Introduction to Agents

1. Attributes of Agents
2. Application Scenarios
3. Requirements for Intelligent Agent Framework
4. Example Systems

Attributes of Agents

Agents, software agents, intelligent agents and mobile agents are started to appear in every area without giving the detailed description of what an agent is. A common perception is that an agent is a software unit that carry out some processes on behalf of its user (people or another agent). Besides this broad description, everybody has his/her own perception of what an agent is. Therefore, we preferred to enumerate some of the features that are hinged to agents instead of trying to give an explicit definition.

Autonomy	:	It runs in a separate thread of execution, i.e., it has its own life cycle.
Reactive	:	It gathers information from the environment through event mechanism and decides about its response about this change in the environment. Note that, it does not require that an agent should react to each change in the environment since this is agent's decision.
Proactive (goal-oriented)	:	It can initiate an activity by itself.
Sociable (communicative)	:	It communicates with other agents as well as the people.
Adaptive	:	Its behavior changes based on its past experience.
Mobile	:	It can transfer itself from host to host.
Character	:	It can have an emotional side and personality.
Flexible	:	Actions are not scripted.

The first four features are almost always observed on the products which are claimed to be a some sort of an agent. These features makes the components a self sufficient living software entity by interacting with people or other agents in the environment. Software agents paradigm is based on the thinking of the information system components (a software component or a web resource) as animates by anthropomorphizing them.

Agents can try to reach a certain goals by themselves or they can build a cooperative or competing societies depending on the problem at hand. These collection of agents are called multi-agent systems.

Application Scenarios

We can define several scenarios where agents can be used efficiently. Some of these basic scenarios can be summarized as:

The Personal Agents learn the user's preferences and provides services for scheduling meetings, travel, and filtering messages coming from various resources. The ultimate result might be to replace the business cards with the address of your Personal Agent so that this allows people to reach you. Personal Agents should learn user's preference and negotiate on behalf of their users. For example, *Portico* from General Magic is a new kind of virtual assistant. *Portico* can take messages, screen your calls, track you down when you are out of the office, check your email, even get you the latest business and stock quotes.

Updating softwares in your computer takes continuos monitoring of software vendors and upgrading your software when something is found. This process can be handled by a software upgrade agent and software upgrade process can be totally automated. A software upgrade agent can collects the new software changes and as soon as it realizes that there is something which might be upgraded then it can do so with or without its user's permission.

A mobile computing is getting widely excepted everyday because of the new cheaper technologies such as laptops. This also redefines some of the old problems and these problems have to be addressed. In mobile computing environment, clients can be disconnected for some time. During this period system should continue to work regardless of whether a client is disconnected or not. In this settings, agents can be used as a representative of the user during the disconnected time so that it can handle a transaction or query on the network as the client is connected. An agent leaves the client machine before it is disconnected from the network, and moves itself back when the client is reconnected.

Information gathering and filtering is an important process for daily life. An agent can be send to a network and it can travel from server to server to collect the necessary information as well as it can execute the necessary operations on particular servers. If the amount of data needs to be transferred between a client and server is large , moving agent from client to server definitely provides performance benefits while utilizing the network bandwith. Similarly, for e-commerce applications, the agents may meet at special mall server or they go from one store to another for shopping. It finds the required product and the best buy on behalf of its user.

Agents can exhibit emotions when they are called *emotional* agents or *anthropoid*. These agents can show some attitude and brings a richer character. These agents are especially useful for virtual training environment and gaming environments where the agents need to interact with people. There are some examples:

Julia	:	An agent in chat rooms. Julia gives directions to the clients as well as can participate the ongoing discussions.
Anthropoid cat	:	This cat has goals and exhibit emotions based on the level of satisfaction or dissatisfaction of its goals.
Herman the Bug	:	An animated insect for teaching plant biology to students.

The fundamental benefits of agent technology can be summarized as:

1. Utilization of network bandwith. Mobile agents push work to the server where the data resides so that it prevents transferring large amount of data from server to the client.

2. Fault-tolerance. In disconnected mode, an agent can handle a transaction or query on the network and as soon as the user reconnects to the system, it moves back to the user's computer. If an agent needs to travel over the network, it can choose intelligent paths and processing order of tasks

according to the current connectivity pattern in the network.

3. Parallel Processing. Concurrency can be obtained by cloning the agent or dividing the activity into multiple agents so that each agent carries out a part of the task.

Requirements for Intelligent Agent Framework

In this section we will try to summarize the possible requirements for the agent framework to develop and deploy agents. The variety of attributes attached to agent unfortunately demands a lot of different requirements spreading all over the computer science sub-topics. The main services required for agents:

1. Knowledge representation and transfer of knowledge between agents: The intelligence attribute requires that agents has some beliefs, intentions, and goals to fulfill. They need to communicate with each other to exchange information or to get service. Therefore, a proper representation and interoperable message exchange format is a required property.

2. Security: Secure communication channels between agents and/or servers is required to protect the sensitive information. This requires the encryption of messages between agents and agents or agents and servers. The authentication of agents and servers is another requirement for the protection of resources.

3. Trusted transaction: It is a main requirement for e-commerce applications. If agents are going to make it in any e-commerce application, then it should provide a transacted and secure environment for operations since business applications always include distributed transaction between components.

4. Ease of use, management and development of agents are important to become a main stream computing technology.

Knowledge representation is a long time studied problem in Artificial Intelligence. There are number of methods that can be used for representing the knowledge of an agent. A knowledge of an agent can be summarized as its beliefs, its rules, and its goals. The followings are some possible techniques:

1. Semantic nets are used to define the meaning of a concept by its relationships to other concepts. A graph data structure is used with nodes holding concepts and links with natural languages labels used to show the relationships.

2. Procedural Representation encodes facts, defines a sequence of operations for using and manipulating those facts. Visual Basic, JavaScript, and Lotus Script can be seen as a language. This representation brings a tight coupling between the knowledge and its manipulation process.

3. Relational Knowledge Representation uses tuples or records of information for an item. Those records contain a set of fields or columns defining a specific attributes and values for that individual. Unfortunately, this representation is insufficient to define relationships between concepts or objects.

4. Hierarchical representation pursues inheritable knowledge. The relationships and shared attributes between classes of objects are expressed in the object hierarchies ("IS-A" relationship).

5. Frame representation defines two abstractions: slot and filler. The slots are data values or container of attribute values. The Fillers are procedures attached to those slots and activated whenever the before, during, or after the value of the slot is changed. Frames can be linked together to build hierarchical relationships ("IS-A") as well as has-part relationship. For example, an instance of a Java object can be used to define a frame. The attributes represent the slots. The filler mechanism is defined based on events. The interested objects register themselves for particular attribute pdates. The updates of those attributes are handled in methods and proper events are produced for the interested objects to inform the change before and after the update.

6. Rule-based representation uses the first order predicate logic to code rules and facts. Rules are composed of two parts: antecedent and consequent. In the antecedent, there may be some logical operations for testing the current knowledge and some functions called sensors() to gather the information about the environment. These sensors allow agents to interact with external entities. Similarly, in the consequent, there can be new facts which can be added to the agent's knowledge base and some functions called effectors() to interact with the environment for various acts such as starting an engine, informing another agent, etc. This technology is widely pursued by Expert Systems and Automatic Theorem Provers.

The beliefs that an agent contains in its internal data structure represents the its Knowledge Base. This may contain facts, rules for inference and ways to interact with its environment to cause a change or to be informed about the change in the environment. Adaptability or learning feature of agent require to process this knowledge and draw a new concepts based on its experience since agents interact with the environment and other agents to accumulate some additional information which can be used to do generalization or make decisions. Algorithms which can be used for learning comes from AI techniques. We can summarize a few here.

1. Decision Tree technique builds a tree based on the information theory. On each level, system tries to split the data set based on the attribute that contains the most information according to Shannon's definition. Leaf nodes represents the final classification and nodes denotes the decision nodes.

2. Supervised Learning Algorithms such as Back Propagation Learning and other neural network learning algorithms. For example, a simple back propagation learning algorithm can try to learn the user's preferences to provide a better service.

3. Unsupervised learning algorithms such as Kohonen Self Organizing Map (SOM) and other similar unsupervised algorithms that we can find any neural network and artificial intelligence text books.

4. Genetic Algorithms can be used for adaptation of rules in the rule-based systems to enrich the knowledge base. Similarly, GA can be used for an optimization problems when a certain fitness function is defined.

5. For rule based knowledge representation systems:

    1. The forward chaining algorithm generates new data by the simple and straightforward application on firing of the rules. An agent keeps its rules and facts in its knowledge base. Algorithm selects the best matching rule and fires the rule. This process is called as

inference and the component doing this process is called **inference engine**. Inference engine uses agent's knowledge base to start with its beliefs (facts) and puts them into another area called working memory. Engine collects all obtained data into working memory. Data in working memory is used to do matching. All the rules are selected when their antecedent is satisfied. These rules constitute a conflict set. Then one rule is selected from this set and its consequent is executed. This phase (firing) might update the facts in the working memory or execute an effector.

Matching phase may take a long time because of the size of knowledge base. A rete algorithm is developed to reduce this time. Algorithm builds a network data structure to manage dependencies between the data, condition tests, and rules, and minimizes the number of tests required for each match operation.

2. Backward chaining (goal directed inferencing) starts from a consequence or goal clause and goes backwards by using rules. It tries to answer questions about whether the consequent is true or not based on the knowledge it has.

Sociable feature requires that agents should communicate with each other to cooperate, compete, or use services. This requires an interoperable communication language, formally Agent Communication Language (ACL). This language can define two things: mental agency or social agency. Mental agency refers to the mental concepts such as beliefs and intentions. Social agency refers to the communication perspective. Interoperability calls for standard organizations. There are several initiatives in this direction such as ARPA sponsored Knowledge Sharing Effort (KSE), The Foundation for Intelligent Physical Agents (FIPA), and OMG Mobile Agent Facility (MAF).

ARPA sponsored Knowledge Sharing Effort (KSE)

ARPA sponsored Knowledge Sharing Effort (KSE) tried to develop techniques and methodology for building sharable and reusable large-scale knowledge bases and knowledge based systems. The Knowledge Sharing Effort is organized around working groups:

1. *Interlingua*: concerned with translation between different representation languages.

2. *KRSS (Knowledge Representation System Specification)*: concerned with defining common constructs within families of representation languages.

3. *External Interfaces*: concerned with run-time interactions between knowledge based systems and other modules in a run-time environment.

4. *Shared, Reusable Knowledge Bases*: concerned with facilitating consensus on contents of sharable knowledge bases, with sub-interests in shared knowledge for particular topic areas and in topic-independent development tools/methodologies.

These working groups produced several specs but the most interesting ones are: The Knowledge Interchange Format (KIF) and The Knowledge Query and Manipulation Language (KQML).

Knowledge Interchange Format (KIF) defines a common language for expressing the context of a knowledge base to exchange. KIF proposed to use the first order predicate logic to describe things within computer systems so that it can be used as "interlingua". The syntax of KIF is a prefix version of first order predicate calculus and provides supports for non-monotonic reasoning and definitions. KIF can be used to encode a knowledge about knowledge. KIF syntax allows to define a procedure.

The Knowledge Query and Manipulation Language (KQML) focuses on message formats and message-handling protocols between running agents. The syntax of KQML uses a balanced parenthesis list like lisp. The first element of the list gives the performative and the rest of the message contains the arguments of this performative keyword/value pairs. KQML describes 32 performatives based on speech act theory.

Speech Act Theory

Speech Act theory proposes that a language is not only used for making statement but it also performs actions. For example, when someone asks to do something, he/she already causes an action. A syntactic form of speech acts starts with "I hereby request ?." or "I hereby declare ?" kind of phrases. Those phrases are also known as "performatives". Ofcourse, someone can think of some other verb that can not be put into this phrase such as "solve", then these verbs can not be accepted as speech acts. Also verbs in speech acts are sub-categorized as the following:

- Assertives (informes)
- Directives (request/query)
- Commissives (promising)
- Prohibitives
- Permissives
- Declaratives (causes events)
- Expressives (expresses emotions)

Using speech act theory eases the some ambiguous semantic resolution compared to the natural languages. But still the semantics of speech acts for a particular agent, completely depends on the agent's belief, intention, knowledge about how to carry out the operation, and the society to whom an agent belongs.

External interactions can be an interaction between knowledge base systems and/or modules during run time. KQML defines a message format and a message-handling protocol. A special agent called facilitator is responsible for name resolution, registration of databases and/or services offered by agents, and communication services. KQML has three layers: the content layer, the communication layer, and the message layer. The content layer represents the content of the message and it can describe the language that the message is coded in. The communication layer adds the lower level communication parameters. For example, it can be the identity of the sender and recipient. The message layer determines the kinds of interactions (speech act/performative) between KQML-speaking agents and the protocol to be used to deliver the message.

A communication facilitator is an agent that performs various useful communication services such as maintaining a registry of service name?s, forwarding messages to named services, routing messages based on content, providing "matchmaking" between information providers and clients, and providing mediation and translation services. The services provided by facilitator agents can be summarized as:

- Subscribe: An agent A request that the facilitator F monitors the truth of the clause and informs the agent A.

- Matchmaking: An agent A requests that the facilitator F finds an agent that can process the required performative. Agents, which can process this performative, register themselves to the facilitator. Therefore, the facilitator knows which agent would like to process what and it sends the process to the particular agent and forwards the reply to the requesting agent.

- Recruiting: It is almost the same process as we saw in "matchmaking" case except that the agent, which carries out the process, informs the requesting agent directly.

- Recommend: An agent A asks the Facilitator F to find an agent that can process the particular performative. When the facilitator finds an agent in its registry, it gives the name of the service agent to the requesting agent.

There are various KQML literate agents but because of dialects they can not communicate with each other. This is partially because of the number of performatives in the KQML since this caused in complex designs or partial implementations.

The Foundation for Intelligent Physical Agents (FIPA)

FIPA model defines a model for the creation, registration, location, communication, migration and retirement of agents. Each entity in the model offers a service called as logical capability sets.

An agent can have one or more service capabilities and it has one or more owners. Each agent is identified by the assigned GUID. Directory Facilitator (DF) offers a yellow page service and a mandatory agent. Each DF defines an Agent Domain. Agent Management System (AMS) is an agent responsible for controlling the access and usage of the trust. AMS keeps the logical agent names (GUID) and their associated transport addresses on the host. Agent Communication Channel (ACC) is a communicator between different hosts and each agent has an access to at least one ACC. Agent Platform is the term referring to the host keeping agents, DF, AMS, ACC and the communication channel between them. This channel is called as Internal Platform Message Transport (IPMT) and left unspecified since it is an implementation specific issue.

FIPA compliant APs communicate with each other various communication technolgies but IIOP is specified as the base line technology. ACCs are connected to each other. ACC accepts a message addressed to an agent and routes this message to the referred agent.

The first AP in which an agent is created is called as Home Agent Platform (HAP) that can vouch for the agent’s identity. Agent’s GUID gives the IIOP-URL of the HAP so that any AP can ask HAP to authenticate the identity of an agent. Therefore, an agent always keeps HAP informed about its location in the Agent Universe.

Agent Platform Security Manager (APSM) places itself on top of IPMT layer so that all communications are tapped by APSM. Security mechanism supports message encryption based on sending agent’s request, host authentication since each AMS keeps a public key and certifate in the DF, and a public-private key pair for each user. User’s keys are used for message encryption.

FIPA is a competing standard with KQML and based on six performatives while it allows composability to express more sophisticated expectations and beliefs. FIPA also defines a security model unlike KQML. There is an essential semantic difference between the FIPA and KQML agents. When KQML Agent A says something to KQML Agent B, Agent A only tells to Agent B what Agent A believes in. When FIPA Agent A says something FIPA Agent B, Agent A wants Agent B to believe that what the Agent A says is true. In this sense, KQML is more cautious then the FIPA model.

OMG Mobile Agent Facility (MAF)

OMG MAF addresses the management, tracking, and transport issues. Each agent conduct its process on behalf of an authority and this authority should be authenticated. Agent system represents an agent server for creating interpreting, executing, transferring and terminating agents. Each Agent System is associated with an authority such as a person or an organization. AS protects the resources of its authority based on the authority's policies. Security features covers the authentication of an agent, mutual authentication of agent systems before transmission of an agent, protected access to the services of resources and agents, and message level encryption.

OMG proposal tries to bring together various languages therefore it introduces new concepts such as agent profile, place, and region. A profile describe what agent system (Aglets, Voyager,..), what language (Java,Telescript,..), and serialization type. Place represents an execution context in which an agent runs and agents are transmitted between places sharing the same agent profile. A region refers to a set of agent systems controlled by the same authority. Regions can be used for load balancing as well as running different agent systems. An address of an agent is defined by the combination of region's name and name of an agent or agent's place.

If agent A on machine B wants to talk to agent C on machine D, then there are several options based of the agent systems:

1. Both agent systems support the same profile so that agent A can be moved to the agent system in which agent C is running.

2. Both agent systems have different profile and there a compatible agent systems running on machine D, then agent A can be moved to this compatible agent system and communicates with agent C through RPC.

3. Both agent systems have different profile and there is no compatible agent system on the machine D, then agents communicate through RPC.

A Mobile Agent Facility (MAF) should provide a support for start, stop, move, and monitor services on the agents as well as the facilitator functionality to allow one find another on the particular universe. These functinalities are realized by MAFAgentSystem and MAFFinder interfaces in OMG MAF proposal. MAFAgentSystem provides life cycle and transmit support for agents. MAFFinder finds the name and location of agents, places, and agent systems. MAFAgentSystem objects, MAFFinder objects, an agents written in CORBA-enabled language can publish its reference to CORBA Naming Service. A handle of MAFFinder object can be obtained from MAFAgentSystem object or from CORBA Naming Service.

An address of an object becomes tricky when it starts moving. If the system is using a naming service, then this value should be updated after each transmit. In this case, a client which has the previous address receives invalid object reference error therefore developers should catch this error and consult with the naming service for the current location of the object. Another solution is to use forward agents. Forward agent is defined on the home agent system when an agent lefts the home site to keep the current address of the agent. A forward agent forwards all the request to the agent. In this technique, agents have to inform their forward agents about their current location for every transmit. Also this techniue makes the home site a single point of failure and the agent becomes unaccessible as soon as the home site goes down. A third approach is to keep the agent's original address fixed and let ORBs handle the location problem. This requires keeping a mapping between the original address of the object and the current address of the object in the ORBs.

Mobility feature requires that the running agent should be moved from one host to another and continue its operation where it left off. This requires taking a snapshot of a running process (state), moving around, and resuming it. After resuming an agent on different host, a references to other agents has to be taken care of.

1. Its data and the state of the thread in which an agent was running. This level of detail can be seen as an instruction-level snapshot. This snapshot requires saving the call stack and the execution context so that an agent can be resumed on another host from exactly the same instruction. Java based systems unfortunately do not support this approach since the standard Java Virtual Machine does not allow it. It can be done by changing the JVM but this JVM won't be compatible with the standard.

2. A coarse-grained snapshot can be used instead of detailed instruction-level snapshot. In this technique, system asks an agent to save and restore its state and informs an agent about the changes of its life cycle.

While we are transferring an agent, we need to transfer the attributes (data) and the code of the agent. There are small variations in this transfer:

1. Data and Code are shipped at once.

2. Only data is shipped and the system requires that the code should be preinstalled on the new host.

3. On-demand code shipping. Data and an address of server from which the system can ask the necessary code components are shipped.

Agents and hosting server needs a name so that agents can be able to address each other and a particular host. For the sake of location-transparency, this names should not directly contain a machine specific address information and it should be human readable. In general, some sort of yellow page server is required.

Resources should define policies for protecting themselves. This can be based on identity of the agent's principal (owner), its role in the organization, or its security classification level. Besides this, resources should be monitored and shared between agents without letting one agent take over the resource. Similarly, a rogue agent, which can damage the system, network, server or the server's resources, be removed if it is not releasing the resource. Some services may require a service fee therefore the agent should be able to pay to the server because the agent is using its services. This requires that an agent should carry digital cash or credit card or be subscriber of the services. If it is a subsciber, then it should prove it.

Privacy and integrity are important issues for agents as well as any mobile code. The state of the agent should be protected from the rouge server that is an aggressive server tries to extract the sensitive information from the visiting agent for malicious purposes. Requiring that the agent could keep its encrypted version of the data could solve this problem. But, in some real time settings we might want to expose the different set of attributes to the different server based on the server's security clearance and this complicates the matter. Note that even though we can solve the protection problem, solving the integrity problem is a tough one since any server can tamper the data and the code. O nly we can do is to verify whether it is tampered or not. This matter can become complicated if the agent is collecting data from various servers since its size changes continously.

Authentication between agents and host servers are important since this allow servers to decide how much resources are going to be reserved for whom, whether the user has the right credentials, or whether the server has the required capabilities. Digital signatures can be used for this purpose.

Itinerary describes which task (method or code) to execute on which host. Some systems execute only a predefined method when an agent arrives and activated on its destination host while other systems allow to define what to execute on the arrival of a particular host.

Fault-tolerance and monitoring are also important issues in mobile computing. Some systems applies checkpoints transparently, some of them applies checkpoints while transmitting agent from one host to another, and some of them provides an interface on which an agent can request a service to be checkpointed. The functionalities of monitoring process can be summarized as obtaining the current location of an agent, calling it back, collecting its exceptions, and terminating it.

Example Systems

Agent Tcl

Agent Tcl executes Tcl code and Java and Schema support will be provided soon. A security support includes encryption of messages between agents and servers, signing messages for authentication and resource manager for controlled access to the system resources. It does not solve rogue server problem. Docking mechanism is provided for mobile clients. For each mobile user, a specific docking server keeps the agent and when the user reconnects, then agents moves itself to the user's computer.

Voyager

Voyager is developed by ObjectSpace. It runs with JDK 1.1 and takes advantage of serialization and reflection APIs. Core concept in Voyager is called Virtual Object that is wrapper object for remote object or agents. Virtual Code Compiler (VCC) takes any existing Java class (source or class file) and produces an associated Virtual Object. If the object contains a reference to other objects, then the whole interrelated objects have to be passed through the VCC. Virtual Object abstraction provides a service for installation, communication, and migration of the wrapped object in the network. Instantiation can be done on remote machine. Communication can be done asynchronously, synchronously, and a future remote method call. Migration can be done from one server to another server as well as one virtual machine to another.

Voyager-enabled program starts threads for providing timing services and distributed garbage collection when it starts up. Each Voyager-enabled program has an address consisting of the host name and the port number on which it listens to in-coming calls.

The virtual class also contains a superset of the original class functions and allows function calls to occur even when objects are remote or on move. The virtual class allows to define a remote instances on another host or on the same host. Each object is assigned a 16-byte globally unique identifier (GUID). Optionaly an alias basen on URL format can be associated with GUID and these two components can be used to reach the object's current location from directory service.

Objects can be transmitted after all pending messages are processed. Object lefts a forward object which can forward incoming messages to the object's new location and informs the clients (virtual references) about the new location.

Agents are derived from COM.objectspace.voyager.Agent class and they have to be passed from VCC. An agent moves itself by denoting the name of the application and the name of the method that will be executed on an agent after its arrival to the destination. Agents can be moved between applications and applets. The security restrictions on applets is solved by using the server as a gateway between applets so that they can talk to each other.

Voyager uses JavaBeans' event model and also supports transmission of these events accross the network without changing anything in the JavaBeans code.

Voyager allows an attachment of property set to an object and these poperties can be used by other objects to obtain their references.

Voyager introduces a VoyagerDB abstraction for persistent storage functionality so that each object can have a persistent storage support. VoyagerDB abstraction can work on various commercial data bases. Objects can initiate this saving process.

Space (Scalable Group Communication) service is based on grouping objects into subspaces and connecting subspaces to build a space so that a broadcasting of message is done through subspaces and subspaces broadcast this message to its objects.

A special agent called message is used for message delivery. Messeges can be synchronous, asynchronous, future-reply, multicast. In multicast mode, publish/subscribe pattern is supported based on selection criteria. Selection can be made by the initiator or object's subscription subject that is a hierarchical string. Voyager also supports a dynamic message definition so that message is executed on the proper object such as CORBA's DII.

Voyager supports a federated directory service where a distributed directory services are linked together. All directory services can also take advantage of persistent storage.

For distributed garbage collection, Voyager defines various life span definition so that an object is garbage collected at the end of the life span. A life span can be an infinite time, a length of time, a point in time, or when the number of references to an object drops to zero.

The Voyager Transaction Service (VTS) is built on the Voyager as a plagible service and compatible with CORBA's OTS and X/Open DTP.

Odyssey

Odyssey is developed by General Magic as a result of a reseach effort. Odyssey defines three base classes specializing in agent management, name resolution, and transmission of agents. Odyssey transport mechanism can be run on top of Sun's RMI (default), OMG's CORBA IIOP, and Mocrosoft's DCOM. Odyssey runs with JDK 1.1 and uses Java RMI and reflection features. Agents are derived from Odyssey Agent class or Odyssey Worker class. Odyssey Worker class defines an itinerary describing which method (task) to execute on which host. An itinerary can be manipulated dynamically. An agent system provides a support for creation, management, execution, transmission, and termination of agents. An agent is executed in a context called place which is a java thread. Each place and agent runs in its own thread group. Odyssey uses standard Java Security features and does not apply any other additional security restrictions.

Aglets

Aglets Workbench is from IBM and runs with JDK 1.1. Agents framework supports applets, AWT callback, JavaBeans and a publish-subscribe message model. Aglets are defined with Aglet and AgletContext object that is very similar to Applet and AppletContext objects in the java.applet package. Aglet can define various property-value pairs to associate with itself and query other Aglets based on their tags. For message exchange between aglets, a synchronous or asynchronous messaging is supported. For filtering, also Aglet class supports a basic registration for the events. It may publish so that filtering can be done at the source level.

Proxy Object can not access the real object when the aglet migrates since this requires either tracking down all the proxy objects and informing them or a global naming mechanism so that a uniquely given name can identities the object whenever the object resides currently.

State information is based on Java Serialization feature. State is based on functional level and aglet receives events according to the life cycle of an aglet.

Aglets send each other message through Message object which can carry extra optional object. Messages can be sent asynchronously or synchronously or future-type. Future-type is an asynchronous message sending but it returns a FutureReply object from which sender can collect the reply of a message. Message arrival is notified through events (handleMessage()). Aglets Framework defines a FIFO message queue for each aglet to keep the in-coming messages.

Aglets Framework has a two layered architecture and communication layer is abstracted away with OMG MAF interface so that communication layer can be put on top of RMI, IIOP, or Agent Transfer Protocol (ATP). ATP is defined to transmit agents between ATP-literate hosts. ATP contains four methods: Dispatch, Retract, Fetch, and Message. ATP messages can be used with HTTP servers through HTTP tunneling. ATP messages are send in the body of HTTP POST message with "x-atp" content type.

Security architecture authenticates servers based on domains. For resources, a policy is defined by using GUI or directly editing policy database. This information is defined for each principal. A principal is defined by the code base and the owner of the aglet.

Tahiti is a visual agent manager to monitor and control aglets executing on the host machine. Fiji is a Java applet to host an aglet in the Web page. It supports Aglet Framework and minimum functionality to create an aglet or retract aglet from the Web page. Fiji can dispatch the aglet to the HTTP server for processing when the server supports ATP.

Ajanta

Ajanta is from University of Minnesota and emphasizes on security issues. Ajanta is Java based agent architecture, and it uses Java Serialization for state saving. Ajanta uses an URN based naming service. Each server keeps an access control list for its entities, authentication of the server, and public key certificates for various principals, that is a creator of agent such as another agent, service provider, hosts, agent servers or human users. The name resolution process starts from the local nameserver. If it can not be found in the local nameserver, then the creator host's name server is queried. Every agent keeps his creator name service posted about its current location.

Resources are exposed with *Resource* interface and agents can ask the available resources on the host. Agent accesses resources through proxy object that contain a private reference to the real resource object and an array of methods which can be invoked by the corresponding agent. Each agent has its own Thread Group and Class Loader so that they can not tamper each others codes.

It defines an Agent Transfer Protocol where sender and receiver servers authenticates each other before the transfer starts. Then receiving server checks the credentials of the agent that wants to move in and decides what to do.

Tacoma

TACOMA (Tromso And COrnell Moving Agents) is a result of joint project netween the University of Tromso (Norway) and the Cornell University (USA). System can run agents written in various programming languages such as C, Tcl/Tk, Perl, Python, and Scheme.

Agents can leave data at a certain site and carry with them. Tacoma introduces a *folder* abstraction which is used for storing data, code, and additional information that an agent wants to carry with itself. A folder is named with ASCII text string. A collection of folders is called as a *briefcase* that is a unit of transmitable item. Briefcases contain a predefined folders such as DATA, CODE, HOST, and CONTACT. Tacoma also introduces a *file cabinet* a stationary folder for persistent storage requirements.

Agents can request a service from another agent by sending a briefcase which contains the necessary information. Agent code is organized a sequence of codes to be executed on different hosts. These codes are stored in the CODE folder. This mechanism can be used to create an agent on another host remotely. Tacoma agents decide when to move to a particular place.

Concordia

Concordia is a Java based mobile agent system developed by Mitshubishi. Each Concordia server manages the life cycle of agents and knows its colliques. Agent's are travelled according to their itineraries which can be specified programmatically or by the user with a GUI or a command line and an agent can initiate the transmission.

Concordia defines a set of extendable manager objects such as Administration Manager, Agent Manager, Security Manager, Persistence Manager, Event Manager, Queue Manager, and Directory Manager. An Administration Manager holds all the manager objects and only one AM is sufficient to adminster the Concordia network. Agent Manager is responsible for life cycle of the agent including creation, execution, destruction, and transmission. Security Manager is responsible for identifying users, authenticating their agents, protecting server resources, ensuring the security and integrity of agents and their accumulated data objects. Persistence Manager provides checkpoint support so that makes the system fault-tolerant. Checkpoint is automatically used when an agent needs to be transmitted and an agent can also initiate its checkpoint operation. Event Manager handles the registration, posting, and notification of events between agents. Queue Manager keeps agents on the queue for scheduling and transmission. On transmission, the state of the agent is kept on the storage areas so that event the client is disconnected it will stay in the queeue until the client server is connected to the network again. Directory Manager provides a naming service. Service Bridge interface is designed for accessing services on the various machines in the network. Inter-agent communication can be done in two ways: distributed events and agent collaboration.

Security credentials and permissions are defined by the user of the agent. The user's credentials and permissions are defined by an Concordia System Administrator. Agent carries this credentials that are authenticated during execution at every server. While agents are in transit, their codes and data are encrypted.

Agent's data is protected on transmission by using Secure Sockets Layer version3 (SSlv3) that supports authentication of both machines on each side of TCP/IP communication and allows encryption of data on the wire. Agent's state is encrypted before storing to the persistent storage by using a symmetric key encryption algorithm. Each Concordia server keeps a public and private key. As soon as an agent arrives to the server, concordia server gives a symmetric key and encrypes this key with its public key for storage. Server's private key is kept in secure area. Concordia can use digital signatures to guarantee that the code of the agent is not tampered during network transmit through untrusted nodes. Digital signatures allows user to configure their system based on the code authors credibility. Concordia extends this view with the addition of the owner of the agent since it is most likely that a person who is an agent is different from a person who wrote it. Therfore, each user represented with a user java object that keeps the name of the user, user's group, and enrcyped password. This information is used as the identity of an agent and travels with an agent. Each node asks this identity to the arriving agent and verifies it. Verification process is done by checking the the user's name and enrcyped password information. Therefore, list of valid users and their enrcyped passwords are kept in a central server that can be accessible via JDBC or URL. This server also keeps this information in encrpted mode. Concordia allows to define restriction on services that can be asked from a resource in addition to the Java Security Manager's services. Each node keeps the list of users/groups and their access permissions to the resources on the server so that a server can check every interaction of the agent with its environment based on this access control list and its identity.

The Persistence Manager and Queue Manager collaborates for transmission of an agent from one host to another. The Queue Manager applies Two-Phase Commit for guaranteed delivery. In transmisson process, Concordia Server check's the agent's itinerary, contact with the destination Concordia server, stores the state of the agent into a persistent area, and transfers the agent. Sender keeps the agent's information until it receives an acknowledgment so that it can provide a docking mechanism for disconnected clients. Concordia provides a fast its own checkpoint mechanism instead of using a database or JDBC Api. Concordia applies transparent checkpoint for fault tolerance and an agent can also ask to be checkpointed to the system.

Concordia provides an interface allowing agents to create other agents and to clone themselves. Ininerary keeps a location of the host and the name of the method to be executed. This object stored in separate location not inside the agent. Concordia does not provide a support to hand in a reference of one agent to another instead it expects that agents communicate with each other through messages or group-oriented communication mechanisms.

Events can be asynchronous distributed events and collaboration. Asynchronous events can be publish-subscibe or a group-oriented manner. In publish-subscribe pattern, aach agent registers itsef to the Event Manager (EM) by providing a list of events that it wants to listen and a location that it wants EM to sent events. In group-oriented pattern events are delivered to the registered group members. Each registered agent can publish and receive published events without any filtering. Collaboration provides a distribuyed synchronization point or collaboration point. Agents within a collaboration unit represented as *AgentGroup* object and each agent access this object through a proxy. When each agent arrives at the collaboration point, it executes *collaborate()* method on the *AgentGroup* object with its results or information that an agent wants to share. Then an agent blocks. The AgentGroup object waits until all the registered agents call the collaborate() method then it calls *analyzeResult()* method on each agent with the collection of results coming from each agent. Agents implements analyzeResult() method to inspect the results collected by other agents. Since some agents may be unavailable, Concordia also allows *weak collaboration* where system can continue its operation when some of the agents are not available. For deadlock prevention, the AgentGroup object also supports time-outs.

AgentSpace

AgentSpace project is developed by Technical University of Lisbon. It runs on top of Voyager and provides additional functionalities which are not supported by the Voyager. AgentSpace server is called AS-server which runs on all the hosts and supports creation of agent and place, execution of an agent, a way to control, transfer, and store an agent into storage areas, generation of unique identity (UID), communication between agents, and administration interface for itself.

AS-Server is identified with machine address (DNS or IP) and a port number. A place identity contains the address of the AS-server and its unique name in that server. An agent identity contains the identity of the place in which it is created and a unique name within that place. Therefore, all components has a global unique names.

User is identified by an instance of User class. This object keeps the user’s name , public key, a set of certificates, e-mail address and organization and country to which the user belongs. A different view is provided for the context, place and agent to the interacing agents based on its user’s credentials. A factory pattern is provided for creating agents, places, and views to agents, places, and contexts.

Agents are extended from the Agent class and they can have callbacks which informs the state change or events. It supports synchronous (*doOperation* ()- *handleOperation* ()) and asynchronous messages (*sendMessage*()-*handleMessage*()).

Future of Agents

Today, agent based technologies suffer from security problems and not easy to use development suites.

Both of them requires more research in the field. Some of the research is focused on visual development environments and programming languages which takes advantage of an agent idea.

Today's Visual Agent development toolkits allows user to define an itinerarry of an agent and monitor its status. For example, Agent Inception System is based on Iconic Modeling Tool (IMT) interface where user uses icons to model the agents and their itineraries. This development toolkits require a visual concepts and metaphors required for clear description and functionality of the agents.

The Agent Oriented Programming (AOP) is the natural way of programming on these systems. This programming style defines an agent as a communicating and evolving unit. In this framework, agents are viewed as entities and the state of an agent consists of mental components such as beliefs, capabilities, choices, and commitments. AOP specifies the syntax and semantics of an interpreted programming language to describe the mental states.