Addis Ababa University
School of Graduate Studies

Incrementally Autonomous Light Weight Agent Architectures for Optimization Task

A Thesis submitted to the Faculty of Technology of
Addis Ababa University
in partial fulfillment of the requirements for
the degree of
Master of Science in
Computer Engineering

Advisor: Prof. Kumudha Raimond and Prof. Dr. Gerald Sommer

By: Nardos Asnake

Department of Electrical and Computer Engineering

March, 2005

Acknowledgment

First of all I would like to thank God for being with me and for helping me in all the ways through.

Unlimited gratitude goes to Prof. Dr. Gerald Sommer and Prof. Kumudha Raimond for being my advisors. I am also grateful for their consistent consultation, encouragement and unreserved support.

Deepest thanks to Yohannes Kassahun for his concern, continual advice and supervision of my work.

I am thankful for German Academic Exchange Service (DAAD) for sponsoring me to do this work.

Finally, I am grateful to my family for unlimited support, encouragement and for being with my side.

Table of contents

Abstract

An agent oriented software engineering paradigm has been applied in solving engineering problems. In this work, agent architecture is developed that is suitable to solve a given problem. The system is developed progressively beginning from simple expert system to a system where agents learn to solve a given task based on some performance measure and agent architecture. Self-organization, emergence of global behavior and learning at population are considered while designing the agent architecture. Different ways of interaction and cooperation between agents are used to realize the collaboration among them. The transfer of problem solving task from human being to agents is clearly elaborated in the test scenario chosen for the work.

1 Introduction

The motivation for this work is the design of computer programs as multi-agent systems, which presents a useful software engineering paradigm where systems are described as individual problem-solving agents pursuing high-level goals. Although having such an abstraction seems promising, its widespread adoption among system designers has not been materialized yet. One of the reasons is that multi-agent system development is technically difficult task. Efforts are challenged not only by known distributed programming issues, but also by the complexities associated with supporting agent collaboration. In this work, agent architecture is developed where agents collaborate and are trained as a group to bring a solution for a problem on the behalf of human beings. Different ways of interaction and collaboration methods are used by these agents to purse their goal.

Empirical observations of smaller-size optimization problems show that the number of solutions increases exponentially with increasing size (Sosi and Gu, 1994). Alternatively, search-based algorithms have been developed. For example, a backtracking search will systematically generate all possible solution sets for optimizations problems (Bitner and Reingold, 1975 and Purdom and Brown, 1983). In practice, however, backtracking approaches provide a very limited class of solutions for large size because it is difficult for a backtracking search to find solutions that are significantly distinct in the solution space (Sosi and Gu, 1994). Several authors have proposed other efficient search techniques to overcome this problem. Introduction a number of efficient search algorithms have been proposed for solving combinatorial optimization and constraint satisfaction problems using an iterative improvement approach. Their idea is to start searching from a possible solution, and to try to improve it in a sequence of successive steps. Several methods have been proposed to overcome this limitation: random-restart hill-climbing, tabu search, simulated annealing, genetic programming.

Agent's paradigm is proposed as a best way to solve optimization problem. There are few agent and multi-agents architecture developed to solve optimization problem. For instance multi agent architecture is proposed to solve the Kinematics inversion problem (Amar Ramdane-Cherif and Nicole Levy, Hicham Djenildi and Chakib Tadj, 2002). Intelligent agents have successfully solved the train pathing problem on a small portion of railroad network. As the railroad network grows, it is imperative that the agents collaborate to operate as efficiently as possible (Blum, J. and Eskandarian, A., 2002).

An agent system paradigm is becoming an important subject in today's large society of researchers and real world application developers. As a result, this thesis has been committed to the development of software application, which can help one to understand and learn the use of agent systems, through solving specific optimization problem. Through the development of this application, it is possible to study the behaviors of an agent system. This is carried out in phases where the phases span from simple backtracking algorithm and genetic algorithm to the use of agent architecture. In each of the phases, the transfer of problem solving task from human being to the system is shown. In the progressive phase the intervention and decision making of a human being is reduced as the system becomes intelligent to solve by its own.

The thesis is organized as follows. In the second chapter, the general and specific property of an agent is explained. The basic kinds of agent programs and the design principles of the software agent are discussed.

The third chapter deals with multi-agent system. The interaction between these agents is presented. Some basic architecture of multi agent-system is also discussed. The needs and capabilities of multi agents system are also dealt.
In the fourth chapter genetic algorithms are presented. The natural way of selection used to find an optimal solution for different searching problems is discussed. Different genetic operators are also explained.

The fifth chapter presents the actual work done in the research. The work is done in separate sequence of phases by taking a test scenario. At each phase, the system becomes more independent and the intervention of a human being in making a decision is reduced. Finally, conclusion and outlook of the work is given.

2 Intelligent Agents

2.1 Genetic Agent

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [8]. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors. A software agent receives keystrokes, file contents and network packets as sensory input and acts on the environment by displaying on screen, and sending network packets. A simple generic agent is illustrated in Fig 2.1

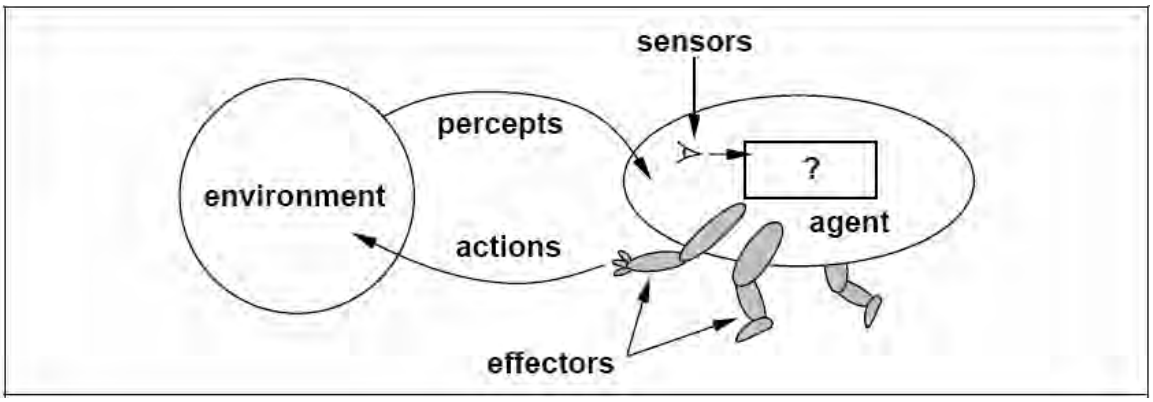

Fig 2.1 Agents interact with environments through sensors and effectors

An agent is an abstract or physically autonomous entity which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The agent should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result.

An autonomous agent is a system situated in and part of an environment, which senses its environment, and acts on it, over time, in pursuit of its own agenda. An autonomous agent senses and acts upon its environment in the service of its own agenda. Examples are to be found among humans (you and I), other animals, autonomous mobile robots, artificial life creatures, and software agents.

2.2 Taxonomy of agents

In thinking about taxonomy of agents two possible models come to mind, the biological model and the mathematical model. The biological taxonomy takes the form of a tree with "living creatures" at the root and individual species at the leaves. For example, we humans are classified as

- kingdom - animal
- phylum - chordata
- class - mammalia
- order - primate
- family - pongidae
- subfamily - hominidae
- genus - homo
- species - sapiens
- where each line represents a branching point of the tree. It is possible to create such taxonomy of autonomous agents [1, 4].

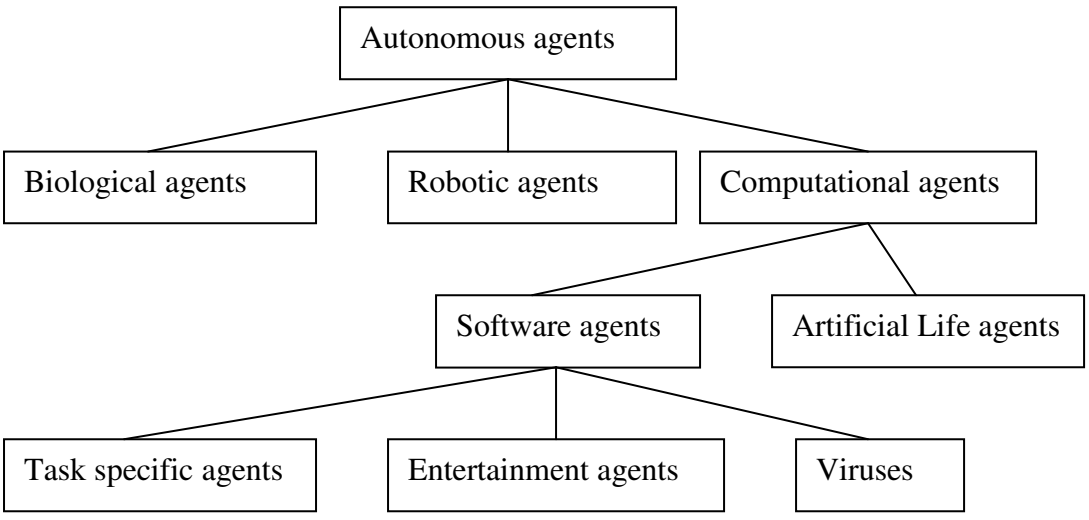

Figure 2.2. Taxonomy for autonomous agents

At the kingdom level let's classify our agents as biological, robotic, or computational, as these seem to be natural kinds. At the phylum level we can reasonably sub-classify computational into software agents and artificial life agents. At the class level, we might sub-classify software agents into task-specific agents, entertainment agents, and computer viruses.

A software agent is characterized by the following properties [1, 9, 19]

- Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

- Social ability: agents interact or communicate with other agents

- Reactivity: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it. The ability of agents to selectively sense and act.

- Pro-activity: agents do not simply act in response to their environment; they are able to exhibit goal-directed behavior by taking the initiative. An agent is capable of handling complex, high-level tasks. The decision as to how such a task is best split up into smaller sub-tasks, and in which order and way the sub-tasks are best performed, should be made by the agent itself.

- Temporal continuity: agents are continuously running processes.

- Mobility: the ability to transport itself from one machine to another, retaining its current state.

- Adaptivity: being able to learn and improve with experience.

The various definitions discussed involve a host of properties of an agent. Having settled on a much less restrictive definition of an autonomous agent, these properties may help us further classify agents in useful ways. Table 2.1 lists several of the properties mentioned above.

Property	Other Names	Meaning
reactive	sensing and acting	responds in a timely fashion to changes in the environment
autonomous		exercises control over its own actions
goal-oriented	pro-active purposeful	does not simply act in response to the environment
Temporally continuous		is a continuously running process
communicative	socially able	communicates with other agents, perhaps including people
learning	adaptive	changes its behavior based on its previous experience
mobile		able to transport itself from one machine (location) to another
flexible		actions are not scripted
character		believable "personality" and emotional state.

Table 2.1 Properties of an agent

2.3 Rational Agent

A rational agent is one that does the right thing. Obviously, this is better than doing the wrong thing, but what does it mean? The right action is the one that will cause the agent to be most successful. Therefore, some way to measure success is needed. Together with the description of the environment and the sensors and effectors of the agent, this will provide a complete specification of the task facing agent.

2.4 Performance measure

A performance measure embodies the criterion for success of an agent's behavior. When an agent is plunked down in an environment, it generates a sequence of action according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well. Obviously, there is not one fixed measure suitable for all agents. The agent could be asked for a subjective opinion of how happy it is with its own performance, but some agents would be unable to answer, and others would delude themselves. Therefore, an objective performance measure is needed, typically one imposed by the designer who is constructing the agent. As a general rule, it is better to design performance measure according to what one actually wants in the environment, rather than according to how one thinks the agent should behave.

2.5 Environment

An agent is coupled to the environment. In all cases, however, the nature of the connection between them is the same: actions are done by the agent on the environment, which in turn provides percepts to the agent. Different types of environment and how they affect the design of agents are next described.

Fully observable vs. partially observable

If an agent's sensors apparatus gives it access to the complete state of the environment at each point in time, then the environment is fully observable. An environment is

effectively fully observable if the sensors detect all aspects that are relevant to the choice of action. Fully observable environment is convenient because the agent needs not maintain any internal state to keep track of the world. An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

Deterministic vs. **stochastic**

If the next state of the environment is completely determined by the current state and the actions executed by the agents, then the environment is deterministic; otherwise, it is stochastic. In principle, an agent needs not worry about uncertainty in fully observable, deterministic environment. If the environment is partially observable, however, then it could appear to be stochastic. This is particularly true if the environment is complex, making it hard to keep track of all the unobserved aspects. Thus, it is often better to think of an environment as deterministic or stochastic *from the point of view of the agent*.

Episodic vs. **sequential**

In an episodic environment, the agent's experience is divided into atomic "episodes." Each episode consists of the agent perceiving and then performing a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. In episodic environment, the choice of action depends only on the episode itself. Many classification tasks are episodic. In sequential environments, on the other hand, the current decision could affect all future decisions. Episodic environments are much simpler than sequential environment because the agent does not need to think ahead.

Static vs. **dynamic**

If the environment can change while an agent is deliberating, then the environment is dynamic for that agent; otherwise it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action,

nor need it worry about the passage of time. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing. If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is semi dynamic.

Discrete vs. **continuous**

The discrete/continuous distinction can be applied to the *state* of the environment, to the way *time* is handled, and to the percepts and actions of the agent. If there are a limited number of distinct, clearly defined percepts and actions then the environment is discrete.

2.6 Basic kinds of agent programs

There are four basic kinds of agent programs having various methods for selecting action and that embody the principles underlying almost all intelligent system [8].

2.6.1 Simple reflex agents

The simplest kind of agent is the simple reflex agent. These agents select action on the basis of the *current* percept, ignoring the rest of the percept history.

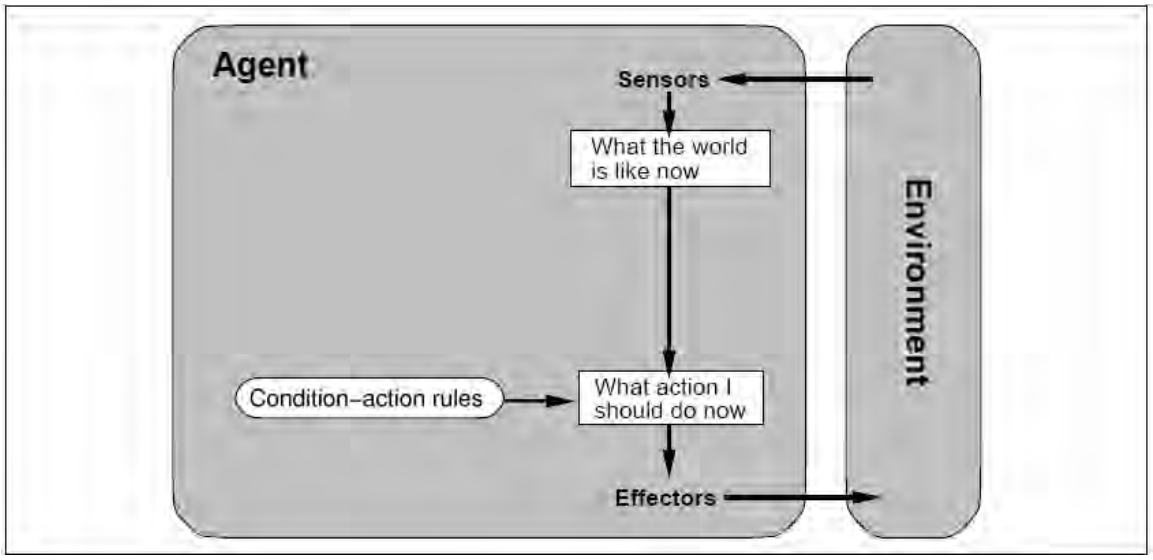

Figure 2.3 Schematic diagram of a simple reflex agent

Figure 2.3 shows the structure of simple reflex agent in schematic form, showing how the condition–action rules allow the agent to make the connection from percept to action. Simple reflex agents have the admirable property of being simple, but they turn out to be of very limited intelligence. The correct decision can be made on the basis of only the current percept, only if the environment is fully observable.

2.6.2 Model-based reflex agents

The most effective way to handle partial observability is for the agent to keep track of the part of the world it can't see now. That is, the agent should maintain some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.

Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program. First, information about how the environment evolves independently of the agent. Second, information about how the agent's own actions affect the environment. This knowledge is called a **model** of the environment. An agent that uses such a model is called a model-based agent.

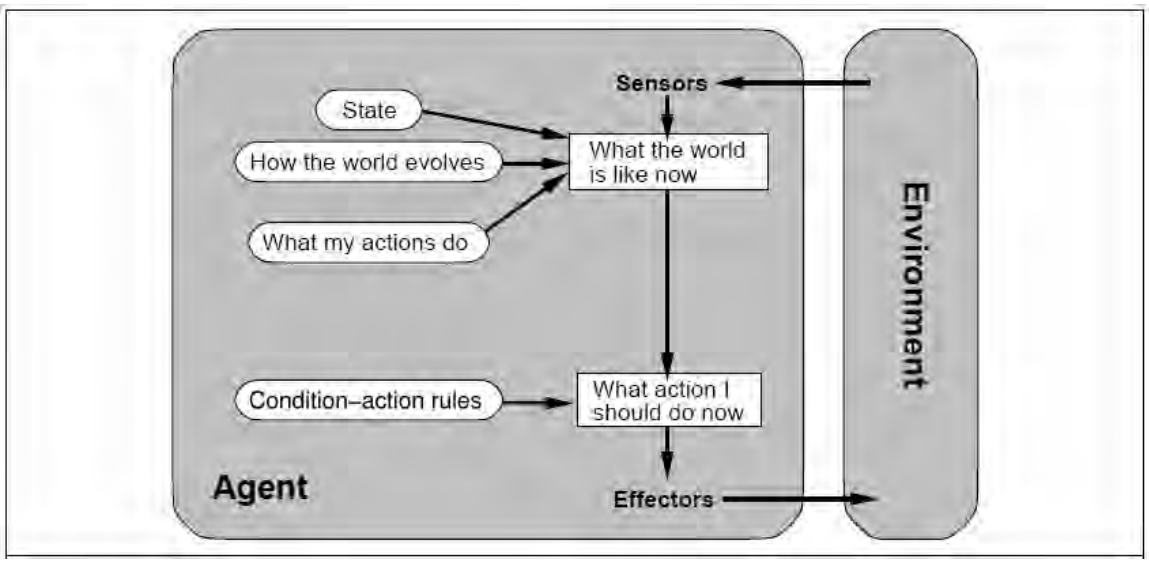

Figure 2.4 A model-based reflex agent

Figure 2.4 shows the structure of the reflex agent with internal state, showing how the current percept is combined with the old internal state to generate the updated description of the current state.

2.6.3 Goal-based agents

Knowing about the current state of the environment is not always enough to decide what to do. The agent needs some sort of **goal** information that describes situations that are desirable. The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose action that achieve the goal.

Figure 4.5 shows the goal-based agent's structure. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will lead to the achievement of its goals.

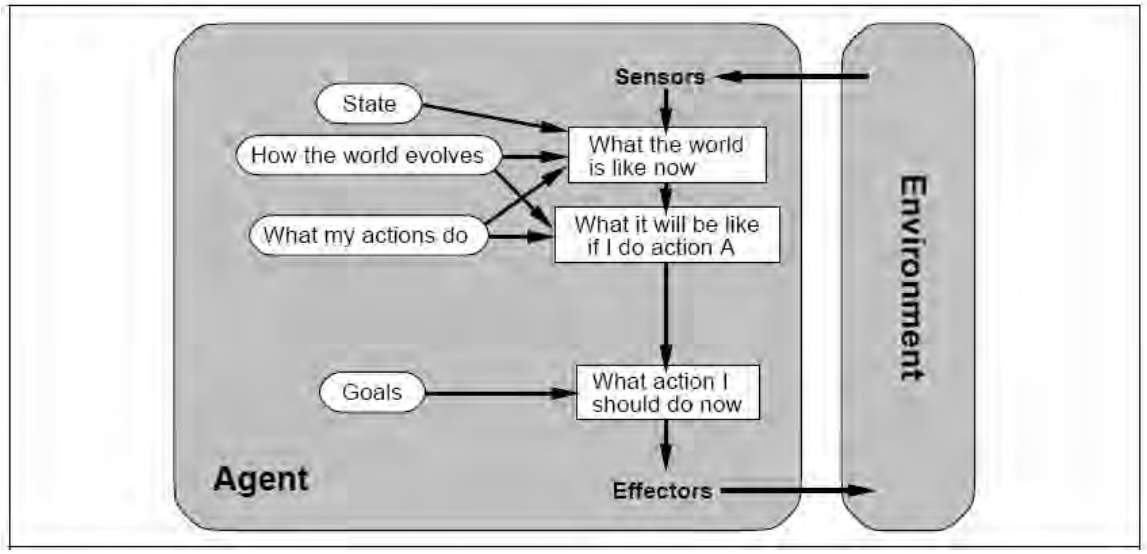

Figure 2.5 An agent with explicit goals

2.6.4 Utility-based agents

Goals alone are not really enough to generate high-quality behavior. There are many action sequences that will lead to achieve the goal, but some are quicker, safer, more

reliable, or cheaper than others. Goals just provide a crude distinction between “happy” and “unhappy” states, whereas a more general performance measure should allow a comparison of different world states (or sequences of states) according to exactly how happy they would make the agent if they could be achieved. Because “happy” does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher **utility** for the agent.

Utility is therefore a function that maps a state onto a real number, which describes the associated degree of happiness. A complete specification of the utility function allows rational decisions in two kinds of cases where goals are inadequate. First, when there are conflicting goals, only some of which can be achieved, the utility function specifies the appropriate trade-off. Second, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.

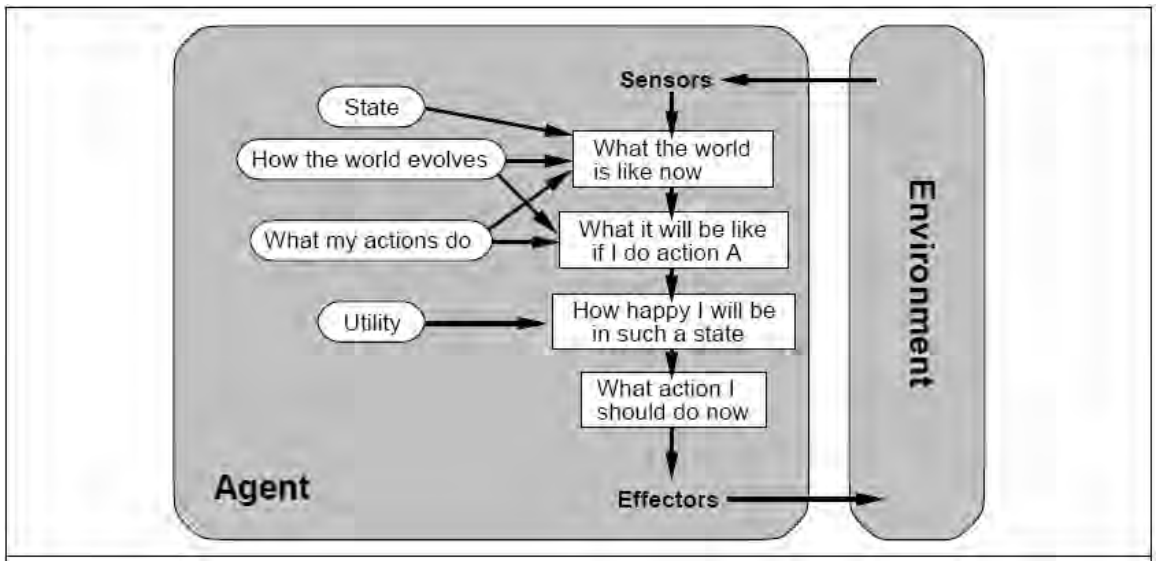

Figure 2.6 A complete utility-based agent

Figure 2.6 shows a utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it choose the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by probability of the outcome.

2.7 Design principles autonomous software agents

The proposed design principles are largely unrelated to each other and that have been derived from an analysis of many autonomous agents. They serve to constrain cognitive agent architectures and, so, will contribute to an eventual theory of cognitive agent design. [4]

Drives: Every agent must have built-in drives to provide the fundamental motivation for its actions. It's included here because autonomous agent designers tend to hardwire drives into the agent without mentioning them explicitly in the documentation and, apparently, without thinking of them explicitly. An explicit accounting of an agent's drives should help one to understand its architecture and its niche within its environment.

Attention: An agent in a complex environment who has several senses may well need some attention mechanism to help it focus on relevant input. Attending to all input may well be computationally too expensive.

Internal models: Model the environment only when such models are needed. When possible depend on frequent sampling of the environment instead. Modeling the environment is both difficult and computationally expensive. Frequent sampling is typically cheaper and more effective, when it will work at all.

Coordination: In multi-agent systems, coordination should be achieved without the high cost of communication.

Knowledge: Build as much needed knowledge as possible into the lower level of an autonomous agent's architecture. Every agent requires knowledge of itself and of its environment in order to act so as to satisfy its needs. Some of this knowledge can be learned.

Curiosity: If an autonomous agent is to learn in an unsupervised way, some sort of more or less random behavior must be built in. Curiosity serves this function in humans, and apparently in mammals. Autonomous agents typically learn mostly by internal reinforcement.

Routines: Most cognitive agents will need some means of transforming frequently used sequences of actions into something reactive so that they run faster.

An autonomous agent is situated within some environment, perhaps our real world, or an artificial environment within a computing system, or within an operating system, a database, or a network. The agent actively senses its environment, and acts upon it so as to effect what it may sense next. This interactive sensing and acting aims to achieve goals that can be expected to satisfy drives. These drives may have evolved, as in us or other animals, or may have been designed and built in as in robots, artificial life agents, or software agents.

An autonomous agent senses and acts upon its environment in the service of its own agenda. An autonomous agent with human-like cognitive capabilities is called a cognitive agent. Such autonomous software agents, when equipped with cognitive (interpreted broadly) features chosen from among multiple senses, perception, short and long term memory, attention, planning, reasoning, problem solving, learning, emotions, moods, attitudes, multiple drives, etc., are called cognitive agents.

“Conscious “software agents are autonomous agents that range in functionality, from academic seminar organizers to navy detailers responsible for naval personnel placement. They sense this environment and act on it over time, in pursuit of their own agenda, in such a way as to possibly influence what they sense at a later time. Autonomous agents are coupled to their environment. In the most generic sense they include numerous animals such as humans along with software agents such as computer viruses and artificial life agents. Many of these agents are also cognitive agents which are

autonomous agent equipped with cognitive features such as concept formation, consciousness, emotions, long and short-term memory, meta-cognition, and perception.

2.8 High-level architectures for cognitive agents

Several high-level architectures for cognitive agents have been proposed. The plan for this architecture is produced by a sequence of refinements beginning with a very simple model [1, 4].

Partition a computing system into input, processing and output for the beginning as shown in Figure 2.7 a.

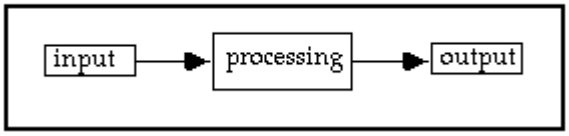

Figure 2.7 a Computing system model

The corresponding diagram for an autonomous agent might look as follows.

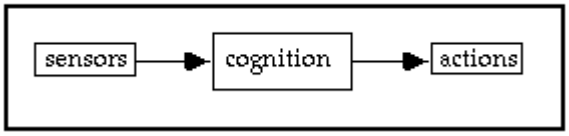

Figure 2.7 b Cognitive agent model

A refinement of Figure 2.7 b, while sensors and actions are explicitly present, drives and action selection are not.

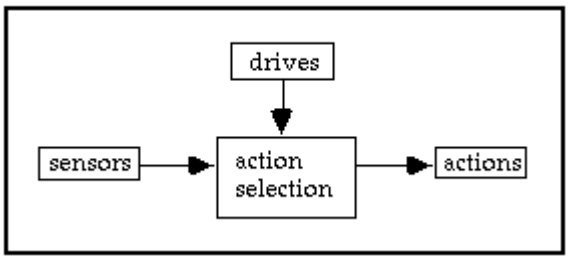

Figure 2.3 c Cognitive agent model

Though drives are explicitly represented in Figure 2.7 c, they may well appear only implicitly in the causal mechanisms of a particular agent. The diagram in Figure 2.7 c is explicitly guided by action selection and drives.

In Figure 2.7 d split perception off from action selection. In further refinements of the architecture, action selection must be interpreted less and less broadly.

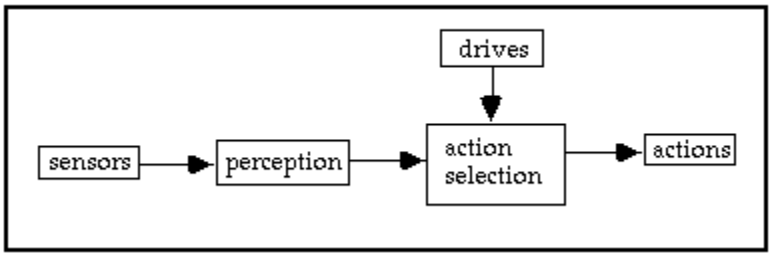

Figure 2.7 d Cognitive agent model

Figure 2.7 e leads to another refinement with the addition of memory. Include both long-term memory and short-term memory (workspace). Though only one memory and one workspace is shown in Figure 2.7 e, multiple specialized memories and workspaces may be expected in the architectures of complex autonomous agents.

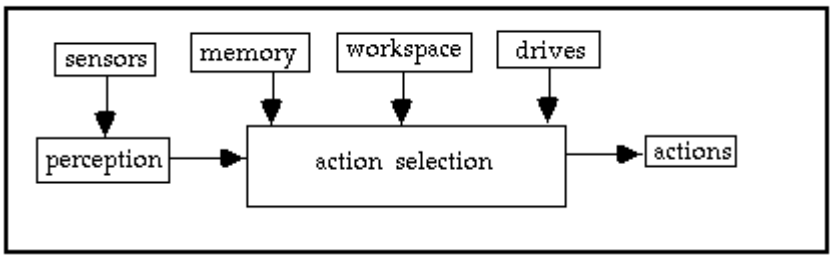

Figure 2.7 e Cognitive agent model

At this point the action selection paradigm of mind gives only general guidance: employ multiple, independent modules and allow for disparate mechanisms. Thus we turn to design principles. The Attention Principle points to an attention mechanism or relevance filter which is shown in Figure 2.7f. Attention will depend on context and, ultimately on the strength and urgency of drives.

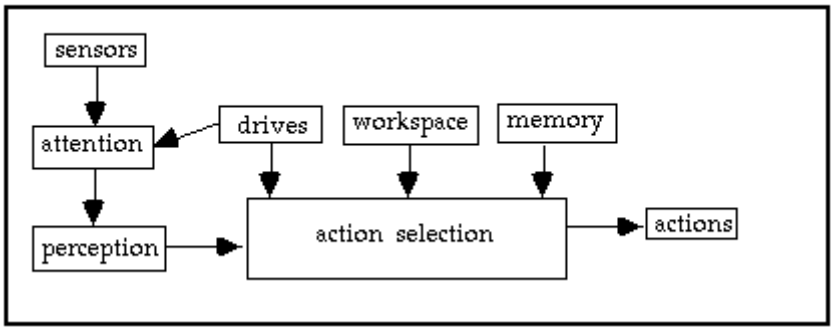

Figure 2.7 f Cognitive agent model

Figure 2.7 g points the distinction between reactive and deliberative action selection. Deliberative actions are selected with the help of internal models, planners, schedulers, etc. These models use internal representations in the strict sense of the word, that is, they are consulted for their content. Internal states that play a purely causal role without such consultations are not representations in this sense. Reactive actions are exemplified by reflexes and routines. They are arrived at without such consultation.

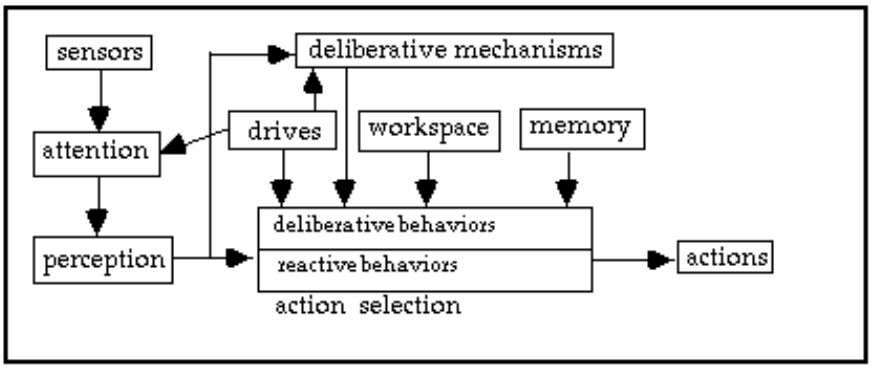

Figure 2.7 g Cognitive agent model

The Coordination Principle warns us against unnecessary communication. Still, for a cognitive agent in a society of other such, communication may well be needed. Compositions of outgoing messages are brought about by deliberative behaviors. Independent modules for understanding messages and for composing messages must be part of general cognitive agent architecture.

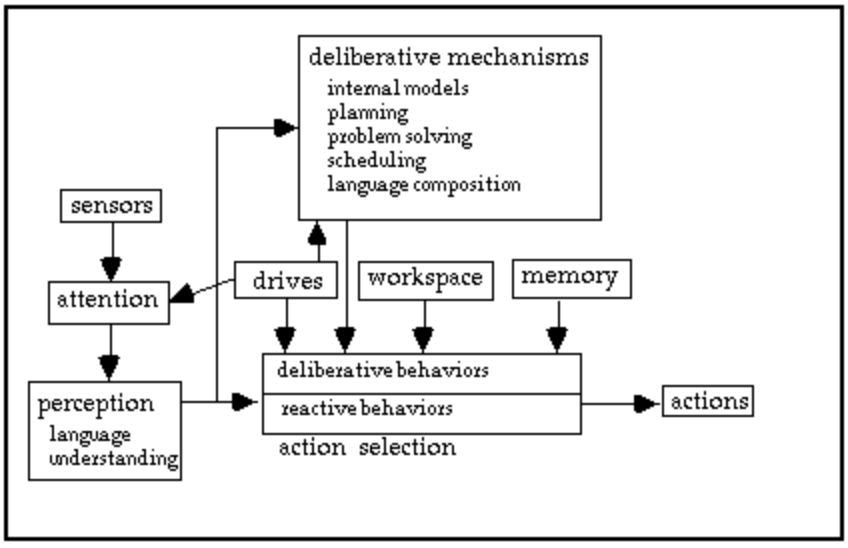

Figure 2.7 h Cognitive agent model

The Knowledge Principle brings up two issues: building knowledge into the reactive behaviors, and learning. By definition, knowledge is built into reactive behaviors casually

through their mechanisms, rather than declaratively by means of representations. This doesn't show up in the diagrams.

The other issue brought up by the Knowledge Principle is learning, which is critical to many autonomous agents coping with complex, dynamic environments, and must be included in general cognitive agent architecture. Learning, itself, is quite complex. It is possible to categorizes learning according to the sophistication of the behavior to be learned as follows habituation-sensitization, signal learning, stimulus-response learning, chaining, concurrent discriminations, class concepts: absolute and relative, relational concepts I: conjunctive, disjunctive, conditional concepts, relational concepts II: biconditional concepts. One might also classify learning according to the method used such as: memory based reasoning, reinforcement learning, supervised learning, and learning by advice from other agents.

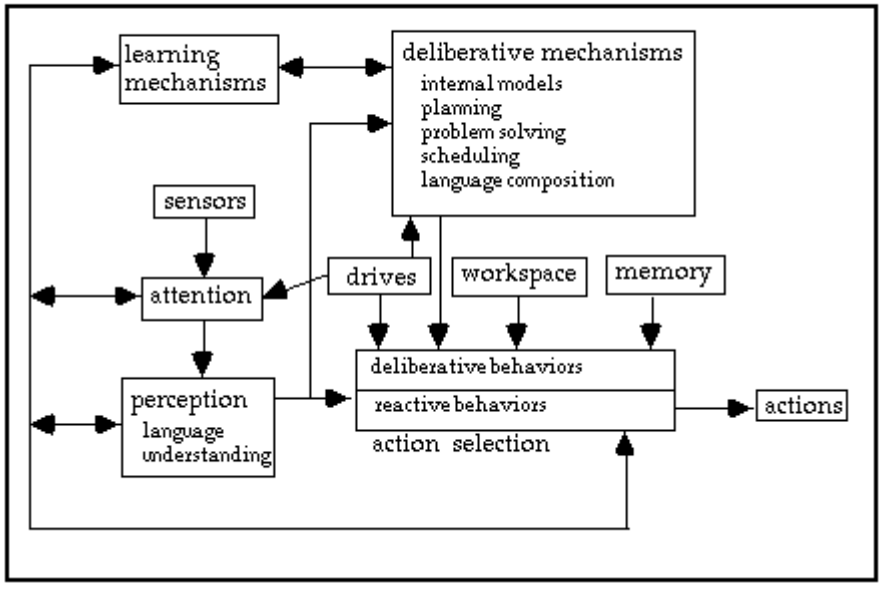

Figure 2.7 i Cognitive agent model

Learning mechanisms should also connect to drives and to memory, essentially to everything. And, there are other connections that need to be included, or to run in both directions.

2.9 Learning agents

Learning allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow. A learning agent can be divided into four conceptual components, as shown in Figure 2.8. The most important distinction is between the learning element, which is responsible for making improvements, and performance element, which is responsible for selecting external actions. The performance element takes in percepts and decides on actions. The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.

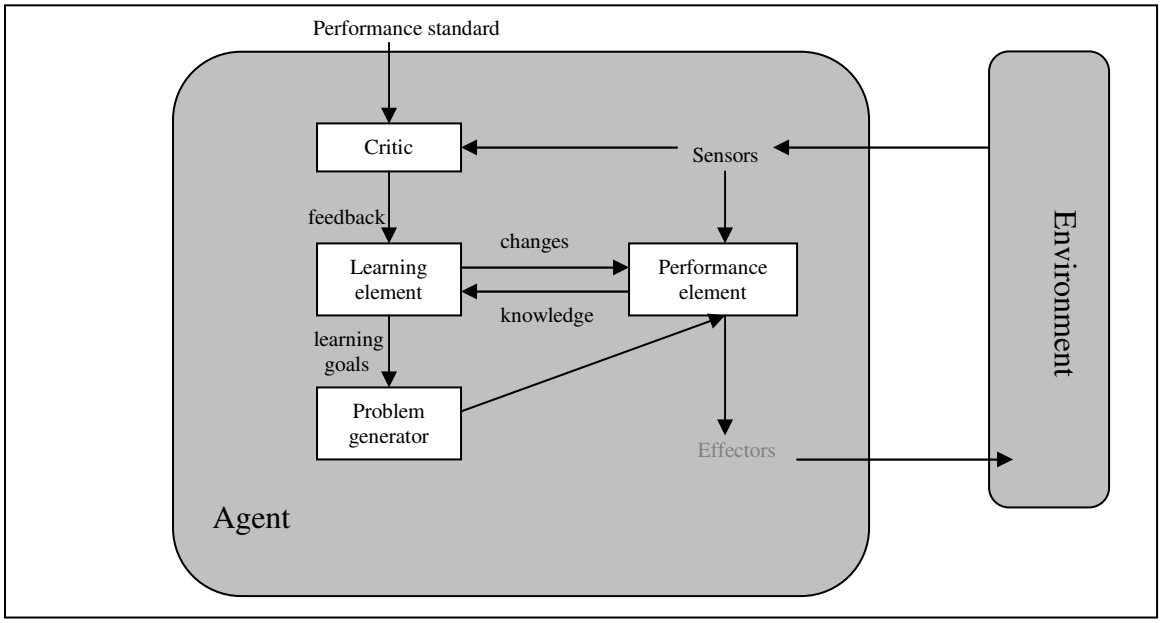

Figure 2.8 A general model of learning agents

The critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success. The last component of the learning agent is the problem generator. It is responsible for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore

a little, and do some perhaps suboptimal actions in the short run, it might discover much better action for the long run. The problem generator's job is to suggest these exploratory actions.

3 Multi Agent System (MAS)

3.1 Basics of a multi-agent system

A multi-agent system [3, 5] is a loosely coupled network of problem-solver entities that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity. A multi-agent system is a system composed of a population of autonomous agents, which cooperate with each other to reach common objectives, while simultaneously each agent pursues individual objectives. More recently, the term multi-agent system has been given a more general meaning, and it is now used for all types of systems composed of multiple autonomous components showing the following characteristics:

- each agent has incomplete capabilities to solve a problem and, thus has a limited viewpoint
- there is no global system control
- data is decentralized
- computation is asynchronous

One of the current factors fostering MAS development is the increasing popularity of the Internet, which provides the basis for an open environment where agents interact with each other to reach their individual or shared goals. To interact in such an environment, agents need to overcome two problems: they must be able to find each other (since agents might appear, disappear, or move at any time), and they must be able to interact.

In order for MAS to solve common problems coherently, the agents must communicate among themselves, coordinate their activities. Coordination and communication are central to MAS, for without it, any benefits of interaction vanish and the group of agents quickly degenerates into a collection of individuals with a chaotic behavior.

There are several reasons why multiple agents need to be coordinated.

- Prevent chaos: no agent possesses a global view of the entire agency to which it belongs, as this is simply not feasible in any community of reasonable complexity. Consequently, agents have only local views, goals and knowledge that may interfere with rather than support other agents' actions. Coordination is vital to prevent chaos during conflicts and to meet global constraints. Agents performing network management may have to respond to certain failures within seconds and others within hours. Coordinating agents' behavior is therefore essential to meet such a goal.
- Agents in MAS possess different capabilities and expertise. Therefore, agents need to be coordinated in just the same way that different medical specialists, including anesthetists, surgeons, ambulance personnel, nurses, etc., need to coordinate their capabilities to treat a patient.
- Agent's actions are frequently interdependent and hence an agent may need to wait for another agent to complete its task before executing its own. Such interdependent activities need to be coordinated.

3.2 Finding Agents

There is a need for mechanisms for advertising, finding, fusing, using, presenting, managing, and updating agent services and information. To address these issues, the notion of middle agents was proposed [10]. Middle agents are entities to which other agents advertise their capabilities, and which are neither requesters nor providers from the standpoint of the transaction under consideration. The advantage of middle agents is that they allow MAS to operate robustly when confronted with agent appearance, disappearance, and mobility.

There are several types of agents that fall under the definition of middle agents. Note that these types of agents, which are described below, are defined so vaguely that sometimes it is difficult to make a clear differentiation between them.

- **Facilitators**: agents to which other agents surrender their autonomy in exchange for the facilitator's services. Facilitators can coordinate agents' activities and can satisfy request on behalf of their subordinated agents.
- **Mediators**: agents that exploit encoded knowledge to create services for a higher level of applications.
- **Brokers**: agents that receive requests and perform actions using services from other agents in conjunction with their own resources.
- **Matchmakers** and **yellow pages**: agents that assist service requesters to find service provider agents based on advertised capabilities.
- **Blackboards**: repository agents that receive and hold requests for other agents to process.

3.3 Agent Interaction

Interaction is one of the most important features of an agent. In other words, agents recurrently interact to share information and to perform tasks to achieve their goals. The following elements are required to achieve multi-agent interaction [3, 11]:

- A common agent communication language and protocol
- A common format for the content of communication
- A shared ontology (specification schemes for describing concepts and their relationships in a domain of discourse)

3.3.1 Agent Communication Language

There are two main approaches to designing an agent communication language. The first approach is procedural, where communication is based on executable content. This could be accomplished using programming languages. The second approach is declarative, where communication is based on declarative statements, such as definitions, assumptions, and the like. Because of the limitations on procedural approaches (e.g., executable content is difficult to control, coordinate, and merge), declarative languages have been preferred for the design of agent communication languages. Most declarative

language implementations are based on illocutionary acts, such as requesting or commanding; such actions are commonly called performatives. One of the more popular declarative agent languages is KQML.

KQML

KQML, which is an acronym for **Knowledge Query and Manipulation Language**, was conceived both as a message format and a message handling protocol to support run-time knowledge sharing among agents [17, 18]. This language can be thought of as consisting of three layers: a communication layer (which describes low level communication parameters, such as sender, recipient, and communication identifiers); a message layer (which contains a performative and indicates the protocol of interpretation); and a content layer (which contains information pertaining to the performative submitted).

```
(register
      :sender       agentA
      :receiver     agentB
      :reply-with   message2
      :language     common_language
      :ontology     common_ontology
      :content      ``(ServiceProvision
Manufacturing:TaskDecomposition)''
)
```

Example of a KQML message.

The format of a KQML message is shown in the example starts with the word ``register", which is the action (performative) intended for the message. The remainder of the message contains keywords needed for the message and communication layers. Keywords used in KQML messages are defined as follows:

- **sender**: agent sending the message.
- **receiver**: agent receiving the message.
- **from**: original sender; used when a message is sent using intermediary agents.

- **to**: final recipient; used when a message is sent using intermediary agents.
- **in-reply-to**: identifier of the message that triggered this message submission.
- **reply-with**: identifier to be used by a message replying to this message.
- **language**: language for interpreting the information in the content field of this message.
- **ontology**: identifies the ontology to interpret the information in the content field of this message.
- **content**: context-specific information describing the specifics of this message.

3.4 MAS Architectures Standardization

Several standardization of multi-agent technology are available, such as those of the Object Manager Group (OMG), the Foundation for Physical Agents (FIPA), the Knowledge-able Agent-oriented System (KAoS) group, and the General Magic group are briefly described below.

OMG's Model

The OMG group proposes a reference model as a guideline for the development of agent technologies. This model outlines the characteristics of an agent environment composed of agents (i.e., components) and agencies (i.e., places) as entities that collaborate using general patterns and policies of interaction. Under this model, agents are characterized by their capabilities (e.g., inferencing, planning, and so on), type of interactions (e.g., synchronous, asynchronous), and mobility (e.g., static, movable with or without state). Agencies, on the other hand, support concurrent agent execution, security and agent mobility, among others.

FIPA's Model

The Foundation for Intelligent Physical Agents (FIPA) is a multi-disciplinary group pursuing the standardization of agent technology. This organization has made available a series of specifications to direct the development of multi-agent systems. Of particular importance are their Agent Management and Agent Communication Language

specifications. FIPA's approach to MAS development is based on a “minimal framework for the management of agents in an open environment.” This framework is described using a reference model (which specifies the normative environment within which agents exist and operate), and an agent platform (which specifies an infrastructure for the deployment and interaction of agents).

KAoS' Model

Another important standardization effort is pursued by researchers of the Knowledge-able Agent-oriented System architecture. This system, which is also known as KAoS, is described as “an open distributed architecture for software agents.” The KAoS architecture describes agent implementations (starting from the notion of a simple generic agent, to role-oriented agents such as mediators and matchmakers), and elaborates on the interactive dynamics of agent-to-agent messaging communication by using conversation policies.

General Magic's Model

General Magic is a commercial endeavor researching mobile agent technology for electronic commerce. Conceptually, this technology models a MAS as an electronic marketplace that lets providers and consumers of goods and services find one another and transact business. This marketplace is modeled as a network of computers supporting a collection of places that offer services to mobile agents. Mobile agents, which are entities that reside in one particular place at a time, have the following capabilities:

- they can **travel**, to move from one place to another
- they can **meet** other agents, which allows them to call one another agent's procedures
- they can create **connections**, to allow an agent to communicate with another agent in a different place

- they have **authority**, which indicates the real-world individual or organization that the agent represents
- they have **permits** to indicate the capabilities of agents

3.5 Abilities of Multi-agent systems

The abilities of multi-agent systems are listed below:

- To solve problems that are too large for centralized agent to solve because of resource limitations or the sheer risk of having one centralized system that could be a performance bottleneck or could fail at critical times.
- To allow for the interconnection and interoperation of multiple existing legacy systems. To keep pace with changing business needs, legacy systems must periodically be updated. Completely rewriting such software tends to be prohibitively expensive and is often simply impossible. Therefore, in the short to medium term, the only way to such legacy systems can remain useful is to incorporate them into a wider cooperating agent community in which they can be exploited by other pieces of software. Incorporating legacy systems into an agent society can be done, for example, by building and agent wrapper around the software to enable it to interoperate with other systems.
- To provide solution to problems that can naturally be regarded as a society of autonomous interacting components-agents.
- To provide solutions that efficiently use information sources that is spatially distributed.
- To provide solution in situation where expertise is distributed.
- To enhance performance.

3.6 Multi-agent systems terminology

Agent Architectures analyze agents as independent reactive/proactive entities. Agent architectures conceptualize agents as being made of perception, action, and reasoning

components. The perception component feeds the reasoning component, which governs the agents' actions, including what to perceive next.

Agent System Architectures analyze agents as interacting service provider/consumer entities. System architectures facilitate agent operations and interactions under environmental constraints, and allow them to take advantage of available services and facilities.

Agent Frameworks are programming tools for constructing agents.

Agent Infrastructures provide the regulations that agents follow to communicate and to understand each other, thereby enabling knowledge sharing. Agent Infrastructures deal with the following aspects:

- **Ontologies**: allow agents to agree about the meaning of concepts.
- **Communication Protocols**: describe languages for agent communication.
- **Communication Infrastructures**: specify channels for agent communication.
- **Interaction Protocols**: describe conventions for agent interactions.

3.7 Objects versus Agents

Agents and objects share many characteristics; this sometimes makes it hard to differentiate between them. For example, agent-oriented programming (AOP) could be considered a specialization of the object-oriented programming (OOP) paradigm. OOP views systems as consisting of objects communicating with one another to perform internal computations, whereas AOP specializes this view to have agents (instead of objects), whose internal computations are based on beliefs, capabilities, and choices, that communicate with each other using messages adopted from speech-act theory.
Although this view allows one to appreciate the similarities between agent and objects, their differences are less obvious. There are three main differences between agents and objects that have been identified [3]:

The first is in the degree to which agents and objects are autonomous. We thus do not think of agents as invoking methods upon one-another, but rather as requesting actions to be performed. In the object-oriented case, the decision lies with the object that invokes the method. In the agent case, the decision lies with the agent that receives the request.
The second important distinction is with respect to the notion of flexible (reactive, pro-active, social) autonomous behavior.
The third important distinction is that agents are each considered to have their own thread of control but in the standard object model, there is a single thread of control in the system.

3.8 Why are Agents important?

Software developers and system designers use high-level abstractions in building complex software for one reason of managing complexity. An abstraction focuses on the important and essential properties of a problem and hides the incidental components of that problem. Agents provide a new way of managing complexity because they provide a new way of describing a complex system or process. Using agents, it is easy to define a system in terms of agent-mediated processes.

Consider, for example, the system design issues involved in building a loan approval application that ties together branch banks, the main bank, loan underwriting companies, and credit reporting companies, and automates much of the loan approval process. Building this system using current technology is a complex and difficult task because the system decomposition forces the developer to deal with relatively low-level concepts (e.g. loan applications, account balances, credit ratings) when defining the overall system architecture. In addition, significant design time must be dedicated to defining the communications protocol and interfaces that will allow the bank to exchange data with the credit reporting agencies and loan underwriters. In an agent-oriented system design, the system solution might include a customer service agent, a loan application analysis agent, an underwriter agent, etc. The focus is placed on the behavior of each of these agents and communication between agents. The problem is made much easier because

the level of abstraction is much higher and the programming problem becomes one of specifying agent behavior.

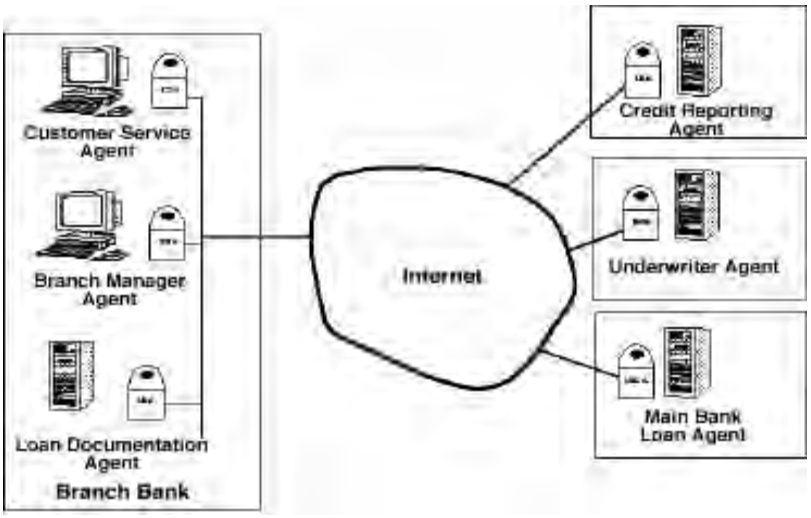

Figure 2.4 Multiple software agents for enterprise application

An agent-based solution is useful and attractive because the various agents used in the solution inherently know how to do many things. For example, agents know how to communicate with other agents. The system developer no longer has to design communication protocols and message formats. The agent provides this capability as part of the basic agent mechanism. Agents have the inherent capability to build models of their environment, monitor the state of that environment, reason and make decisions based on that state. All the software developer needs to do is simply specify what the agents are to do in any given situation.

4 Genetic Algorithms

4.1 Basics of Genetic Algorithms

Genetic Algorithms (GAs) are adaptive heuristic search algorithm inspired from natural evolution. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an

intelligent exploitation of a random search within a defined search space to solve a problem.

Genetic algorithms are optimization techniques based on selection and recombination of promising solutions. The collection of candidate solutions is called populations of genetic algorithms whereas candidate solutions are sometimes named as individuals, chromosomes, etc. Each individual is an encoded representation of variables of the problems at hand. Each component (variable) in an individual is termed as gene. Sometimes the components (genes) are independent of one another and sometimes they are correlated. The values that can be taken by a gene are known as alleles. But always a communication and information exchange among individuals in the population is maintained through selection operator and variation operator.

In genetic algorithms, solutions of a problem to be solved are represented in some way and stored in the computers memory and then modified much the same as generation of organisms evolving under natural selection.

As shown in Figure 4.1, the first step to apply genetic algorithms is to create an initial population of random individuals. The individuals must in some way represent a candidate solution to the problem. Variations among the individuals lead to that some of them are more fit to the environment than others. Fitness function is used to guide the selection of new candidate solutions that might appear in the next population (generation). Candidates are then reproducing with some probability to generate a new generation. This creates a new population with better average fitness value. This cycle will go on until one individual that is fit enough is found in a population.

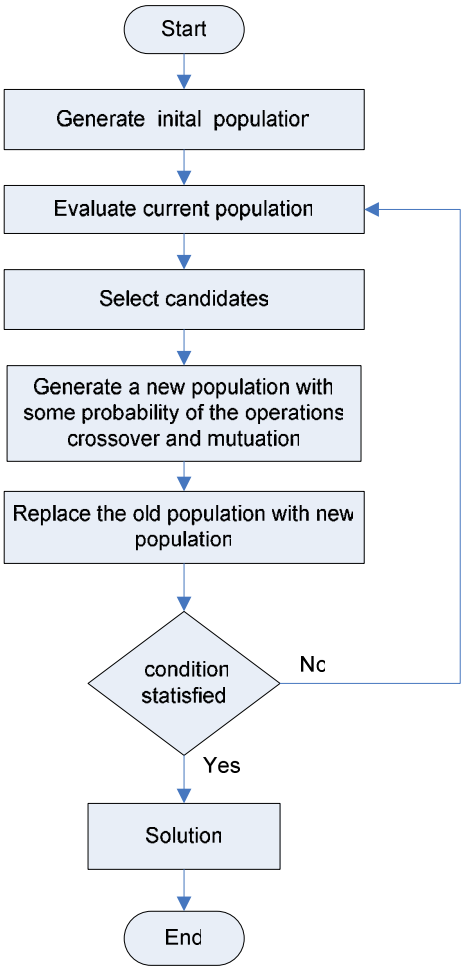

Figure 4.1 Flow chart for genetic algorithm

After the initial population is created the genetic algorithms repeats the following steps until the solution is found:

1. Evaluate current population.
2. Select candidates.
3. Create a new population.

4.2 Fitness function

The function that provides the mechanism for evaluation of candidate solutions is called the fitness or evaluation function. Its purpose is to measure the performance of

individuals and its range of values varies form problem to problem. The selection mechanism makes more fit individuals live on and reproduce.

4.3 Individual representation

Fundamental to genetic algorithms is the encoding of the problem variables. The encoding mechanism depends on the nature of the problem. The variables have to be encoded onto individuals that uniquely encode all possible solutions to the problem.

4.4 Selection operators

The criteria used for the selection of individuals, who pass their genetic information from one generation to the next, are one of the key aspects that determine the success or failure of a genetic algorithm. Genetic algorithm follows different selection criteria, which are closely related to selection among natural organisms.

Fitness-proportionate selection

Stochastic selection schemes is used to answer the following question: Which individuals are to be reproduced such that promising regions of the search space are discovered and further exploited? The individual's probability of survival is computed by comparing an individual finesses value to the sum population fitness value. The probability of an individual being reproduced into the next generation is proportional to its fitness value.

Rank-based selection (Truncation selection)

The individuals are sorted according to their fitness. Selection is not determined by the actual fitness value but by an individual's position within a fitness rank. The worst individual will have rank 1, second worst 2 etc. and the best will have fitness N (number of chromosomes in population). Only few best individuals are selected to be parents for the next generation on the bases of this fitness rank. Rank-based selection

is used to avoid undesirable convergence effects, where differences between fitness values are normalized.

4.5 Variation operators

In the classic genetic algorithm approach there are two variation operators that are used to generate the offspring for the following generation.

Crossover operator

Crossover is a partial exchange of genetic material between solution candidates. When two parents are selected a crossover point is randomly chosen. Let us say that we have a binary representation of two individuals MOM= 11110000 and DAD=00001111 and the crossover point is 4. This means that the first offspring ,KID1= 11111111 gets the first 4 bits from parent MOM and the next 4 bits from parent DAD, KID2= 00000000 gets its first 4 bits from DAD and the next 4 from MOM. It is also possible to have multiple crossover points.

Mutation operator

Mutation is just random bit flips which are applied to some bit in an individual with small probability. This means, KID3 = 11100000 is mutated version of Mom at point 4.

In real world, those who do not fit in the environment die and those who fit live on to reproduce. When the organisms reproduce, the offspring inherit the parents' fitness. Sometimes mutations on the offspring occur and might lead to an even better fit new organism. In this way, the evolution goes on and it seems to work well in the nature. This way of successful evolution is interpreted in computer science as genetic algorithms. The behavior of the genetic algorithms depends on the choice of genetic operators: selection, crossover, mutation, probabilities of crossover and mutation, population size, rate of generational reproduction, number of generations etc.

4.6 Genetic algorithms versus standard search techniques

Genetic algorithms differ from standard search techniques in, essentially, the following ways:

1. Genetic algorithms require encoding of candidate solutions of the current problem into unique individuals.
2. Genetic algorithms search simultaneously from a population of candidate solutions, climbing many peaks in parallel therefore reducing the probability of a false pick.
3. Genetic algorithms use probabilistic rules rather than deterministic rules to guide their search.
4. Genetic algorithms use a fitness function to evaluate each individual in the population.

5. System development and Implementation

Agent architecture analyzes agents and independent reactive and proactive entities. It conceptualizes agents as being made of perception, action, and reasoning components where the perception component feeds the reasoning. Based on the definition of agent architecture, light weight agent architecture is developed to solve a problem. The architecture is developed by considering a specific problem as test scenario.

5.1 What is done?

The work is divided into five different phases where each of the phases has contribution for the next phase and for the objective of the work that is the development of agent architecture for solving a problem.

Phase 1: An expert system is developed to solve a problem. All the possible conditions to solve the problem are hard coded in the system.

Phase 2: The problem is coded on chromosomes and the system is given a certain quality of measure. The system solves the problem using the quality of measures available.

Phase 3: A multi-agent system is developed to solve a problem. According to the definition of multi-agent system, each of the agents has incomplete capacity to solve the problem. These agents are autonomous, reactive, proactive and adaptive. These agents also communicate and coordinate with each other so as to give solutions to the problem.

Phase 4, 5: Maintaining the concepts mentioned in phase 3, only the way agents cooperate and communicate is enhanced. This enhancement helps the agents to interact without sharing a global system.

5.2 Test scenario

N-queens problem [6, 7] is selected as a test scenario for each phase of the system development.

What is n-queens problem?

The problem is to find possible ways of placing *n* non-attacking queens on n by n chessboard. A queen attacks all queens that are in the same row, column, and diagonal as that of the queen. In simpler words, the objective of the *n-queens problem* is to place *n* queens on n by n chessboard in such a way the no two queens are on the same row, column and diagonal.

The *n queen problem* is originally derived from *the eight queens problem*. The original *eight queens problem* is to place eight queens on a chessboard so that no queen would attack any other queen. An alternate way of expressing the problem is to place eight things on an eight by eight grid such that none of them share a common row, column or diagonal. There are 92 solutions for *eight queens problem*, of these 12 are unique solutions and the remaining solutions can be transformed into one of these 12 unique patterns using rotations and reflections.

The number of solutions for n=1, 2…15, is 1, 0, 0, 2, 10, 4, 40, 92, 352, 724, 2680, 14200, 73712, 365596, 2279184.

Application of *n queens problem*

The *n queens problem* is really a puzzle but, there are some practical applications such as parallel memory storage schemes, Very Large Scale Integration (VLSI) testing, traffic control, and deadlock prevention [7, 30, 31]. The problem is also illustrative of more practical problems whose solutions are permutations such as traveling sales person problem: find the shortest route on a map that visits each city exactly once and return to the starting city. Furthermore, the solution technique,

backtracking is very important, and is best learned on simple examples such the *n queens problem*.

5.3 Phase 1

The objective is to develop a system like an expert system that can solve the problem. All the possible conditions are hard coded in the system by the domain expert. An expert system is developed that finds all possible solution by backtracking rules from domain expert.

Backtracking is a scheme for solving a series of sub-problems each of which may have multiple possible solutions and where the solution chosen for one sub-problem may affect the possible solutions of later sub-problems. To solve the overall problem, we find a solution to the first sub-problem and then attempt to recursively solve the other sub-problems based on this first solution. If we cannot, or we want all possible solutions, we backtrack and try the next possible solution to the first sub-problem and so on. Backtracking terminates when there are no more solutions to the first sub-problem.

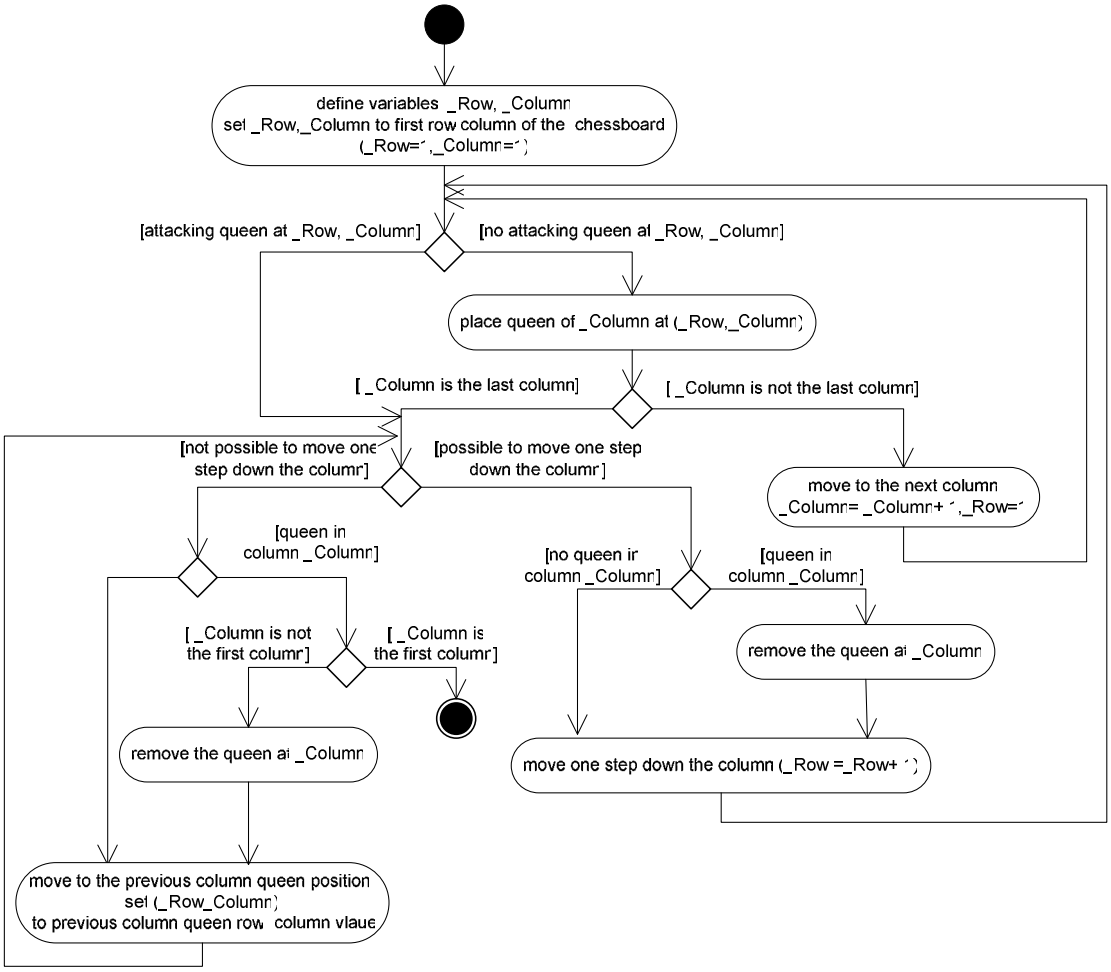

Figure 5.1 Activity diagram for phase 1

The system is modeled as n by n chessboard, n queens, and n queens' solver. Initially no queen is placed on the chessboard. The n by n chessboard is modeled as a one dimensional array that stores only reference to the queen. Each queen has an identity, row and column. Figure 5.2 shows the representation for each of the abstraction. The identity of the queen can be the same as the column value where a queen is located. So, each queen can move down its column which avoids the queens from attacking other queen column wise.

The chessboard checks if two queens are attacking row wise and column wise. Queens with the same row value attack each other row wise. Two queens attack each

other diagonally if the absolute value of the difference of rows and columns of queens is equal. The n queens' solver is designed to solve the problem using the hard coded rules from the domain expert. Figure 5.1 shows the backtracking algorithm in solving the *N-queens problem*. It is possible to find all the solutions by using the algorithm.

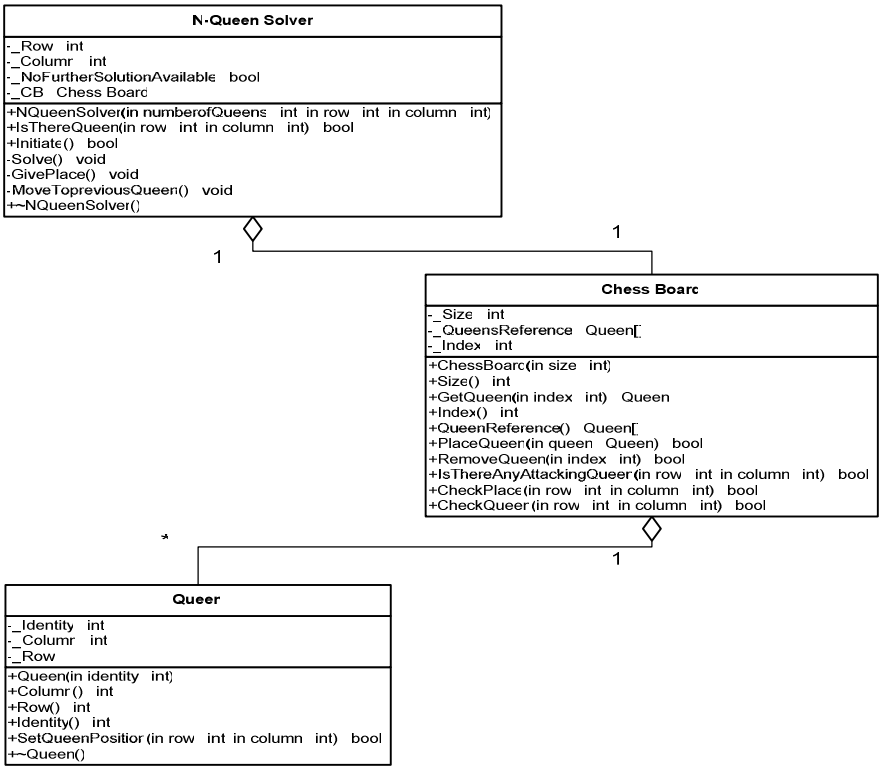

Figure 5.2 Class diagram for phase 1

5.4 Phase 2

The objective of phase 2 is to incorporate some level of autonomy to the system in solving the problem. The problem is coded on a chromosome like data structure and a certain measure of quality is defined. The system looks for an optimal solution depending on the given measure of quality. Genetic algorithm is adopted to find the optimal solution for this phase so as to incorporate some level of independence in the system.

The idea of genetic algorithm comes from the nature's way of evolving successful organisms. It contains data structures which look like chromosomes. The chromosomes can be described as strings of many thousands of smaller units called genes. The values that can be taken by a gene are known as alleles. The details of genetic algorithm are explained in chapter 4.

The first step, when applying a genetic algorithm is to create an initial population of random individuals. The individuals must in some way represent a candidate solution to the problem. Variations among the individuals lead that some of them are fitter to the environment than others. Fitness function is used to guide the selection of new candidate solutions that might appear in the next generation. Candidates are then reproducing with some probability to generate a new generation. This cycle will go on until an individual that is fit enough is found in the population.

How to apply genetic algorithm to solve the problem?

Representation

The representation of an individual (chromosome) for the *n-queens problem* is a list of unique integers [1 to n] that encodes a solution. Each gene is a single queen which is represented by a single integer. The number of genes found in a chromosome depends on number of queens available. Figure 5.3 shows an example for the representation for the *4 queens problem* which is [1, 3, 2, 4]. The position within the list denotes the chessboard column and the number stored at that position is chessboard row. An individual (chromosome) modeled as n number of queens together with their position. Clearly this type of representation of an individual avoids attacking queens' column wise since each of the integers in the list represents a queen in the separate column.

Q

Q

Q

Q

[1, 3, 2, 4]

Figure 5.3 An example of representation of n-queen problem for n=4

Creation of first generation

An individual is created by generating random unique integers in [1to n] number of population times. Attacking of queens row wise is avoided by generating unique value of each integer in the list since the value stored in the list denote the row in the chessboard. An individual created in this fashion is called a valid solution. Valid solution means an individual having queens that do not attack each other either row wise or column wise.

Operation

To use the classic variation operators directly would cause invalid solutions in many cases. Applying the classic crossover operator usually results in an invalid new solution. For example, if we have two individuals with [1,3,2,4] and [2,4,1,3] and if we make a crossover at gene two, the offspring will be [1,3,1,3] and [2,4,2,4] , which are invalid solutions.

The mutation operator must also be modified; it can not just change one number in the list to another. The modified mutation operator selects two distinct points in the list and swaps their values. This creates a new potential solution that is very similar to its parent and preserves the validity of the solution.

We replace the crossover with the current mutation operator. That means the same operator was used for both crossover and mutation. This exchange of operator makes the algorithm converge more quickly.

Fitness function:

The fitness value of an individual is evaluated by counting the diagonal attacking queens in a given chromosome. In other words, how many queens on the board are placed diagonally with other queens in a given chromosome? An individual with a few diagonal conflicts is considered fitter than one with many diagonal conflicts. The diagonal conflict is evaluated by comparing the absolute value of the difference between rows and columns values of the queens. If the absolute value of the difference between rows and columns values of two queens is equal then the two queens attack each other diagonally.

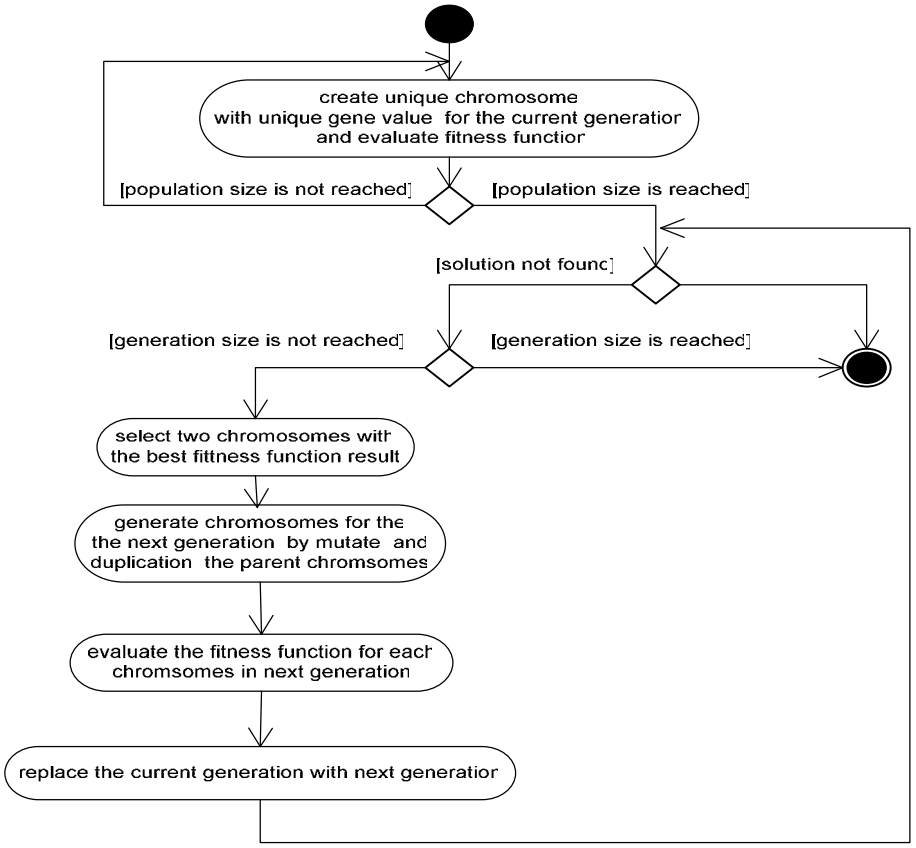

Figure 5.4 Activity diagram for phase 2

As shown in the Figure 5.4, the first generation chromosomes with unique value of genes are created. The fitness function is computed for every chromosomes and the best individual is selected. Then we will continue to create the next generation from the selected individual. This will continue till an individual which is good enough is

found. Using this algorithm, an optimal solution for the n queen problem is found.

The system is modeled with two major classes GeneticAlgorithm and Chromosome. GenticAlgorithm class is responsible for the number of chromosomes created, evaluate the fitness function of individuals (chromosomes) and to select the best individuals. The Chromosome class creates chromosomes with distinct values and mutates the chromosomes to generate offspring. See Figure 5.5.

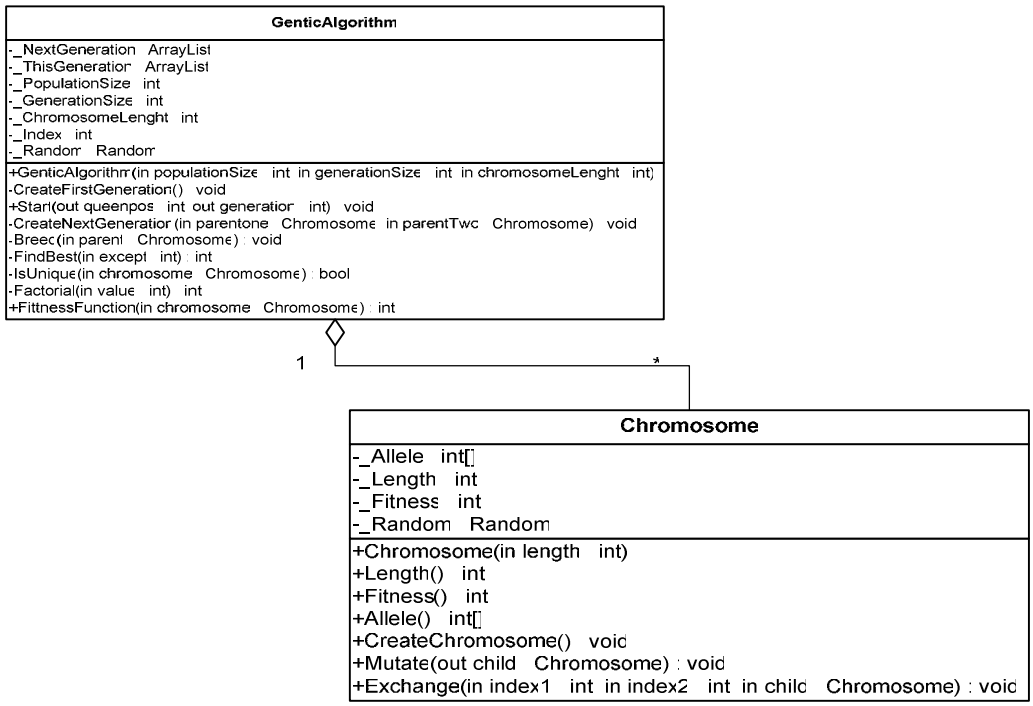

Figure 5.5 Class diagram for phase 2

Figure 5.6 shows the average fitness development of the best individual and the population for the n-queen problem of size 8. For each generation the population size is twice n. The fitness value is measured by counting the number of diagonal conflicts. So, the lower the fitness value of an individual, the fitter it will be. The maximum fitness value of an individual is zero. In this figure, it is shown the negative

of the fitness value for visualization. The number of generations shown in the figure is not large enough to show a clear convergence.

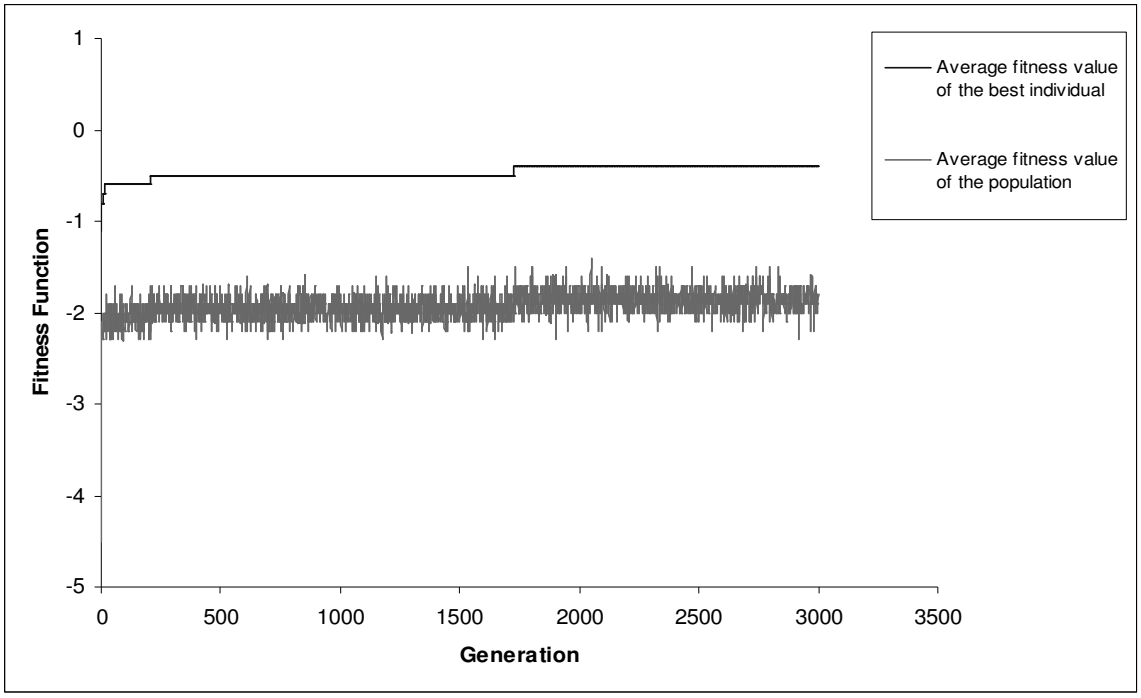

Figure 5.6 The average fitness value of the best individual and the population.

5.5 The basic concepts considered in the agent architecture

In the remaining 3 phases, a light weight agent architecture is developed to solve the problem. The agent architecture is modeled by taking the following concepts into consideration.

Self-organization

Self-organization is a process in which a pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern.

Self-organization describes the time evaluation of a system into an ordered state in the absence of external pressures.

Emergence

Emergence is the appearance of a property of feature not previously seen. It is the appearance of coherent global behavior from interaction between the low level parts. Emergent behavior is robust since it is not particularly sensitive to initial conditions or perturbations. In systems that demonstrate emergence order arises from many different initial states.

Communication

The communication can be local or global. In the case of global communication there is a central system that is commonly accessed by all the systems that are communicating. And the systems communicate through the commonly access system. On contrary, for local communication there is no system available that is shared for the interaction. Every system interacts with each other directly.

Loosely coupled asynchronous parallel processes

Each of the agents should have a separate thread of control. The agents should be coupled strongly through the interaction with the environment.

These agents are characterized by:

- Situatedness: agents receive input from the environment and perform an action that may change the environment.
- Autonomy: agents operate without the direct intervention of human being or other agents. They are created by separate thread.
- Reactivity: agents take timely action in response to changes in the environment.

- Proactivity: agents not only react, but also exhibit goal – oriented actions.
- Adaptivity: agents can learn to self-organize itself; as a result a global behavior emerges.

The agent's computational structure contains

- Static knowledge on itself and on other agents (acquaintances).
- Expertise knowledge that represents treatments and actions that an agent is able to carry and which can be described in various forms (production rules, frames, logical expressions, etc.).
- Reasoning: the inferences which draw the problem resolution (reasoning).
- Communication: the communication protocols between the agents.
- Cooperation strategies used by the agents to cooperate with others.

For the particular *n-queens problem*, we model each queen as a separate agent. Every agent has identity, character and goal. As a multi agent system working together they also have a global goal. Each agent communicates and interacts with other agents. The agents are capable of sensing the environment and taking actions to achieve their goal.

The identity of an agent is the same as the column on the chessboard where the agent located. The character of an agent shows the location of the agent which is perceived as a state of the environment (chessboard). Each of the agents has local goals and global goal. The local goal is to place itself in the chessboard without attacking the agents next to it. While the global goal, is to place itself in the chessboard without attacking any other agent in the chessboard. Initially the agents perceive the state the environment and this state of the environment will be the default character for the agents. The state of the environment shows the location of the agent that is the row value along the column that the agent is placed. Each of the agents interacts and modifies its character to achieve both the local and global goal. Message sending and receiving is used by the agents for the interaction. The message shows the identity and the character of a particular agent. Eventually, every agent will self organize

itself and results in the emergence of a global behavior. As the global behaviors emerge we can say to some level that each of the agents learns how to arrange itself in the chessboard.

5.6 Phase 3

The objective of this work is to solve the problem using a multi agent system where communication between agents is global. The communication between agents is achieved through a global system called *white board*. The *white board* is used as repository for the available information about agents. An agent will get information about all the other agents form the *white board*.

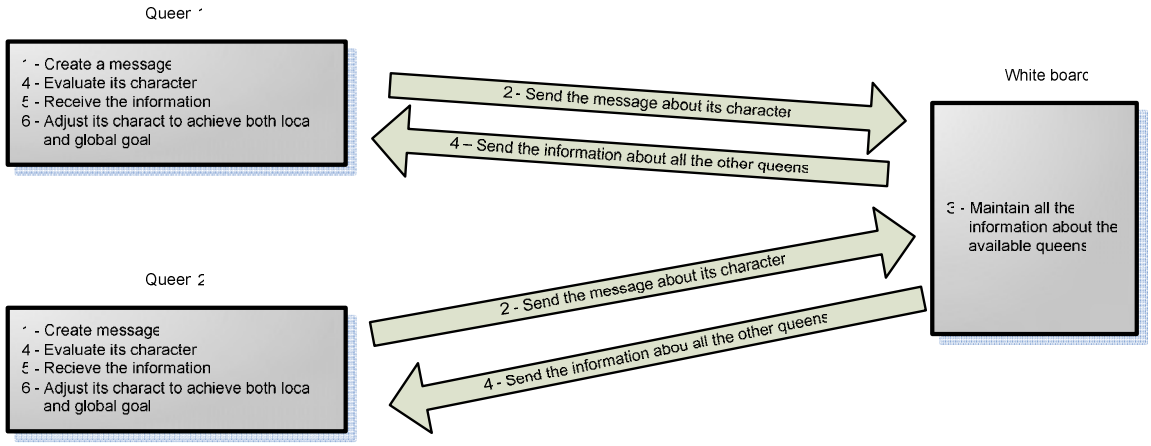

Figure 5.7 System architecture of the agents' interaction in phase 3

Figure 5.7 shows how the agents interact through the white board to fulfill the local and global goals. A queen (agent) posts a message about its character on the white board. A queen reads the information about all other queens from the white board. Since a single queen has the information about all the other queens, it evaluates and modifies its character according to local goal and global goal. Every queen will perform similar tasks asynchronously. Posting, reading message and evaluating, modifying character will continue till the local and global rule are fulfilled. Finally, the global behavior emerges as the queens position themselves accordingly in the

system. Figure 5.8 shows the activity diagram for the algorithm of the system. The classes diagram for the system is shown in figure 5.9.

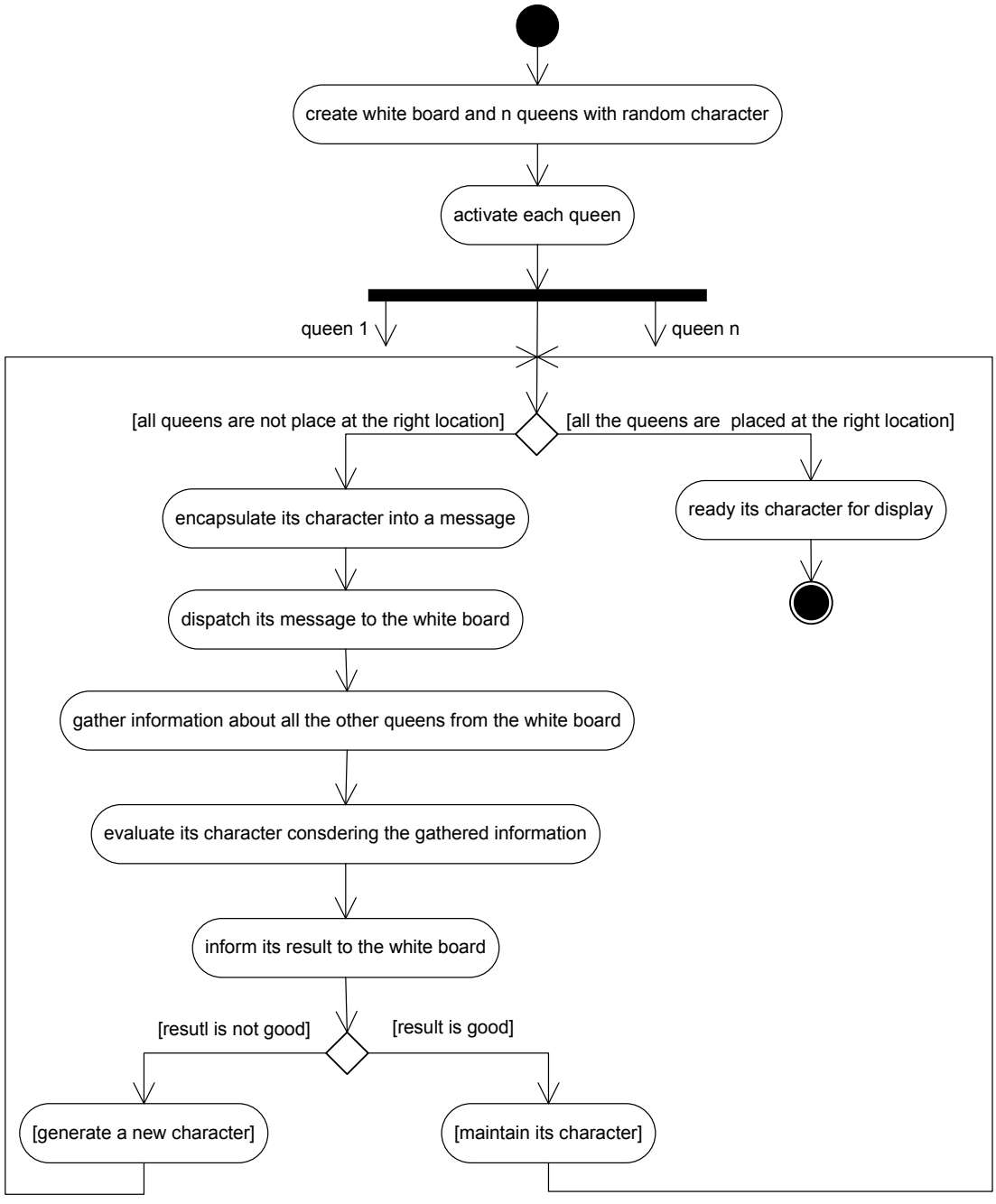

Figure 5.8 Activity diagram for phase 3

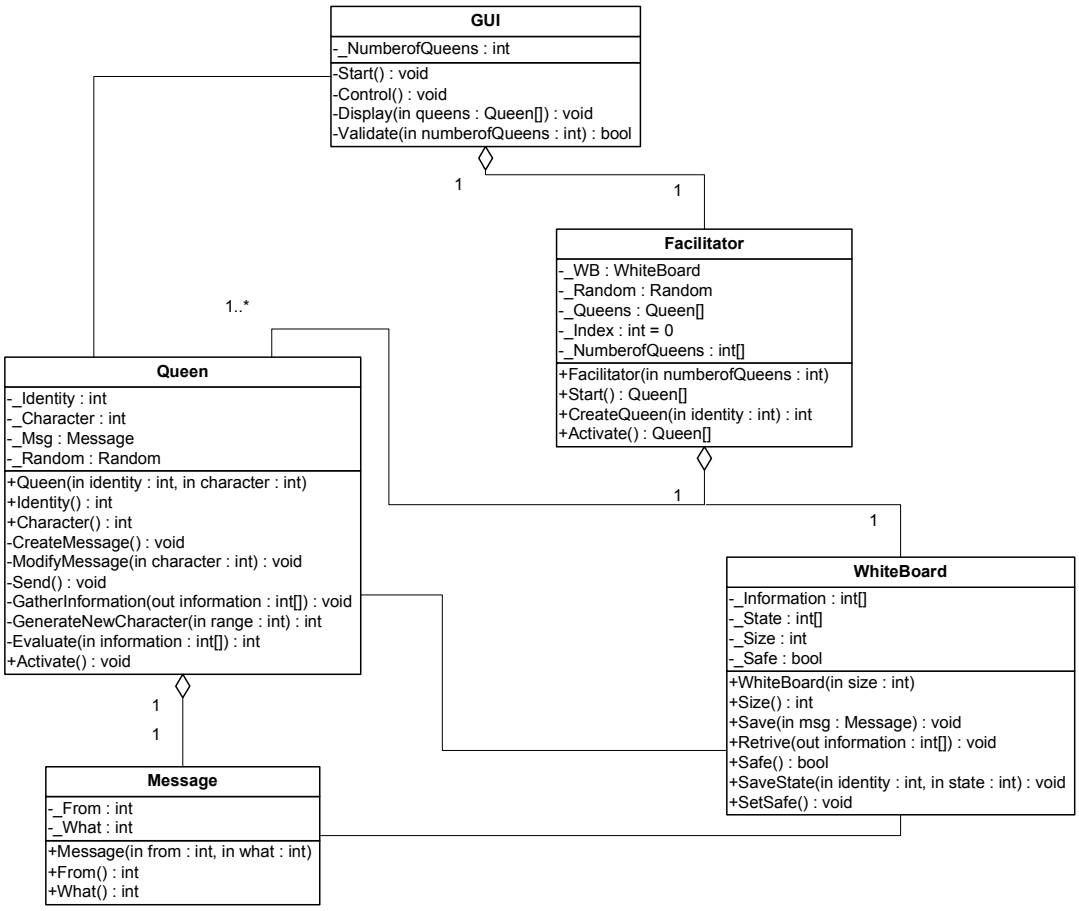

Figure 5.9 Class diagram for phase 3

5.7 Phase 4

The objective is to modify the way the agents interact with each other. Agents communicate with each other by sharing a global system called *communication center*. The difference between *communication center* and the *white board* mentioned in phase 3 is that the *communication center* is active. This means the *communication center* filters the information before sending it to an agent. It filters the information in such a way that an agent can only have information about agents next to it (neighboring agents).

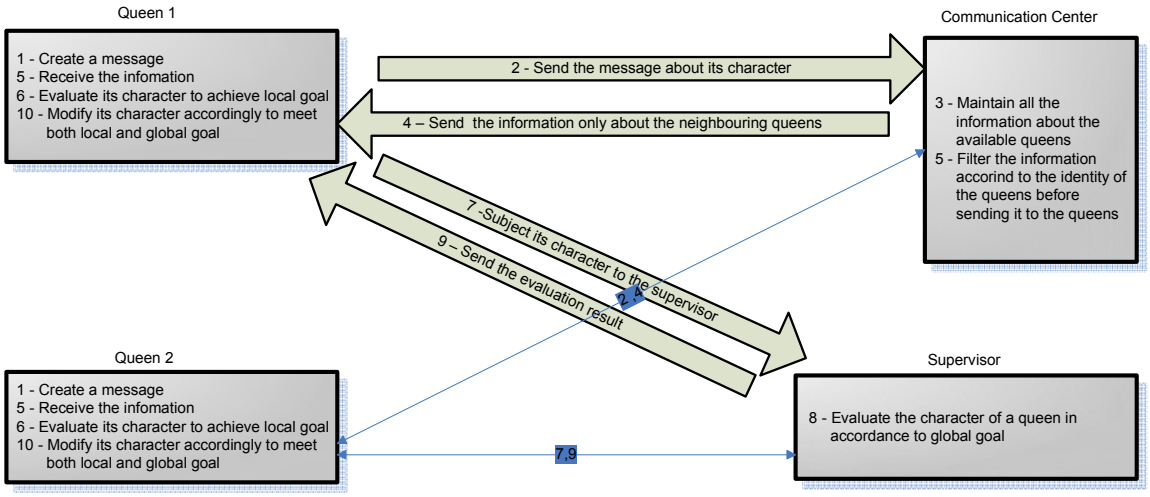

Figure 5.10 System architecture of the agents' interaction in phase 4

As shown in the figure above, every queen will dispatch a message about its character to the *communication center*. The *communication center* accepts the message, uses the identity of the agent to filter the agents next to the particular queen and sends back a message only about the neighboring agents. Having information only about the neighboring queens, the particular queen will be able to evaluate and modify its character only in accordance to the local goal. On the behalf of the evaluation for the fulfillment of the global goal there will be a supervisor. The agent adjusts its character in accordance to the local goal and subjects its character to the supervisor. The supervisor checks the character of an agent if it is good enough to satisfy the global goal. The activities stated above will be performed by each queen asynchronously till the local and global goal is fulfilled simultaneously. Finally, the global behavior emerges as the queens position themselves accordingly in the system. Figure 5.11 and 5.12 show the activity diagram and class diagram for the implementation phase 4.

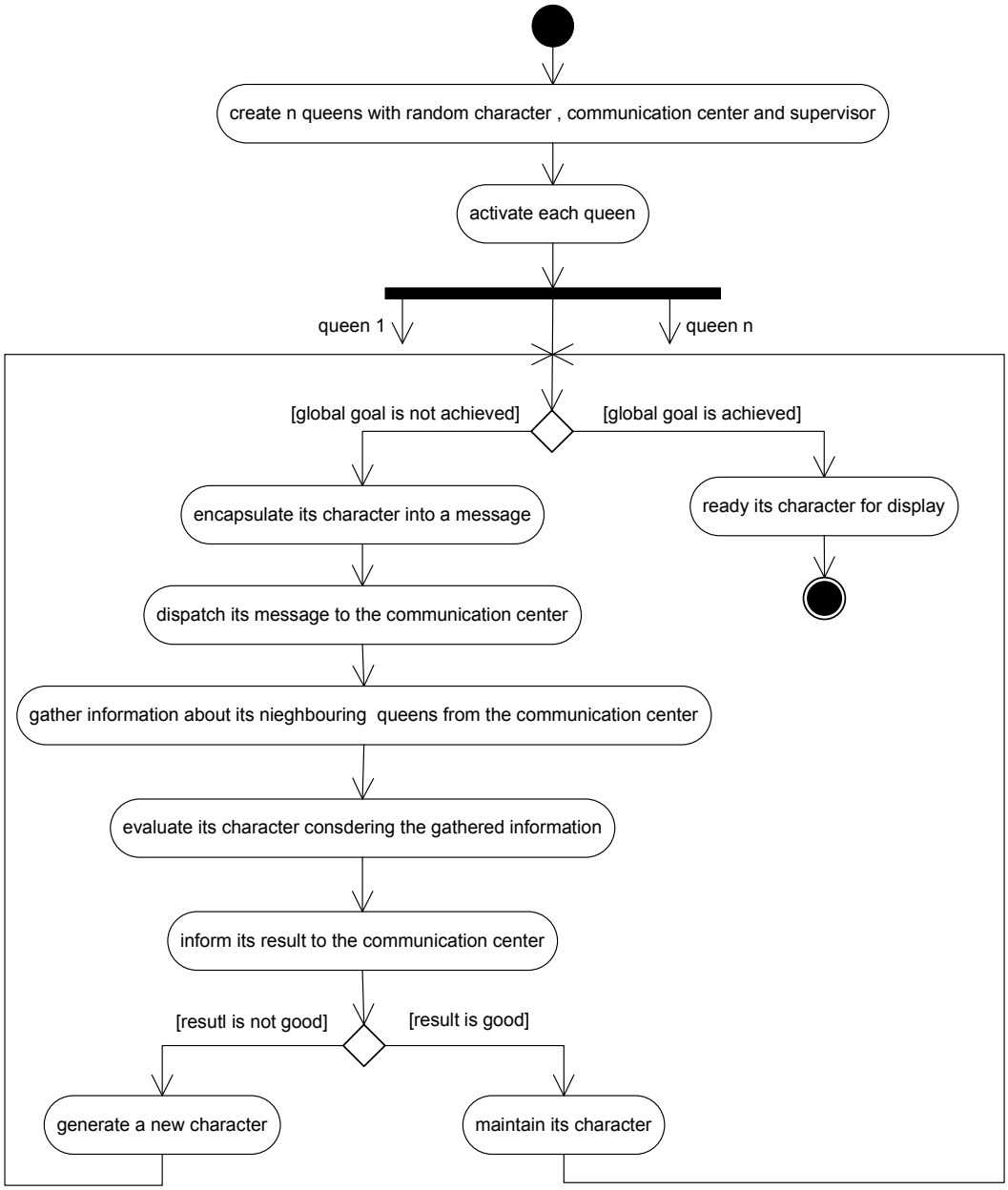

Figure 5.11 Activity diagram for phase 4

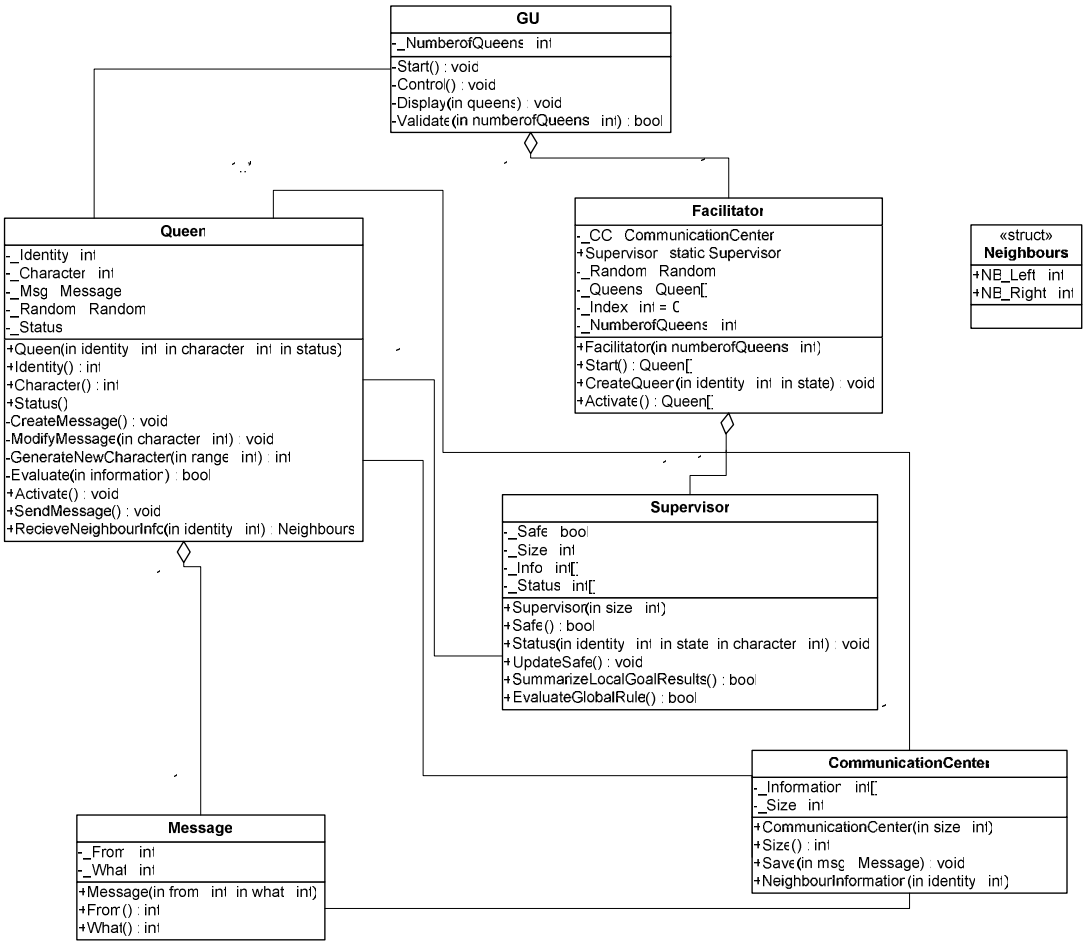

Figure 5.12 Class diagram for phase 4

5.8 Phase 5:

The objective of this phase is to further modify the way the agents interact with each other. Agents interact and cooperate directly with each other through local communication without using any shared system. An agent can only communicate with agents next to it (neighboring agents).

To make communication between agents local, we use a concept of *publishing and subscribing of event*. This means a class can raise an event and publish a message to convey information whenever there is a change of character or fulfillment of certain requirement. And the conveyed information can only be listened by those classes who subscribe for that particular event.

We can adopt the above concept to create a communication between agents. Every agent raises an event and publishes a message when there is a change in character or fulfillment of certain requirement. An agent can subscribe for event only from the agents next to it.

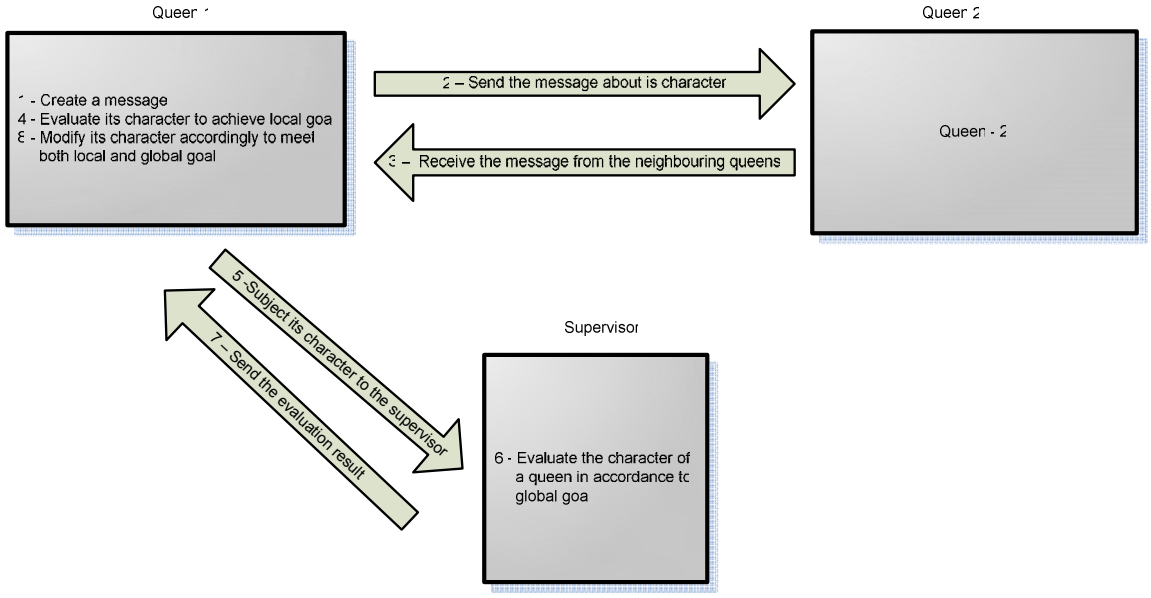

Figure 5.13 System architecture of the agents' interaction in phase 5

Every queen communicates only with its neighboring queens directly. Since a queen has information only about the queens next to it, it can evaluate and modify its character only in account to the local goal. There should be a supervisor that evaluates the character of the queens in accordance to the global rule. The supervisor manages to gather information about the queens by subscribing for events raised and message published when there is a change in the queens' character.

The queens evaluate and modify their character till the local goal is fulfilled then subject their character to the supervisor to check if the global goal is also satisfied. This will continue until the local and global goal is achieved simultaneously.

In the same manner the global behavior emerges as the queens eventually position themselves accordingly in the chessboard. Figures 5.14 and 5.15 show the activity diagram and the class diagram for the implementation of phase 5.

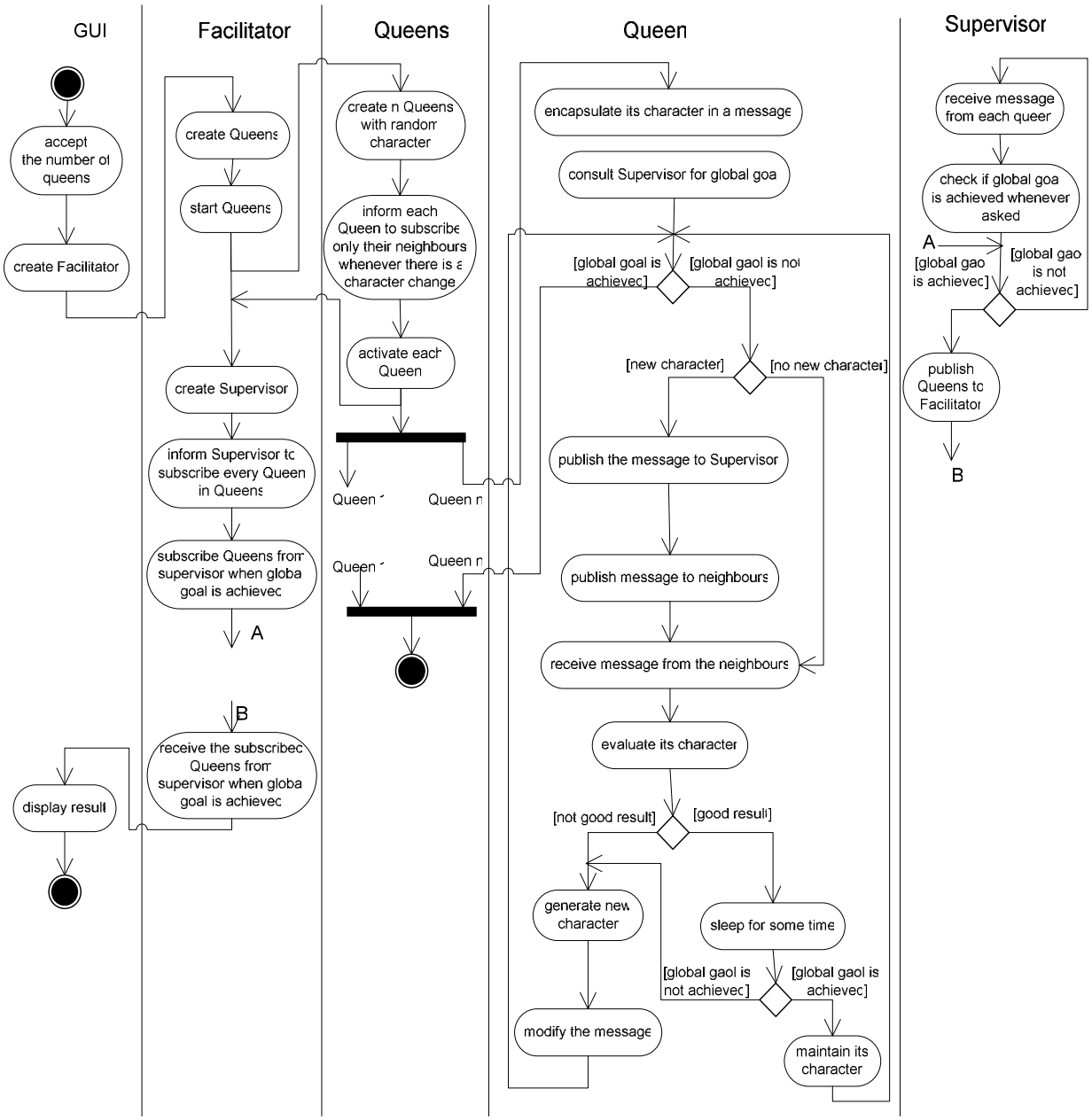

Figure 5.14 Activity diagram for phase 5

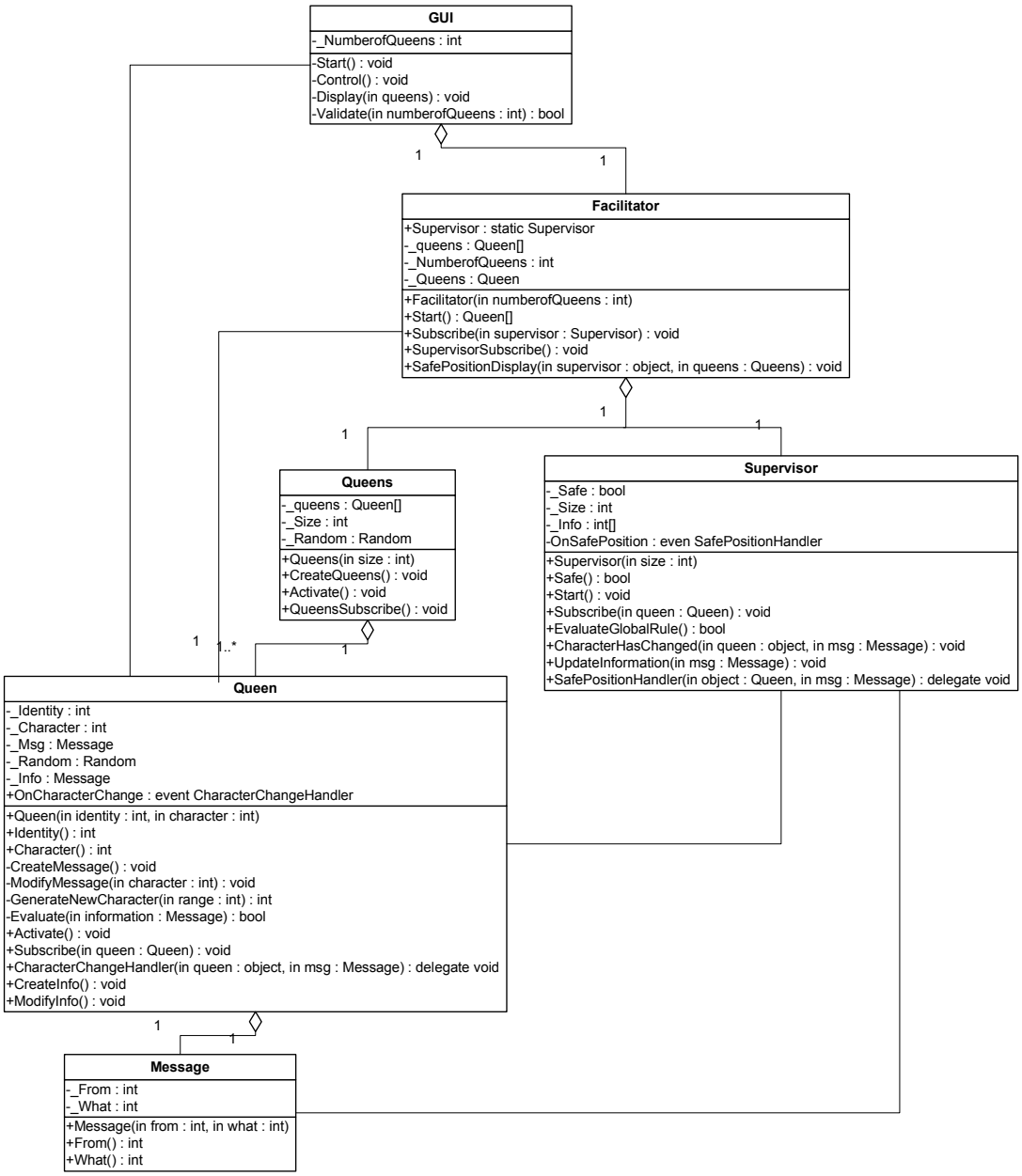

Figure 5.15 Class diagram for phase 5

5.9 Phase comparison

In the first phase, a simple backtracking algorithm is employed to develop the expert system that solves the n-queen problem. All possible conditions are hard coded in the system resulting in the transfer of all knowledge from the domain expert to the system. This holds the system from making an autonomous decision by itself. Using this approach it is possible to find a solution for a large number of queens with a relatively short time. However, there is a noticeable delay as the number of queens grows more than 15.

The second phase uses a genetic algorithm. This phase models a set of *n* queens with their positions as chromosome representing an individual in a population. This leaves the system to search for an optimal solution by evaluating its fitness proximity using a certain known measure of quality as a fitness function. That is, the system will have a limited level of independence in the process of finding the optimal solution. The amount of time required to find a solution depends on factors such as the number of queens and the size of populations. The probability of finding the optimal solution increases as the size of populations grows. The probability of finding a solution for a given evolution increases as the number of generations increases.

In the remaining phases, agent architecture is introduced where agents make independent decisions without interference from the human being. A communication scheme provides a channel for agents to cooperate and coordinate among themselves. The agents communicate (directly or indirectly) with other agents to adaptively respond to changes in the environment it perceive. These agents make decisions by evaluating their behavior according to their local and global goals. The third and fourth phases utilize a globally shared communication system to achieve coordination. Global communication creates a bottleneck for the system performance and is also prone to a single point of failure.

The third phase has a globally accessible *white board* for communication between agents. In this type of communication method, an agent can obtain information about

all other agents that are participating in the system from the global white board. Therefore, agents participating in this type of communication will store their information on the white board and perceive information about the environment from the white board. These agents respond to changes in the environment by adapting themselves accordingly to meet both the local and global goals. This communication usually hinders the performance of the system in terms of the amount of time that is required to result in a solution. However, the agents can communicate with their environment, which leaves the control to agents to respond adaptively to environmental changes.

Real world situations do not allow an agent to have knowledge over all its environment variables. The extent of information an agent can obtain from the environment can be limited to the neighboring agents. Phase four enforces this limitation by modifying the global communication model, using a *communication center* that allows an agent to obtain information only about neighboring agents. Due to this limitation of available information to an agent, an individual agent is forced to act only to achieve its local goal. As a result, a supervisor which is in charge of evaluating the agents' performance in accordance to a global goal comes to the picture of the agent architecture.

The communication model can be again modified to closely mimic real world behavior by leaving agents to communicate directly with each other. Phase five is a demonstration for this type of agent communication method. Agents have information only about their neighboring agents. Each of the agents adapts to the change of the environment to achieve a local goal. These agents subject their change of state to the supervisor to evaluate their performance in account of the global goal.

The agent architecture helped us to model an intelligent system. Based on this agent architecture, agents are developed having ability to perceive, react and adapt to the changes in the environment. These agents perform their activities and make decision without direct human intervention. Finally, in phase five, the agent architecture

introduces an efficient way of direct communication into the system. For the agent architectures, the time complexity increases dramatically as the number of queens increases. This behavior is outlined in Figure 5.16. The resource required for computation (CPU utilization) also increases very dramatically as the number of queens increases. This clearly shows that as a system becomes independent, it will require more computational resources, computational time, and CPU and memory utilization.

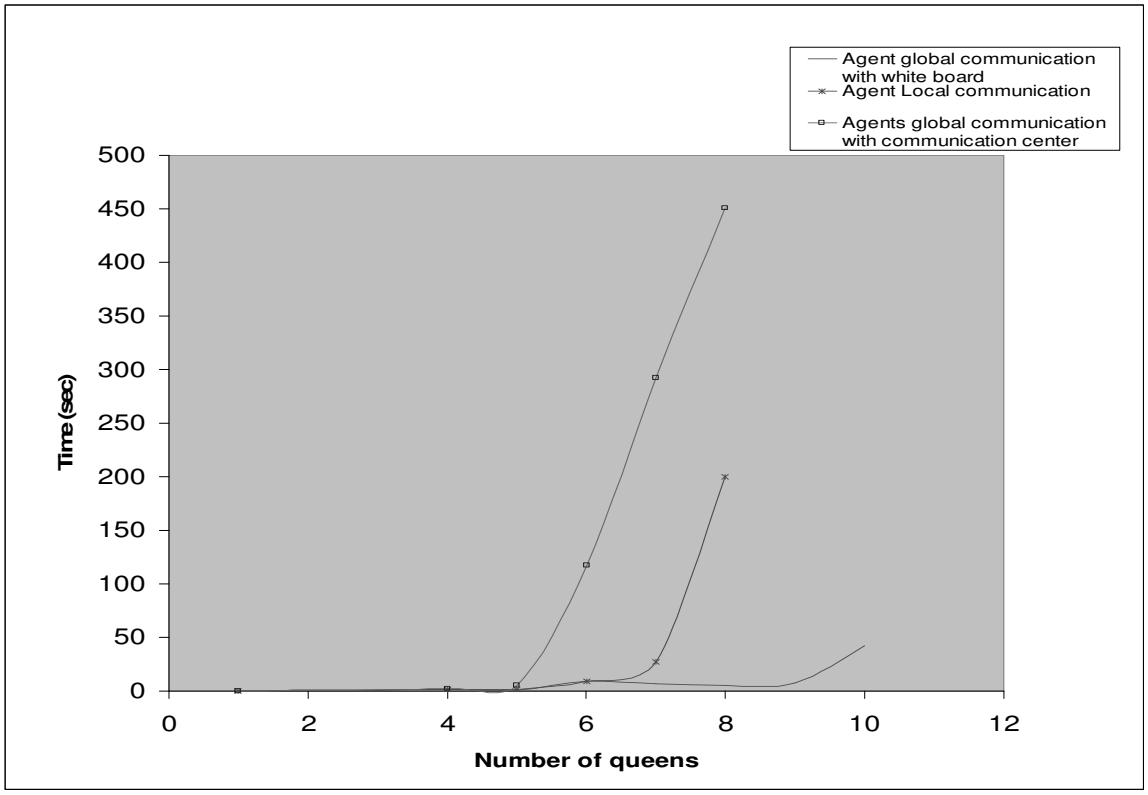

Figure 5.16 Time complexity vs. problem complexity diagram for the agent architectures.

6. Conclusion and outlook

Intelligent agents are suitable to solve complex real world problems, especially problems that involve complex decision tasks. Agents systems are also useful to smooth the transition from the analysis of a system to its construction (design and implementation), which generally reduces the time that is required to construct a software system.

An agent system paradigm is becoming an important subject in today's large society of researchers and real world application developers. As a result, this thesis has been committed to the development of software application, which can help one to understand and learn the use of agent systems, through solving the famous n-queen problem.

Through the development of this application, it is possible to study the behaviors of an agent system. This is carried out in phases where the phases span from simple backtracking algorithm and genetic algorithm to the use of agent architecture. In each of the phases, the transfer of problem solving task from human being to the system is shown. In the progressive phase the intervention and decision making of a human being is reduced as the system becomes intelligent to solve by its own.

At the beginning, an expert system is developed where the system gets all the knowledge from the domain expert. A simple backtracking algorithm is used to solve the n-queen problem. In the next phase a genetic algorithm is used to provide an optimal solution to the problem. The system makes decisions and exercises some level of independence by finding an optimal solution for the given problem. Problem solving and decision making task is shared by system.

For the remaining phases, the light weight agent architecture has been developed, where agents entertain properties like suituatedness, autonomy, reactivity, proactivity and adaptivity. Each of these problem solving agents peruse to achieve their goal. The agents cooperate and interact with each other asynchronously to fulfill their team goal

as a multi-agent system working together. Low level interaction and communication between the agents began by sharing a system where each agents has unlimited viewpoint of the environment. In the next phase, the global communication among the agents is limited between neighboring agents. Finally, the direct communication between agents is developed where each of the agents have limited view only about the agents next to it. From these lower-level interactions at each phase, a defined pattern at the global level of a system emerges solely. As a result, agents can self-organize themselves and show the emergence of coherent global behavior. From this, we can say these agents can learn and adapt themselves to the environment at population level.

It is evident that many ideas and additional improvements and utilization of other agent system technologies, such as mobility, exist. However, it can be stated that the goal of this thesis has been achieved by developing a simple and clear application that demonstrates the different elements and behaviors in an agent system.

The system can be further enhanced by making agent to fully manifest their properties. The architecture can also be extended to an agent system architecture where an agent can provide and consume service from any problem in different environment. Further more learning at individual level can be incorporated.

References

[1] S. Franklin and A. Graesser, Is it an Agent or just a Program?: A Taxonomy for Autonomous Agents, 1996

[2] P. Busetta, R. Ronquist, A. Hodgson, A Lucas, JACK Intelligent Agents - Component for Intelligent Agents in Java, Technical Report 1, and Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1999

[3] R. A. Flores-Mendez , Towards a Standardization of Multi-Agent System Frameworks,1999

[4] S. Franklin, Autonomous Agents as Embodied AI Cybernetics and Systems, special issue on Epistomological Issues in Embodied AI, 1997.

[5] K. P. Sycara, MultiAgent Systems ,1998

[6] http://www.durangobill.com/N_Queens.html

[7] http://www.schoolnet.ca/vp/AMOF/e_queenI.html

[8] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 1995

[9] http://www.asap.cs.nott.ac.uk/themes/ma.shtml

[10] K. Decker, K. Sycara and M. Williamson, Middle-Agents for the Internet, 1997

[11] T. Finin, Y. Labrou and J. Mayfield , KQML as an Agent Communication Language, 1997

[12] M.N. Huhns, and M.P. Singh, Agents and Multi-agent Systems: Themes Approaches, and Challenges, 1998.

[13] T. Kumar. Paul and H. Iba, Linear and Combinatorial Optimizations by Estimation of Distribution Algorithms, 2002

[14] A. Davidsson and J. Lindgren, An Introduction to Genetic Algorithms, 1999

[15] C. Jacob, Illustrating Evolutionary Computation with Mathematica , 2001

[16] P. J. Bentley and D. W. Corne, Creative Evolutionary Systems, 2002

[17] M. Genesereth, and R. Fikes, Knowledge Interchange Format, 1992

[18] K. M. Kavi and D. Kung, A Framework For Designing, Modeling and Analyzing Agent Based Software Systems.

[19] M. Fowler and K. Scott, UML Distilled Applying the stander object modeling langue, 1998

[20] J. Liberty, Programming C#

[21] Grosso, A. Gozzi, M. Coccoli, A. Boccalatte, An Agent Programming Framework Based onthe C# Language and the CLIA, University of Genova, DIST,Via Opera Pia, 13 – 16145 Genova, Italy

[22] C. Vecchiola, A. Gozzi, M. Coccoli, A. Boccalatte, An Agent Oriented Programming Language Targeting the Microsoft Common Language Runtime, University of Genova, DIST Via Opera Pia, 13 – 16145 Genova, Italy

[23] D. Chauhan, JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation, ECECS Department Thesis, University of Cincinnati, Cincinnati, OH, 1997.

[24] M. Williams, "Microsoft Visual C# .NET", Microsoft Press, Redmond, Washington, 2002.

[25] D.S. Platt, "Introducing Microsoft .NET", Microsoft Press, Redmond, Washington, 2001.

[26] D. Parks, "Agent Oriented Programming Languages: a Practical Evaluation", available on the web at www.cs.berkeley.edu/~davidp/cs263/, December, 1997.

[27] A.V. Aho, R. Seti, J. D. Ullman, Compilers: Principles, Technicques and Tools, Addison Wesley, Reading, Massachussets, 1986.

[28] S. C. Johnson, YACC – Yet another Compiler Compiler, Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[29] K. Boukreev, Genetic Algorithms and the Traveling Salesman Problem, 2001.

[30] http://www.cs.mcgill.ca/~ramadan/projects/605/project-summery.html

[31] http://www.cit.gu.edu.au/~sosic/abstracts/sigart90.html