Primitive Interaction Protocols for Agents in a Dynamic Environment

Roberto A. Flores (1) and Niek J.E. Wijngaards (2)

(1) Department of Computer Science, University of Calgary, Canada T2N 1N4
   Email: robertof@cpsc.ucalgary.ca

(2) Department of Artificial Intelligence
    Vrije Universiteit, The Netherlands
    Email: niek@cs.vu.nl

ABSTRACT

This paper describes a generic multi-agent systems architecture for dynamic collaboration in an open environment. This architecture is described as composed of basic role-oriented processes communicating through primitive interaction protocols. The architecture is aimed to provide an enabling communications framework upon which multi-agent system (MAS) models can be organized and built. In particular, we plan to use this architecture in future research as an experimental test bed on which to study the suitability of human, face-to-face interaction factors—as described in the social sciences—as guidelines for analyzing and improving the interaction abilities of software agents.

Keywords: Agents, interactions, enabling architecture, multi-agent systems.

1. INTRODUCTION

This paper presents the initial step toward a study of human social factors as potential features to model the interaction of software agents. This long-term research aims to study and apply to software agents the social aspects relevant to humans in face-to-face interactions, namely, language use, shared knowledge, and the environment where interactions are pursued (Clark, 1996). Sociologists have recognized the importance of these aspects in the process of human interaction, and have taken face-to-face as the basic setting for language use due to the richness of the resources available to the interacting entities.

Our approach to the subject will be to study the factors in human face-to-face interaction as a guideline to model the interaction of less able entities in less-rich settings than face-to-face. This approach has also been pursued in the field of human-computer interaction, as described by L.A. Suchman (1987), who focused in the use of graphical user interfaces for communication between computer processes and human operators. In our case, we are attempting to analyze human interaction factors in the expectation to apply them in an even more restricted setting: that of software processes in a generic and distributed environment. To that end, our objective is to first
attempt the definition of a software agent architecture to support agent interoperability in
dynamic and open environments. This architecture is attempted as the basic framework for
experimenting with relevant human social factors in software agents.

The paper is divided in the following way: Section 2 briefly accounts the definition of agency as
understood by researchers of computer science, followed by a short description of the
motivations underlying psychologists and sociologists in the study of humans as cognitive and
interacting entities, respectively. Following sections address the agent architecture by describing
the requirements we have identified for the architecture (Section 3), the basic roles defined in it
(Section 4), and the basic interaction templates for agents in the architecture (Section 5). Section
6 describes the conceptualizations we use to implement a simple version of the architecture, and
Section 7 presents a test case on which the architecture is put to use to support a simple agent-
based chat system. A brief account of some related efforts in the study of agent architectures is
covered in Section 8 and, finally, the conclusions are drawn in Section 9.

2. AGENTS AS SOCIAL ENTITIES

For researchers of computer science, the notion of agency has been found useful as a metaphor
for the design of complex systems (Wooldridge and Jennings, 1998, p.1), where agents are
conceptually defined as (relatively) autonomous entities that perceive and act (Russell and
Norvig, 1995, p.7). Additionally, other definitions indicate the existence of an environment
where other agents exist and interact (Shoham, 1997, pp. 271-272; Wooldridge and Jennings,
1995, pp. 4-5). Such definitions ascribe agents with cognitive and social characteristics that—at
first sight—might not be unlike those shown by humans.

In the psychological sciences, scholars see cognitive processes as located inside the head of the
actor, which include the formation and effect of beliefs, desires, intentions, and the like. In this
discipline, students of cognition make use of symbolic representations as the means by which
“the entities that are imagined to be inside the mind are modeled on a particular class of entities
that are outside the mind” (Hutchins, 1995, p. 357). Since computers are devices that efficiently
support the manipulation of mechanized versions of formal systems, they have naturally been
viewed as artefacts that might be able to exhibit intelligence by manipulating such symbolic
representations. It is in this avenue—the use of formal symbolic systems to simulate reasoning—
that scholars in artificial intelligence have invested time and effort through the years. Although
this is a laudable, worth-pursuing effort, our focus rather resides on the human social aspects—
those pertaining the interaction of agents—than on the internal processes ascribed to rational
organisms.

For researchers of the social sciences, the processes of interest are those of circumstantial
interaction, defined in the relationships among actors, and between actors and their embedding
situations (Suchman, 1987, p.2). Basic to the notion of interaction is the idea that interacting
actors do not perform loosely coupled, autonomous actions, but a highly coordinated set of
participatory actions that makes their interaction a joint action (Clark, 1996, p. 19). For example,
playing the piano solo is an individual action, but playing a duet is a joint one, and different
processes are to account even when both types of actions can be observed as identical. In brief,
Clark’s study of language use is centred on a number of insights. First, that language—the
linguistic and non-linguistic signals used in communication—is an instrument by which people
coordinate their individual actions when performing a joint action (ibid, p. 387). Second, that
language use takes place in a setting, which is composed of a scene—the environment where language use takes place, and medium—the conduit through which communication is achieved (ibid, p. 4). Finally, that interacting actors possess “a great mass of knowledge, beliefs, and suppositions they believe they share. This I will call their common ground” (ibid, p. 12). These social aspects of language use may account for the achievement of mutual intelligibility in the coordination of speaker’s meaning and addressee’s understanding during interactions.

It is the importance of these social factors in human interactions, and their possible application to software agents where our research efforts will be focused in the future. We expect to develop and report more on this subject as time advances.

3. ARCHITECTURE REQUIREMENTS

Our goal is to define a generic software agent architecture enabling agent interoperability in dynamic and open environments. This architecture is aimed to define the initial settings on which agent interaction can be accomplished. On the basis of these initial settings (also seen as interaction primitives) more complex patterns of agent interaction may be constructed. In this section, we make an account of the requirements set for the architecture.

The architecture is based on the principle that agents participate as requesters and/or providers of services, and they play at least one of these roles to accomplish interaction. The architecture makes use of specialized facilitator agents to organize, advertise, and locate agents and services.

An overall requirement on the agents in this architecture is that a ‘skeleton’ interactional structure of an agent needs to be provided, such that basic (or primitive) interactions are pre-defined. More complex interactions (domain and application specific interactions) can then be achieved by invoking these pre-defined interactions in the standard, skeleton part of an agent. This approach allows for designers of multi-agent systems to abstract from the physical, detailed situation in the multi-agent system model, and focus on when agents wish to communicate, with whom, and how agents wish to be contacted.

On those grounds, our initial concerns are to provide a supportive structure in the architecture for agents willing to contact or being contacted by other agents. To that extent, agents need to

- (requirement 1) register as active participants in the architecture,
- (requirement 2) advertise their services (if willing to be contacted),
- (requirement 3) request other agents’ advertised services (if willing to make contact), and
- (requirement 4) be able to concurrently communicate to a number of agents.

These requirements must also account for the reversing of actions (e.g., to un-register as active part of the architecture, to withdraw an advertised service). These requirements will be used to define the architecture and they will be addressed in the following sections.

4. REFERENCE MODEL

This section describes the organizational and behavioural functions we use to describe entities in the architecture in compliance with the requirements defined. In brief, to satisfy the requirement of allowing agents as active participants in the architecture (requirement 1) we define the concepts of area and local area coordinator; to satisfy the requirements on advertisement (requirements 2 and 3) we define the concepts of yellow pages; and, to satisfy the requirement of
concurrent communication of agents (requirement 4) we define the concept of cooperation domains.

4.1. Areas and Local Area Coordinators

To satisfy the requirement of “being part of the architecture” (requirement 1) we are defining two entities that support the concept of residence: areas, and local area coordinators.

An area can be conceptualized as a bounded location in which agents reside. This definition comprises two aspects: area boundaries, and agent residency. The first aspect is that of area boundaries. We conceive an area as a function of the computer resources made available to it. On this view, it is possible to have several areas in one computer, up to one area encompassing several networked computers. Moreover, computer resources could be shared among areas, or they could be exclusive to one area. Areas could also be organized in terms of other areas, where areas could be deemed as federations (where hierarchical dependencies exist among them), or as democracies (where areas does not have such hierarchical dependency). The second aspect is that of agent residency. An agent is considered resident of an area after it has succeeded to register with the local coordinator of that area. Any agent can be registered to an area as long as the coordinator accepts the registration. The registration process does not necessarily require agents to physically reside in the area’s resources to be considered part of the area. Therefore, agent residency is logical by default (since it only results from the process of registration), although it could be constrained to be physical if local area coordinators require agents to be present in one of the area’s resources.

Local area coordinators (LAC) are mandatory facilitator agents to which other agents can request area services. They are mandatory in the sense that one—and only one—must exist as part of any area. They are considered facilitators since they help agents to gain access to agents registered in the area. The fundamental service provided by LAC is that of registration, and agents need to go through a registration process to be considered part of an area. As part of this process, agents will surrender some privileges to the LAC in exchange for assistance to engage in agent-to-agent communications. At the very least, privileges granted must allow the LAC to communicate back to the agents; however, these privileges could go as far as controlling the processing of agents, such as invoking, halting, suspending or resuming their execution. To recall which agents are in their area, local area coordinators should keep a listing of registered agents. This white pages listing of agents could be discretionally made available as a service provided by the LAC.

4.2. Yellow Pages Agents

To satisfy the requirement of “advertising of services and their enquiry” (requirements 2 and 3) we define an entity called yellow pages agent (YP).

Yellow pages agents are service provider agents that allow the advertisement and enquiry of advertised agent services. Their functionality should allow agents to add, modify, and delete their own advertised services, and to query about advertised information from other agents. YP give agents the means to find each other under uncertainty; where agents might unpredictably appear, disappear or migrate, causing known agent locations to become unreliable for future interactions.

4.3. Cooperation Domain Agents
To satisfy the requirement that “the architecture must support concurrent communication among agents (requirement 4) we define an entity called Cooperation Domain Agent (CDA).

Our approach to concurrent communication is in the form of a service called cooperation domain (CD), which is provided by a basic agent called cooperation domain agent. A cooperation domain can be conceptualized as a delimited type of setting on which interactional resources are made available to participants of the domain. The functionality of cooperation domains can be paralleled to that of Internet multi-user dungeons, where participants enter rooms on which objects are located and shared, and their actions in that environment can be mutually perceived. We anticipate that cooperation domains could be used in domains were data is to be shared among participants, such as in distributed databases and collaborative automotive designs. One practical example is given by Flores (1997), who describes the application of a specialized cooperation domain in the field of diagrammatic representations.

Our rationale on pursuing this functionality is based on the fact that meaningful interactions in dynamic environments cannot be accomplished in the sole basis of message exchanging. As described by Hutchins (1995, p. 238): “Meanings can only be imagined to be in the messages when the environment about which communication is performed is very stable and there are very strong constrains on the expectations.”

On this view, we devise cooperation domains as entities where messages—which rely in great measure on language use—can also be linked to a context on which to anchor the meaning they are intended. Suchman (1987, p.58) gives an account of this fact when describing that “our shared understanding of situations is due in great measure to the efficiency of language…[which] is due to the fact that, on the one hand, expressions have assigned to them conventional meanings, which hold on any occasion of their use. The significance of a linguistic expression on some actual occasion, on the other hand, lies in its relationship to circumstances that are presupposed or indicated by, but not actually captured in, the expression itself.”

Although it is early in this research to prove the adequacy of cooperation domains to this task, we can foresee that they can act as enablers to provide agents with representational media (i.e., tools) that can be used to coordinate their actions. On those lines, Hutchins (1995, p.154) details that “tools provide two things simultaneously. First and most apparent, they are representational media in which the computation is achieved by the propagation of representational state. Second, they provide constraints on the organization of action.” At this point in time, we cannot specify how such tools are defined, or which tools are better than others for certain situations, but we are certain that valuable results can be achieved with further efforts in this subject.

5. INTERACTION MODEL

While the previous section described the functionality intended for the behavioural processes in the architecture, the current will explain how interaction at the architecture level is to be regulated. Again, it is important to remark that these basic interaction primitives are intended as building blocks which MAS designers can use to abstract from the low-level, physical agent-to-agent interactions and focus on the application specific interactions of their proprietary MAS models.

For researchers of language structures, languages are meant to serve for communication among entities. Clark (1996, p.296) describes two categorizations for the transfer of goods among agents.
according to the extent on which the condition of social interaction have been prescribed to actors prior to their interaction. Such categories are routine procedures, where interaction is “one that is almost prescribed by the social situation,” and close procedures, where the actors’ “situation is tightly circumscribed [and] fixed.” While routine procedures dynamically emerge from mutual agreement among interacting agents, closed procedures are defined in advance, and “participants know their roles, rights and duties, and potential joint purposes. All they need to establish is the joint purposes for that occasion... The first partner initiates the routine, often with a phrasal utterance, and the second partner completes it by complying” (ibid, p.296). A similar categorization of interaction conventions has also been identified by researchers in computer science, which have called them off-line design—i.e., close procedure, and emergence from within the system—i.e., routine procedure (Walker and Wooldridge, 1995, p. 1).

In the case of off-line design, conventions are built at design time, and included as a predefined part of any agent. Therefore, all conventions are well known by agents at the time of interaction. On the other hand, "emergence from within the system" defines methodologies to allow conventions to emerge from mutual agreement among interacting agents. This approach results in agents that are packed with knowledge on how to dynamically create conventions based on peer interaction. Although this approach allows a great deal of flexibility in interactions, it does so at the cost of higher complexity. In the case of off-line design, its application reduces flexibility, but it simplifies reasoning mechanisms that might be too onerous to support.

In the case of our architecture, we have decided to use off-line interaction conventions to satisfy the requirements we described for the architecture. In our view, having less uncertainty on the dynamics of the basic interactions at the architecture level poses less constrains on the decision-making mechanisms in agents and enables less-rational types of agents to join the architecture. As suggested by Wooldridge and Jennings (1998, p.5), it is better “to build agents with a minimum of AI techniques: as success is obtained with such systems, they can be progressively evolved into richer systems.”

We must remark that, regardless of the convention we are selecting at the architecture level, MAS modellers are not restricted on which approach they take: domain specific interaction patterns could be predefined for a model, or they could emerge as the interaction among agents progress (for this, cooperation domains could be a suitable setting of interaction, since they could provide the context on which mutual intelligibility can be achieved). In our view, off-line designed basic interaction protocols can be used to construct (emerging) domain specific protocols.

5.1. Basic Interaction Protocols

In this section, we describe a set of basic interaction protocols for the architecture. We describe interaction protocols as consisting of different message types, interaction-oriented message dependencies (called conversation policies), and agent-oriented message dependencies (represented by conversation perspectives).

Interaction protocols are patterns of communication to be adhered to by interacting agents. The function of the protocols is twofold: firstly, they define causal-relation sequences of messages communicated among types of agents; and, secondly, they describe how these agents should react to these messages during interactions. According to the requirements set for the architecture, the following protocols are defined:
• **Registration**: To be registered/unregistered as part of an area (requirement 1).
• **YP Locations**: To query about known yellow pages agents (requirements 2 and 3).
• **Advertisement**: To advertise/de-advertise services (requirement 2).
• **Search**: To query on advertised services (requirement 3).
• **Cooperation Domain Subscription**: To join/withdraw from a cooperation domain (requirement 4).
• **Cooperation Domain Invitation**: To invite an agent to join a cooperation domain (requirement 4).

Note that requirement 4 is covered by two interaction protocols addressing two different dispositions. In the case of the first protocol (joining a cooperation domain), an agent is willing to become a participant in a cooperation domain. In the case of the second protocol (inviting an agent to participate in a cooperation domain), an agent shows the disposition to interact with the addressed agent in a proposed cooperation domain.

We have organized interaction protocols as consisting of three elements: messages, interaction-oriented message dependencies (using conversation policies), and agent-oriented message dependencies (using procedural perspectives). These elements are further discussed in the next sub-sections.

### 5.1.1. Messages

As already mentioned, our aim is to define a generic MAS architecture for agent interoperability in dynamic and open environments. When addressing the issue of genericness, we looked at ways on which software processes are usually designed to interact among themselves. In our view, commonly found software systems allow inter-process communication via direct communication (e.g., remote method invocation, IP sockets), or through a shared resource (e.g., memory segments, data files).

One characteristic that is common to these mechanisms is that interactions between a “speaker” and an “addressee” are usually accomplished in a discrete rather than in a continuous progression; this is, data is assembled by the “speaker” and sent as a unit to the “addressee” for evaluation, rather than transmitted in real time as it is produced—which is a more natural process in human face-to-face interactions. This characteristic found in software processes is alike to a written (and differently and more limited than a spoken) setting. As explained by Clark (1996, p. 9), “most spoken settings allow the participants to produce and receive simultaneously, but most written settings do not. Being able to speak and listen simultaneously gives people in conversations such useful strategies as interrupting, overlapping their speech, and responding “uh-huh,” and these are rule out in most written settings.”

Having observed this limitation on the communication medium among agents, we continue to define those units of data submitted among interacting agents as **messages**. We categorize basic messages in the architecture as either request, reply or inform messages\(^1\); where requests are used

---

\(^1\) Note that this categorization is different from that used by researchers of language, such as Searle (1975), who usually classify illocutionary verbs (by their publicly intended effect) as either assertives, directives, commissives, expressives, or declarations. This difference on categorizations does not pose a limitation to our intentions to analyze language use using the architecture, since intentional verbs can be implemented as messages in a domain specific MAS model designed for such purpose.
to ask for the provision of services, replies to answer requests, and informs to notify agents without expecting response. The request, reply, and inform messages defined for the architecture are listed in Tables 1, 2, and 3, respectively.

<table>
<thead>
<tr>
<th>REQUEST MESSAGE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>To request being registered/unregistered as part of an area (requirement 1).</td>
</tr>
<tr>
<td>YP Locations</td>
<td>To request the location of currently active yellow pages agents (requirements 2 and 3).</td>
</tr>
<tr>
<td>Advertisement</td>
<td>To request the advertisement/de-advertisement of a service (requirement 2).</td>
</tr>
<tr>
<td>Search</td>
<td>To request a query on advertised services (requirement 3).</td>
</tr>
<tr>
<td>Cooperation Domain Subscription</td>
<td>To request joining/withdrawing from a cooperation domain (requirement 4).</td>
</tr>
<tr>
<td>Cooperation Domain Invitation</td>
<td>To request an agent to join a cooperation domain (requirement 4).</td>
</tr>
</tbody>
</table>

Table 1. Request messages defined for the architecture.

<table>
<thead>
<tr>
<th>REPLY MESSAGE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>To reply to a Registration request (requirement 1).</td>
</tr>
<tr>
<td>YP Locations</td>
<td>To reply to a YP Locations request (requirements 2 and 3).</td>
</tr>
<tr>
<td>Advertisement</td>
<td>To reply to an Advertisement request (requirement 2).</td>
</tr>
<tr>
<td>Search</td>
<td>To reply to a Search request (requirement 3).</td>
</tr>
<tr>
<td>Cooperation Domain Subscription</td>
<td>To reply to a Cooperation Domain Subscription request (requirement 4).</td>
</tr>
<tr>
<td>Cooperation Domain Invitation</td>
<td>To reply to a Cooperation Domain Invitation request (requirement 4).</td>
</tr>
</tbody>
</table>

Table 2. Reply messages defined for the architecture.

<table>
<thead>
<tr>
<th>INFORM MESSAGE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>To notify of an unexpected event in ongoing or attempted interactions. This message has been added to support communications, and its intent is to flag the unilateral termination of an interaction.</td>
</tr>
</tbody>
</table>

Table 3. Inform messages defined for the architecture.

5.1.2. Conversation Policies

In accordance to the off-line design principle we adopted for the architecture, messages are not independent entities that are interpreted as unrelated to each other. Instead they are bounded by a set of orderly and meaningful dependencies according to the intents given by the interaction protocols. We have adopted the notion of conversation policies (Greaves, Holmbach, and Bradshaw, 1999) to represent these dependencies. Conversation policies are diagrams similar to finite-state-machine diagrams that are depicted as nodes and directed arcs, where a node represents one state in an interaction, and an arc linking a pair of nodes represents a message that
can be chosen to move from one state to the next. In these diagrams there is one node marking the beginning of the conversation, and there are one or more nodes representing terminal states. In addition, messages are labelled with the types of agents acting as the speaker and addressee for each message in a diagram.

Following is an account of the conversation policies corresponding to each of the interaction protocols defined earlier in this section. These conversation policies describe the flow of messages under 'uneventful' interactions, and do not show failures in the exchange of messages (i.e., the error message is not used).

**Registration.** Agents willing to become part of an area need to register with the area's LAC. As shown in Figure 1, the dynamics of this conversation are: an agent sends a request for registration message to the LAC, which then replies with either a confirmation of acceptance or a rejection. Agents wishing to unregister from an area they have previously registered to also follow this same process. A special case of registration for an agent is that of yellow pages agents, which as part of their "normal" agent registration also inform LAC of their ability to act as YP. This way, agents in an area can dynamically query their LAC to find out about active YP in the area at any point in time.

![Figure 1. Registration conversation policy diagram.](image)

**YP locations.** Once registered as part of an area, agents may need to know the location of a YP to advertise their services and to query about advertised services. However, agents should not solely rely on previously known YP locations to plan future interactions, since—as any other agent—YP can unpredictably appear and disappear between interactions. To help on such regards, agents can request their LAC for a list of active YP the LAC is aware of. One way on which such list is kept updated in the LAC is through the registration process followed by YP. The dynamics of this conversation policy are shown in Figure 2.

![Figure 2. YP locations conversation policy diagram.](image)

**Advertisement.** Agents wanting to advertise services or withdraw an advertisement from a YP follow this conversation policy. The dynamics of this conversation, which is shown in Figure 3, are as follows: first, the agent with a service to advertise sends a message to the LAC where the YP resides. The LAC can forward the request to an active yellow pages agent in the area, or it can reject this request, e.g., in the case that a yellow pages agent is no longer available. If the
request is forwarded, the yellow pages agent can accept or reject the request in a reply message sent to the requesting agent.

![Advertisement conversation policy diagram.](image)

**Figure 3. Advertisement conversation policy diagram.**

**Search.** Search is the complementary action to advertise, as 'advertise' is about making services public, and 'search' is about querying for advertised services. The search conversation policy follows similar dynamics to the advertisement conversation policy. First, a message containing the query for service information is sent to the LAC where a yellow pages agent resides, which can reject the request or forward it to a yellow pages agent. If forwarded, the yellow pages agent can then accept or refuse to satisfy the query. This outcome is reflected in a message that is sent back to the requesting agent.

**Cooperation Domain Subscription.** Agents wanting to join a cooperation domain can do so by using this conversation policy. The dynamics of the conversation policy are as follows: an agent sends a request for subscription message to a LAC where a cooperation domain agent exists, which can either forward the request to the cooperation domain agent, or reply back to the agent denying the forwarding of the request (for example, when the agent is not available, or it is no longer registered in the area). If the request was forwarded to the cooperation domain agent, this agent could accept or refuse the subscription. This outcome is indicated in a reply message sent back to the agent. If such result was successful, the agent can expect to send and receive messages from other participants through the subscribed cooperation domain.

**Cooperation Domain Invitation.** This is the most intricate of the basic conversation policies in the architecture. Its purpose is to present an invitation request to an agent for it to join a cooperation domain. This is a complex conversation policy since the agent being invited will try to subscribe to the conversation domain before acknowledging or rejecting the invitation. As implied, the cooperation domain subscription policy is also part of this conversation policy.

### 5.1.3. Conversation Perspectives

Although conversation policies are useful for describing the expected flow of messages between agents during interactions, they are not transparently enough to show how individual agents react to such messages. As a result, we include in the architecture the concept of *conversation perspectives* to represent the message sequences that agents in a specific participatory role follow.

---

2 A yellow pages agent could refuse an advertisement request when, for example, an agent attempts to advertise a service to specialized yellow pages agents which only accepts requests from pre-defined agents, or only on services for specific domains.
when engaged in a conversation policy. Therefore, one conversation perspective is defined per agent-role in a conversation policy.

To represent conversation perspectives, we have adopted finite state machine diagrams based on those described by Barbuceanu and Fox (1995). Similarly to conversation policies, conversation perspectives are also made of nodes representing states, and directed links representing messages through which state transitions are achieved. They differ in that conversation perspectives use a different message notation, of the form "message/" and "/message" (which indicate message receptions and submissions, respectively), and in that conversation perspectives only represent the sequence of messages from the perspective (thus the name) of one agent-role in a conversation policy.

Our depiction of conversation perspectives differ from that of Barbuceanu and Fox in that we have extended the original notation for nodes to reflect the expectation of the agent toward messages when in those states. These extensions (depicted in Table 4) show the following state categorization: receptive (the agent expects a message from another agent), productive (the agent is expected to send a message to another agent), and terminal (which indicates to the agent the end of an interaction).

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>EXPECTATION TOWARD MESSAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>□</td>
<td>Receptive only: An message arrival is expected.</td>
</tr>
<tr>
<td>♯</td>
<td>Receptive expected: A message arrival is expected, but an inform message could be sent.</td>
</tr>
<tr>
<td>○</td>
<td>Productive only: A message submission is expected.</td>
</tr>
<tr>
<td>♯</td>
<td>Productive expected: A message submission is expected, but an inform message could arrive.</td>
</tr>
<tr>
<td>☰</td>
<td>Terminal success: Interaction completed successfully.</td>
</tr>
<tr>
<td>☱</td>
<td>Terminal error: Interaction completed with errors.</td>
</tr>
</tbody>
</table>

**Table 4.** Extended notation for states in conversation perspective diagrams.

There are two types of receptive states: receptive-only, and receptive-expected. In receptive-only states, agents wait for oncoming messages before moving to the next state. In the case of receptive-expected states, agents wait for oncoming messages, but they could produce an inform message (such as a time-out error message) that allows them to change the flow of an interaction. Reception-only states are usually found as initial states, and reception-expected states are mostly used as intermediate (i.e., not initial or terminal) states.

Productive states are also found in two types: productive-only, and productive-expected. In productive-only states, agents generate messages without requiring a reply to continue to the next state. In the case of productive-expected states, agents generate messages but may receive an error message from an interacting agent informing of the termination of the interaction. Productive-only states are found as initial states, and as intermediate states in a concatenation of outgoing messages. Productive-expected states are only found as intermediate states.

Figure 4 shows an example of the conversation perspective diagrams for the Registration conversation policy. There are two diagrams in this figure: one for the agent seeking registration (Agent A), and the other for the LAC upon which registration is proposed. The diagram for Agent A starts with a message submission to the LAC. This action changes agent A’s state to
state 1. In this state, agent A waits until a reply is received from the LAC (which could be either an error or acknowledgment), or until the agent deems that an error message needs to be issued (e.g., as a result of a timeout). In the case of the LAC, its conversation perspective diagram starts from a receptive-only state upon which a message for registration can be received; once such a message is received, its state is changed to state 1. Although it is expected that an acknowledgment or an error message will be issued in this state, an error message could also be received from the registering agent to cancel the registration process. Any of these actions will lead to a different terminal state in the LAC.

Figure 4. Conversation perspective diagrams for the Registration conversation policy.

Note that the above conversation perspectives are described as if agents could only carry them on sequentially (i.e., one conversation perspective at a time), rather than concurrently (several conversation perspectives at a time). In our view, there is no obstacle on having an agent concurrently engaged in several interactions; however, it is left to agents’ designers to follow good software engineering practices to keep the state of an agent consistent. This task is their sole responsibility, and the architecture does not take any steps to prevent or correct problems arising in such circumstances.

6. IMPLEMENTATION ASPECTS

This section addresses design issues on how to implement an agent “skeleton” for agents built in the architecture. Note that our long-term research focus is to study the interaction aspects of agents, and not how agents are internally represented. Nevertheless, we need to have an approach for the implementation of agents if we are to test the feasibility of agent interactions with this architecture. It is important to emphasize that our agent conceptualization is not intended as the ultimate guide for implementing agents in the architecture—or in any domain whatsoever, and it only shows a view we currently have found useful to approach the subject.

The approach we have chosen to conceptualize agents’ internals is shown in Figure 5. In this view, agents are conceptually organized in three modules: a communications module responsible for handling incoming and outgoing messages; an interactions module responsible for maintaining current, ongoing interactions; and a knowledge module responsible for the domain-specific functionality of the agent. The latter module also includes a repository (or library) with the conversation perspectives an agent is capable to participate on, e.g., a LAC is expected to have on its library a Registration conversation perspective template for the LAC role.
We have defined the communications module as consisting of a communications channel for incoming messages, and a communications channel for outgoing messages. Each channel has a buffer on which messages are cached after reception (in the case of incoming messages), and for their transmission (in the case of outward-bound messages). Currently, we are using KQML (Finin, Labrou, and Mayfield, 1997) as the format on which messages are structured for communication. In other words, this format is used ‘wrap up’ domain specific messages delivered between interacting agents. Recipient agents unwrap the domain specific message, which is then interpreted in the knowledge module.

Messages queued in the incoming buffer are retrieved by the interactions module, which tries to match them with one of the conversation perspectives maintained in a list of currently engaged in conversation perspective instances (CPI). If a match does occur, the message will be passed to the CPI for processing—i.e., for moving to the next state allowed by the received message in that conversation perspective. If a match does not occur, the interactions module will pass this message to the knowledge module, which will try to match it with one of the conversation perspective templates stored in the library of conversation perspectives. A match will occur if the type of this message is identical to that of the expected initial message in a conversation perspective template. If a match is not found, the knowledge module sends an error message back to the sender—via the interactions and communications module. Otherwise, if a match is found, a new instance of that conversation perspective is created and handed over to the interactions module, which will add it to the list of current CPI. At this point, this instance will be considered as one more of the ongoing, concurrent interactions of the agent.

---

3 It is possible that there were different conversation perspectives with identical starting messages, therefore creating a conflict on which perspective to select. This could be resolved by analyzing additional information (e.g., the history of previous interactions with the requesting agent) in order to select the most appropriate conversation perspective in each case. Another possibility is to combine the offending conversation perspectives into one composite template. This composite conversation perspective will contain the inferencing mechanisms to direct the flow of interaction to the appropriate message sequence, thus avoiding any non-standard mechanisms that might affect the genericness of the knowledge module.
An additional characteristic of the knowledge module is that it not only acts as a reactive process to messages received, but it can also pro-actively initiate an interaction by instantiating conversation perspectives. One example of the latter could be that of a manufacturing agent engaged in an interaction with a client requesting an order to make a product. Since the product is made of different parts, the manufacturing agent first needs to secure the supply of the parts prior to committing to the client. To that end, the manufacturer—still in an ongoing interaction with the client—spawns several interactions with its suppliers to request the required parts. Once the suppliers have assured the supply of parts, then the manufacturing agent can commit to the client agent.

One of the most challenging parts on building the agent “skeleton” was the implementation of conversation perspectives. In our design, each conversation perspective is designed as an object composed of a set of functions (which implement the states in the conversation perspective), and a series of mechanisms to support the reception and submission of messages. On the one hand, functions implement the processing expected on each state. However, it is often the case that this processing makes use of resources in the functionality component of the knowledge module (e.g., a LAC could maintain there a list of registered agents). Since this opens the possibility of simultaneous access to resources by different CPI, concurrency control mechanisms are implemented to maintain the consistency of these resources. In the case of MAS designers, this same practice is expected and should be compulsory for any agent with domain specific conversation perspectives that could concurrently access domain specific functionality. On the other hand, conversation perspectives use a buffer to store incoming messages and implements mechanisms to change states based on the status of message submissions and receptions. Each conversation perspective has its own thread of control.

For this implementation we decided to use the Java programming language (Arnold and Gosling, 1998), not only for their portability characteristics, but also for their multi-threading capabilities and comprehensive networking functions. This decision was also taken after resolving to use IP sockets as the communication channel between agents—this, in order to support the inherent distributed nature of multi-agent systems.

7. EXAMPLE

As an initial proof of concept for the architecture, we chose to implement an unsophisticated chat system on which agents act as intermediaries for the exchange of messages typed by human operators.

In this test case, chat agents register to pre-defined areas; where they request the location of known yellow pages agents; after being given that information, they query yellow pages agents and locate cooperation domain agents offering chat cooperation domains; after that, these chat agents choose and subscribe to a specific cooperation domain, engaging in the exchange of messages with other subscribed participants in the cooperation domain. Messages exchanged contain text typed by humans through the use of a graphical user interface displayed by each agent.

For this test case, we designed a simple scenario consisting of two areas, one yellow pages agent, one chat cooperation domain agent, and two chat agents. Although this scenario could have been
expanded to allocate more actors (to demonstrate the scalability of the architecture, for example), we decided to keep it minimal for demonstration purposes. In this scenario, we initially created the two areas (by initializing their LAC), followed by the activation of the yellow pages agent, the cooperation domain agent, and—finally—the chat agents. Figure 6 shows the distribution we pursued in this scenario. In brief, the dynamics of interaction were as follows:

- When the LAC started, they each initialized a network port as a communications channel from where to receive requests from agents. The address of the network port, along with the location of the area where the YP was to be located, were given as pre-defined parameters to each of these LAC.

- As part of its registration process, the YP informed its LAC (which is the LAC for the predefined area known to contain the YP) of its role as yellow pages agent. This way, the LAC could know which of its agents to contact when a request for a YP arrive.

- After registering to its area, the CDA inquired its LAC about known locations of yellow pages agents (in this case, just one). This (pre-defined) information is provided to the CDA, which then proceeds to advertise its services as a chat cooperation domain.

- After registering to their areas, the chat agents request the location of YP, to which they query for a chat cooperation domain service provider. Once the location of the (only active) chat CDA is identified, agents request to join a CD in this agent (which, in this example, only supports one chat CD). Once admitted as participants, agents engage in the exchanging of messages with the other participants of the domain.

Although the chat system exemplified is a simple MAS model, it should be seen—nevertheless—as a fully capable MAS model. This model is based on two conversation policies for the exchange of messages within the realms of a chat cooperation domain. These conversation policies are: Message-for-CD (for agents to send a message to the cooperation domain), and Message-for-Agents (for the CDA to broadcast a message to the participants of the cooperation domain). These two simple conversation policies, along with their corresponding conversation perspectives are shown in Figure 7.
One of the disadvantages of our current approach of implementing cooperation domains as broadcasters of messages by means of interaction protocols is that it requires the creation of one CPI per agent in the cooperation domain each time that a message is to be broadcasted. This characteristic makes our approach inefficient if compared to state-of-the-art broadcasting technologies such as multi-casting. It is possible that—in the future—we will examine the feasibility of such technologies for the implementation of cooperation domains.

8. RELATED WORK

Even though agent research started many years ago, it was not until recently that several industrial and research groups started to pursue the standardization of multi-agent technology. Prominent efforts, such as those of the Foundation for Intelligent Physical Agents (FIPA), General Magic, the Knowledge-able Agent-oriented System (KAoS) group, the Object Manager Group (OMG), and the Zeus toolkit at BT Laboratories are briefly described in the following sections. We feel obliged to acknowledge that several of the concepts we included in our architecture were inspired on the notions outlined by these research projects.

**FIPA's Model.** FIPA’s framework specifies a normative environment within which agents exist and operate, and a physical infrastructure for the deployment of agents (FIPA, 1997a). This framework is similar to our architecture in some notions, such as those of areas, local area coordinators, and yellow pages agents, which are similar to FIPA’s agent platforms (which are infrastructures in which agents can be deployed), agent management systems (described as agents that manage the creation, deletion, suspension, resumption, authentication, and migration of agents, and which also provide a “white pages” directory service), and directory facilitators (which are agents that provide a “yellow pages” directory service), respectively. One of the main differences between these architectures, however, is that FIPA’s approach to agent interaction is through a proprietary agent communication language called Arcol (FIPA, 1997b; Singh, 1998, p. 41), while our approach is using KQML.

**General Magic's Model.** This model conceptualizes multi-agents systems as electronic marketplaces where providers and consumers of goods and services find one another and transact business. Such marketplaces are modeled as networks of computers supporting a collection of places that offer services to mobile agents (White, 1997). This framework is similar to our architecture in that it describes the notion of places in a distributed environment, but they
differ—however—in that General Magic’s model is explicitly intended as a framework to support agent mobility, a characteristic that is not essential for agents in our architecture.

KAoS’ Model. The KAoS architecture is been designed with the same objective as our architecture: to support “an open distributed architecture for software agents” (Bradshaw, Dutfield, Benoit, and Woolley, 1997, p. 378). An important characteristic of this architecture is that it pioneers conversation policies as protocols to represent the interaction dynamics of agents’ interactions, an approach that is also followed in our architecture. Additional similarities include KAoS’ notions like agent domains—described as bound spheres of agent activity (a notion akin to that of areas), and domain managers—described as the agents controlling the entry and exit of other agents within an agent domain (a functionality related to that of LAC). KAoS also defines concepts that have no parallel functionality in our architecture, such as proxies and mediators—which mediate the inter-communication among KAoS agents from different object models, and the inter-communication with non-KAoS agents, respectively; and an agent structure with mechanisms to manage facts, beliefs, desires, intentions, and capabilities (ibid, p. 383).

OMG’s Model. This model is described as an agent environment composed of agents (i.e., components) and agencies (i.e., places) (Virdhagriswaran, Osisek, and O’Connor, 1995), where agents and agencies can interact with other agents and agencies using defined policies of interaction. In this model, agents are classified by their capabilities (e.g., inferencing, planning), type of interactions supported with other agents (e.g., data, domain-specific semantic requests), and mobility (e.g., static, movable with or without state). Agencies, on the other hand, are described to support concurrent agent execution, security, and agent mobility, among others. Our architecture is similar to this model in that agents exist within agencies (which are akin to areas), and that agents can communicate in different cardinalities, i.e., 1-1, 1-n, n-n (which is a functionality supported by cooperation domains).

BT Labs’ ZEUS Toolkit. The Zeus toolkit—developed at the British Telecommunications Laboratories—implements a rapid application development for multi-agent systems in task-oriented domains (Nwana, Ndumu, Lee, and Collis, 1999). The ZEUS toolkit consists of a set of three components: an agent component library (which provides the agent-level functionality—communications, reasoning, planning, and the like), an agent-building tool (which supports interactive creation of agents by visually specifying their attributes), and a suite of utility agents consisting of a nameserver agent (supporting a white pages directory), a facilitator agent (supporting a yellow pages directory), and a visualiser agent (for debugging societies of ZEUS agents). Our architecture is conceptually similar to ZEUS, although the latter includes an architecture for generic agents, and a comprehensive graphical user interface environment for the development of plan-oriented multi-agent systems.

9. CONCLUSIONS

This paper presented an overview of the ongoing research on a multi-agent systems architecture being developed at the University of Calgary. This architecture was described as composed of specialized basic agent-roles that interact by following low-level conventions named interaction protocols. These protocols are described using messages, which are the basic interaction elements, conversation policies, which define message dependencies in an interaction, and
conversation perspectives, which describe the state transitions that each interacting agent follows according to their participatory role in a specific conversation policy.

On the basis of these primitives for communication, more elaborate patterns of communication can be realised by (domain specific) applications. This is, MAS designers can use these primitive interaction protocols in two ways. First, new domain specific interaction protocols can be devised, which are supported by existing primitive interaction protocols. Secondly, complex, composite interaction patterns can be devised, which exist from numerous instances of the primitive interaction protocols.

Our main goal—as stated in this paper—was to define a generic multi-agent systems architecture for dynamic collaboration in an open environment. We are aware that the presented ideas and concepts represent only the initial steps in the evolutionary process of defining a reliable multi-agent systems architecture. As asserted by Wooldridge and Jennings (1998, p. 4), “to develop a new architecture that is both reliable and that offers sufficient power to be usable takes years of effort—not person years, but years.” While in the process of perfecting this work, we plan to use the architecture as a test bed on which to study the implications for software agents by researching and applying known social factors impacting the performance of humans in face-to-face interactions. In addition, researchers at CAG plan to use this architecture as an experimental vehicle on which to test multi-agent system models for manufacturing.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Douglas Norrie and researchers at the Collaborative Agents Group (CAG), specially to Rob Kremer, Bob Brennan, Fuhua Lin, Sudong Shu, Matsumoto Tsutomu, Mihaela Ulieru, and Xiaokun Zhang for their invaluable comments and suggestions to improve the material presented in this paper.

Financial support for Roberto A. Flores has been made available by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES


