Intelligent Matchmaking for Polar Ice Sheet Data Collection and Delivery

Costas Tsatsoulis, Sudha Sivashanmugam and Steven Perry
Department of Electrical Engineering and Computer Science
Information and Telecommunication Technology Center
The University of Kansas
1520 West 15th St., Room 2001, Lawrence, KS 66045, USA

Abstract— **The PRISM (Polar Radar for Ice Sheet Measurement) project is developing mobile, autonomous sensors for the measurement and study of the mass balance of the polar ice sheets. These sensors consist of intelligent radars integrated into robotic vehicles. They autonomously decide where and how to measure by examining a variety of information including onboard sensor data and collections of a priori knowledge. These data include the health and status of the rover, health and status of the sensors themselves, the state of the environment as measured by the sensors, satellite measurements of the area indicating expected ice sheet motion, and so on. All of this information is used to direct the data collection process by allowing for the dynamic configuration of sensors and the motion of the rovers that carry them. The PRISM intelligent sensor and rover control system is built upon a multiagent collaborative architecture that involves a number of distinct data collection and data dissemination agents functioning continuously and autonomously in a distributed computing framework. A critical component of this system is an agent service called the Matchmaker. The Matchmaker coordinates requests for information and services within the agent community and allows decision-making agents to locate and communicate with the data source-agents that can fulfill these requests.**

Keywords- intelligent agents; matchmaking; polar ice sheet measurements; autonomous radar and rover control;

I. Introduction

The PRISM (Polar Radar for Ice Sheet Measurement) project is developing mobile, autonomous sensors for the measurement and study of the mass balance of the polar ice sheets. The primary sensors include a Synthetic Aperture Radar (SAR) sensor and a dual-mode wideband radar sensor. The SAR sensor can operate in monostatic and bistatic mode at 60, 150 and 350 MHz. Its main purpose is to generate reflectivity maps of the bed which are used in the determination of basal ice sheet conditions, specifically the presence and distribution of basal water. The wideband dual-mode sensor features a radar depth sounder which is used to map deep layers and an accumulation radar to map near-surface layers.

These sensors are integrated into two vehicles; an autonomous robotic vehicle and a base vehicle. The base vehicle incorporates one dual-mode radar, the SAR transmitter, and monostatic SAR receiver. The robotic vehicle operates another dual-mode radar as well as the bistatic SAR receiver. These two vehicles are linked by a wireless network and an information system that allows them to share sensor data. The base vehicle features an additional communication system that can relay data back to a central location for further processing and long-term storage.

One important piece of this system is the ability of sensors to autonomously decide where and how to measure. These decisions are guided by a variety of information including onboard sensor data and collections of a priori knowledge. These data include the health and status of the rover insofar as it affects mobility of the sensors (e.g. power remaining, fuel level, status of motor, etc.), health and status of the sensors themselves, the state of the environment as measured by the sensors (e.g. snow cover depth, thickness of the ice sheet, condition of the basal level, i.e. roughness and presence of water, etc.), satellite measurements of the area indicating expected ice sheet motion, and so on. All of this information is essential in determining the operating range, mode and frequency of the sensors, as well as the motion of the rovers that carry them.

The PRISM intelligent sensor and rover control system is built upon a multiagent collaborative architecture that involves a number of distinct software agents functioning continuously and autonomously in a distributed computing framework. This community of agents is comprised of agents that represent data sources (source agents), and consumers of data, i.e. agents that require data to make decisions about the sensors and rovers (decision-making agents). A source agent's primary responsibility is to make data from a particular sensor or a priori information source available to the other agents in the community. A decision-making agent is responsible for controlling a particular parameter of the mobile radar system. These parameters include the radar frequency and mode as well as the speed and scan-path of the rover (a series of waypoints).

A decision-making agent reasons about the operation of the component it controls based on information obtained from source agents. In order to acquire this information, the decision-agents must know what types of data are required to make a decision and which agents can provide these data. Furthermore, a decision-making agent may need different kinds of data at different rates, depending upon the state of the mobile radar. For instance, when the SAR sensor is in monostatic mode, the rover should travel in a relatively straight line, thus the decision-making agent that controls the rover's

This work was supported in part by the National Science Foundation (Grant #OPP-0122520), the National Aeronautics and Space Administration (grants #NAG5-12659 and NAG5-12980), the Kansas Technology Enterprise Corporation, and the University of Kansas.

path does not need to know the direction in which the rover is heading, only the waypoint at the end of the path. However, when the SAR sensor is in bistatic mode, the rover must make repeat passes across the swath being measured, so the decision-making agent that directs the rover's path must continually monitor the rover's heading at each point along that path.

A central feature of the multiagent architecture is a service called the Matchmaker. The Matchmaker coordinates requests for information and services within the agent community and allows decision-making agents to locate and communicate with the source-agents that can fulfill their requests. The Matchmaker must know about all the agents in the virtual multiagent environment, what types of data they can provide, and how often they can provide it.

When a sensor is turned on, the source agents associated with it awaken. For example, when the rover is started, the temperature, fuel level, power, speed, GPS, etc. source agents are started. The first thing a source agent must do upon awakening is to register with the Matchmaker. During registration, a source agent announces its existence and tells the Matchmaker about the data it can provide and the frequency by which it can provide the data. Decision-making agents are awakened in response to the activation of the system components they control. For example, when the radar is started the decision-making agent that controls it is also started. Decision-making agents also register with the Matchmaker. When a decision-making agent is started it notifies the Matchmaker of its existence, then tells the Matchmaker about each type of data it requires and how frequently it requires that data. The Matchmaker then brings together the source agent that produces the data and the decision-making agent that requires it.

When a decision-making agent requires a new type of data, or requires data at a different rate, it again queries the Matchmaker. The Matchmaker analyzes the new needs of the decision-making agent and attempts to match those needs to the source agents that can fulfill them. If no source agent is available to fulfill a need, the Matchmaker informs the decision-making agent so it can modify its behavior accordingly.

The Matchmaker allows for matching between agents that require data and other agents that can provide that data. It builds flexibility into the agent community because it allows for agents to dynamically issue and retract statements about their capabilities and needs. Without the Matchmaker, each agent would have to know specific information about every other agent that it communicates with. A Matchmaking multiagent architecture can provide direct, timely, and intelligent data collection.

II. MATCHMAKING

Multiagent systems are distributed systems composed of many intelligent, autonomous agents that work together to accomplish common goals. Each agent has a limited viewpoint or provides a certain set of capabilities to the community of agents. In multiagent systems where different agents provide different capabilities, agents must communicate with each other in order to work towards the common goal.

Multiagent systems that feature communication between agents all face the problem of connecting information providers to information consumers. Kuokka and Harada proposed matchmaking as a solution to this problem. They describe matchmaking as a cooperative partnership between information providers and consumers assisted by an intelligent facilitator [1]. The PRISM Matchmaker is an example of such a facilitator.

Matchmaking attempts to dynamically match capabilities and needs and notifies information providing agents and information consuming agents about potential match-ups. Since matchmaking is a type of automated reasoning, it requires that capabilities and needs be represented in a machine-readable, formal knowledge-sharing language. In multiagent systems, these are often message and content languages. Some researchers have experimented with specific languages for expressing agent capabilities, such as Gil and Ramachandran's EXPECT language [2].

There are several different patterns for implementing a matchmaker service. In one pattern, the consumer notifies the matchmaker of its need and requests that the matchmaker suggest one or more providers that can fulfill the need. Under this model consumers are responsible for selecting the preferred provider if more than one is suggested by the Matchmaker. After selecting a provider from the list of matches returned by the matchmaker, the consumer initiates communication with it. In other matchmaking systems, the consumer notifies the matchmaker of a need and stipulates that the matchmaker forward the request directly to the provider with the understanding that future interactions between the provider and the consumer will bypass the matchmaker. The major difference between this approach and the previous one is that instead of merely making a recommendation, the matchmaker selects the best match and initiates communication between the consumer and the provider. Finally, in brokered matchmaking, all interactions between the provider and the consumer are facilitated by the matchmaker. Each of these different approaches to matchmaking has a different impact on communication efficiency and bandwidth utilization.

The PRISM multiagent framework allows for multiple source agents to provide the same capability. This is especially useful because it means we can add redundant sensors at a future date. As in other multiagent systems, agents communicate by passing messages to each other. Agents also

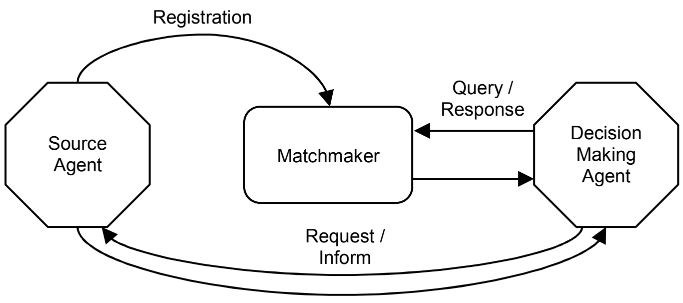

Figure 1. Agent interactions with the Matchmaker

communicate with the Matchmaker through messages. These messages are FIPA (Foundation for Intelligent Physical Agents) compliant ACL (Agent Communication Language) messages in XML (eXtensible Markup Language) format. In the PRISM system, different agents can reside on different physical computers, so message passing occasionally involves communication over a rover's wireless network. Because several source agents can provide the same type of data, and because the network bandwidth between the rovers is limited, the PRISM Matchmaker takes the first approach to matchmaking: in response to a request for service from a consumer agent, the Matchmaker returns a list of matching source agents that can fulfill the request. The decision-making agent then selects the appropriate source agent from this list of suggested matches and initiates communication with it. This approach allows for matching a single consumer request to multiple providers, and also minimizes bandwidth usage.

III. Representing and matching capabilities

In the PRISM multiagent system, different source agents provide different capabilities. Most of these capabilities are quite simple to represent. For example, the temperature agent can provide the internal temperature of the rover as a floating point number. Other capabilities are slightly more advanced in that they require some kind of input from a requesting agent in order to provide data. For instance, the scientific interest agent, given a location's GPS coordinates, can provide a pre-determined estimate of the scientific interest in that location by examining an onboard knowledge base. Despite the large number of capabilities provided by source agents, most of these capabilities consist of the simple data types returned from sensors or well-defined a priori data sets. For this reason the PRISM multiagent system does not require a full-fledged ontology.

In their basic form, capabilities are represented in the PRISM multiagent system by a simple string name and an identifier of the data type they return. For instance, the temperature agent returns the internal temperature of the rover as a floating point number. If a source agent defines a capability, then it must be able to fulfill one-time requests to provide that capability. In a one-time request, the decision-making agent sends a request message to the source agent. The source agent examines the message, and if it is understood, the source agent will send an inform message back to the sender that contains the current value of the data source represented by the requested capability.

In addition to simply being able to respond to a one-time request, many source agents can provide regular updates about the recent data obtained from their data source. In the PRISM content language, this is called a subscription. Once the Matchmaker has suggested a source agent that can be subscribed to, the decision-making agent sends special type of request message to the source agent. The source agent will respond with one or more inform messages over time until the subscription expires or is cancelled.

The PRISM framework allows for three different types of subscriptions. An always-update subscription request has a single parameter: the capability to subscribe to. Always-update subscriptions allow the source agent to control the message frequency (the time interval between subsequent inform messages). Most agents respond to an always-update subscription request by sending a response as frequently as possible. The always-update subscription request uses the FIPA request-whenever communicative act. A periodic subscription allows the decision-making agent to attempt to set frequency of inform messages and can have an optional expiration time. For example, a decision-making agent would use a periodic subscription to ask a source agent to send data every 100 milliseconds for the next two minutes. Periodic subscriptions also use the request-whenever performative and have three parameters: the name of the requested capability, the desired interval between inform messages, and an optional expiration time for the subscription. Finally, the PRISM framework allows for event based subscription. Event based subscriptions use the request-when performative and request that the source agent sends an inform message to the decision-making agent whenever some statement about the data source wrapped by the source agent is true. An event-based subscription has two parameters, the requested capability and a conditional statement with which to evaluate the current value of the data source. All subscription requests are expressed in the PRISM content language which is an extension of the FIPA RDF (Resource Description Framework) standard. A basic library of conditionals is defined in the RDF content query language and includes functions such as equals, greater-than, less-than, among others. These conditionals can be combined with basic Boolean expressions. A decision-making agent would use an event-based subscription to ask a source agent to send it data every time the value of that data is greater than some threshold.

Since all subscriptions start with a single request from the decision-making agent to the source agent, and are followed by multiple inform messages sent from the source agent to the decision-agent, not all capabilities can be subscribed to. For example, the scientific interest capability cannot be subscribed to because it requires additional input in order to fulfill a request for information.

When a source agent is started, it immediately connects to the Matchmaker in order to register itself. During registration an agent announces its existence and the capabilities it can provide to the agent community. These capabilities include not only one-time access to specific data sources, but also subscriptions to constantly-changing data sources. For each capability it provides, a source agent must notify the Matchmaker of the capability's name, any input parameters it might take, as well as its return type (String, integer, floating point, etc.). Additionally, the source agent notifies the Matchmaker of the different types of subscriptions that are offered for the capability. Minimum limits for subscription parameters such as the periodic interval subscription are specified as well. All of this information is encoded in an advertise message. The *advertise* performative is a PRISM-specific extension to the FIPA Communicative Acts library.

When a decision-making agent needs a new type of data or needs to change its subscription frequency in response to some change in the environment, it must query the matchmaker. It is during this query process that the Matchmaker matches an agent's need to the capabilities provided by other source agents in the community. In the PRISM multiagent framework, matching is a relatively simple process.

Before matching, the decision-making agent first formulates a query message which will be sent to the Matchmaker. The query message contains a special PRISM content language statement that specifies the requested capability name as well as the request type (simple request, always-update subscription, periodic subscription, or event-based subscription). If the request type requires the specification of additional parameter values (such as the period and timeout values for a periodic subscription), then the values of these parameters must also be included in the query message.

When this query message is sent to the Matchmaker, it examines the message and searches the capabilities list to first find out which, if any, agents provide this capability. This consists of a string match on the name of the capability. If the Matchmaker finds potential matches, it next examines the specifics of the request type. If the query refers to a simple request, the Matchmaker immediately returns the identities of all matching source agents to the decision-making agent. However, if the request is for a subscription type, the Matchmaker checks each match to make sure that it allows subscriptions of this type to this capability. If any source agent matches on capability, but not on subscription type, it is removed from the list of possible matches. Finally, the matchmaker sorts the remaining possible matches based on how close the values of their subscription parameters match the specific minimum parameters. For example, assume that a decision agent requires a periodic subscription to the GPS position capability that will inform it of the rover's position once a second. If two agents can provide periodic subscriptions, and one of the two can only provide this information once every ten seconds, while the other can provide this information every two seconds, the Matchmaker will prioritize the list so that the source agent that can send at two second intervals is preferred.

This flexible matchmaking system does not try to provide an exact match, merely the best possible match. It was designed with the idea that decision-making agents would rather receive infrequent updates than no updates at all.

IV. Implementation

The PRISM multiagent framework, which includes the Matchmaker, and all source and decision-making agents, was developed in the Java programming language. Each agent is designed as a thread and contains its own message interpreter, scheduler, and message-to-Java binding system. The Matchmaker was designed as a special type of agent that implements the Mediator agent design pattern. All other agents implement the Collaborator agent design pattern. Agents communicate over the network by Java Remote Method Invocation (RMI). Messages are encoded in XML and sent as Strings between agents using a remote interface.

When a source agent is started, it first looks up the Matchmaker in the RMI registry. After getting a remote interface to the Matchmaker, the agent can announce itself and advertise its capabilities by sending a single message. When a decision-making agent queries the matchmaker for source agents that can fulfill a need, the decision-making agent again talks to the Matchmaker via XML messages over RMI. After matching the request to one or more source agents that can fulfill it, the Matchmaker returns a list of remote interfaces for the matching agents. The decision-making agent can then initiate communication with the source agents directly. RMI is also used for normal inter-agent communication.

RMI allows the option of running different agents on different Java virtual machines. Because the rovers are linked with an 802.ll wireless network, different agents can run on different machines on different rovers and all form a single virtual community.

V. Results and Conclusions

The PRISM multiagent system and Matchmaker have been implemented and tested on a single autonomous vehicle with several sensors. Work is ongoing and the custom radar sensors are still in development as of the writing of this paper. However, we have constructed the agents that will wrap these data sources and have tested them with simulated data sources in a large agent community that mixes agents wrapping both simulated and actual sensors.

We have shown that the PRISM Matchmaker and multiagent system perform well in the near real-time setting of field experiments. Though decision agents made decisions based in part on simulated radar data, all agents initialized and registered, capability advertisements and matchmaking worked properly, and agents fulfilled a variety of requests and subscriptions. Computing and network utilization were all within acceptable limits. Finally, we examined the flexibility provided by the Matchmaker architecture in tests where some agents were forced to drop out of the agent community and then later were restarted. This tested the behavior of the system when the Matchmaker failed to match a request to a source agent. The decision making agents that made the request were properly notified by the Matchmaker of a failure to match and adjusted their decisions accordingly.

In future the PRISM system will be extended and simulated data sources will be replaced by actual radar sensors.

References

[1] Kuokka D. and Harada, L. "Supporting Information Retrieval via Matchmaking," Working Notes of the AAAI Spring Symposium Series, pp. 111-11, Stanford University, March 27-29, 1995.

[2] Gil Y. and Ramachandran, S. "PHOSPHORUS: A Task-Based Agent Matchmaker," Proceedings of the Fifth International Conference on Autonomous Agents, pp. 110-11, Montreal, Canada, May 2001.

[3] Kendall E. A, Murali Krishna, P.V., Pathak, C.V. and Suresh, C.V. "Patterns of Intelligent and Mobile Agents," Proceedings of the Second International Conference on Autonomous Agents, pp. 92-9, Minneapolis, Minnesota, USA, May 1998.

[4] Cruickshank D, Moreau, L., and De Roure, D. "Architectural Design of a Multi-Agent System for Handling Metadata Streams," Proceedings of the Fifth International Conferent on Autonomous Agents, pp. 505-12 , Montreal, Canada. 2001