Chapter 2

LITERATURE REVIEW

Natural form of interaction helps to minimize the artificiality that exists in human-computer interaction. Software agents which are inherently delegatable entities need to provide for this form of interaction so that it is easy to delegate tasks in human terms. The foremost pre-requisite for this is to provide for an intelligent interface that facilitates a natural form of interaction. This requirement has been highly felt since long and agents have been developed under different nomenclature and functions supporting different forms of interaction facilities. This chapter aims to provide a review of these agents with respect to their language abilities in providing for interaction.

The objectives of this chapter are:

- To introduce software agents, describe its properties, agent architectures, and agent-based systems using the contributions of the various researchers working on agents.
- To describe the role of intelligent interfaces and its key characteristics that help to realize a paradigm shift from direct manipulation to indirect management of user interface in human-agent interaction.
- Presenting the results of the study of the existing agents with language ability in the form of an evolutionary taxonomy.
- Delineating the advantages, limitations and drawbacks in the language ability of the various agent types in the taxonomy.

2.1 Software Agents

Software Agents are a prominent milestone in Artificial Intelligence research. They have been perceived, architected and realized to augment the behavior of a software to that of digital assistants that perform a designated function on behalf of users who use them. This notion of agents made many researchers in different domains to take
interest in agents. Each of them has defined agents using their own discernment of the concept of agents. Two distinct but related approaches to the definition of Software Agent have been attempted: One is based on the notion of agent-hood as an ascription made by some person, and the other based on a description of the attributes that software agents are designed to possess.

**Agent as an ascription**

Software Agents originated as a result of the research which focused upon realizing the software counterparts of the hardware robots. The idea of the agent was conceived by John Mc Carthy and the term was coined by Oliver G Selfridge in 1950 (Kay, 1984). Their view of an agent was that of a system which given a goal could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck. An agent would be a soft robot living and doing its business within the computers world (Kay, 1984).

Though the agents have their roots in Distributed Artificial Intelligence (DAI), the concept of an agent was taken up by researchers pertaining to varied fields such as Distributed Object Computing, Human Computer Interaction, Intelligent and Adaptive Interfaces, Intelligent search and filtering, Information Retrieval, Knowledge Acquisition etc and different types of agents were developed. This list of fields is ever growing. Researchers pertaining to these fields have made their own ascriptions to the term agent. For example, some researchers ascribed the term agent to a software that manifested characteristics of DAI (Moulin and Chaib Draa, 1996), some to a software that served as a mediating role between people and the software (Coutaz, 1990, Weiderhold, 1989, Weiderhold, 1992), some to a software that performed the role of an intelligent assistant (Boy, 1991, Maes, 1997), some to a software that abstracts out the details of differences between information sources or computing services (Knowblock & Ambite, 1997) and some to a software that implements a cognitive function (Minsky, 1986, Minsky and Reicken, 1994). This reveals that there is no concise definition of an agent and no consensus is reached in the definition of an agent, because, as obvious from the above discussion, the concept of an agent is to a greater degree dependent on the eye of the beholder. Stating simply, it can be said that ‘an agent is what an agent does’ (Van de Velde, 1995).
Agent as a description

This corresponds to the more specific approach used in defining an agent, whereby, an agent is characterized using certain unique properties. In order to be realized as an agent, any software is expected to possess these properties. It is these properties that help to distinguish an agent from conventional software.

2.2 Agent Properties

Many researchers have worked on Agents and they have identified certain properties that should be possessed by the agent. The agents are either defined using textual descriptions comprising of the identified properties or the properties are given as a separate list along with explanations of what is attributed by the corresponding property to the agent. The textual descriptions of properties given by the various researchers are given below.

(Maes, 1995) Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

(Brenner et al, 1998) Intelligent software agents are defined as being a software program that can perform specific tasks for a user and possesses a degree of intelligence that permits it to perform parts of its tasks autonomously and to interact with its environment in a useful manner.

(Hayes Roth, 1995) Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment, action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.

(Russel & Norvig, 1995) An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.

(Franklin and Graesser, 1996) An autonomous agent is a system situated within a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.
The list of properties given by various researchers are given below in Table 2.1.

Table 2.1: Agent properties given by various researchers

<table>
<thead>
<tr>
<th>Name of Researcher</th>
<th>Agent Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Etzioni and Weld (Etzioni and Weld, 1995)</td>
<td>Reactivity, Autonomy, Collaborative behavior, Knowledge-level communication ability, Inferential capability, Temporal Continuity, Personality, Adaptivity, Mobility</td>
</tr>
<tr>
<td>Wooldridge and Jennings (Wooldridge and Jennings, 1995)</td>
<td>Autonomy, Social Ability, Reactivity, Pro-activeness, Knowledge, Belief, Intention, obligation, mobility, veracity, benevolence, rationality</td>
</tr>
<tr>
<td>Nwana (Nwana, 1996)</td>
<td>Autonomy, Cooperation, Learning</td>
</tr>
<tr>
<td>Franklin and Graesser (Franklin and Graesser, 1996)</td>
<td>Reactive, Autonomous, goal-oriented, temporally continuous, Communicative, Learning, Mobile, Flexible, Character</td>
</tr>
<tr>
<td>Brenner, Zarnekow, and Wittig (Brenner et al, 1998)</td>
<td>Reactivity, Proactivity / goal-orientation, Reasoning / Learning, Autonomy, Mobility, Character, Communication, Cooperation, Coordination</td>
</tr>
</tbody>
</table>

From the above table, it is obvious that there is no consensus on the list of properties of an agent. But, most of them agree on the following properties, namely,

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

- **Reactivity**: agents perceive their environment, and respond in a timely fashion to changes that occur in it.

- **Pro-activity**: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

- **Social Ability**: agents interact with other agents via some kind of agent communication language.

### 2.3 Types of Agents

The properties possessed by an agent depend upon the nature of the function to be performed by the agent, or the environment in which it has to work. Based on the subset of the properties possessed by the agent several types of agents have been arrived upon by the researchers. For example, Wooldridge & Jennings (Wooldridge & Jennings, 1995) identify two types of agents namely, Weak Agents which possess only autonomy, reactivity, proactivity and social ability and Strong Agents which...
possess the other properties listed by them as given in Table 2.1. Using the various combination of the properties possessed by an agent—cooperation, learning and autonomy, Nwana (Nwana, 1996) has identified the types of agents depicted in Figure 2.1.

![Figure 2.1: Typology of Agents](image)

Similarly Franklin and Graesser (Franklin & Graesser, 1996) consider that autonomy, reactivity, proactivity and social ability properties to be essential to realize an agent. They specify that the inclusion of the additional optional properties help to identify the different types of agents, for example, Mobile Learning Agents, etc. Other types of agents, based on the functions performed by the agent have also been identified by the researchers (Nwana, 1996), (Franklin & Graesser, 1996), (Sycara and Zeng, 1996), (Brenner et al, 1998).

### 2.4 Agent Architectures

The architecture of an agent could be very abstractly represented as shown in Figure 2.2 (Genesereth and Nilsson, 1987), (Wooldridge, 1999).

The agent can be represented by the following functions:

- **See** \( S \rightarrow P \) (\( S \) – set of environment states, \( P \) – set of percepts)
- **Next** \( I \times P \rightarrow I \) (\( I \) – set of internal states)
- **Action** \( I \rightarrow A \) (\( A \) – set of agent actions)
The agent starts in some initial internal state \( i_0 \). It then observes its environment state \( s \), and generates a percept \( \text{see}(s) \). The internal state of the agent is then updated via the next function, becoming set to \( \text{next}(i_0, \text{see}(s)) \). The action selected by the agent is then \( \text{action}(\text{next}(i_0, \text{see}(s))) \). Different ways of implementing or realizing the above functions has led to the identification of the following four types of agent architectures:

- Logic-based
- Reactive
- Deliberative / BDI
- Layered

2.4.1 Logic-Based Architecture (Genesereth and Nilsson, 1987) (Wooldridge, 1999)

This architecture is realized using the traditional approach to building artificially intelligent systems, which suggests that intelligent behavior can be generated in a system by giving that system a symbolic representation of its environment and its desired behavior and syntactically manipulating this representation. This syntactic manipulation corresponds to logical deduction. The symbolic representation is made using first-order logic.

Let \( L \) be the set of sentences of classical first-order logic and let \( D = \rho(L) \) be the set of \( L \) databases and \( A \) the set of agent actions. The internal state of an agent is then an element of \( D \). Now the above shown agent functions are as follows:
2.4.2 Reactive Architecture (Brooks, 1986) (Wooldridge, 1999)

There are two defining characteristics of the subsumption architecture. The first is that an agent's decision-making is realized through a set of task accomplishing behaviors, where each behavior may be thought of as an individual *Action* function as defined above. These task accomplishing modules are assumed to include no complex symbolic representations and are assumed to do no symbolic reasoning at all. The behaviors are implemented as rules of the form:

\[ \text{Situation} \rightarrow \text{Action} \]

which simply maps perceptual inputs directly to actions. The second defining characteristic of the subsumption architecture is that many behaviors can fire simultaneously. In order to choose between the different actions, the modules are arranged into a subsumption hierarchy with the behaviors arranged into layers as shown in Figure 2.3. Lower layers have higher priority and are able to inhibit the higher layers which implement more abstract behaviors.

![Figure 2.3: Reactive Agent Architecture](image)

2.4.3 Deliberative / BDI Architecture (Rao and Georgeff, 1991) (Wooldridge, 1999)

Rao and Georgeff specify the following seven main components to a BDI agent.

- A set of current beliefs (*Bel*), representing information the agent has about its current environment.
- A belief revision function (brf), which takes a perceptual input and the agent's current beliefs, and on the basis of these, determines a new set of beliefs
\[
\text{brf} : \rho(\text{Bel}) \times P \rightarrow \rho(\text{Bel})
\]
- An option generation function (options), which determines the options available to the agent (its desires - Des), on the basis of its current beliefs about its environment and its current intentions (Int)
\[
\text{Options} : \rho(\text{Bel}) \times \rho(\text{Int}) \rightarrow \rho(\text{Des})
\]
- A filter function (filter), which represents the agent's deliberation process, and which determines the agent's intentions on the basis of its current beliefs, desires, and intentions
\[
\text{Filter} : \rho(\text{Bel}) \times \rho(\text{Des}) \times \rho(\text{Int}) \rightarrow \rho(\text{Int})
\]
- A set of current intentions, representing the agent's current focus - those states of affairs that it has committed to trying to bring about
- An action selection function (execute), which determines an action to perform on the basis of current intentions
\[
\text{Execute} : \rho(\text{Int}) \rightarrow A
\]

This architecture is depicted in Figure 2.4.

2.4.4 Layered Architectures (Muller et al, 1995) (Wooldridge, 1999)

Reactive architecture is suitable for providing reactive behaviors and Deliberative architecture is suitable for providing proactive behaviors. In order to design an agent capable of reactive and pro-active behaviors, the layered architectures were identified. In this layered architecture, the various subsystems of the agent capable of providing reactive and proactive behaviors are arranged in a layered structure. Typically, there will be at least two layers, to deal with reactive and pro-active behaviors respectively. Two types of control flows can be identified within layered architectures viz.,
- Horizontal Layering
- Vertical Layering.
In horizontally layered architectures, the software layers are each directly connected to the sensory input and action output as shown in Figure 2.5a. Depending upon the percept, any of the layer(s) takes control. The decision regarding which layer gains control is made by a mediator function. This forms a bottleneck when the number of layers increases as the mediator function has to consider all possible interactions between layers before deciding which layer of the agent should take control.

**Vertical Layering**

Vertically Layered architectures can be classified into one pass architectures and two pass architectures. In one-pass architectures, control flows sequentially through each layer, until the final layer generates action output. In two pass architectures, information flows up the architecture and control then flows back down. This is depicted in Figures 2.5b and 2.5c. This architecture helps to alleviate the bottleneck...
problems encountered in the Horizontal Layering, but at the same time suffers from the problem of fault intolerance, as failure in any one layer can have serious consequences for agent performance.

![Diagram of Horizontal and Vertical Layering](image)

Figure 2.5 (a): Horizontal Layering
Figure 2.5 (b): Vertical Layering with One Pass
Figure 2.5 (c): Vertical Layering with Two Pass

### 2.5 Agent-Based System

An agent-based system is one in which the key abstraction used is that of an agent. It contains any non-zero number of agents (Jennings and Wooldridge, 2002). These agents work together to solve a problem. This helps to visualize, comprehend and develop complex systems by dividing the complex problem into smaller ones each of which can be solved using specialized agents. This approach brings along with it the advantages of modularity, distribution, abstraction and intelligence required for dealing with complexity. The issues to be addressed with respect to agent-based system development are:

- Types of Agent-based system
- Coordination mechanism
- Agent-based system development.

#### 2.5.1 Types of Agent-based System

Research into systems composed of multiple agents was carried out under the banner of Distributed Artificial Intelligence (DAI) (Weiss, 1999) and has historically been divided into two main camps as follows:
Distributed Problem Solving Systems (DPS) and Multi-agent Systems (MAS).

**Distributed Problem Solving Systems** (Durfee, 1999)

Distributed Problem Solving is the name applied to a subfield of DAI, in which the emphasis is on getting agents to work together well to solve problems that require collective effort. Due to an inherent distribution of resources such as knowledge, capability, information and expertise among the agents, an agent in a distributed problem-solving system is unable to accomplish its own tasks alone, or at least can accomplish its tasks better when working with others. It considers how a particular problem can be solved by a number of modules which cooperate in dividing and sharing knowledge about the problem and its evolving solutions. All interaction strategies are incorporated as an integral part of the system. It presumes the existence of problems that need to be solved and expectations about what constitute solutions.

The main steps involved in distributed problem solving are:

- Task Decomposition. Decomposition of large tasks into subtasks that could be tackled by different agents
- Task Allocation. Assigning subtasks to appropriate agents
- Task Accomplishment. Accomplishment of tasks by the corresponding agents
- Result Synthesis: Composing the results of individual agents into an overall solution.

The required agents for performing the above steps are identified and designed to work together at the time of development itself. Hence, the system is said to be closed with respect to the agents.

**Multi-agent System**

A Multi-agent system can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver (Durfee et al., 1989). These problem solvers - agents - are autonomous and may be heterogeneous in nature. This is because the system may be constituted by agents developed by different people. The system is also open with respect to the agents as new agents can become part of the system and leave the system dynamically. This has the advantage that multi-agent systems can
solve problems that have not been envisaged during system development unlike the distributed problem solving systems. The disadvantage is that due to the autonomous and heterogeneous nature of the agents, it cannot be assumed that all agents are interested in achieving an overall goal in the most efficient manner. Conflicting goals occur as every agent of a multi-agent system is primarily interested in achieving its own particular goal. Hence, coordination is an imminent aspect of MAS in order to resolve conflicts and work together in a coordinated manner.

2.5.2 Coordination Mechanism

Huhns and Stephens (Huhns and Stephens, 1999) specify that coordination is a property of a system of agents performing some activity in a shared environment. The degree of coordination is the extent to which they avoid extraneous activity by reducing resource contention, avoiding livelock and deadlock, and maintaining applicable safety conditions. Two types of coordination are possible in an agent-based system, viz.,

- Cooperation
- Negotiation.

Cooperation is coordination among non-antagonistic agents, while negotiation is coordination among competitive or simply self-interested agents. From this, it is obvious that Distributed Problem Solving systems are inherently cooperative and negotiation is the coordination mechanism usually prevalent in a Multi-agent System.

2.5.3 Agent-based System Development

Agent-based system development is about creating new techniques and methodologies that exploits the essential abstractions of agency in the software (Singh, 2004). This is because, the software to be designed has to deal with new concepts such as agent, goal, task, services, organization, interactions, environment etc. Also, the agent-based software is usually used to solve complex and distributed problems. These requirements can be viewed from two perspectives as follows:

- Designer's perspective
- Developer's perspective.
From a designer’s perspective, the need for methodologies that must provide the designers with abstractions that capture the essential features of agency is realized. The methodologies should also enable for agent-based system engineering by providing a rigorous process which enable the generation of a set of models describing the different aspects of the software to be designed. Also, it should help the designers to manage the inherent complexity of agent-based systems.

From a developer’s perspective, development constructs that help to realize the design models with the design abstractions used, is required. The subsequent section explores further into both these perspectives.

2.5.3.1 Designer’s Perspective

The requirements from the designer’s perspective are fulfilled by the Agent-Oriented Analysis and Design methodologies that are available. Since agent-based systems are multi-agent organizations, abstractions from social and organizational systems theory have been imbibed in these methodologies. This has been done by identifying the analogy that exists between an organization and a multi-agent system. This analogy can be inferred from the very definition of organization given by Wooldridge and Jennings and that of Gasser as follows:

*We view an organization as a collection of roles that stand in certain relationships to one another and that take part in systematic institutionalized patterns of interactions with other roles (Wooldridge and Jennings, 2000)*

*An organization is an element that groups agents, which play roles and have common goals An organization hides intra-characteristics, properties and behaviors represented by agents inside it (Gasser, 1992).*

From these definitions, it is possible to derive the following main features of organizations (Ferber, Gutknecht and Michel, 2003),

- An organization is constituted of agents (individuals) that manifest a behavior.
- The overall organization may be partitioned into partitions that may overlap.
- Behaviors of agents are functionally related to the overall organization activity (concept of role).
- Agents are engaged into dynamic relationship (also called patterns of activities) which may be typed using a taxonomy of roles, tasks or protocols, thus describing a kind of supra-individuality.
- Types of behaviors are related through relationships between roles, tasks and protocols.

These features reveal that the notion of role is an important constituent of an organization. This has been adopted as the basic abstraction for designing Agent-based systems because the conceptualization of roles is appropriate in the context of agents (Zhu, 2006) (Cabré et al. 2005) also as obvious from the elaboration on roles given below.

2.5.3.2 Roles as Design Abstractions for Agent-Based Systems

A role defines a normative behavioral repertoire for agents (Odell et al., 2003). Defined in the context of an organization, it guides and restricts the behavior of an agent or an object in the organization (Silva et al., 2003). They provide both the building blocks for agent social systems and the requirements by which agents interact (Odell et al., 2003). Each agent is linked to other agent by the roles it plays by virtue of the system's functional requirements – which are based on the expectations that the system has of the agent (Odell et al., 2003). It describes the constraints (obligations, requirements, skills) that an agent will have to satisfy to obtain a role, the benefits (abilities, authorization, profits) that an agent will receive in playing that role, and the responsibilities associated to that role (Ferber et al., 2003). A role is also the placeholder for the description of patterns of interactions in which an agent playing that role will have to perform (Ferber et al., 2003). It also represents the functional position of an agent in a group. An agent must play a role in a group, but an agent may play several roles. Roles are local to groups and a role must be requested by an agent. A role may be played by several agents (Ferber et al., 2003).

The characteristic features that a role should meet, which have been identified by Steinmann (Steinmann, 2000) (Cabot and Raventos, 2006) are as follows:
Ownership: A role comes with own properties and behavior.

Dependency: An instance of a role is related to a unique instance of its entity type and its existence depends on the entity type to which it is associated to. That is, it is not possible to have an instance of Student not related to an instance of Person.

Diversity: An entity may play different roles simultaneously, i.e., an instance of a Person may play simultaneously the role of Student and Employee.

Multiplicity: An instance of an entity type may play several instances of the same role type at the same time. For instance, a person that registers to more than one university has multiple instances of Student related to it.

Dynamically: An entity may acquire and relinquish roles dynamically, i.e., a Person may become a Student, after some years become an employee, finish his/her studies, become a project manager, start another program and so forth.

Control: The sequence in which roles may be acquired and relinquished can be subject to restrictions. For eg., a person may not become an Employee after he/she retired or when he/she is also studying two degrees. This does not prevent an employee from studying two degrees in the future. The restriction needs to be true only when hiring the employee.

Roles can play role: This mirrors that an instance of Person can play the role of Employee and an instance of Employee can also play the role of Project Manager.

Role Identity: Each instance of a role has its own role identifier, which is different from that of all other instances of the entity to which it is associated with.

Adoption: Roles do not inherit properties from their entity types. Instead instances of roles have access to some properties of their corresponding entities, i.e., Student may adopt name and address properties of Person, but neither religion nor marital status properties.

Relationship Independency: A role is meaningful even out of the context of a relationship. Example, a Person may play the role of student or employee without necessarily being tied to a university or a company respectively.
Common role for unrelated types: A set of unrelated types may play the same role. For eg., a Project Manager may be the role of both employee and external service provider.

Sharing structure and behavior: Roles may have some common structure and behavior. For instance, the constraint that Maria may not become an Employee before she is 16 years old should apply also to Project Manager.

The Role concept has been widely used in the literature and different patterns for the semantic representation of roles are available. Table 2.2 gives a summary of the same. Another such set of semantic patterns is given by Fowler (Fowler 1997), which is given in Table 2.3.

2.5.3.3 Developer’s Perspective

From a developer’s perspective, programming abstractions that capture the notion of roles is required. Kendell (Kendall, 1999) has identified that of all other approaches used for implementing roles, the Aspect Oriented Programming approach is best suited for realizing roles. This is because aspects match almost every characteristic of roles. A comparison of the similarities of aspects and roles is also given in (Hanenberg and Unland, 2002).

2.5.3.4 Aspect Oriented Programming and Roles

Aspect-Oriented Programming (AOP) is a new programming paradigm developed at Xerox Parc. AOP strives to help the developer to separate concerns to overcome the problems with cross-cutting concerns. Cross-cutting concerns are behaviors that span multiple, often unrelated, implementation modules. In addition, crosscutting concerns cannot be neatly separated from each other. AOP makes it possible to program crosscutting concerns in a modular way, and get the usual benefits of improved modularity – simpler code that is easier to develop and maintain, and that has greater potential for reuse. What OOP has done for object encapsulation and inheritance AOP does for crosscutting concerns.
<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
<th>Properties supported</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role as the name of the participant in a relationship type.</td>
<td>A Role is merely represented as a name assigned to the participation of an entity type in a relationship type.</td>
<td>Dependency, Diversity, Multiplicity, Dynamicity.</td>
<td>Roles are transformed into attributes of entity types.</td>
</tr>
<tr>
<td>Roles as a sort of subtypes or supertypes of the natural entity type.</td>
<td>Role entity types are represented as subtypes of the entity type playing them.</td>
<td>Ownership, Dependency, Diversity, Dynamicity, Roles can play roles. Relationship independency, different roles may share structure and behavior.</td>
<td>Implemented indirectly using approaches as described in (Fowler, 1997).</td>
</tr>
<tr>
<td>Roles as interfaces.</td>
<td>Roles are represented as interfaces.</td>
<td>Dependency, Diversity, Relationship independency, different roles may share structure and behavior.</td>
<td>Implemented using interfaces provided in object-oriented languages.</td>
</tr>
<tr>
<td>Roles as a distinct element from an entity type but coupled to it</td>
<td>Roles are represented as reified entity types of a relationship type between the entity type playing the role and another entity type.</td>
<td>Ownership, Dependency, Diversity, Dynamicity, Roles can play roles. Role identity. different roles may share structure and behavior.</td>
<td>The reified types are designed and implemented as classes with a set of single-valued additional attributes, which refer to each of the participants of the relationship type.</td>
</tr>
<tr>
<td>Roles as Entity Types.</td>
<td>Represents roles as an entity type related to the entity type playing the role by means of a special kind of relationship type, the 'RoleOf relationship' type.</td>
<td>All</td>
<td>Creating separate classes for the natural entity type and its roles.</td>
</tr>
</tbody>
</table>
Table 2.3: Semantic Representation of roles and their corresponding implementation methods (Fowler, 1997)

<table>
<thead>
<tr>
<th>Role Representation</th>
<th>Implementation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Role Type</td>
<td>All behaviors of the different roles are implemented within a single class.</td>
</tr>
<tr>
<td>Separate Role Type</td>
<td>Every role is implemented as a separate class.</td>
</tr>
<tr>
<td>Role sub-type</td>
<td>• Every role is implemented as a separate interface that extends a common interface.</td>
</tr>
<tr>
<td></td>
<td>• Every interface implements its own behaviors, in addition to implementing the common behaviors.</td>
</tr>
<tr>
<td></td>
<td>• Delegation of role behavior is through the super type.</td>
</tr>
<tr>
<td>Role Object</td>
<td>• Every role is implemented as a separate interface that extends a common interface.</td>
</tr>
<tr>
<td></td>
<td>• The user directly invokes the role behavior after obtaining the required role from the super type.</td>
</tr>
<tr>
<td>Role Relationship</td>
<td>The role behavior is implemented as a super class which is extended by the two classes that are related by the role relationship (Dahchour et al. 2004).</td>
</tr>
</tbody>
</table>

An aspect is, by definition, modular units that cross-cut the structure of other units. An aspect is similar to a class by having a type, it can extend classes and other aspects, it can be abstract or concrete and have fields, methods, and types as members.

Aspect-Oriented Programming involves three different development steps as follows.

- Aspectual decomposition
- Concern implementation
- Aspectual Recomposition

During Aspectual decomposition, the requirements are decomposed to identify crosscutting and common concerns. The core concerns are separated from the crosscutting system level concerns.

In the second step, each concern is implemented separately.

In Aspectual Recomposition, an aspect integrator specifies recomposition rules by creating modularization units (aspects). The recomposition process also called as weaving or integrating uses this information to compose the final system.

Kendell (Kendall, 1999) points out six basic options for designing and implementing roles in an aspect-based way as follows:
• Option A covers the situation where a static aspect introduces new elements of a role in one of the core classes.

• In Option B, the methods and variables needed for the role behavior have to be declared in the interface of the core class. An aspect instance will add the role behavior to a core instance by adapting the relevant elements of the core object.

• Option C separates the role elements from the core functionality, comparable to Option A, but here both the role and the aspect are on instance level.

• Option D builds on Option B by adding the complete role behavior to the core class. The aspect instance will filter out the unwanted role elements.

• In Option E, both a role and a core class are regular classes, glued together by a static aspect instance.

• Finally, in Option F, the role interface first is introduced at compile time to the core class. At runtime, the implementation of the role behavior is added to a core instance executing that role.

From the above discussion it can be inferred that similar to aspects, roles are nothing but cross-cutting behaviors of objects. So from a developer’s perspective, they are fully appropriate for the implementation of roles.

2.6 Human-Agent Interaction and Intelligent Interfaces

Donald A. Norman in his work on ‘How might people interact with agents’ (Norman, 1997) specifies that agents are capable of taking over human tasks, and interact with people in human-like ways perhaps with a form of natural language. They also have the potential to form their own goals and intentions, to initiate actions on their own without explicit instruction or guidance and to offer suggestions to people. They can simplify the use of computers by allowing moving away from the complexity of command languages or the tedium of direct manipulation toward intelligent, agent-guided interaction called as the indirect management of user interface. From the user-interaction point of view, they help to provide an increased form of abstraction which is described below.
2.6.1 Direct Manipulation to Indirect Management / Intelligent Interface

Jeffrey Bradshaw (Bradshaw, 1997) specifies that a distinct but complementary motivation for software agents is in overcoming problems with the current generation of user interface approach which is the direct manipulation approach. Direct Manipulation approaches are a distinct improvement over command-line interfaces and they require software objects to be visible. The users are constantly informed about the kinds of things they can act upon. But, the advantages of direct manipulations begin to fade as tasks grow in scale or complexity. The following are the common problems

*Large search space:* In large distributed systems, it is difficult to find what is needed though browsing or the use of traditional indexing methods. What is practical and possible for a few hundred items becomes unwieldy and impossible for several thousand.

*Actions in response to immediate user interaction only.* Sometimes instead of executing an action immediately, it may have to be scheduled for a specific time in the future, or the system may have to automatically react to system-generated events without user intervention.

*No composition:* With most direct manipulation interfaces, it is not easy to compose basic actions and objects into higher-level ones.

*Rigidity:* The consistency that makes passive artifact interfaces predictable and easy-to-learn for simple tasks makes them brittle and untrustworthy for complex ones.

*Function-Orientation:* Software is typically organized according to generic software functions rather than the context of the person’s task and situation.

*No improvement of behavior:* Traditional software does not notice or learn from repetitive actions in order to respond with better default behavior.

To overcome the above problems, the researchers and developers have explored the indirect management style of interaction (Kay, 1990). In such an approach, users would no longer be obliged to spell out each action for the computer explicitly;
instead, the flexibility and intelligence of software agents would allow them to give
general guidelines and forget about the details. Specifically, the use of software agents
will eventually help to overcome the limitations of direct manipulation by supporting
the following features:

*Scalability:* Agents can be equipped with search and filtering capabilities that run in
the background to help people explore vast sources of information.

*Scheduled or event-driven action:* Agents can be instructed to execute tasks at specific
times or automatically wake up and react in response to system-generated events.

*Abstraction and delegation:* Agents can be made extensible and composable in ways
that common iconic interface objects cannot.

*Flexibility and opportunism:* Because they can be instructed at the level of goals and
strategies, agents can find ways to work around unforeseen problems and exploit new
opportunities as they help solve problems.

*Task orientation* Agents can be designed to take the context of the person’s tasks and
situation into account as they present information and take action.

*Adaptivity* Agents can use learning algorithms to continually improve their behavior
by noticing recurrent patterns of actions and events.

With all the above features, the agents would provide for an intelligent interface. In
order to provide for the above, the interfaces of agents should fulfill certain
properties, which are explained in the subsequent section.

### 2.6.2 Key characteristics of Intelligent Interfaces and their language
requirements

Laurel (Laurel, 1997) specifies the following four key characteristics of interface
agents. When considered for an individual agent, these properties apply to its
interface.

- Agency
- Responsiveness
- Competence
- Accessibility.

Agency

An agent is one who is empowered to act on behalf of another. Hence, they should provide for expertise, skill and labor. That is, they must be capable of understanding the user needs and goals in relation to it, translating those goals into appropriate set of actions, performing those actions, and delivering the results in a form that could be used by the user.

Responsiveness

Most other forms of human-computer interface exhibit responsiveness, that is, user and system communicate through a series of highly constrained, explicit transactions. The users must parse their actions and intentions in terms dictated by the system and must express them explicitly. Because an agent has to take action on behalf of a user, it should have the ability to formulate and execute a set of actions solely on the basis of a user’s goals. Whether those goals are explicitly stated by the user or inferred by the system, the way an agent interprets and attempts to meet them constitutes implicit responsiveness

One aspect of implicit responsiveness is the ability of an agent to tune its actions to the user’s traits and preferences. Knowledge about the user can be both explicitly obtained and inferred. Since the user’s traits and preferences keep changing over time, responsiveness requires that the agent has access to a dynamic model of the user, or a log of his experience in a particular application or environment with rules for interpreting that experience when formulating actions.

Competence

An agent must be competent in the domain of the application or environment in which it operates. In order to serve the user well, the agent must possess both meta-knowledge and multiple representations. Meta-knowledge is knowledge about problem-solving in a domain and the latter corresponds to the ability of the agent to provide multiple representations of information. For eg., if the domain is a table of airline schedules and fares, the meta-knowledge consists in knowing how to formulate
a travel plan based on both domain information and the preferences of the traveler and then how to present it in a clear and actionable way.

**Accessibility**

An agent's traits and predispositions must be made accessible to the user. Perceptually, users must be given cues by the external representation of an agent which allow them to infer its internal traits. On the conceptual level, an agent is accessible if a user can predict what it is likely to do in a given situation on the basis of its character. Equally important is the criterion that an agent must be conceived by users as a coherent entity. It is in the area of accessibility that the idea of structuring agents as dramatic characters has the greatest value.

The whole of the discussion above describes the requirements of intelligent interfaces, and its key properties. The description is from a functional perspective. But, to realize the above, the agent needs language ability to fulfill the following requirements. These reasons have been derived by considering the interactional aspects of the characteristics of intelligent interfaces described above.

- To comprehend the user's request given in natural language
- To infer the task goal from an abstract request, by collaborating with the user.
- To understand any suggestions given by the user or to proactively offer suggestions during the course of task execution.

Thus, language capability forms an important aspect of human-agent interaction. The subsequent section delves into the language capabilities of the agents and provides an exhaustive overview of the language capabilities available in the existing agents.

**2.7 Study of Existing Agents with Language Ability for Human-Agent Interaction**

The environment of an agent is composed of the following entities namely.

- Agents
- Software
- Users.

When agents have to interact with other agents, formal Agent Communication Languages like KQML, ACL (Huhns and Stephens, 1999) is used. For interaction
with other software, which may be either system software or application software or network software, the API provided by the corresponding software is used.

When an agent has to interact with users, it requires language ability to support for natural form of interaction. An agent has to interact with the user to obtain specification of the task it has to perform, to convey the result of the task execution to the user and to provide or obtain any suggestions to or from the user. Apart from interaction, some agents possess language ability at the functional level as the designated task may be language-based requiring language ability.

Since language forms an important aspect of human-agent interaction, different forms of language abilities have been attributed to many agents. The field of computational linguistics forms the major contribution for attributing language abilities to agents. This section explores the language ability that has been incorporated to agents that are available.

To study the language ability of agents, the interaction capabilities of various agents are explored into. This helped to discern the language ability available in them and also arrive upon a taxonomy which classifies the agents into various types depending upon the natural language ability available in them. This taxonomy is evolutionary in nature whereby it gives account of the language abilities of agents in every stage of the evolution starting from the lowest form of language abilities available in earlier agents to the most sophisticated form of language abilities that are available in today's agents. This taxonomy is depicted in figure 2.6.

The following discussion delves into every agent type identified in the taxonomy to provide an overview of the language ability available in the corresponding agent type. The overview consists of the following:

- **Nomenclature** attributed in this thesis corresponding to the type of language ability in the agent.
- **Description of the language ability** of the corresponding type of agent.
- **Few examples** of such agents that possess this type of language ability.
- **Advantages and disadvantages / limitations** of the language ability of this type of agent.
Figure 2.6: Evolutionary Taxonomy of language ability of Agents
2.7.1 Keyword-Based Agents

These agents work based on the keyword that they extract or identify from the task specification given by the user. The keyword is matched with the required target entity to perform search or filter functions of the agent. The agents reply using static predefined messages that are chosen by matching the context of the message to be conveyed, which is again a keyword, with the available set of predefined messages. Thus, these agents could be considered to possess the most primitive form of language ability as they do not require natural language ability to comprehend the task specification or perform the task or interact with the user.

This is because the early research on agents and the then developed agents strived to accommodate for the properties of autonomy, reactivity, pro-activity and social ability in the functional behavior of the software. Thus, the primary focus was at achieving functional intelligence, whereby natural language ability of agents did not gain much interest.

Depending upon whether they extract or identify the keywords in the given task specification, these agents could be classified respectively as follows:

- GUI-based agents
- Conversation-based agents.

2.7.1.1 GUI-Based Agents

This was the most primitive form of interaction supported by the agents to obtain the details of the task to be executed. GUI-based agents are of two types namely,

- Form-based
- Wizard-based.

Form-Based Agents

The form-based agents obtain the input or the specification of the delegated task through a GUI form. The form contained menus with corresponding entry columns for collecting all the details regarding the task the agent is to perform. The user has to either fill in the entry columns or choose from the options available against the entry
columns to provide the task specification. The filled in values have to be comprehensible by the agent without the need for additional processing.

The agent works by extracting only the inputs from the entered columns. The entry columns are mapped to predefined internal fields that store the inputs given by the user. These parameters are matched with available documents or data to perform a particular function. The reply is given in the form of static predefined messages as in the examples given subsequently.

Examples
Mladenic (Mladenic, 1999) provides a survey of several text learning agents that are used for information retrieval or assisting the user in web browsing. Though the machine learning technique that they use for information retrieval are different (content-based approach, collaborative approach), they rely on the keywords provided by the user to perform a context-sensitive retrieval or information filtering or web browsing assistance to the users. These agents use form-based interface to obtain the task specification from the user.

Maes (Maes, 1994) discusses about several agents like that of Maxims (email filtering), Newt (news filtering) that help to reduce the work and information overload of the users. These agents too make use of keywords obtained through forms to perform email filtering, news filtering etc.

NewsHound (Caglayan, 1997) is a personalized newspaper that searches the stories in the San Jose Mercury News as well as several other newspapers to find articles that match a user’s profile. Selected articles matching the user’s profile are sent to the user’s Internet address by email.

Advantage
• From a developer’s perspective, keyword based agents have the advantage in that it is very simple to implement as the language ability of the agent is limited to keyword processing.
Limitations

- From a user's perspective, the form-based agents pose a difficulty in that, they require that the user is generally proficient in using software - to be aware of which of the menus to open and which forms to fill for which task or options to be invoked by him in the agent.

- The user's interactions are constrained in that, he has to choose from the available options against the various entries or provide keyword-based responses or input parameters that could be understood by the agent without further processing. This form of interaction is not suitable to be used in sophisticated environments of today because of the high degree of direct manipulation of the agent imposed on the user.

- The human-agent interaction is available only for giving specifications of the task and the agent does not have interaction abilities to provide or get suggestions while executing the task, which is an important of interaction during task delegation.

- Global use of these agents is limited because of the support of a single language only

Wizard-based Agents

A Wizard is a help system that guides the user through a specific task using a dialog metaphor. It collects all the required information regarding the task to perform through a series of questions posed by the agent requiring answers from the user. The questions are posed and answers are obtained through GUI forms. The forms provide the entry columns to obtain the user response regarding the required details of the task specification. As described in the form-based agents, the form entries made by the user are constrained. The agent conveys its response by static predefined messages. Thus, as is the case with form-based agents, the wizard-based agent does not have or need natural language ability for task comprehension, execution and conveying response to the user.

Examples

Wizards are used in Microsoft desktop applications to provide task-oriented help. For example, the chart wizard in Microsoft Excel helps the user for constructing charts.
Lotus Smart Assistant, MacOS Expert, and Claris Assistant are other examples of wizard-based agents.

Advantages

- Since the wizards provide for an interactive question answer type of interaction with the user, it gives the illusion as though the agent has natural language interaction capability.
- The user need not worry about which of the menus to open and entries to be filled because the wizard takes control of the entire interaction process by asking the relevant queries required to collect the complete task specification from the user.

Limitation

- The messages or questions are predefined and are displayed to the user in a predefined order. This requires only message sequencing skills to identify which of the response message to display based on the task specification to be collected or reply given by the user. Hence, natural language ability is not involved or necessary in this form of agents.

2.7.1.2 Conversation-based Agents

The user can carry out a free conversation with the agent in natural language. The agent also responds to the user in natural language. Internally, it does not perform any natural language processing to comprehend the user’s input or natural language generation to provide response to the user. It only identifies the keyword in the user’s conversation and provides a predefined response that is appropriate to it.

Example

For example, Julia is a conversational agent (chatterbot) (Caglayan, 1997) that can carry out conversation with the user regarding many topics. Its topic-oriented responses are encoded in an activation network. The conversation generation is only a retrieval of the appropriate conversation from the network. It creates the illusion of natural language interaction by employing several tricks like,

- Repeating user’s input in questions
- Admitting ignorance
- Changing the topic of conversation
- Using conversational statements
- Using humorous statements, etc.

Advantage

- These agents bring the user outside the confines of the GUI-based interface by enabling him to interact naturally with it.

Limitations

- These agents though provide the illusion of natural language conversation, they work only based on identifying the keyword in the conversation of the user.
- Since the agent basically works only on keyword matching, it does not possess the language ability that is required to support indirect manipulation of user interface demanded by the modern dynamic environments.

2.7.2 Natural Language Understanding Agents

Information Management is one of the major applications where agents are extensively being used (Jennings & Wooldridge, 2002). The information management functions pertain to information retrieval, search, filtering etc. The use of agents helps to provide a tremendous increase in performance by enabling a more contextual and comprehensive information management functions. These functions are fundamental utilities for IT savvy users as well as novices. To accommodate the latter type of users, the need to provide for natural form of interaction was realized. Hence, the agents were incorporated with natural language understanding capability to comprehend the natural language requests and thereby service all classes of users. This helps to eliminate the constraints in the form of interaction with the agent as that of GUI-based agents.

Natural Language Understanding was facilitated by using natural language processing techniques that were contributed by the field of Computational Linguistics. By this, the agent can comprehend the user's request with a higher degree of precision than that of keyword-based technique and perform the appropriate required function.
However, the agent is attributed with language ability that is required to comprehend the request only. The agent conveys the result of execution along with static messages and collaboration is not in the scope of the agent.

**Examples**

NIAGENT (Yang et al, 1997) is a natural language processing based agent that understands a user’s natural query and searches for the required information in WWW.

Bindiganavale et al (Bindiganavale et al, 2000) describe about how Smart Avatars’ (agents) behavior could be altered by using natural language instructions. Smart avatars are virtual human representations controlled by real people. Given instructions interactively, smart avatars can act as autonomous or reactive agents. During a real-time simulation, a user should be able to dynamically refine his or her avatar’s behavior in reaction to simulated stimuli without having to undertake a lengthy offline programming session.

The Smart Avatars require language ability to comprehend the user’s instructions. The result is conveyed in the form of change of behavior that is graphically visible to the user in the simulation environment. Collaboration is beyond the scope of the avatars.

A similar work is reported by Tanaka et al (Tanaka et al. 2002) about the Kairai system which is composed animated agents that perform actions based on the instructions given by the user.

Several search agents and information retrieval agents listed in the agentland.com (ws1) and MIT agent projects (ws2) also provide for natural language interface.

**Advantage**

- These agents are better than the keyword-based agents in two ways viz.,
  - Providing for natural form of interaction
  - Comprehending the request by using natural language processing.

The former removes the interaction constraints of GUI-based interaction whose limitations are given before. Secondly, they do not work on mere keywords, but
possess language ability to comprehend the request using NLP and then act based on the comprehension. Hence, reaction has a higher level of precision.

Limitations

- They do not possess natural language responding and collaboration capabilities which are beyond the scope of these agents.
- They support a single language and hence cannot cater to the global users.

2.7.3 Multilingual Natural Language Understanding Agents

The necessity for global reach imposes the requirement on agents to support multiple languages. Hence, agents supporting multiple languages are also available. These agents can be further classified as follows based on the technique that they use for supporting multiple languages.

- Natural language Processing based Multilingual NLU Agents
- Translation based Multilingual NLU Agents.

The need to support multiple languages provides alternatives regarding how the agent comprehends the multiple languages. The agent could either comprehend every language individually using the corresponding NLP or translate the task specification presented to it in any of the supported languages to a single target language and then comprehend the request using the target language NLP. This resulted in the above two types of agents respectively. Agent implementations in both alternatives are available.

2.7.3.1 Natural language Processing Multilingual NLU Agents

These agents understand a request by performing NLU in the corresponding language.

Examples

Jabot (Read and Barcena, 1998) is a multilingual Java-based intelligent agent for web sites. It accepts queries in Spanish and English to readily provide details regarding the contents of the web site to which it is associated with.

Another example is the Intelligent Agent-based Multilingual Information Retrieval System (Idicula and Peter, 2005) which helps for information retrieval from a database using natural language. The system uses NLP techniques to extract the
meaning of the query and retrieve information from data stores and present the information to the user. The languages supported are Malayalam and Hindi. This system makes use of agents for the various function of NLP viz., Morph Agent, Parser Agent, etc. Each of these agents is capable of performing the corresponding functions in all the languages supported.

2.7.3.2 Translation-based NLU Agents

These agents use translation to process a natural language request. Usually, these agents are available as simple translators or part of translator systems and multi-agent systems. In a translator system, it contributes to the translation process. In a multi-agent system, it translates a natural language request to interpret the natural request so that other functional modules or agents can act on the request. This agent is unlike the above Natural language processing Multilingual NLU agents that perform a specific function in addition to processing the natural language request. An example of each of the above types of translation agents is given below.

Examples

Globuddy2 (ws3) is a translator agent developed by MIT Media Laboratories. It runs on a PDA and can translate phrases given to it. It is developed for people who are traveling in foreign countries. It is better than other translators in that it does not require the user to type the entire text to be translated. It has a vast knowledge base of common sense facts and relationships. Using this it is able to expand on the user's translation request and provide words and phrases related to the user's situation.

Multitrans4 is a comprehensive suite of next-generation translation software that employs a translation agent. Multitrans4 could be integrated with any document editors or content management systems to perform the translation of the associated contents.

Hamard et al (Hamard et al, 1999) propose a digital libraries system based on Multi-level agents (Multi-agent system). Here, the translation agents are used to translate user's natural language request for a library access and transfer it to the query handling agent which proceeds to manipulate the query with the help of other agents.
in the multi-agent system. Translation agents for handling English and French languages are available in the system.

**Advantage**

- Multilingual Natural language understanding enables an agent to service users of multiple languages enabling global reach. The two techniques viz. NLP and translation available for natural language understanding provide alternative ways for understanding a request.

**Drawbacks and Limitations**

- Among the two techniques used for achieving multilingualism, using translation for handling multiple languages has many drawbacks as follows.
  - Inherent ambiguities of translation
  - Comprehension of request and hence precision of action performed depends upon the precision of translation and
  - Overhead of translation before carrying out NLP in target language.
- Though natural language understanding ability helped in comprehending the request, it did not help the agent in communicating back with the user naturally which is an important requirement of agents. This capability is very much required when the agent has to proactively offer suggestions to the user or collaborate with the user to enable him to complete a task.

To fulfill the above requirement, the agents with dialoguing capabilities were developed. This is described in the subsequent section.

**2.7.4 Dialoguing Agents**

Software agents became extensively prevalent in the software domain due to their unique features such as their independent control and decision making skills, adaptation, intelligence, flexibility etc., which they could exhibit in their behavior. These features enabled them to be used in service-oriented domains like manufacturing, process control, e-commerce, healthcare, entertainment etc., and the list is ever-growing. In these domains, the agents have to service common user
requests and require extensive interaction with the user to enable him to carry out his request successfully.

For these types of agents, NLU alone would not be sufficient to provide their services. Even to carry out a simple interaction with the user, the agents need language generation to realize the context to be conveyed in a particular language and dialogue management facility to keep track of the context of the interaction, in addition to NLU. NLU, dialogue management, and language generation together constitute for a simple dialogue system. These dialogue systems are an important contribution of the field of computational linguistics. Mc Tear provides a survey of these dialogue systems (Mc Tear, 2002). These contributions were profoundly used in developing simple dialogue agents that can carry out simple forms of interactions with the user. Since, the language handling ability extends beyond natural language understanding, it could be considered as the next level of language ability attributed to agents. Since spoken mode of interaction was also facilitated by the renditions of the field of computational linguistics, most of the dialogue agents have spoken capabilities and are known as conversational agents. Some agents also facilitated multi-modal interaction by supporting interaction through keyboard, voice, mouse, gestures, etc.

Although the contributions of computational linguistics rendered an agent interactive, from an agent perspective, the interactive capability had to be augmented to accommodate the ability to provide for proactive help, cooperate with the user and enable him to achieve his task. These requirements resulted in the evolution of the simple dialoguing capability of the agent to provide for collaborative dialoguing, which fulfills the above requirements. The following types of dialogue agents could be discerned from the literature along the evolution contour. The evolution constitutes a contour because the degree of language ability of agents and the complexity of the dialogue manager also increases with every evolution.

- Agents with simple interaction
- Agents with required interaction
- Agents with reasoned interaction
- Agents with collaborative interaction.
This is because the nature of interaction slowly migrates from a static-based form to a more dynamic and comprehensive form. This is described below.

2.7.4.1 Agents with simple interaction

These agents use natural language interaction to collect the task specification from the user. Here, the control of conversation lies solely with the agent, whereby, it asks the user a series of questions, ignoring anything the user says that is not a direct answer to the system’s question and then goes on with the next question until it obtains the required task specification from the user. This is similar to the wizard that guides the user through an interactive question answer session using forms to obtain the user specification of a task. The only difference is that the interaction is carried out in natural language. Hence, the interaction is more of a static form as it poses the required predefined sequence of questions to collect the task specification. These agents are also called system-initiative agents (Jurafsky and Martin, 2000).

Another alternative is the user-initiative dialogue agents (Jurafsky and Martin, 2000) which are initiated only when the user gives a query. They provide the answer to the queries posed by the user. Thus, the dialogue process is controlled by the user. These agents are generally used for stateless database querying systems, where the user asks single questions of the system, which the system converts into SQL database queries, and returns the results from some database.

Advantage

- Helps to provide a flexible form of interaction as it supports NLU, NLG and dialogue handling.

Drawbacks and Limitations

- In both the above types of agents, there is only single initiative, either by the agent or by the user. These agents are not practically useful while considering a majority of problems. For example, the system-initiative agent requires that the user answers exactly the question that the agent asked. But, most often, users tend to provide more answers in a single reply which the agent does not accommodate. This may lead to an annoying conversation.
In both the above agents, language ability is required for processing the reply given by the user to extract the answer for the posed query. Language generation is more of a static template-based form, and most work is required only for sequencing of queries to be posed to the user.

2.7.4.2 Agents with Required Interaction

The main drawback of the above type of agent is that the control of conversation lies only with the agent and the agent ignores any other details given by the user except the direct answer for the posed query. These could be overcome if the agent allows the user to take initiative and express his requirements freely and the agent interprets every detail given by the user. The agent takes the initiative to disambiguate the user’s request only if anomalies exist. Thus, the initiative or control of the conversation alternates between the agent and the user. Therefore, these agents are also called as mixed-initiative dialogue agents (Jurafsky and Martin, 2000). The Mercury flight reservation system is an example that follows this type of interaction.

Advantage

- These agents possess a higher degree of natural language understanding to process and analyze a request and extract the details whichever is relevant for a request, and determine which of them are missing

Limitation

- It needs language generation to pose queries only for the missing details, which again is of a fixed template-based structure.

2.7.4.3 Agents with reasoned interaction

The above two types of agents are only capable of limited domain-specific conversations. This is because, semantic interpretation and language generation processes in them are based only on what is needed to procure the required task specification details from the user, which is similar to a form-based agent, except for the natural language interaction. In order to be usable for more than just form filling applications, a conversational agent needs to be able to reason why the user has asked
a question, made a proposal, or rejected a suggestion and ask clarification questions, and suggest plans. This necessity was addressed to only from this agent onwards.

**Examples**
The TRIPS system (Allen and Ferguson, 2001) and the DIPPER (Bos et al, 2003) system are few examples that support this kind of reasoned dynamic interaction using the Information State Update approach (Traum and Larsson, 2003). A probabilistic architecture which is an extension of the information-state approach – the Markov decision process model has also been developed. This model uses a probabilistic way of deciding on the proper actions given the current state. (Levin et al, 2000) propose an architecture using this model.

**Advantage**
- This agent possesses the necessary language ability in its NLU component, more than that of the above two types of dialogue agents, to support and comprehend a free unconstrained form of interaction unlike in the above two agents. The NLG also supports for a more dynamic form of interaction suitable to express the dialogue act generated verbally.

**Limitation**
- This type of agents do not possess collaborative capabilities to provide alternate suggestions to the user when his request could not be exactly satisfied

2.7.4.4 Agents with Collaborative Interaction

Providing for collaborative interaction is the next challenge to be handled by the dialogue agents. This necessity was realized for two reasons:

- To provide indirect manipulation of human interface
- To realize anthropomorphic / embodied agents.

Research in developing adaptive interfaces pertaining to the field of human-computer interaction emphasized that the agent should be able to act on very abstract task specifications imposing minimal overhead on the user to specify the required task. This requirement was also necessitated by the need to accommodate novice users who
feel at home in interacting naturally as well as who may be prone to providing ambiguous task specifications due to their low profile proficiency in interacting with agents, or in general, any software. In these cases, the agent has to cooperate with the user to obtain the required details, offer necessary alternative suggestions and enable him to complete his task.


The collaborative interaction is achieved using a plan-based approach. This is an advanced approach used in developing dialogue agents. Its design is based on the BDI model (Wobcke et al, 2005). In this approach, every action performed by the agent is implemented as a plan. Every plan has a set of pre-conditions, post-conditions / effects and a plan body. A plan is chosen for execution only if its pre-conditions are satisfied. The post-conditions become true on execution of the plan. The body of the plan is a set of partially ordered goal states that must be achieved in performing the action.

Examples
Today, there are a number of collaborative dialogue agents available. They are also used for a variety of purposes as follows:

Dan Bohus (ws6), Larsson (ws7) and Mc Tear (ws8) provide a list of dialogue agents / conversational agents that perform various functions. From these lists, it is found that dialogue agents / conversational agents are prevalently used in travel planning and arrangement domains.
(ws2) gives a recent list of agent projects that provide for collaborative natural language interaction for the delivered functionality. Some of them are Storied Navigation (Shen, 2007), Goal-Oriented User Interface for Personalized Semantic Search (Faaborg and Leiberman, 2006), Reducing Complexity in Consumer Electronic Interfaces (Leiberman and Espinosa, 2007), i-Seek: An Intelligent System for Eliciting and Explaining Knowledge (Kumar, 2005), Woodstein (Wagner and Lieberman, 2003), and Collaborative Storytelling with an Embodied Conversational Agent (Wang, 2003) and Aria: An Agent for Integrated Annotation and Retrieval of Images (Leiberman and Liu, 2002). CALO (ws13) is another new project on collaborative agents initiated under PAL (ws14).

Microsoft agents (ws5) provide interfaces to Microsoft Office suite of applications like office, excel, powerpoint, etc.

SmartKom (Reithinger et al, 2003) is an adaptive and flexible multi-modal access interface to multiple applications that range from consumer electronics control to mobile services. This enables the user to get a homogeneous and pleasing interaction experience through an anthropomorphic, personalized interaction agent called Smartakus to whom the user delegates the task to be solved. Both communication partners collaborate during the problem solving process in which the personalized agent accesses the background services.

The Open Agent Architecture (Martin et al, 1999) consists of a user-interface agent that interfaces between the user and multiple back-end agents. It receives task specification from the user through its multimodal interface, uses a natural language processing agent to interpret the task and delegates the work to the appropriate task agent in the MAS to carry out the task. The result of execution is also conveyed through this user-interface agent.

A spoken dialogue agent architecture is proposed by Araki (Araki, 2003), which can control virtual electronic devices via web services. Each device is assumed to have web service interface which can register its control functions and command ontology to the server. The dialogue agent acquires dialogue knowledge from the server and dynamically changes its dialogue transition rule according to the status of the device.
Advantage

- This type of agent possesses a dynamic and flexible form of interaction as required by intelligent interfaces.

Limitation

- Since the agent could support a single language only, it cannot cater its services to global users.

Thus, research in dialogue agents resulted in developing agent with the various desired forms of interaction that could be used to interface various back-end systems as described above. Another parallel research that happened in dialogue agents is in developing multilingual dialogue agents. Here, the focus was on making the agent support the interactions in more than one language. This is described in the subsequent section below.

2.7.5 Multilingual Dialoguing Agents

The necessity to address the language diversity that exists globally was also realized in the dialogue agents’ domain. This resulted in the development of multilingual dialogue agents that supports multiple languages and can interact with the user in his required language (Massaro et al, 2006), (Papadakis et al, 2006), (Pelé et al, 2005). Thereby, the agent is rendered useful to the people of the corresponding languages.

The various ways by which the agents were made to support multiple language interactions are using

- Parallel language agents
- Internationalization and Localization
- Translation
- Ontologies.

2.7.5.1 Multilingualism Using Parallel Language Agents

Here, multilingual support is achieved by using individual agents or NLP resources each of which is capable of handling a particular language.
Examples

King et al (King et al, 2003) describe about a bilingual conversational agent called Kare which supports the languages English and Maori. The agent possesses a collection of NLP resources pertaining to English and Maori.

(Turunen & Hakulinen, 2000) propose Jaspis which is an agent-based adaptive multilingual speech application development framework. In Jaspis, different parts of the interaction process are distributed into independent units that are specialized in small tasks. Each unit is implemented using various agents. To handle multilingual dialogues, input agents and presentation agents that are specialized in the required languages are used. The input agent of a particular language processes the inputs received in the corresponding language. To deliver response, the required presentation agent that generates messages in the required language is chosen by the presentation evaluator.

Advantages

- Support for multiple languages.
- Using individual agents for every language provides for a simple modular and elegant design for handling multilingual interaction.

Drawbacks

- System is overloaded with many agents.
- Management of agents becomes complex with the increase of the number of agents.
- Development cost, resources and time is incurred for inclusion of every language.
- Cost of update is high.
- Additional space requirements to accommodate all agent versions.
- Language ability of an individual agent is limited to a single language only.

2.7.5.2 Multilingualism Using Internationalization and Localization

Internationalization or 118N (18 letters between I and N) is the process of designing and developing software so that it can be adapted to various locales and regions.
without further engineering changes (Huang et al). A locale is a set of conventions affected or determined by human language and customs, as defined within a particular geographical location (Joshi et al, 2001). For example, language, other cultural information such as date, time, font, currency etc. of a particular location constitute for a locale.

Localization (L10N) is the subsequent process of translating and adapting an internationalized product to a given market’s cultural conventions, that is, customizing a product for a particular locale (Joshi et al, 2001) (ws4).

Examples

Connell (Connell, 2000) has emphasized the necessity to make agent benefits available across cultural and linguistic boundaries and proposed the use of the above method for localizing agents. This literature reports a pilot study in designing the look, feel, sound and dialogue of an autonomous intelligent agent accessed through a web browser. Machine translation is to be used for localizing of agent interactions.

Microsoft agents (ws5) support for multilingualism by providing individual localization modules for the various languages supported by it. The agent software is provided in terms of core components that are independent of language and localization modules that help to contribute locale aspects to the software. Since it provides for multi-localization, the same agent software can be localized to multiple languages, thereby helping to provide for interaction in the corresponding languages.

Advantage

- Internationalization and Localization provides a better method for achieving multiple language support. This is because, it helps to minimize the cost of development as incurred in the parallel language agent versions as only localization effort is required for every language to be included.

Limitation

- The localized agent can handle only a single language to which it is localized. Hence, the language ability of the agent is limited to a single language only.
2.7.5.3 Multilingualism using Translation

Here, the agent supports for multiple language by using a translator that translates one language to another.

Example

Ritter et al (Ritter et al, 1999) propose a translation agent that translates a spoken utterance to another language. This agent also supports for multi-modal interaction. Here, language translation is between the languages English and German.

Advantage

- Translation provides for a simple means of supporting multiple languages.

Drawbacks

- Ambiguities of translation.
- Performance of agent depends on performance of translation
- Overhead of translation process.

2.7.5.4 Multilingualism Using Ontologies

Ontologies are explicit specifications of conceptualizations which are now recognized as important components of information systems and information processing (Estival et al. 2004) This is because conceptualizing a domain is a prerequisite for understanding the domain and processing information about the domain. Also, information systems are no longer isolated, but, are parts of the global information system and need to be interoperable. Hence, conceptualizations and ontologies are required for all kinds of information systems and information processing (Estival et al, 2004). Ontologies are found to be very useful also because they facilitate better communication between people and organization, facilitating better specification in systems engineering, reliability in development of information systems (Ren and Shi, 2000). In recent years, the use of ontologies for electronic commerce, enterprise integration, knowledge management, and access to information on the web is widely prevalent (Ren and Shi, 2000).

Since, ontologies help to conceptualize the domain of an information system, they were naturally found suitable for natural language processing, whereby the natural
language request could be processed and represented in terms of the domain concepts and their relations. Thus, ontologies were used for the following purposes:

- Natural language processing
- For providing multilingual support in many types of information systems
- For developing agents that can support multiple languages.

The following discussion gives examples of each of these.

**Examples**

Estival et al (Estival et al, 2004) describe about ontology-based natural language processing that was used in the FOCAL (Future Operations Centre Analysis Laboratory) project. FOCAL is a research project whose goal is to pioneer a paradigm shift in command environments through a superior use of capability and greater situation awareness. It contains a large-screen semi-immersive virtual reality environment as its primary display, allowing vast quantities of information to be displayed. Spoken dialogue with virtual characters known as VAs (Virtual Advisers) is one of the means of delivering information.

Multilingual Knowledge-Based European Electronic Marketplace (MKBEEM) is a research project that makes use of ontologies to mediate between languages and to infer answers to questions in MKBEEM (Lehtola et al, 2003). The languages supported are Finnish, French, English and Spanish.

Pazienza et al (Pazienza et al, 2003) propose an ontology architecture for organizing the information provided by different resources in several languages. This ontology is present inside CROSSMARC which is an European research project supporting development of an agent-based multilingual information extraction system from web pages. CROSSMARC ontology aims to support all the different activities carried on by the system’s agents through the use of this ontology.

Pazienza et al (Pazienza et al, 2005), describe about an ontology-based question answering for web sites where the agents involved in the system use ontology mapping to handle multilingual queries.
Ren and Shi (Ren and Shi, 2000) propose a general ontology based multilingual multi-function multimedia intelligent system called as MMM-IS. MMM-IS is a complex system with multiple functions that can deal with multiple languages and multiple media. It supports multiple natural languages like English, Chinese and Japanese. The system accepts and manipulates various forms of information such as text, sound and image.

Advantages

- The use of ontologies has brought about a major break through in handling multilingualism. This is because unlike the other approaches of handling multilingualism, ontologies help to abstract domain concepts from language expressions.
- It naturally helps to achieve language independence amidst the presence of multiple languages.
- It also provides for inclusion of further languages with minimal overhead as only the ontology mapping of the language entities to the domain concepts has to be carried out.
- Though 118N also helps to realize language independent globalized software that can be localized to different languages, it cannot provide the advantages of ontologies like, interoperability, communication, reliability etc.
- This approach helps to overcome the drawbacks of using parallel language agent versions, 118N and L10N and translation.

2.8 Summary

This chapter provided an overview of the agents with respect to their definition, properties, architectures, and agent-based systems. It also provides a review of the language ability of existing agents. The review explores into the depth of language ability starting from simple keyword matching, to natural language processing, and to collaborative dialoguing. The breadth-wise dimension of the review helps to explore the span of language ability of agents starting form a single language support to multiple language support and the techniques used for handling multiple language support in agents. Along with the description of each of these agents, the advantages
and limitations highlight the strong and weak points of the corresponding type of agent with respect to its language ability.

This review indicates the following:

- Language ability has been an inevitable aspect of agents.
- Language ability of agent varies from the simplest keyword processing language ability to collaborative multilingual dialoguing handling language ability.
- Many techniques like NLP, translation, ontologies contributed by the field of Computational Linguistics are used for realizing the language-based functions involved in human-agent interaction.
- Many techniques like parallel language resources, Internationalization-Localization, Translation and ontologies are used for achieving multilingual support in agents.
- Agents with language ability are used as single functional agents or language ability attributing agents for back-end systems, suite of applications, multi-agent systems and web service mediation in a variety of application domains.
- An evolutionary taxonomy is arrived upon based on the language abilities available in them.

From a critical analytic perspective, the review also indicates that,

- Only the dialoguing agents that support for collaborative dialoguing help to provide a comprehensive form of interaction.
- Except for the ontology-based approach, all the other approaches used for achieving multilingualism have their own drawbacks or limitations.