A Declarative Approach to Agent-Centered Context-Aware Computing in Ad Hoc Wireless Environments

Gruia-Catalin Roman[1] Christine Julien[1] Amy L. Murphy[2]

[1]Department of Computer Science
Washington University
Saint Louis, MO 63130
{roman, julien}@cs.wustl.edu

[2]Department of Computer Science
University of Rochester
Rochester, NY 14627
murphy@cs.rochester.edu

ABSTRACT

Much of the current work on context-aware computing relies on information directly available to an application via context sensors on its local host, e.g., user profile, host location, time of day, resource availability, and quality of service measurements. We propose a new notion of context, which includes in principle any information reachable via the ad hoc network infrastructure but is restricted in practice to specific views of the overall context. The contents of each view will be defined in terms of data, objects, or events exhibiting certain properties, associated with particular application components, residing on particular hosts, and part of some restricted subnet. Location, distance, movement profile, access rights, and a lot more will be available for use in view specifications. The underlying system infrastructure will interpret the view specifications and continuously update the contents of user-defined views despite dynamic changes in the specifications, state transitions at the application level, mobility of hosts in the physical space, and the migration of code among hosts.

1. INTRODUCTION

The foundation of this work is the notion that context-aware computing holds the key to achieving rapid development of dependable mobile applications in ad hoc networks. *Context-aware computing* refers to the explicit ability of a software system to detect and respond to changes in its environment, e.g., a drop in the quality of service on a video transmission, a low battery level, or the sudden availability of much needed access to the Internet. Most current facilities supporting context-awareness are relatively simple and limited in scope. When the needs of the application must reach beyond the basics (e.g., the application requires access to services available at a remote location), the programmer needs to contend with more complex processes that include discovery and communication. While these extra costs may be acceptable in wired networks where connections persist over extended periods of time, in ad hoc networks the complexity of managing frequent disconnections can significantly increase the programming effort. Yet, mobile systems do need access to a broad range of resources, maybe even more so than distributed applications.

Of interest to us is the ease with which resources can be acquired and retained in the presence of mobility. Our idea is to extend the notion of declarative specifications to a broader set of resources and to provide the mechanisms needed to maintain access to the specified resources despite rapid changes in the environment caused by the mobility of hosts, migration of software components, and changes in connectivity. For instance, an application on a palmtop should be able to declare its need for printer access and, as the owner travels along, a printer should always appear on the desktop, as long as some printer is within wireless communication range. Of course, building such an application with today's technology is feasible, but coding it cannot be reduced to the simple act of providing a declaration in the program. We contend that we can accomplish the latter (and a lot more) by extending the notion of context-aware computing and by developing a software infrastructure that continuously secures the resources declared in the application program.

2. DECLARATIVE SPECIFICATION OF VIEWS

We assume a computing model in which hosts can move in physical space and the applications they support are structured as a community of software components called agents that can migrate from one host to another whenever connectivity is available. Thus, an agent is the unit of modularity, execution, and mobility, while a host is a container characterized, among other things, by its location in physical space. Communication among agents and agent migration can take place whenever the hosts involved can physically communicate with each other, i.e., they are connected. Since the notion of context is always a relative one, we will use the term *reference agent* to denote the agent whose context we are about to consider, and we will refer to the host on which this agent is located as the *reference host*. An agent's location is always a host, while a host's location is always a

.

point in some physical or logical space.

An ad hoc network is defined as a closed set of reachable hosts. In principle, the context associated with a given agent consists of all the information available in the ad hoc network. We will refer to it as the *maximal context* of the reference agent. Of course, such broad access to information is generally costly to implement. In addition, various parts of the same application may need different resources at different times during the execution of the program. For this reason, we believe that it is important to structure the context in terms of fine-grained units which we call views. A *view* is a projection of the maximal context together with an interpretation that defines the rules of engagement between the agent and the particular view.

The concept of view is agent centric in the sense that every view is defined relative to a reference agent and with respect to its needs for resources from and knowledge about its environment. An agent sees the world through a set of views. The set may be altered at will by defining, redefining, and deleting views as processing requirements demand. The expected software engineering gains will be determined to a great extent by the level of flexibility and simplicity we can offer to the application programmer. Our strategy focuses on declarative specifications. A rich set of criteria can be employed towards this purpose. For instance, one ought to be able to describe the view contents in terms of phrases such as:

> *All specials (reference to objects) posted by family restaurants (reference to agents) within a mile (implicit reference to hosts) of my current location (property of the reference host).*

In general, constraints on the attributes of the desired resources (data or objects) and the agents that own them are an effective way to restrict a view's contents. They must be combined, however, with constraints on the attributes of the hosts on which the agents reside and with properties of the ad hoc network in the immediate vicinity. Security and network considerations will likely emerge as important research issues in any effort to design a language for view specification. At the network level, for instance, it may be desirable to limit context to a connected subnet of the ad hoc network forming a region around the reference host. The network topology, geometry, physical distribution in space, and security enforcement procedures may play a role in determining the shape of the region of interest. These considerations are new to context-aware computing and are injected by our focus on ad hoc mobility.

The adoption of a declarative context specification is motivated by the expectation that transparent context management will shift to the underlying middleware many of the burdens programmers face in the development of applications for use over ad hoc networks. Moreover, the programmer can be given explicit control over the cost associated with context management. The programmer controls the scope of the view (a large or small neighborhood), the size of the view (the range of entities included), and the relative cost of executing a particular operation on that view (by defining the level of consistency, e.g., best effort versus transactional semantics). Finally, by having access to a complete specification of the needs of all agents residing on the same host, opportunities for optimizations arise naturally.

3. MODELS OF CONTEXT-AWARENESS

Because we see ad hoc mobility as a fundamental challenge to developing the next generation of consumer, industrial, and military applications, we seek to develop new models of context-awareness able to accommodate the complexities of mobile computing, to build middleware that embodies these models, and to evaluate both on interesting application test beds. This section offers a broad-brush discussion of the four types of context-awareness models we are currently investigating. Our models reflect those popular in distributed computing, but we expect new technological advances to result from our special focus on their applicability to ad hoc networks, the introduction of declarative specifications of context, and automatic context maintenance.

To show how an agent's interaction with the view differs among the four models, we introduce an example that we will revisit throughout this section. Consider a team of robots exploring an uninhabited planet. The robots need to perform experiments that require precise relative locations and instrumentation that no single robot can carry. For example, some robots may be able to precisely sense their locations, some may be able to sense the ambient temperature, others may sense atmospheric pressure, and still others may collect data about the soil composition. All of these pieces of information have the potential to contribute to the operating context of any agent in the system. Now consider a specific agent that requires two pieces of location information from other robots (for determining relative locations) and a single piece of temperature information (for performing its experiment). To satisfy its needs, the agent defines two views. The first is defined to contain the location data items that are between some minimum and maximum distance from the agent's robot. The second view contains temperature data items within a specified number of network hops. The agent can dynamically adjust its view specifications as its needs change. The agent's style of interaction with these views depends upon the features of the context-awareness model in use by the system. As we describe the several context-awareness models, we will revisit this example to elucidate how the agent interacts with its temperature view in each model.

Context-Sensitive Data Structures. In many distributed systems, data access serves as the primary form of interaction among components. In mobile computing, several systems have used shared tuple spaces as a coordination medium. MARS [1] is concerned with interactions among mobile agents and employs a single tuple space per host to facilitate coordination among co-located mobile agents. LIME [6] relies on transient sharing of tuple spaces among agents on the same host and among hosts within communication range. This enables LIME to provide support for both physical and logical mobility. Other systems have explored different data structures. PEERWARE [3], for example, stores documents in trees and adjusts the tree structure to account for mobility. All these systems assume a symmetric model of sharing. When a group of components is formed, they all share the same data, and they perceive it in the same manner. By contrast, our proposed model allows each individual agent to define its own perspective of the data available in the world in terms of one or more views. This asymmetry, a distinguishing feature of our model, allows each agent to assume responsibility for and control over the size and scope of the data it accesses. For example, an

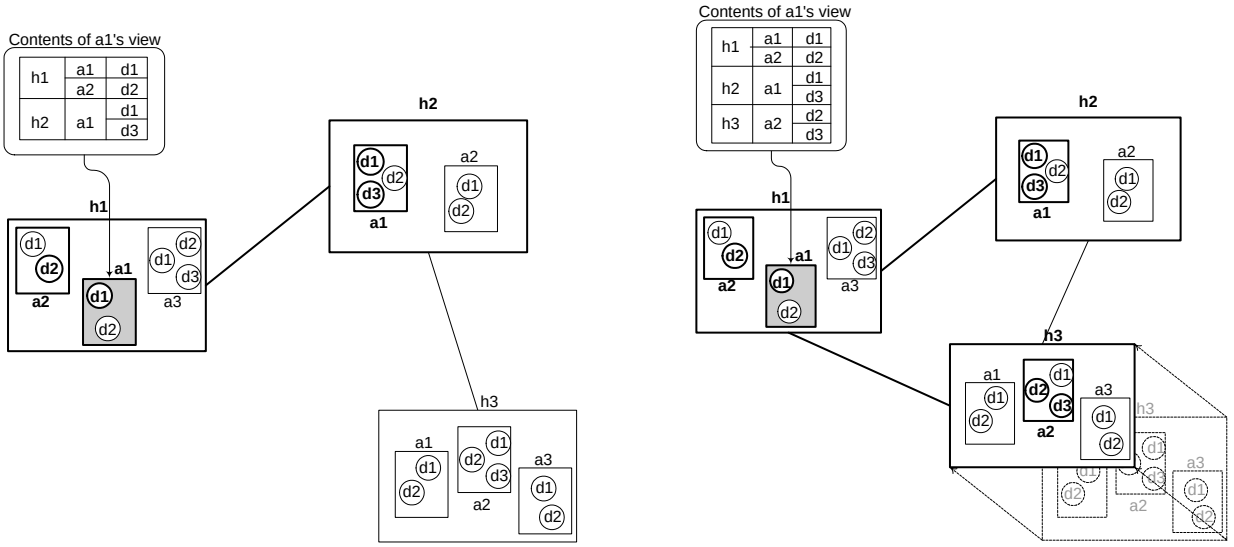

Figure 1: View dynamics. Data items visible to the reference agent a1 located on host h1 before and after h3 moves into the range of h1. Hosts, agents, and data items with darkened borders contribute to the view, while ones with lighter borders do not satisfy the specification.

agent associated with a managing robot that monitors the activities of other robots in its vicinity might define a view that includes the locations and activities of all other robots within a certain distance, which may be continuously adjusted as the exploration progresses.

In general, we envision allowing the agent's view to contain a *representation* of a subset of the data available in the ad hoc network. The choice of representation is a defining feature of each specific instantiation of the general model. In the context-sensitive data structures model, the view's representation is a simple data structure (e.g., a tuple space). The three remaining models build on this foundation. The choice of data included in the view, i.e., its *contents*, is determined by the view specification. The latter is given in a declarative manner by stating constraints on the network, hosts, agents, and data that contribute to defining the view. One can impose restrictions on network properties (e.g., number of hops, distances, bandwidth, etc.) so as to define a connected subnet immediately surrounding the reference host. We expect this kind of locality to help control the context maintenance costs while meeting the needs of most mobile applications. Within this contextual setting, further restrictions can be imposed on the properties of the physically mobile hosts (e.g., power availability, devices supported, etc.) in the subnet and of the mobile agents supported by the admissible hosts. Finally, data associated with the remaining eligible agents can be filtered to produce the actual contents for that view. As hosts and agents move and properties of the network components change over time, the contents of the view must be transparently updated for the reference agent.

The dynamic nature of the view definition is illustrated in Figure 1, where the depicted view of agent `a1` changes as the distance between hosts `h1` and `h3` decreases. Agent `a1` is grayed to indicate that it is the agent specifying the view. Hosts, agents and data items that contribute to the view are shown with darkened borders. In part (a) of the figure, because of `a1`'s specification, only hosts `h1` and `h2` qualify to contribute agents to the view. Because of the restrictions on agent properties and data properties, only certain data items on certain agents on these hosts appear in the view. The balloon pointing to `a1` shows a table of the hosts, agents, and data items contributing to `a1`'s view. As part (b) shows, when host `h3` moves closer to `h1`, it satisfies the view's constraints. Again, only certain data items on certain agents appear in the view. Exactly which hosts, agents, and data items contribute is determined by the application-provided view specification.

In this model, the view representation takes the form of a standard data structure. For the purposes of discussing our example, we assume the view's data structure is a tuple space with which the robot agent interacts by performing standard tuple space operations. Figure 2 shows this general pattern of interaction. This figure and all subsequent ones show a virtual picture of an agent's view where both remote and local tuples are included in a single "soup". The actual distribution of the information in logical and physical space (as shown in Figure 1) is omitted. The tuple space operations, or requests, include reading and removing data from the view. Additionally, the tuple space might provide reactive behaviors whereby the robot agent can react to the appearance of new data items in the view. Tuples match operations or reactions through content-based pattern matching, i.e., an agent selects data by specifying values for fields in the tuples. For example, a robot agent might gain an initial temperature reading by performing a read operation for a tuple corresponding to any probe (`p`), labeled as temperature data (by the string `temp`), with any

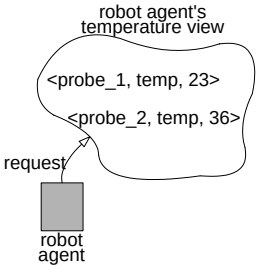

Figure 2: Agent/view interaction in the context-sensitive data structures model.

temperature value (v):

$$\mathsf{read}(\langle \mathit{probe} : \mathsf{p}, \mathsf{temp}, \mathit{value} : \mathtt{v} \rangle)$$

If the robot wants to receive subsequent readings from the same temperature probe, it might register a reaction:

$$\mathsf{react\ to}(\langle \mathsf{p}, \mathsf{temp}, \mathit{value} : \mathtt{v} \rangle, \mathsf{A})$$

The action A will be performed whenever the temperature probe p outputs a new temperature reading that satisfies the view specification.

Context-Sensitive References. One can easily extend the context-sensitive data model so that the view contains objects and object references instead of data items. An agent obtains an object reference and description from the view. The agent then uses this information to interact with the remote object directly by invoking methods in it. The agent can continue to use the reference but receives no guarantees regarding the stability of the remote object because the interaction occurs outside the view.

In using the context-sensitive references model in the robot environment, the temperature data is encapsulated in objects. Instead of reading data items directly from the view, the robot agent reads an object reference based on requirements it provides. The information returned indicates the remote object's location and information about how to interact with the object (i.e., its available methods). Figure 3 shows this style of interaction. A robot agent might request from the view a reference to a temperature object at a location (l) within 2 meters:

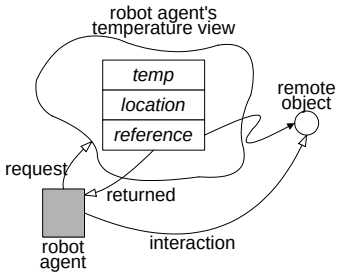

Figure 3: Agent/view interaction in the context-sensitive references model.

$$\mathsf{read}(\langle \mathit{object\ reference} : \mathtt{r}, \mathsf{temp}, \mathit{location} : \mathtt{l} :: \mathtt{l}{-}\mathtt{here} < 2\mathsf{m} \rangle)$$

For a more complicated request, the agent could require that the object reference returned provide a particular method. The robot agent can interact directly with the remote object by invoking methods on it. For example, a temperature object might have methods `getCelsius`() and `getFahrenheit`(), and the robot agent could call either method depending on its needs:

`r.getCelsius`()

The agent can hold the reference as long as it desires, however, if the reference object disappears, an exception will be generated if the robot agent attempts to use the stale reference. In such a case, the agent must obtain a new reference from the view.

Context-Sensitive Bindings. Traditional distributed systems, like CORBA-compliant systems [5] and Jini [4], hide many of the details of object distribution from the programmer. The general pattern of interaction requires a client to find an object in the lookup service and then bind to it, allowing the programmer to invoke methods on the remote object as if it were local. If the remote object fails, the client must revisit the lookup service to retrieve a new reference. In a mobile environment, objects move, and bindings are more likely to break. The middleware supporting the view concept transparently manages bindings, hiding both the lookup service and object mobility from the programmer. In general, the view contains a set of objects owned by connected agents. The set of available objects depends on the reference agent's view specification. However, the programmer does not access this set of objects directly. Instead he can request bindings to objects in the view. As agents and the objects associated with them move, the bindings are maintained and transparently updated to select new objects as needed. As an example of a view, consider a reference agent responsible for printing documents. Its view might contain all printers available on the current floor in the current building. The agent might then request a binding to the highest quality printer. As the agent moves, the set of available printers changes, and therefore the binding automatically changes. Our current implementation plan focuses on adding this functionality as a thin veneer over the context-sensitive references model. This veneer will hide the view contents and will service a binding request by locating an object in the view that matches the binding specification and by creating the connection to it for the agent. The layer will also have to respond to changes in the available set of objects in order to maintain, update, and break bindings when necessary.

Because the robot agent requires a single temperature reading, when using the context-sensitive bindings model, the agent would request a single binding to a temperature object. This model allows the agent to specify a binding policy which helps select the "best match" for the binding from among the objects in the view. In our example, the agent might request to bind to the temperature probe with the highest precision. A binding request might look like:

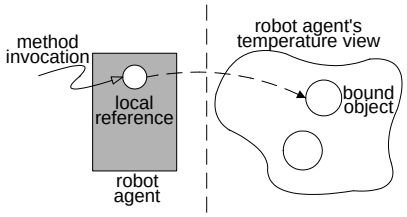

Figure 4: Agent/view interaction in the context-sensitive bindings model.

$$\mathsf{bind}(\langle \mathit{object\ reference} : \mathtt{r}, \mathsf{temp} \rangle)\ \mathsf{highest\_precision\_policy}$$

The agent interacts with the object by invoking methods on the binding:

`r.getCelsius`()

Figure 4 shows these interactions. If the bound object disappears or a new object appears that better satisfies the binding policy, the middleware automatically updates the binding. The system generates an exception only when no object in the view satisfies the binding request.

Context-Sensitive Events. Another model we plan to investigate allows agents to interact through a language of events. In this case, the view contains events generated by components in the system. For example, an agent monitoring robot activity might define a view containing events generated when new robots (hosts) connect and are within a certain physical distance. Event-based interactions have become common in distributed systems. The JEDI system [2], for example, uses a distributed event dispatcher through which active objects communicate by generating events and

registering to receive events. The view concept we propose introduces allowances for ad hoc mobility and the capability to restrict the scope of visible events based on the network, hosts, agents, objects, and the events themselves.

In this case, the objects themselves are not directly visible to the reference agent, only the events they generate are visible. These events are filtered by an event specification. Agents operate on this resulting view of events by binding callback functions to the events or prescribed sequences of events which pass through the filter. Any application-defined object can generate events, allowing agents to respond to both application specific events as well as generic events such as a change in an object data field. To ensure a unified treatment of all events and uniformity of the view contents, we expect to postulate (for specification purposes) the existence of virtual objects so named as to refer to application agents, hosts, and network resources abstractly. Their role will be to pass on system generated events to the context, but their implementation will be hard coded in the middleware. The implementation of context-sensitive events will appear as a veneer over the context-sensitive data structures model.

In this model, the robot agent interacts with events generated by temperature objects instead of interacting with the data or objects themselves. The robot agent registers to receive temperature events from its view. As shown in Figure 5, this registration attaches a callback function provided by the agent to the generation of satisfying events. As the figure indicates, this style of interaction completely hides the view's contents from the robot agent. The agent will, however, receive all events generated by all temperature probes in the view. To handle this, the agent has several choices. One is simply to filter these events locally, using the information only as the application desires. A second option would detect a single "first" event and remember the source, `r`. The callback for this event would deregister the initial registration and register the agent for only events originating at `r`. Of course, this option requires the agent to explicitly handle the failure or disappearance of `r`.

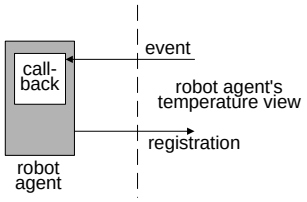

Figure 5: Agent/view interaction in the context-sensitive events model.

Even for this simple example, all four models have advantages and disadvantages. The model chosen for use depends on factors as varied as the guarantees required by the system and the application developer's preferred programming paradigm.

4. DISCUSSIONS

Our experiences in the ongoing development of the LIME middleware provides us with a foundation for beginning this model's implementation. An initial implementation of the basic context-sensitive data structures model is in progress. This prototype builds directly on top of the LIME middleware and provides most of the capabilities required by the model. This initial prototype allows us to begin the development of the applications that spurred this investigation. Further work on the middleware's development will replace LIME with the provision of true asymmetric behavior and will allow for an empirical evaluation of the performance of the applications. We approach this development effort from a bottom-up perspective. The lowest level requires algorithms and protocols for gathering information from sensors and disseminating that information in a timely fashion. We have already developed an algorithm for consistent group membership [7] that uses location information to provide the appearance of announced disconnection in spite of host mobility. Other work on providing an abstraction of the network based on properties of network paths [8] provides a foundation for implementing the view abstraction required in this model. At each layer of the implementation, key issues related to the highly dynamic ad hoc environment will have to be addressed. As mentioned in the introduction, one such issue concerns the application's ability to specify the level of consistency guarantees it requires for particular operations over particular views. As always, another key element of the final implementation involves tradeoffs between system expressiveness and the efficiency of its implementation. Specifically, the view specification language should be as flexible as possible without losing the efficiency gains associated with the provision of the asymmetric model. The initial prototype will be useful in evaluating possible specification mechanics, but the final evaluation will come with the full implementation of the asymmetric model.

5. CONCLUSIONS

As software must function in settings that are increasingly open and highly dynamic, software development is becoming more complex. While we cannot eliminate the intrinsic complexity of software artifacts operating under such demanding circumstances, we can reduce the complexity of application development by shifting much of the burden onto the system support infrastructure. Programming power can be amplified by allowing the developer to think at a new and higher level of abstraction. Effective use of the limited resources often associated with mobile systems can be achieved by having the system infrastructure explicitly know what the application needs are at any given point in time.

6. REFERENCES

[1] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: A programmable coordination architecture for mobile agents. *Internet Computing*, 4(4):26–35, 2000.

[2] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, September 2001.

[3] G. Cugola and G. Picco. PEERWARE: Core middleware support for Peer to Peer and mobile systems. Technical report, Politecnico di Milano, 2001.

[4] K. Edwards. *Core JINI*. Prentice Hall, 1999.

[5] W. Emmerich. *Engineering Distributed Objects*. John Wiley and Sons, Ltd., 2000.

[6] A. Murphy, G. Picco, and G.-C. Roman. LIME: A middleware for physical and logical mobility. In *Proc. of the 21st Int'l. Conf. on Distributed Computing Systems*, pages 524–533, April 2001.

[7] G.-C. Roman, Q. Huang, and A. Hazemi. Consistent group membership in ad hoc networks. In *Proc. of the 23rd Int'l. Conf. on Software Engineering*, May 2001.

[8] G.-C. Roman, C. Julien, and Q. Huang. Network abstractions for context-aware mobile computing. In *Proc. of the 24th Int'l. Conf. on Software Engineering*, (To appear).