Chapter 2

SEMANTIC WEB & INTELLIGENT AGENTS: A REVIEW

2.1 SEMANTIC WEB

The Semantic Web (SW) aims towards transformation of information oriented web into knowledge oriented web. Semantic web makes product and service information more abundant and improves search mechanisms thus resulting in user satisfaction. The idea of SW was coined by Tim-Berner Lee in ‘The Scientific Americans’ [128] in 2001. The article described the evolution of a web that comprised largely of documents containing data and information for computers to manipulate.

“The SW is a web of actionable information i.e. the information which is derived from data by operating it with semantic theory of interpreting symbols. The semantic theory provides an account of “meaning” in which the logical connection of terms establishes interoperability between systems” [98]. SW could not be realized at the time of its inception due to lack of proper tools & techniques. Recent technological improvements have made the semantic web a reality and it has started transforming information oriented web into knowledge oriented web. Actually the term semantic web signifies an intelligent web that not only processes the information for the users but also for machines in such a way that machines can interpret and exchange information on the web, thereby, increasing the probability of relevant information retrieval. The initial waves of semantic web could be realized in the form of changes in web services and ease of internet surfing.

Unlike traditional web, semantic web is a highly dynamic and distributed system that contains incomplete and uncertain knowledge, therefore an autonomous and distributed
software system was desired to optimally exploit the knowledge and information available on the web. Tim Berners-Lee [128],[98] sensed the power of software agents [74] as these could be designed so as to infer new facts, link scattered but conceptually related data and provide higher-level algorithms necessary for managing complex systems [148],[3]. Software agents and multiagent systems have proved to be instrumental in SW implemented so far. The upcoming section discusses the details about semantic web and information about software agents is given in subsequent sections.

2.1.1 Semantic Web Architecture

Since the time of inception and publication of the original SW vision by Lee et.al [127],[128] Berners-Lee proposed four versions of a Semantic Web architecture. All of the reference architecture versions were presented by Berners-Lee in different presentations; they were never published in literature or included as part of World Wide Web Consortium (W3C) Recommendation. The different versions are depicted as V1 to V4 in Fig. 2.1 given below.

In order to support the idea of SW [127], Lee proposed first version (V1) of layered architecture in 2000 [129] containing eight layers. Lee presented the second version (V2) of the SW reference architecture thrice at different conferences [130],[131],[132] in 2003 only. In contrast to Version V1, V2 contained eleven different layers as depicted in Fig. 2.1. Berners-Lee proposed third version (V3) having twelve layers [133],[134] and fourth version [134] again having eleven layers in 2006 both of which were improvements over V2 but still these could not fulfill the expectations of Internet users. Version V4\(^1\) was later improved with the aim to enhance the functionality and efficiency of semantic web.

Fig. 2.1 The four versions of the Semantic Web reference architecture (V1-V4) proposed by Berners-Lee [16]

Although the latest version seemed to be complete and supportive enough but the major shortcoming of all of these architectures is that there exist no concise description of the intended meaning of the various layers. In fact, there seem to be inconsistencies at different levels of the layered architecture. The architectural layers are portraying both functionality as well as technology. For example, in V1, ontology is functionality required in layer 4 whereas XML is technology in layer 2, lacking the explanation of the functionality that the specific technology embodies. Also, the intention behind the stacking of layers is not clear. As shown in Fig. 2.1, few layers comprises of more than one technology leading to overloading and also ambiguity in the functioning of various layers. All the architectures contain vertical layers but the available literature does not reflect the intention behind this deviation from the orthogonal structure.
Gerber et. al in [15] evaluated the above mentioned architectures and showed that they were *not in compliance to the evaluation criterion* of the layered architecture [39],[55] and thus still there was a scope of improvement. Their work [15] highlighted following major discrepancies and irregularities in the reference architecture:

1. **Side-by-Side layers**: As shown in Fig. 2.1, URI and Unicode are depicted on the bottom layer as two separate blocks or side-by-side layers. It can be assumed that the intended implication is that the two technologies both reside on the bottom layer. However, this is contentious and inconsistent when referring to the presentation of other layers containing more than one technology such as in layer3 (refer V1) that depicts the RDF and RDF Schema technologies in one layer.

2. **Triangular Structure of the layered architecture**: The layers are staggered into a triangular structure with the lower layers wider than upper layers. It is not clear whether this means that the upper layers use only part of what is provided by lower layers, or whether a lower layer specifies additional functionality that is not used for the purpose of the Semantic Web.

3. **Mixing technologies and functionality descriptions in the naming of layers**: Since certain layers are labeled using technologies whilst others are labeled using functionality descriptions, it is not clear what the layers in any of the four versions represent. In all versions the bottom layers represent technologies while higher layers are labeled with functionality descriptions.

4. **Vertical layers**: Vertical layers depict various conventions of cryptography such as digital signatures in V1, signature and encryption in V2, V3 and crypto in V4. The precise meaning of any of these vertical layers is not specified in literature. It is possible to speculate that these layers are included in all the other layers, or that these layers reside alongside the other layers, or even that these layers only depict technologies building on top of their lower layers and excluding the upper layers.

In order to cope up with above mentioned discrepancies and irregularities, Gerber et. al [16] (2007) proposed an orthogonal layered architecture called *Comprehensive Functional Layered(CFL) architecture* for the semantic web as shown in Fig. 2.2.
CFL architecture of SW comprises of two stacks:

- Language Stack
- Security Stack

Language Stack contains eight layers [10] each containing only one element namely **Unique Identification Mechanism (UIM), Syntax Description Language (SDL), Meta-data Data Model (MDM), Ontology, Rules, Logic Framework, Proof and Trust** where:

- Unique Identification Mechanism ensures that the terms and concepts are tied to a unique definition and refer to an address on the web.
- Syntax Description Language involves languages necessary to make information available on the web pages readable for machines. XML (eXtensible Markup Language) is one of the most popular SDL, since it has been recommended as the language for the publication of web document in W3C-2006.
- Meta-data Data Model is based on RDF which is framework for describing information about resources on the web, using XML tags. RDF/RDF Schema
facilitates semantic interoperability. Recently W3C has recommended RDF to standardize the definition and use of metadata descriptions of Web-based resources.

- Ontology layer works on the terminology of the domain under consideration. Ontology is an explicit specification of conceptualization. It provides a common platform for understanding the terms and concepts of a domain for communication between people and application systems.

- Rules layer contains rules for converting a document from one RDF schema into another one. This layer includes inference rules without having negation. Using inference rules, one can draw inferences on similarity of two properties, defined in two different schemas.

- Logic Framework provides facility of writing logic into documents thus providing rules for deduction of one type of document into another type. This layer includes Predicate Logic and Quantifiers so as to facilitate deductions. Knowledge Interchange Format (KIF) is the language used to specify logic in this layer.

- Proof layer involves proof checking for validation of identity. The proof will be a chain of assertions and reasoning rules with pointers to all the supporting material.

- Trust layer requires that the reasoning system must include signature verification system. This will result into a system which can express and reason about relationships across the whole range of public key based security and trust systems.

Although, the language stack of the proposed CFL architecture for SW overcomes the shortcomings of Lee’s architecture and confirms to the existing evaluation criterion. However, the refinement of the security stack for the semantic web is still a topic of research.

Next section provides the insight into the agent technology and multiagent systems.
2.2 INTELLIGENT AGENTS

The implementation of Semantic Web revolves round intelligent agents and their societies termed as Multi-agent systems. The field of agents has many diverse researchers, approaches and ideas, which has helped to create one of the more dynamic research areas in recent years. Actually the presence of agent technology and its promising features lead to inception of idea of Semantic Web [127],[128]. The huge popularity of agent research has arisen at the time when object-oriented programming languages such as Java and C++ are proving such a success. It can be demonstrated by an overview given in the upcoming subsections, which will unfold few different agent frameworks based on various available taxonomies.

2.2.1 Origin of Software Agents

Agent research stems from the work in Distributed Artificial Intelligence (DAI) conducted in the 1970’s. Carl Hewitt proposed an Actor system [26] where each Actor had an explicit internal state and had the capability to respond to the messages of other Actors. The subsequent years focused on the more theoretical aspect of bringing intelligence to software agents, whilst the last decade has seen a huge expansion of systems to solve practical problems drawing on advances in object-oriented programming, distributed processing, the Internet, the Web and the increased digitization of information and services. The increased popularity of agents produced many different types of agents such as Belief Desire Intention (BDI) agents [17], weak or strong agents [93], negotiation agents [110],[38], distributed or multi-agent systems [78], autonomous agents [61] and even emotional agents [65]. An extensive and varied set of domain of applications of agents include military intelligence gathering, industrial production planning, network resource management, aircraft maintenance tools, financial portfolio management; and a host of e-business and internet applications such as online auctions, web mining and comparison shopping.
2.2.2 Origin of Intelligent Agents

The agent paradigm is an extension rather than a replacement of conventional systems that are either object-oriented or component-based. A software object in an object-oriented system cannot perform any action on its own where as the software components, which are logical extension of already existing objects can sometime act autonomously and can be reused in building new applications rather than starting from the scratch. Components that preserve the degree of autonomy and are able to carry out certain operations in parallel by using sensors and effectors are known as software agents [74]. A system comprising of software agents is termed as an Intelligent Software System especially when it incorporates knowledge-based technology and acts proactively on perceiving the dynamic state of its environment.

A four-tier framework shown in Fig. 2.3 forms the basis of intelligent software agents. The framework is a hierarchical structure comprising of objects, components, active documents and software agents.

![Fig. 2.3 Origin of Intelligent Agents [17]](image)

More precisely, the software components are the logical collection of homogeneous objects i.e. an object whose inner functionalities remain abstracted from the outside world and it just provides an interface to the environment acting as a component. It is an independent entity that can perform a task autonomously by taking input from its surroundings and further plays the role of an agent especially when it carries out an action that only changes the state of environment, preserving the autonomous behavior.
A single component might not be able to achieve the desired goals and hence a finite number of components may be allowed to cooperate to achieve a goal. These mutually cooperative components collectively form an active document, as shown in Fig. 2.4. An active document is a set of components that has specific contents, a structure and a proactive behavior.

![Fig. 2.4 An Active Document](image)

An active document possessing mobility, autonomy, sociality, veracity, benevolence and rationality is called as *strong agent*. It is an active and an intelligent software module that acts proactively. But acquiring all these properties doesn’t seem to be feasible due to physical, social and economical reasons; therefore instead of referring it as a weak agent, an active document is referred to as an *Active Agent* or an *Intelligent Agent*.

### 2.2.3 Intelligent Agent Taxonomies

Researchers have proposed number of contemporary taxonomies of agents and criteria required for being an intelligent software agent.

Nwana(1996) in [61] identified three primary attributes namely autonomy, learning and cooperation. Autonomy refers to the principal that agent can operate on its own without the need for human guidance. For an agent to be referred to as an intelligent agent, it must have learning abilities as they interact with their external environment. The software agents need to possess social ability i.e. the ability to interact with other agents
and possibly humans via some communication languages. The author further classifies these agents as collaborative agents, reactive agents, interface agents, information agents, hybrid agents, mobile agents and ants.

Wooldridge and Jennings (1995) in [91] distinguish agents to be weak agents and strong agents. A weak agent is autonomous, social, reactive and proactive. To be reactive, agents must perceive their environment, and respond in time to changes that occur. To be proactive, agents just don’t simply respond to their environment but they can also take the initiative and carry the task with goal-directed behavior. A strong agent follows Belief-Desire-Intention (BDI) model ([123],[17]) and have human characteristics such as trust and competence.

Jennings, Sycara & Woodridge (1998) in [118] defined agents as a computer system in an environment that can take autonomous actions to meet the objectives. According to the author, the agent receives sensory input from its environment and its actions affect the environment. The agent in such an environment is flexible enough to be responsive, pro-active and social.

Magedanz, Rothermel & Krause (1996) in [136] divided agents as local agents, networked agents and mobile agents for single-agent systems, DAI-based agents and MultiAgent Systems (MAS) respectively. Still there exist advisory agents acting as personal assistants in email management, web browsing, searching etc.

Number of researchers has classified agents as task-specific and performative agents [57] such as search agents [91], navigation agents [46], information agents [74], softbots [91],[99],[100], knowbots [67] and so on. Depending upon the responsibilities to be carried out and aims to be achieved, agents with different attributes are being designed.

2.2.4 Intelligent Agent Characteristics

The invasion of various approaches under the banner of ‘agents’ caused a need to classify and define this term. However, it quickly became apparent that everyone had their own
definition [61] due in part to the historical relationship with the AI community and the vague notion of intelligence. Numerous definitions for the agents have been proposed, but in core most have a set of defining characteristics that every agent must demonstrate. For instance, BDI agents must show explicit beliefs (knowledge perceived to be true), desires (goals) and intentions (plan to obtain goals)[77], while Woodridge and Jennings’ weak agents [91] should be autonomous, reactive and social. A definition from Franklin & Graesser [126] lists autonomous, reactive, communicative, adaptive, mobile, flexible, goal-oriented, continuous and with some form of character or emotion. Using such definitions from the literature, it has been possible to create a set of primary, secondary and tertiary agent characteristics. Primary characteristics are inherent to most of the popular agent definitions while secondary characteristics are the set of extended characteristics usually associated with agents. The third set of characteristics contains more abstract, desirable, human-like features.

**Primary Characteristics**

- **Autonomous** - An agent should be able to execute without the need for human interaction, although intermittent interaction may be required.

- **Social / Communicative** - An agent should have a high level of communication with other agents. The most common protocol for agent communication is the Knowledge Query and Manipulation Language (KQML) [135].

- **Reactive / Responsive** - An agent should be able to perceive its environment and react to changes in it.

**Secondary Characteristics**

- **Proactive** - Proactive agents do not just react to their environment but can take active steps to change that environment according to their own desires.

- **Adaptive** - Adaptive agents have the ability to adjust their behavior over time in response to internal knowledge or changes in the environment around them.
Goal-oriented / Intentions - These agents have an explicit internal plan of action to accomplish a goal or set of objectives.

Persistence / Continuous - Persistent agents have an internal state that remains consistent over time.

Mobility - Mobile agents can proactively decide to migrate to a different machine or network while maintaining persistence.

Tertiary Characteristics

Emotion - Agents with the ability to express human-like emotion or mood. Such agents might also have some form of anthropomorphic character or appearance.

Intelligence - Agents with the ability to reason, learn and adapt over time.

Honesty - Agents that believe in the truthful nature of the information they pass on.

These agent characteristics lead to many advantageous features. The very nature of agents as independent, social entities that can respond to and change their environment provides a strong foundation for building reliable, robust, flexible, extensible and scalable systems. Agents can help ease user tasks and adapt to user requirements. Despite of their many benefits, agents are not the solution to every problem. One major disadvantage of building agent systems is that the complexity of agent interactions and dynamic nature of the agents themselves make it difficult to predict agent behavior. It can cause problems in safety critical environments where outcomes need to be assured. However as the computer world is becoming increasingly networked and distributed, agents are likely to become the next engineering paradigm for system development.

2.2.5 Ontologies for Agents

Ontologies [144],[97], a term borrowed from ancient Philosophy, originated in the knowledge acquisition field and was originally defined as an “explicit specification of a conceptualization”[143]. Ontologies represent a world-view in a particular domain,
which consists of concepts, definitions and concept-relationships. The term conceptualization is described as “an intentional semantic structure, which encodes the implicit rules constraining the structure of a piece of reality” [144]. Ontologies [88] are useful because they facilitate a shared understanding in a particular domain of interest. They have a wide range of applications including knowledge engineering, knowledge representation and sharing, database design, information modeling and agent-based system design [22],[30].

The agent community has borrowed ontologies because they provide a shared worldview in which each agent can ground its beliefs and actions. Ontologies provide agents with a powerful domain of discourse. Before conversing, agents can agree to use a particular ontology, which defines some aspect of the world. Using such a shared ontology, agents can then discuss a topic with complete confidence that other agents have the same understanding of the items discussed. For instance, mutually exchanged ‘airplane’ ontology allows one agent to talk about a particular airplane e.g. Boeing 747 or Airbus320, and for the other agents to understand that Boeing 747 is a type of airplane object, which has several properties. Ontologies are playing significant role in SW [98] since it aims at automation of searching and extraction of useful information spread across the web through multiagent systems, which exploit ontologies for understanding their domain.

2.2.6 Multi-Agent Systems

Multiagent Systems (MAS)[147] evolves as a result of sociological relationships among agents. An individual agent can achieve a goal either by using its own capabilities or by successfully exploiting the capabilities of other agents. MAS are defined as community of agents arranged in a predefined manner in which at least one agent referred to as Coordinator or Parent Agent must have an ascribed set of goals and other agents in the system may adopt the goal. The goal adopting agents are called as Sociological Agents as shown in Fig.2.5.
Once an agent adopts the goal of goal originating agent, a relationship is created between the two. If the goal cannot be achieved at this level it may again be forwarded to different homogeneous or heterogeneous adjunct agents. More precisely, MAS can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities of knowledge of each problem solver.

The significant reasons for having MAS are:

1. Incomplete viewpoint of the world for each agent.
2. Lack of global system control.
3. Decentralization of data
4. Asynchronous computation.

![Multi-Agent System](image)

As shown in Fig. 2.6, a MAS inherits most of the advantages of distributed intelligence over centralized, sequential processing, a MAS is expected to be:

1. **Reliable**- MAS is more fault-tolerant and robust.
2. **Scalable**- agents can be added or deleted without greatly disrupting the system.
3. **Adaptive**- agents can re-configure themselves to suit system changes such as noise, resource allocation and faults.

4. **Concurrent**- agents can reason and perform system tasks in parallel and asynchronously, resulting in faster and flexible execution of tasks.

5. **Dynamic** – agents can collaborate to form dynamic groups to solve specific problems, pooling together resources and disband after the problems are solved releasing resources to local usage.

![Fig. 2.6 Characteristics of MAS](image)

**2.2.6.1 Classification of Agents in MAS**

The invasion of various agents in MAS caused a need to classify the agents according to the behavior these pose in a particular agent-based system. According to the behavior being posed locally or globally and using definitions from the literature, it has been possible to classify the agents as **Internal Agents, External Agents and Mediator Agents** [91] as shown in Fig. 2.7.
Internal Agents

Internal Agents are local to a particular architecture and are classified according to the role being played within the environment i.e. how they behave inside the system and with each other. Internal Agents can be

- **Cooperative Agents**: Share some common goals.
- **Self-Interested Agents**: Possess distinct goals.
- **Competitive Agents**: Possess mutually exclusive goals.
- **Destructive Agents**: Direct co-agents off beam.
- **Interface Agents**: Take the raw input and ultimately deliver the output by delegating the task to other agents. Interface agents model human managers and learn from them how to manage networks.
- **Informative Agents**: Store and forward information to co-agents.
- **Task-Oriented Agents**: Responsible to carry out the main goal. These are the actual workers.
- **Reactive Agents**: Senses the input from environment and reacts as directed.
- **Proactive Agents**: Senses the input and if not directed, decides on its own to achieve the probability of success.

**External Agents**

External Agents are referred to as dynamic agents as they possess the ability to change their residing locations. These agents move out of a system to perform a task and these may or may not return to the originating node. Those external agents that move out to different locations to gather the desired information to carry out a task and then return to originating node are referred to as *Mobile Agents* [112]. Mobile agents spread intelligence across networks. The mobility allows them to be created, deployed and terminated without disrupting the network configuration. External agents that originate at one node, keep on changing their locations and may die on any other node are referred to as *Ants* [84]. Ants follow the principle of “STIGMERGY”. *Stigmergy* is a form of indirect communication through an environment. The insect ants when move in search of food stimulate a hormone named as pheromones, which attracts other surrounding ants. The routing of an ant-based agent is pheromone distribution dependent where pheromone distribution depends upon the environment in which the agent’s properties are utilized. Swarm intelligence stems from the work of ants in which unintelligent Internal and External agents possibly belonging to heterogeneous platforms, work independently or with relative small amount of collaboration to achieve a greater goal that requires intelligence.

Internal Agents initially try to achieve the goal on their own but if they are not able to make it, then the assistance of external agents may be desired and for this kind of interaction, an interface between internal agents and external agents is required. Such an interface is accomplished through Mediator Agents (MA) that also makes service requester and provider compatible.

Table 2.1 classifies agents according to types, characteristics and use attributes. These categories are not mutually dependent and different types of agents can have
different combination of these characteristics. Some agent applications can use more than one type of agent.

**TABLE 2.1 CLASSIFICATIONS OF AGENTS**

<table>
<thead>
<tr>
<th>TYPES</th>
<th>CHARACTERISTICS</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Agents</td>
<td>Autonomy</td>
<td>Process Control</td>
</tr>
<tr>
<td>Cooperative/Domains</td>
<td>Co-operation</td>
<td>Manufacturing</td>
</tr>
<tr>
<td>Intelligent Agents</td>
<td>Learning Ability</td>
<td>Air Traffic Control</td>
</tr>
<tr>
<td>Information Agents</td>
<td>Flexibility</td>
<td>Internet Agents</td>
</tr>
<tr>
<td>Pro-active Agents</td>
<td>Resource Provider</td>
<td>Electronic Commerce</td>
</tr>
<tr>
<td>Behavior Agents</td>
<td>Social</td>
<td>Business Process</td>
</tr>
<tr>
<td>Collaborative Agents</td>
<td>Mobility</td>
<td>Medical</td>
</tr>
<tr>
<td>Learning Agents</td>
<td>Beliefs, desires &amp; Intentions</td>
<td>Finger-Print Recognition</td>
</tr>
<tr>
<td>Hybrid Agents</td>
<td>Emergent Behavior</td>
<td>Biometric Analysis</td>
</tr>
<tr>
<td>Reactive Agents</td>
<td>Co-ordination</td>
<td>Authentication</td>
</tr>
<tr>
<td>Self-Interested</td>
<td>Mutually Exclusive</td>
<td>Search Agents</td>
</tr>
<tr>
<td>Destructive Agents</td>
<td>Directs off beam</td>
<td>Viruses</td>
</tr>
<tr>
<td>Task-Oriented</td>
<td>Workers</td>
<td>Personal Assistants</td>
</tr>
<tr>
<td>Mobile Agents</td>
<td>Emergent Behavior</td>
<td>Intrusion-Detection</td>
</tr>
<tr>
<td>Ants</td>
<td>Pheromones/ Stigmergy</td>
<td>Swarm Intelligence</td>
</tr>
</tbody>
</table>

### 2.2.7 Standardization Efforts

There are currently a wide range of different agent architectures, frameworks and systems developed for both research and industrial purposes. To unify these approaches three standardization efforts have appeared with the overall aim of increasing interoperability between agent systems.
MASIF - The Mobile Agent System Interoperability Facility (MASIF) [92],[36] has been in development by the Object Management Group (OMG) since 1995 to promote interoperability and mobility among agent platforms.

KQML - The Knowledge Query Meta Language (KQML) [135] is one of the most popular and widely used protocols for defining agent-to-agent communication. KQML is the oldest project, developed in 1992 by the DARPA Knowledge Sharing Effort consortium.

FIPA - The most recent addition is Foundation for Intelligent Physical Agents (FIPA) [107], a non-profit organization created in 1996 aimed at developing software standards for maximizing interoperability within and across agent-based systems.

Of these three approaches, MASIF uses a procedure-oriented interaction model using Remote Procedure Calls (RPC) or Remote Method Invocation (RMI) technology, while both KQML and FIPA both specify a message-oriented Agent Communication Language (ACL). The ACL model used in both FIPA and KQML is based on speech act theory [91], a field of research aimed at analyzing the semantic content of vocalized messages. These standards facilitate agent interaction across hardware platforms, operating systems, programming languages and agent platforms. Recent FIPA compatibility tests have already shown successful interoperability through the transfer of ACL messages between several FIPA compliant frameworks.

2.2.8 Agent Frameworks

According to the technical report by Nguyen & Dang [47] there are over 100 products in this category. This section will focus on elaborating most appealing & promising six toolkits among the available choices.

On the most fundamental level, each framework supports three features for agent developers:
- **Creation:** Provides the ability to quickly create and run agents within a supported environment.
- **Communication:** Supports agent-to-agent communication using speech acts.
- **Discovery:** Allows agents to find new agents using a service-based discovery mechanism.

Each framework offers a unique set of additional features such as standards compliance, mobility, interoperability, knowledge-based ontologies, graphical interfaces etc.

(a) **JADE**

Java Agent DEvelopment Framework (JADE) [40],[41] is an open source Java implemented middleware framework that became one of the early systems to adopt the FIPA specifications and become FIPA compliant. It provides an efficient, scaleable, distributed environment that fully implements all aspects of the FIPA communications model. JADE was designed to ease the process of developing agent applications and it uses several mechanisms for accomplishing it. JADE abstracts the communications infrastructure away from the programmer, providing transparent mechanisms for locating both local and remote agents and initiating conversations with them. Each JADE agent runs as a single thread within an agent container, and collections of containers run within a single JVM platform as shown in Fig. 2.8.
Each container has a dedicated directory facilitator that acts as yellow pages, providing
the other agents in the container with a means to look up agents based on the services
they offer. The JADE platform also contains a front-end container with a graphical user
interface agent that allows users to interact with the platform and provide a mechanism
for controlling each agent. The interface provides methods for starting, suspending and
stopping agents as well as debugging features such as activating a sniffer to intercept
communication packets, and sending custom messages to agents.

The distributed nature of JADE allows multiple JADE Agents and other FIPA-
compliant agent platforms to communicate via the Internet Inter-ORB Protocol (IIOP).
IIOP is a message transport protocol, allowing objects to be sent across networks in a
platform independent manner. An agent communication channel, which is physically
distributed across all JADE platforms, provides agents with a transparent mechanism for
converting incoming ACL messages from an external platform into the ACLMessage
format used internally in each JADE platform and visa versa. Between containers, Java’s
Remote Method Invocation technology operates as the transport mechanism, and agents
in the same container will communicate directly with each other. When an agent
communicates with another agent, the JADE framework will select the most efficient
communications method and therefore appears transparent to the agent and the agent’s
programmer. Tests of the JADE communications model have shown linear scalability,
which mimics that of RMI when agents are distributed across a network (Vitaglione,
Quarta and Cortese (2002)). The latest version of JADE is 4.0.1 released in July 2010,
conforms to the FIPA2000 specification and currently many international companies,
projects and universities are using the framework

(b) IBM-Aglet or Aglet Software Development Kit (ASDK)

ASDK or IBM-Aglet [32],[108] is an environment for developing mobile agents based
application in JAVA. It is an open source freely available toolkit, with latest version
Aglet 2.5 alpha. It provides good graphical user interface for agent development.
It mainly comprises of two packages-The Aglet Building Environment (ABE) and the Aglet Workbench. Aglet workbench aims at developing stand alone mobile agents. The ABE (SDK) comprises of Aglet API, the Aglet Server known as Tahiti and the Agent Web Launcher called Fiji along with documentation and sample Aglets.

Aglets are basically java objects comprising of two major components i.e. Aglet Core & Aglet Proxy. Core is holder of all the internal variables and methods of an agent whereas proxy acts as an interface to the core, shielding it from any malicious interference from the outside world.

Aglet server Tahiti is an application program that works as agent server for aglets. It provides users with a good GUI and allows users to create & dispatch an agent, monitor it, dispose it off when required. It gives user the ability to set agent’s access privileges on the server. For an aglet to move to a remote host, it must have Tahiti server installed on it, which solves some of the security problems. Fiji is an applet in java which can create Aglets or retract an existing aglet into client’s web browser. This applet accepts an agent’s URL as parameter and can be embedded in a web page using HTML, like any other applet.

Aglets support both agent mobility as well as predefined movement of the agent on the network also called as Itinerary. Although aglet provides weak agent mobility but that too is restricted to its own servers. Aglet works on Mobile Agent System Interoperability Facility (MASIF). Agent migration is implemented using socket mechanism. Communication among agents is achieved using synchronous and asynchronous message passing. Agent Transfer Protocol (ATP) along with Java Agent Transfer and Communication Interface (J-ATCI) also help achieve the same.

Although Aglet platform has wide user acceptance but it doesn’t provide much security. Its security is knitted in the concept of restricting transfer of aglets only to its own servers. Due to lack of security, state of aglets can’t be stored on any other host. No such method is provided by this tool. Scalability is another problem, since aglets are not
interoperable with other platforms or their agents, due to their restriction of working with their own server. Figure 2.9 given below illustrates the structure of an aglet

![Aglet Structure](image)

**Fig. 2.9 Structure of an Aglet (clements, Papaioannou & Edwards (1997))**

(i) **Voyager**

Voyager [32],[71],[149] is an agent development tool developed by ObjectSpace, in mid-1996. ObjectSpace has been taken over by Recursion Software Inc. since 2001 and it’s now their commercial product. Latest version available is Voyager 8.0. It’s a simple yet powerful technology for creating mobile agents in Java. It was an improvement over already existing platforms like Aglets, Odyssey, Concordia etc. which only allowed developers to create agents and launch them into a network to fulfill its mission. But none allowed sending messages to a moving agent, which made it difficult to communicate with an agent once it has been launched and also for agents to communicate with other agents.

Voyager seamlessly integrated fundamental distributed computing with agent technology. It treats an agent like a special kind of object which can move independently and can continue its execution while moving around. The agents and objects are different only because an agent can move autonomously whereas an object can’t.

It allows an agent to send and receive Java messages to and from other agent, even while traversing the network, irrespective of its position in the network. It supports
synchronous, one-way and future message modes. Whenever an agent moves, it leaves behind a forwarder object which forwards the message to its new location.

Voyager provides flexible life spans for agents, by supporting a variety of life span methods:

- An agent can live until it has none local or remote references (default life span of an agent).
- Agent can live for a certain amount of time (by default for a single day).
- It can live for a particular point in time.
- It can live until it remains inactive for a specified time.
- An agent can live forever.

Moreover, an agent’s life span can be changed flexibly as required. Another attractive feature of this tool is its support for directory service, which is particularly important in launching a mobile agent from one application to another and for locating an agent after it moves to some other location in the network. Its directory structure allows creating and connecting network directories together to generate a large interlinked directory structure.

Voyager supports weak mobility of agents using RMI technique. It allows all serializable objects to be mobile using Virtual Code Compiler (VCC)[32]. VCC utility accepts any .class or .java file and produces a new remote enabled virtual class. This virtual class is used in further communications with that agent/object. Agents use moveTo() function and a callback function for migrating to a remote host. On reaching new host, the agent retrives the callback function that it sent and resumes its execution.

Voyager has an associated server called ‘voyager’ but it’s not necessary to have such server installed on all nodes in the network. Due to this reason agents created using voyager are provided restricted access on the host servers. Thus, provision of agent and host security is weak in this tool.
(d) Anchor

Anchor [124] agent toolkit is developed by Lawerence Berkeley National Laboratory, U.S.A. It facilitates the transmission and secure management of mobile agents in a heterogeneous distributed environments. This toolkit is available in BSD style license. Its architecture comprises of an agent viewer graphical user interface, Agent API, Anchor server, Anchor security manager (ASM), Anchor class loader (ACL), secure agent transfer protocol (satp) handler, Anchor Java Naming and Directory Interface (AJNDI) & Anchor Java Native Interface (AJNI) components. Fig. 2.10 given below provides the architecture of an anchor agent.

![Fig. 2.10 Architecture of Anchor Agent [124]](image)

Agent model in Anchor is based on that of Aglets. Agents are serializable java objects capable of migrating in the network. Agents are created in contexts, where context is a namespace under which agents are grouped together. Agents are accessed through their proxies, which protect the agent from any attempt of direct access to its code and methods. Also it provides location transparency to agent which means an agent is represented in a machine even if it has migrated to some other machine in the network. All messages to that agent are forwarded to its location through its proxy.

Agent server in Anchor is a run time environment which acts as backbone of this toolkit. It runs on a host and works on a specific port. It performs all system related
functions. Anchor server supports the agent migration through satp. Security is a major concern in this toolkit. Mutual authentication between agent systems is established through secure socket layer (SSL). Agents are authenticated by signing their byte codes with their private keys.

An interesting component of Anchor toolkit is Akenti which is an access control system designed to ensure controlled access of distributed resources. This component uses Public key infrastructure. Access control decisions are made using digitally signed certificated based on X.509 standard.

Integration of Akenti component in Anchor provides it strong security. AJANDI component in Anchor provides a naming service through which every agent can register and publish its current information. It also provides a directory service to enable effective searching of agents. Agent viewer component supports features like creating an agent, its dispatch, retraction, disposal, activation and deactivation and also cloning of an agent.

(e) Zeus

Zeus [29],[47],[162] is an integrated environment for the rapid development of collaborative agent applications, developed by Advanced Applications & Technology Department of British Telecommunication labs\(^2\). It is open source freely available toolkit. It is purely implemented in Java which makes it compatible with most hardware platforms. It also complies with FIPA standards.

Zeus provides support for generic agent functionality and has sophisticated support for the planning and scheduling of an agent's actions. It provides a set of software components and tools used to design, develop and organize agent systems. It has good graphical user interface and embedded components like report generation tool, statistical tool, agent and society viewer tool etc. which help in observation of application under development. Also, it allows the designers to use different negotiation techniques for testing implemented agents.

\(^2\) [http://www bt lab com](http://www.bt.lab.com)
Communication among agents is performed using Agent Communication Language (ACL) or Knowledge Query Manipulation language (KQML). Communication security is provided using public key, private key cryptography and digital signature technologies.

Major drawbacks of this toolkit include lack of support for agent mobility and its weak documentation which leads to difficulties in creation of new applications.

2.2.9 Intelligent Agent Architecture

The main aim of an agent is to fulfill user’s belief, desires and intentions without the direct intervention of human user. In MAS, various software agents that interact to achieve the human generated goals have a predefined knowledge base and an optional learning system about their user’s goals and wishes and, through adaptive reasoning, use this information to execute their user’s request. Intelligent agents should minimally possess autonomy, social aspects, veracity, benevolence and rationality. Intelligent agents are much more powerful and versatile applications than traditional object-oriented architectures.

(a) Agent’s Input Sensing Mechanism

The Intelligent Agent senses the input from its environment and reacts in accordance with the goal set, coordination rules, and knowledge base. In order to increase the probability of success, an agent is allowed to communicate with its environment or with other agents in the pool and this conversation is normally conducted with message passing mechanism. Agents coordinate their activities by sending and receiving messages. Agents perceive their inputs by either sensing from environment or through an event generated by another agent and this is communicated in the form of message as shown in Fig. 2.11.
(b) General Framework of an Intelligent Agent

The general framework of an intelligent agent is primarily based on five attributes namely *Design Autonomy, Communication Structure, Language, Mediation, and Security*. The main features of the attributes are listed in Table 2.2.

The framework for an agent’s architecture using any object-oriented language encapsulates a collection of beliefs or behaviors known as goals and the methods for selecting them. Each intention triggers itself when it matches the input in the form of desires either being sensed from the environment or generated due to an event. Depending upon the learning and sensing abilities of agents, new behavioral actions may be added periodically. The agent thread constantly checks the environment to ensure if the current desire is applicable. If it finds the desire to be a false application, it stops its intention by calling the respective methods. The agent on its own is responsible for executing and terminating itself.
### TABLE 2.2 ATTRIBUTES OF AGENTS

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>FEATURES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design Autonomy</strong></td>
<td>• Platform Dependent/ Independent</td>
</tr>
<tr>
<td><strong>Communication Structure</strong></td>
<td>• Shared or Message Based</td>
</tr>
<tr>
<td></td>
<td>• Connected or E-mail based</td>
</tr>
<tr>
<td></td>
<td>• Multi-cast, broadcast or point to point</td>
</tr>
<tr>
<td></td>
<td>• Push or Pull</td>
</tr>
<tr>
<td></td>
<td>• Synchronous/Asynchronous</td>
</tr>
<tr>
<td></td>
<td>• Serial/parallel</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>• KQML, HTTP, HTML, OLE, CORBA</td>
</tr>
<tr>
<td><strong>Mediation</strong></td>
<td>• Ontology Based/Transactional</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>• Time-based/Authentication</td>
</tr>
</tbody>
</table>

The BDI-based architecture of intelligent agent primarily comprises of five components namely an Agent, an Environment, a BeliefComponent, DesireComponent and IntentionComponent. The class diagram of design of intelligent agent architecture [17]) is shown in Fig. 2.12. Jam-packed agent-based software requires message communication mechanism, information gathering and dissemination services, event notification and lot of signaling information. These are usually provided by agent-friendly platform such as JADE.
2.2.10 Applications of Agent-Oriented Systems

According to the classification of agents, agent technology finds its usage in variety of applications given as follows:

- **Process Control:** A natural application for intelligent agents and multi agent systems are process controllers, since these are themselves autonomous reactive systems.
- **Manufacturing**: Manufacturing is classified as a complex domain organized on a distributed hierarchy. Ultimate goal is to efficiently manage production processes at the plants. A multi agent approach viewing each factory and factory component as an agent finds its application in manufacturing.

- **Telecommunication**: It consists of large, distributed networks of interconnected components, needing monitoring and managing in real time. Agents can be used in a number of different ways.

- **Air Traffic Control**: Consists of real world autonomous components, which interact (aircraft and air traffic controllers). Agents can help in managing the system.

- **Transportation Systems**: The domain of transportation is ideally suited to an agent-based approach due to its distributed nature and multi component features.

- **Information Management**: Information has grown in richness and diversity and support is required for filtering and gathering information. Using agents to search, filter and manage information can reduce information overload.

- **Electronic Commerce**: Although principally a human interaction, it can be automated in part. The simpler components like search and product matching are good examples.

- **Business Process Management**: Managers make informed decisions based on a combination of judgment and information from many depts. Agents can be used to search and provide the information required and carry out some elementary decision-making.

- **Medical Applications**: A major growth area, especially in using agents in patient monitoring and providing agent based distributed health care.
2.2.11 Advantages of Agent-Oriented Systems

The agent characteristics lead to many advantageous features. The very nature of agents as independent, social entities that can respond to and change their environment provides a strong foundation for building reliable, robust, flexible, extensible and scalable systems. Agents can help ease user tasks and adapt to user requirements. The major advantages of the agent paradigm is that it can overcome network latency, reduce network load, execute asynchronously and autonomously, adapt dynamically, can operate in heterogeneous environments, and usually have robust and fault-tolerant behavior. Since the design approach is modular, the agent architecture also reveals software reusability. Secure communication mechanisms exists to mitigate agent-to-agent, agent-to-platform, and platform-to-agent security risks. Once designed, agents can interchange their roles in order to fulfill the user’s demand. Agents are atomic and isolated i.e. they don’t get distracted easily during processing of any task. Agents can be programmed to avoid flaws of reasoning that usually humans have. The agent concept provides a scalable, modular, flexible and a robust architecture. Also if the goal is safe, agents are more reliable than object-oriented or component-based solutions. Agents are useful in reducing manual load and can serve as knowledge resources. Agents never get tired.

However the flexibility provided by the autonomy of agents, also leads to security problems since agent’s are self motivated and their intensions can’t be guided always. Agent communication which requires use of ontology, suffers due to lack of standard mechanism of ontology development, exploitation and mapping from one MAS to the other. Thus although agent technology has evolved as a potential solution for many problems but still there are miles to cover.

2.3 Conclusions

This chapter provided an insight into Semantic Web. The evolution of SW architecture along with the dimension which still needs to be worked on has been elaborated. Second part of the chapter provided details of agent technology. All intrinsic details required to have an overview of this technology, have been covered.