical, Industrial, and Human Factors Engineering, Dayton Ohio

Nonnweiler, J. (1999). *Software Domain Model Integration Methodology for Formal Specifications*. MS thesis, Air Force Institute of Technology, WPAFB, OH, March 1999. AFIT/GCS/ENG/01M-07.

Prasad, R. G. K. (2001). *Contextual informational retrieval*. Masters dissertation, Wright State University, Department of Biomedical, Industrial, and Human Factors Engineering, Dayton Ohio.

Self, Athie L., Captain, USAF. Design & Specification of Dynamic, Mobile, and Reconfigurable Multiagent Systems. MS thesis, AFIT/GCS/ENG/01M-11. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 2001.

Sparkman, Clint H., Lieutenant, USAF. Transforming Analysis Models into Design Models for the Multiagent Systems Engineering Methodology. MS thesis, AFIT/GCS/ENG/01M-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 2001.

Wood, Mark, Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems. MS thesis, AFIT/GCS/ENG/00M-26. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 2000.

**Scientific Publications and Presentations**

`http://www.cs.wright.edu/~mcox/DAGSI/abmic_pubs.html`

Numerous publication were submitted, accepted and/or published. Most of the following publications are available in postscript, Word, or PDF form at the above URL and from a links available at the ABMIC Home page (see the subsequent subsection for details).

*Published*

Bryson, J., Decker, K., DeLoach, S. A, Huhns, M., & Wooldridge, M. (2000). Agent Development Tools, In C. Castelfranchi & Y. Lesperance (Eds.), *Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop.* Berlin: Springer Verlag.

Cox, M. T. (2000). A conflict of metaphors: Modeling the planning process. In Proceedings of 2000 Summer Computer Simulation Conference (pp. 666-671). San Diego: The Society for Computer Simulation.

Cox, M. T. (Ed.) (1999). Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence. Menlo Park, CA: AAAI Press.

Cox, M. T., Edwin, G., Balasubramanian, K., & Elahi, M. (2001). Multiagent goal transformation and mixed-initiative planning using Prodigy/Agent. In N. Callaos, B. Sanchez, L. H. Encinas, & J. G. Busse (Eds.), *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics, Vol. VII* (pp. 1-6). Orlando, FL: International Institute of Informatics and Systemics.

Cox, M. T., Kerkez, B., Srinivas, C, Edwin, G., & Archer, W. (2000). Toward agent-based mixed-initiative interfaces. In H. R. Arabnia (Ed.), *Proceedings of the 2000 International Conference on Artificial Intelligence, Vol. 1* (pp. 309-315). CSREA Press.

Brown, S., & Cox, M. (1999). Planning for information visualization in mixed-initiative systems. In M. T. Cox (Ed.), *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence* (pp. 2-10). Menlo Park, CA: AAAI Press.

Deckard, M., Narayanan, S., & Cox, M. T. (2000). Natural system metaphors for supporting collaboration in Air Force applications. In *Proceedings of the 2000 Human Factors & Ergonomics Society/Industrial Ergonomics Meeting, Vol. 2* (pp. 614-617).

DeLoach, S. (in press). Multiagent Systems Engineering. To appear in the *International Journal on Software Engineering and Knowledge Engineering*.

DeLoach, S. A. (2001). Analysis and Design using MaSE and agentTool, In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*. Miami University, Oxford, Ohio, March 31 - April 1, 2001.

DeLoach, S. A. (2001). Specifying agent behavior as concurrent tasks: Defining the behavior of social agents. In *Proceedings of the Fifth Annual Conference on Autonomous Agents*, Montreal Canada, May 28 - June 1, 2001.

DeLoach, S. A. (1999). Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. *Agent-Oriented Information Systems (AOIS)*. Seattle Washington: May 1999.

DeLoach, S. A., Matson, E. T., & Li, Y. (in press). Applying agent oriented software engineering to cooperative robotics," To appear in *Proceedings of the The 15th International FLAIRS Conference (FLAIRS 2002)*. Pensacola, Florida. May 16-18, 2002.

DeLoach, S. A., & Wood, M. (2001). Developing Multiagent Systems with agentTool. In *Intelligent Agents VII. Agent Theories Architectures and Languages*, 7th International Workshop ( ATAL 2000, Boston, MA, USA, July 7-9, 2000), C. Castelfranchi, Y. Lesperance (Eds.). Lecture Notes in Computer Science. Vol. 1986, Springer Verlag, Berlin.

DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multiagent systems engineering, *The International Journal of Software Engineering and Knowledge Engineering*, Volume 11 no. 3.

Edala, N. R., Garay, M., Narayanan, S. & Eggleston, R. G. (1999). A distributed simulation environment for analyzing adaptive agent behavior in supervisory control systems. *Proceedings of the 1999 Summer Computer Simulation Conference.*

Edwin, G. (2001). *Coordination in multi-agent systems*. Masters dissertation, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Edwin, G., & Cox, M. T. (2001a). COMAS: CoOrdination in MultiAgent Systems. In N. Callaos, B. Sanchez, L. H. Encinas, & J. G. Busse (Eds.), *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics, Vol. VII* (pp. 7-12). Orlando, FL: International Institute of Informatics and Systemics.

Edwin, G., & Cox, M. T. (2001b). Resource coordination in single agent and multiagent systems. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence* (pp. 18-24). Los Alamitos, CA: IEEE Computer Society.

Garay, M., Edala, N. , Narayanan, S. & Eggleston, R. (1999). An architecture for studying mixed-initiative planning and control in a complex, dynamic environment. *Proceedings of the 1999 AAAI Workshop on Mixed-Initiative Intelligent Systems*, 115-118.

Hartrum, T. C., Graham, Jr., R. P. (2000). The AFIT wide spectrum object modeling environment: An AWSOME beginning. In *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference (NAECON)*. Dayton OH: October 2000.

Kerkez, B. (2001). *Incremental case-based keyhole plan recognition* (Tech. Report No. WSU-CS-01-01). Doctoral proposal, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Kerkez, B, & Cox. M. T. (2001). Incremental case-based plan recognition using state indices. In D. Aha & I. Watson (Eds.), *Proceedings of the Fourth International Conference on Case-Based Reasoning* (pp. 291-305). Berlin: Springer.

Kerkez, B, & Cox. M. T. (2000). Planning for the user interface: Window characteristics. In *Proceedings of the 11th Midwest Artificial Intelligence and Cognitive Science* (pp. 79-84). Menlo Park, CA: AAAI Press.

Kerkez, B, Cox. M. T., & Srinivas, C. (2000). Planning for the user interface: Window content. In H. R. Arabnia (Ed.), *Proceedings of the 2000 International Conference on Artificial Intelligence, Vol. 1* (pp. 345-351). CSREA Press.

Kern, S., & Cox. M. T. (2000). A problem representation approach for decision support systems. In *Proceedings of the 11th Midwest Artificial Intelligence and Cognitive Science* (pp. 68-73). Menlo Park, CA: AAAI Press.

Lee, P. (2002). *Dimensional indexing for case-based retrieval*. Masters dissertation, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Lacey, T. & DeLoach, S. A. (2000). Verification of Agent Behavioral Models. *Proceedings of the International Conference on Artificial Intelligence (IC-AI'2000)*. CSREA Press, Vol II, pp 557-563. June 26 -29, 2000, Las Vegas, Nevada.

McDonald, J. T., Talbert, M. L., & DeLoach. S. A. (2000). Heterogeneous Database Integration Using Agent Oriented Information Systems. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'2000)*. CSREA Press, Vol III, pp. 1359-1365. June 26 - 29, 2000, Las Vegas, Nevada.

Moss, P. (2001). *Modeling human performance in a game of chance using machine learning*. Masters dissertation, Wright State University, Department of Biomedical, Industrial, and Human Factors Engineering, Dayton Ohio

Mulvehill, A., & Cox, M. (1999). Using mixed initiative to support force deployment and execution. In M. T. Cox (Ed.), *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence* (pp. 119-123). Menlo Park, CA: AAAI Press.

Nonnweiler, J. (1999). *Software Domain Model Integration Methodology for Formal Specifications*. MS thesis, Air Force Institute of Technology, WPAFB, OH, March 1999. AFIT/GCS/ENG/01M-07.

O'Malley, S. A., & DeLoach, S. A. (2001). Determining When to Use an Agent-Oriented Software Engineering Paradigm, In *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, Canada, May 29th 2001.

O'Malley, S. A., Self, A. L., & DeLoach, S. A. (2000). Comparing Performance of Static versus Mobile Multiagent Systems, *National Aerospace and Electronics Conference (NAECON)* Dayton, OH, October 10-12, 2000.

Prasad, R. G. K. (2001). *Contextual informational retrieval*. Masters dissertation, Wright State University, Department of Biomedical, Industrial, and Human Factors Engineering, Dayton Ohio.

Prasad, R. G. K., & Narayanan, S. (2001). Personalization of information retrieval through user profiling. *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics*. Phoenix, AZ, CD-ROM publication.

Raphael, M. J. & DeLoach, S. A. (2000). A Knowledge Base for Knowledge-Based Multiagent System Construction, *National Aerospace and Electronics Conference (NAECON)* to Dayton, OH, October 10-12, 2000.

Srinivas, C., Cox, M. T., & Laxminarayanan, V. (2000). *GTrans User manual and reference* (Tech. Rep. No. WSU-CS-00-02). Dayton, OH: Wright State University, Department of Computer Science and Engineering.

Wood, M. & DeLoach, S. A. (2001). An Overview of the Multiagent Systems Engineering Methodology, In *Agent-Oriented Software Engineering*. P. Ciancarini, M. Wooldridge, (Eds.) Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January 2001.

*Accepted for Publication*

Cox, M. T. (in press). Toward tailored information presentation in support of collaborative planning. To appear in B. Bell (Ed.), *Proceedings of the AAAI Fall Symposia on Team Intent Inference*. Menlo Park, CA: AAAI Press / The MIT Press.

Lee, P., & Cox, M. T. (in press). Dimensional indexing for targeted case-base retrieval: The SMIRKS system. To appear in *Proceedings of the 15th International FLAIRS Conference*. Menlo Park, CA: AAAI Press.

*Submitted for Publication*

Prasad, R. G. K., & Narayanan, S. (2001). Contextual electronic information retrieval. Submitted to *Journal of the American Society for Information Science and Technology.*

*Presentations*

Cox, M. T. (2000, January). Interfaces for mixed-initiative planning. Position paper, *IUI'2000 Workshop on Using Plans in Intelligent User Interfaces*. Cambridge, MA: MERL.

Cox, M. T. (2000a, April). *Multi agent coordination and planning*. Invited presentation, 11th Midwest Artificial Intelligence and Cognitive Science (MAICS-2000). University of Arkansas, Fayetteville, AR.

Cox, M. T. (2000b, April). *Planning is NOT search: Why humans and machines have trouble working together.* Invited panel discussant, Aerosense 2000. Orlando, FL.

Cox, M. T. (1999a, May). *Agent-based mixed-initiative collaboration*. Invited presentation, Computer Science Colloqium Series, Department of Computer Science and Engineering, University of Connecticut, Storrs, CT.

**Internet Presence**

One of the deliverables for this project is the establishment of an internet presence that will facilitate dissemination of research results and help establish links to related research projects.

*Project Home Page*

`http://www.cs.wright.edu/~mcox/DAGSI`

The main Web page from which all public information concerning the project may be obtained. It contains list of participants and organizations, public software, publication lists, contact points, general descriptions and pointers to related sites and associated information.

*AAAI Symposium of Mixed-Initiative Intelligence*

`http://www.cs.wright.edu/~mcox/mii/`

This is the home page for a symposium on Mixed-Initiative Intelligence organized by the PI. This event brought together most of the members of the project along with academic participants and governmental representatives to share research results in the area. The symposium was sponsored by the American Association on Artificial Intelligence and took place in Orlando, FL, during the first quarter of the project.

*agenTool site*

`http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm`

Dr. Scott DeLoach has recently left the Air Force and assumed a faculty position at kansas State University. His agentTool web site has moved there. The software has received much access across users of and researchers on the internet.

**Graduate Research Course**

`http://www.cs.wright.edu/~mcox/Teaching/Cs790/`

In the Spring Quarter of 2000, Drs. Cox and DeLoach taught an advanced level course in Artificial Intelligence (AI) concentrating on Multiagent Systems and Mixed-Initiative Planning. Students taking this class have previously taken an introductory class in AI at the graduate level. The course introduced the student to basic concepts in distributed intelligent planning using agent-oriented methods. Subsequently the instructors developed the concept of mixed-initiative planners (i.e., a planning system that actively includes humans in the planning process). Finally the students studied a novel research paradigm that incorporated teams of intelligent agents and humans that together solve difficult collaborative planning problems.This term project explored open issues within this paradigm and provided students with hands-on experience in one of the newest research areas within AI.

This class is being taught again during the Spring Quarter of 2002.

## VII References

Allen, J. F., Ferguson, G., and Schubert, L. K. (1996). Planning in Complex Worlds via Mixed-Initiative Interaction (pp. 53-60). *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, AAAI Press.

Blum, A. and Furst, M. (1997). *Artificial Intelligence, 90* (1&2): 279-298.

Brown, S., & Cox, M. (1999). Planning for information visualization in mixed-initiative systems. In M. T. Cox (Ed.), *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence* (pp. 2-10). Menlo Park, CA: AAAI Press.

Burstein, M. (Ed.). (1994). *ARPA/Rome Laboratory Knowledge-based Planning and Scheduling Initiative Workshop Proceedings*. San Francisco: Morgan Kaufmann.

Carbonell, J. G., Blythe, J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Perez, A., Reilly, S., Veloso, M. M., and Wang, X. (1992). *Prodigy4.0: The Manual and Tutoria*l. Technical Report, CMU-CS-92-150. Computer Science Department., Carnegie Mellon University.

Cox, M. T. (2000). A conflict of metaphors: Modeling the planning process. In Proceedings of 2000 Summer Computer Simulation Conference (pp. 666-671). San Diego: The Society for Computer Simulation.

Cox, M. T. (Ed.) (1999). *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence*. Menlo Park, CA: AAAI Press.

Cox, M. T., Edwin, G., Balasubramanian, K., & Elahi, M. (2001). Multiagent goal transformation and mixed-initiative planning using Prodigy/Agent. In N. Callaos, B. Sanchez, L. H. Encinas, & J. G. Busse (Eds.), *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics, Vol. VII* (pp. 1-6). Orlando, FL: International Institute of Informatics and Systemics.

Cox, M. T., Kerkez, B., Srinivas, C, Edwin, G., and Archer, W. (in press). Toward agent-based mixed-initiative interfaces. In *Proceedings of the 2000 International Conference on Artificial Intelligence*. CSREA Press.

Cox, M. T., & Veloso, M. M. (1997a). Controlling for unexpected goals when planning in a mixed-initiative setting. In E. Costa and A. Cardoso (Eds.), *Progress in Artificial Intelligence: Eighth Portuguese Conference on Artificial Intelligence* (pp. 309-318). Berlin: Springer.

Cox, M. T., and Veloso, M. M. (1997b). Supporting combined human and machine planning: An interface for planning by analogical reasoning. In D. B. Leake and E. Plaza (Eds.), *Case-Based Reasoning Research and Development: Second International Conference on Case-Based Reasoning* (pp. 531-540). Berlin: Springer-Verlag.

Cox, M. T., and Veloso, M. M. (1998). Goal Transformations in Continuous Planning. In M. desJardins (Ed.), *Proceedings of the 1998 AAAI Fall Symposium on Distributed Continual Planning* (pp 23-30). Menlo Park, CA: AAAI Press / The MIT Press.

Das, J. P., Kar, B. C., and Parilla, R. K. (1996). *Cognitive planning: The psychological basis of intelligent behavior.* London: Sage Publications.

Deckard, M., Narayanan, S., & Cox, M. T. (2000). Natural system metaphors for supporting collaboration in Air Force applications. In *Proceedings of the 2000 Human Factors & Ergonomics Society/Industrial Ergonomics Meeting, Vol. 2* (pp. 614-617).

DeLoach, S. A. (1999). Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. *Agent-Oriented Information Systems (AOIS)*. Seattle Washington: May 1999.

DeLoach, S. A. (2001). Specifying Agent Behavior as Concurrent Tasks: Defining the Behavior of Social Agents. In *Proceedings of the Fifth Annual Conference on Autonomous Agents*, Montreal Canada, May 28 - June 1, 2001.

DeLoach, S. A., Wood ,M. F., & Sparkman, C. H. (2001). Multiagent Systems Engineering, *The International Journal of Software Engineering and Knowledge Engineering*, Volume 11 no. 3..

Edwin, G. (2001). *Coordination in multi-agent systems*. Masters dissertation, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Edwin, G., & Cox, M. T. (2001a). COMAS: CoOrdination in MultiAgent Systems. In N. Callaos, B. Sanchez, L. H. Encinas, & J. G. Busse (Eds.), *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics, Vol. VII* (pp. 7-12). Orlando, FL:

International Institute of Informatics and Systemics.

Edwin, G., & Cox, M. T. (2001b). Resource coordination in single agent and multiagent systems. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence* (pp. 18-24). Los Alamitos, CA: IEEE Computer Society.

Ferguson, G., and Allen, J. F. (1998). TRIPS: An Integrated Intelligent Problem-Solving Assistant (pp. 26-30). In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Ferguson, Myers, and Smith, S. (Eds). (1998). *Proceedings of the Interactive and Collaborative Planning Workshop*. Held in Conjunction with the Fourth International Conference on Artificial Intelligence Planning Systems. CMU, Pittsburgh, PA.

Fikes, R. E., and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*: 189-208.

Finin, T.; McKay, D.; and Fritzson, R. 1992. *An Overview of KQML: A Knowledge Query and Manipulation Language*. Univ. of Maryland, CS Dept.

Haller, S., McRoy, S., and Kobsa, A. (Eds.). (1998). *Computational Models of Mixed-Initiative Interaction*. New York: Kluwer.

P. Harmer and G. Lamont. (2000). An Agent Architecture for a Computer Virus Immune System. *Workshop on Artificial Immune Systems at Genetic and Evolutionary Computation Conference*, Las Vegas, Nevada, July.

E. Kendall. (1998). Agent Roles and Role Models: New Abstractions for Multiagent System Analysis and Design. *Proceedings of the International Workshop on Intelligent Agents in Information and Process Management*, Bremen, Germany, September.

E. Kendall, U. Palanivelan, and S. Kalikivayi. Capturing and Structuring Goals: Analysis Patterns. *Proceedings of the Third European Conference on Pattern Languages of Programming and Computing*, Bad Irsee, Germany, July 1998.

Hartrum, T. C., & Graham, Jr., R. P. (2000). The AFIT Wide Spectrum Object Modeling Environment: an AWSOME Beginning. In *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference (NAECON)*. Dayton OH: October 2000.

Hendler, J., Tate, A., and Drummond, M. (1990) A Review of AI Planning Techniques (pp.

26-49). In J. Allen, J. Hendler, and A. Tate (Eds.), *Readings in planning*. San Francisco: Morgan Kaufmann.

Kambhampati, S., Parker, E., and Lambrecht, E. (1997). Understanding and extending Graphplan (pp. 260- 272). In *Recent advances in AI planning: Proceedings of the Fourth European Conference on Planning, ECP97*. Berlin: Springer.

Kambhampati, S., and Srivastava, B. (1996). Universal classical planner: An algorithm for unifying state-space and plan-space planning (pp. 61-75). In M. Ghallab and A. Milani (Eds.) *New Directions in AI Planning*. Amsterdam: IOS Press

Kautz, H., and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search (pp. 1194-1201). In *Proc. of the Thirteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Kent, G. A., and Simons, W. E. (1994). Objective-based planning (pp. 59-71). In P. K. Davis (Ed.), *New challenges for defense planning: Rethinking how much is enough*. Santa Monica, CA: RAND.

Kerkez, B. (2001). *Incremental case-based keyhole plan recognition* (Tech. Report No. WSU-CS-01-01). Doctoral proposal, Wright State University, Computer Science and Engineering Department, Dayton Ohio.

Kerkez, B, & Cox. M. T. (2001). Incremental case-based plan recognition using state indices. In D. Aha & I. Watson (Eds.), *Proceedings of the Fourth International Conference on Case-Based Reasoning* (pp. 291-305). Berlin: Springer.

Kerkez, B, & Cox. M. T. (2000). Planning for the user interface: Window characteristics. In *Proceedings of the 11th Midwest Artificial Intelligence and Cognitive Science* (pp. 79-84). Menlo Park, CA: AAAI Press.

Kerkez, B, Cox. M. T., & Srinivas, C. (2000). Planning for the user interface: Window content. In H. R. Arabnia (Ed.), *Proceedings of the 2000 International Conference on Artificial Intelligence, Vol. 1* (pp. 345-351). CSREA Press.

T. Lacey, S. DeLoach. (2000). Automatic Verification of Multiagent Conversations. *Proceedings of the Eleventh Annual Midwest Artificial Intelligence and Cognitive Science Conference,* pages 93-100. AAAI.

McDonald, J., Talbert, M., & DeLoach, S. A. (2000). Heterogeneous Database Integration Using Agent Oriented Information Systems. *Proceedings of the International Conference on Artificial Intelligence (IC-AI) 2000*. pages 1359-1366, CSREA Press.

Mulvehill, A., and Cox, M. (1999). Using Mixed Initiative to Support Force Deployment and Execution. In M. T. Cox (Ed.), *Proceedings of the 1999 AAAI-99 Workshop on Mixed-Initiative Intelligence* (pp. 119-123). Menlo Park, CA: AAAI Press.

Nonnweiler, J. (1999). *Software Domain Model Integration Methodology for Formal Specifications*. MS thesis, Air Force Institute of Technology, WPAFB, OH, March 1999. AFIT/GCS/ENG/01M-07.

Raphael,M. ( 2000). *Knowledge Base Support for Design and Synthesis of Multi-agent Systems*. MS thesis, AFIT/ENG/00M-21. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, USA, March.

Robinson, D. (2000). *A Component Based Approach to Agent Specification*. MS thesis, AFIT/ENG/00M-22. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, USA, March.

Saba, M. G., & Santos, E. (2000). The multi-agent distributed goal satisfaction system, *Proceedings of the International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce* (MAMA '2000) 2000.

Szyperski, C. (1999). *Component Software: Beyond Object-Oriented Programming*. New York: ACM Press.

Thaler, D. E., and Shlapak, D. A. (1995). *Perspectives on theater air campaign planning*. Santa Monica, CA: RAND.

Veloso, M. M., Carbonell, J. G., Perez, A., Borrajo, D., Fink, E., and Blythe, J. (1995) Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental Artificial Intelligence. 7(1)*: 81-120.

Veloso, M. M., Mulvehill, A. M., and Cox, M. T. (1997). Rationale-supported mixed-initiative case-based planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference* (pp. 1072-1077). Menlo Park, CA: AAAI Press.

Veloso, M. M., Pollack, M. E., & Cox, M. T. (1998). Rationale-based monitoring for continuous planning in dynamic environments. In R. Simmons, M. Veloso, & S. Smith (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems* (pp. 171-179). Menlo Park, CA: AAAI Press.

## References

S. Kern, M. Cox, and M. Talbert. (2000). A Problem Representation Approach for Decision Support Systems. *Proceedings of the Eleventh Annual Midwest Artificial Intelligence and Cognitive Science Conference*, pages 68-73. AAAI.