Abstract
The distributed computing modeling techniques have become widely acceptant recently for designing complicated software. Naturally, there are a number of different approaches in realizing such design. In this paper, the concept of Conversational Agents system design is discussed. Its architecture technology including communication protocol is briefly discussed. This document also briefly compares its technology with other common modeling techniques that are available. The materials presented in this document are based on the information obtained from the class and, as well as, technical articles or papers with related information found on the “Web”.

Introduction
As the complexity of the software increases, software decomposition is required to improve maintainability and reusability. One of the methods that have been introduced to address the decomposition of software is the concept of distributing computing. That is, the software is to be broken into multiple components, and each can be run independently on a separate machine (computer) across the network. A new design paradigm known as agent-based design approach has been developed for analyzing and implementing software to be run in distributed environment. One of the main goals of agent-based design approach is reusability.
Before staring the discussion on topic of Conversational Agents, it may be necessary to clarify what a software agent is. A lot of efforts have been put into agent-based design research. However, researchers in the field have offered various definitions [5] for a software agent. In general, a software agent can be defined as a software entity that carries out a set of operations on behalf of the user or another program with some degree of independence or autonomy. Typically, a software agent possess some or all of the following characteristics:

1. Autonomous
2. Adaptive/Learning
3. Mobile
4. Persistent
5. Goal oriented
6. Communicative/Collaborative
7. Flexible
8. Active/Proactive

The Conversational Agents model is one of the more popular distributed computing models, that has been adopted in many software projects. Other modeling concepts that have been adopted in the software industrial include Distributed Objects, Jini, Blackboard, Publish and Subscribe, and Open Agent Architecture (OAA). This document only focuses on the discussion on Conversational Agents, and it will briefly discuss other techniques for comparisons purpose only.

One of major differences between these techniques lies on the inter-agent communication design. Conversational Agent technologies model the inter-agent communication and cooperation through messages exchange based on “speech act” theory [3]. Speech act theory was invented in the fifties and sixties to help understand human language. The idea was that with language you not only make statements, but also perform action [5]. The reason Conversational Agents based their inter-agent communication on “speech act” theory to enable the agents to have a language-enriched interchanges as human; thus, given its name “Conversational” agents.

One of the original computer languages that were developed based on “speech act” theory is Knowledge Query and Manipulation Language (KQML). Another more recent language that was also developed based on “speech act” is the Foundation for Intelligent Agents (FIPA) Agents Communication Language (ACL). Both are being formalized, so that independently developed agents conforming to the standard will understand each other.

Since these two are the foundation technologies for developing Conversational Agents, this paper will devote the next two sections in going over some of the syntax of these two communication languages.

**KQML**

KQML was developed under the Defense Advanced Research Projects Agency (DARPA) sponsored Knowledge Sharing Effort [6]. KQML is built upon a layered architecture:
1. Content layer – contains the content of the message exchanged between the applications. The content in this layer is typically written in some representation language, such as the Knowledge Interchange Format (KIF) [7] or Structured Query Language (SQL) for databases. KQML ignores the content of this layer. Thus, as long as the two agents are using the same representation language, they can communicate with each other over KQML.

2. Communication layer – contains communication parameters such as the identity of the sender and recipient, and unique identifier associated with the communication.

3. Message layer – identifies the protocol to be used to deliver the message.

Basically, KQML provides a way to structure the messages, but lets the agent designers worry about what is in them.

KQML message is based on a balanced parenthesis list, which is similar to Common Lisp. It consists of a performative, its associated arguments which include the real content of the message, and a set of optional arguments transport which describe the content and perhaps the sender and receiver. Its message structure can be expressed as follows:

\[(\text{performative-name} \quad : \text{sender A} \quad : \text{receiver B} \quad : \text{content X} \quad : \text{language L} \quad : \text{ontology N} \quad : \text{reply-with W} \quad : \text{in-reply-to P})\]

For example, a message representing a query about the price of a share of IBM stock might be encoded as:

\[(\text{ask-one} \quad : \text{content } \text{(PRICE IBM ? price)} \quad : \text{receiver stock-server} \quad : \text{reply-with ibm-stock} \quad : \text{language LPROLOG} \quad : \text{ontology NYSE-TICKS})\]

In this message, the KQML performative is “ask-one”, the content is (price IBM? price), the receiver of the message is “stock-server”, and the expected return type is of “ibm-stock”. The query is written in “LPROLOG” and the ontology of the query is “NYSE-TICKS”.

The KQML specification has defined a set of reserved performatives; it is neither a minimal required set nor a closed one. A KQML agent only needs to implement enough performatives to allow other agents to request the services it provides. Here are some of the pre-defined performatives [6]:

**Basic query performatives:**
- evaluate, ask-if, ask-in, ask-one, ask-all, …

**Multi-response query performatives:**
- stream-in, stream-all, …

**Response performatives:**
- reply, sorry, …

**Generic informational performatives:**
- tell, achieve, cancel, untell, unachieve, …

**Generator performatives:**
- standby, ready, next, rest, discard, generator, …

**Capability-definition performatives:**
- advertise, subscribe, monitor, import, export, …

**Networking performatives:**
- register, unregister, forward, broadcast, route, …

---

**FIPA ACL**

Recently, FIFA has proposed a standard for Agent Communication Language (ACL). It is developed based on KQML. FIPA ACL is very similar to KQML in terms of message syntax and rules. The main differences are:

1. FIPA ACL specifies less performatives than KQML.
2. FIPA ACL allows agents to combine performatives to express more complex beliefs and expectations. For example, an agent can request to be informed about one of several alternatives.
3. KQML provides a number of mechanisms for handling multiple queries at the message level (ask-one, ask-all, stream-all, standby, ready, next, rest, discard). In FIPA ACL, the responsibility of managing multiple solutions resides on the content language.
4. FIPA ACL specification comes with a formal semantics. This, in general, is a strong point because it guarantees that there is only one way to interpret an agent’s communications.

---

**Facilitators**

A facilitator is an agent that performs various useful communication services such as maintaining a registry of service names, forwarding messages to named services, routing messages based on content, providing matchmaking between information providers and clients, and providing mediation and translation services [6]. In Conversational Agents system, facilitators act as a communication bridge between agents. For example, consider a case where an agent A would like to execute service X, if agent A doesn’t know which agents provide service X, it can ask facility F to “recommend” an agent to whom it should send the request X to. Once F learns that B is able to provide service X, it can reply to A with the name of agent B. Agent A can send the service request X directly to B as shown in Figure 1. The role of this facilitator is acting as a service broker and a name server. Figure 2 shows the case where Agent A already knows as which agent can provide service X.
Distributed Computing Models Comparison

Similar to Conversational Agents system, the requesting object in Distributed Objects and Jini systems retrieves a reference object from a registry. And the requesting object dictates as how it interacts with the service object [1].

The Blackboard, and “Publish and Subscribe” approaches are slightly different from Conversational Agents approach. In Blackboard systems, requester posts requests on the Blackboard and poll for available results; providers poll to obtain service requests, and use the Blackboard to post results. Similarly, in “Publish and Subscribe” system, requesters issue a request to the broker that broadcasts it to available providers; their responses are reflected through the broker to the requester [1].
Conclusions
Selecting the right agent-based model for the project can be quite difficult. Each model has its Pros and Cons. For instance, Conversational Agents system has its weakness in having the requesters hard-coded all the interactions between the components. This results in making the services inflexible and difficult to reuse and extend. The Blackboard approach is quite flexible in that it allows dynamic and flexible composition of distributed components because the component interactions are not pre-defined or hard-coded; rather, their interactions are dynamically determined at runtime. However, this approach doesn’t have any programmatic control in place to guide the operation so that the service is to be completed only by provider with the best meets known requirements.

References


