

Middleware for Protocol-Based Coordination in Mobile Applications

Kurt Schelfhout, Danny Weyns, and Tom Holvoet • Katholieke Universiteit Leuven

In distributed mobile applications, component interaction is complicated by dynamics in the environment. ObjectPlaces middleware supports the development of interaction protocols in dynamic, mobile environments to facilitate component coordination.

Distributed mobile applications offer application developers unprecedented flexibility because mobile network nodes move around freely while also staying connected to the wireless network. Distributed applications gain mobility at a price, however: because application components are deployed on mobile nodes, the components must contend with the context changes that a dynamic network entails. This is particularly challenging for coordination, which involves aligning distributed components' behavior to achieve an application's overall requirements.

This coordination is typically accomplished through a distributed protocol—that is, an exchange of multiple, related messages among application components on different nodes. In mobile applications, nodes continuously come and go and change interaction partners, complicating this exchange. For protocol-based coordination, existing middleware approaches—such as publish/subscribe systems or tuplespaces-based systems (see the "Related Work" sidebar)—support the initial interaction partner discovery, but don't support interaction partner maintenance over prolonged interaction sessions or easy protocol modularization.

We propose extending these middleware approaches with suitable abstractions to better support protocol-based interaction in mobile applications. To test our approach, we created ObjectPlaces, a middleware that uses roles as its main abstraction. A role encapsulates an application component's behavior during an interaction protocol. A role's behavior is a black box to ObjectPlaces; rather than focus on specifying a role's behavior, we emphasize the role abstraction. This lets ObjectPlaces manage

- the setup of interaction sessions among several roles played by application components on different nodes and
- active session maintenance in a mobile and dynamic application.

We applied ObjectPlaces in the domain of *automatic guided vehicle* control and evaluated our abstractions by applying them to the AGV application's coordination problems.

Problem illustration

We undertook a research project with an industry partner, Egemin, to explore a decentralized approach's feasibility for AGV control. AGVs are unmanned, battery-powered vehicles that transport loads through a warehouse or factory. They communicate over a wireless network to divide tasks; avoid collisions, deadlocks, and traffic jams; and achieve the warehouse's desired load throughput. Here, we focus on AGV collision avoidance.

AGV collision avoidance

To avoid collisions, AGVs execute a mutual exclusion protocol,¹ designating a critical section on the factory floor that only one AGV should cross at a time. Using the protocol, each AGV tries to lock an area before

driving over it. The AGV's hull projection determines this area (see figure 1). An AGV's hull is the physical area the vehicle occupies on the floor. A hull projection is the union of a set of hulls, projected along the AGV's intended path in small increments. To lock a hull projection, each AGV asks nearby AGVs whether it can drive over the hull projection safely, then locks the hull projection if all the nearby AGVs consent. Every requested hull projection contains a priority, and all such priorities are totally ordered.¹

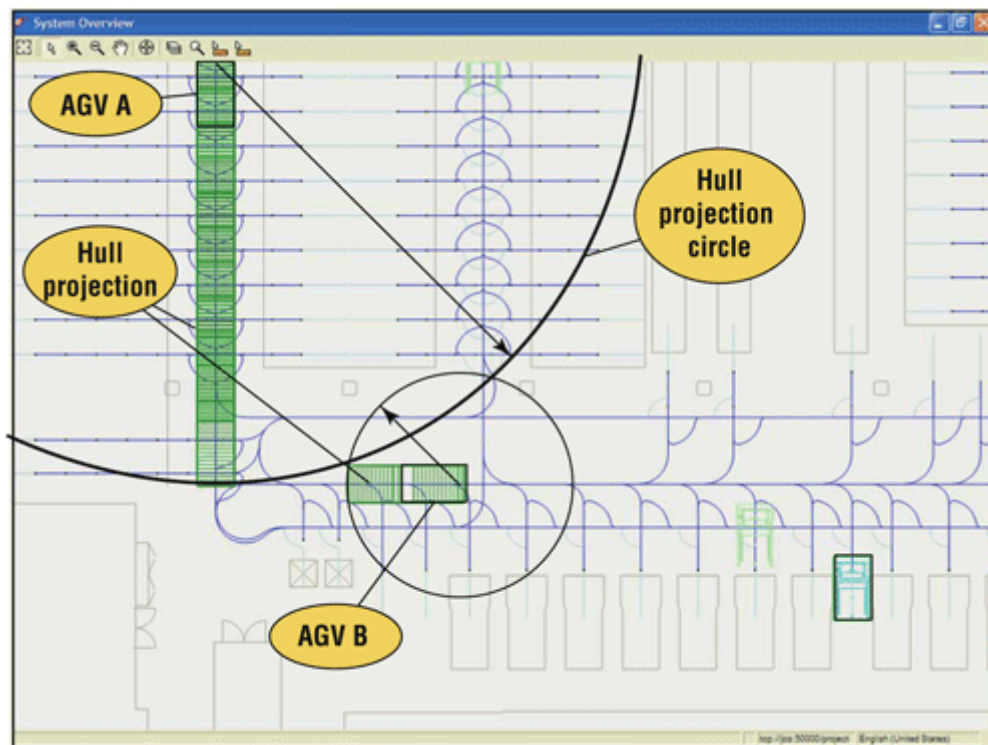


Figure 1. A top-down view of two automatic guided vehicles and their hull projections (green) on a factory floor. The paths over which AGVs can drive are fixed (blue). AGVs A and B are in a collision range because their hull projection circles overlap.

When a voting AGV receives a request, it sends an `allow` vote if

- the requested hull projection doesn't overlap with any of its requested or locked hull projections, or
- the requested hull projection overlaps with its requested hull projections, but the requesting AGV's hull projection has a higher priority.

The voting AGV defers its `allow` vote in all other cases; it re-evaluates deferred requests whenever it has driven over an area and thus unlocks a part of its locked hull projection. To save bandwidth, we want to avoid executing the protocol with all AGVs for every request. However, to guarantee safety, the subset of AGVs that a requester interacts with must include all AGVs with which it might collide.

Figure 1 shows how we determine this safe subset: AGVs in collision range are those with overlapping *hull projection circles*. We define the hull projection circle by a center point (the AGV's position) and a hull length (the distance between the AGV and the furthest point on its hull projection). So, overlapping circles indicate, to a first approximation, that two AGVs are within collision range. This approximation narrows the possible candidates for interaction significantly. To determine interaction partners, each AGV needs only the other AGVs' positions and hull lengths.

The subset of AGVs within a requesting AGV's collision range can change at any time. If a new AGV enters

collision range during a pending request, the requester should detect this and send the new AGV a request. If a voting AGV leaves collision range, the requesting AGV can disregard its vote.

Protocol-based coordination issues

Generally, collision avoidance illustrates two key issues with protocol-based coordination in mobile applications. First, the set of interacting participants is most easily defined by a constraint on node properties, as opposed to an enumeration of node identifiers (such as an IP address). In this case, for example, AGVs interact based on AGV position and hull length.

Second, given the dynamics of node properties that mobility entails, the changing number of interaction participants affects the ongoing interactions. For example, when executing a collision avoidance protocol, a requesting AGV must account for new AGVs entering the collision range.

ObjectPlaces

To support protocol-based interaction, ObjectPlaces offers two services. First, it sets up an interaction session between a set of roles on nodes defined by a constraint on the nodes' properties. ObjectPlaces then maintains the interaction session as the nodes' properties change. The roles are responsible for executing the protocol on behalf of the application components playing the roles.

Interaction session setup

An application component can start an interaction session by specifying the *initiator role* it wants to play, a *group definition* and a *participant role's name*. The initiator role starts and stops the session; there is only one initiator role per interaction session. In our collision avoidance interaction, an application component on the requesting AGV plays the requester role, which is an initiator role. Participant roles—the voter roles in our example—activate in response to initiator role activation. So, an application request activates initiator roles, while ObjectPlaces activates participant roles.

How it works. To determine which nodes participant roles must be activated on, the initiating component provides a group definition that defines a constraint on node properties:

$$f(\text{initiator-node-properties}, \text{participant-node-properties}) \rightarrow \{\text{true}, \text{false}\}$$

The group definition function takes as arguments the values of the initiator node's properties, as well as the values of the candidate participant nodes. It returns true only if the candidate participant is to be part of the interaction.

On all nodes that satisfy the group definition, ObjectPlaces activates the participant role with the given participant name, but only if there are application components on that node that are capable of playing the participant role. To let the middleware know this, an application component should register the names of the participant roles that it is capable of playing. Even if a node's properties satisfy the node constraint, a participant role with the given name is only activated if the role was registered by an application component on that node.

Once a node's participant role is activated, ObjectPlaces notifies the initiator. It then opens an asynchronous communication channel between the initiator and participants so they can execute the protocol. To let ObjectPlaces establish which nodes in the network satisfy a group definition, each node's application maintains the node's property values. (The application need maintain only the properties of the node itself.)

AGV example. For collision avoidance, an application component updates the AGV's position and hull length in ObjectPlaces to determine where to activate voters. When trying to lock a new hull projection, the application activates a requester role, asking ObjectPlaces to activate voter roles on AGVs within collision range using a group definition. ObjectPlaces then finds the nodes that satisfy the group definition and

activates their voter roles. Next, it notifies the requester, which executes the mutual exclusion protocol with the voters.

Expressivity. The group definition’s expressivity is a trade-off between usability and feasibility. While an application developer wants as expressive a group definition function as possible, ObjectPlaces must be able to find the group of nodes efficiently.

The group definition enables constraints based on each participant node’s individual properties and on relations between initiator node and participant node properties. The latter enables typical distance-based constraints. Constraints can’t be expressed on aggregate properties of more than two nodes, such as average or maximum CPU power. ObjectPlaces thus avoids the overhead associated with building a consistent global state of all node properties. Such expressive group definitions are as difficult to resolve as global predicate evaluation,^{2,3} which determines whether a global predicate is true in a distributed system.

In the AGV application, we found the group definition function to be expressive enough. In practice, the interaction protocol can handle a more fine-grained selection of interaction partners on the application level, to avoid high overhead for protocols that don’t need it.

Interaction session maintenance

Due to network dynamics, once an interaction session is established, nodes might appear that satisfy an executing interaction session’s group definition. For example, a new AGV might enter collision range after a requester role has sent out its requests but before all votes have come in.

ObjectPlaces therefore continuously monitors node properties and activates the participant roles on new nodes when necessary. As figure 2 shows, when ObjectPlaces activates a new participant role, it notifies the initiator role. The initiator role can then incorporate the new participant in the protocol. Similarly, if a participating node no longer satisfies the group definition, ObjectPlaces notifies the initiator role and cleans up the participant role. The process terminates when the initiator indicates that the interaction session is over.

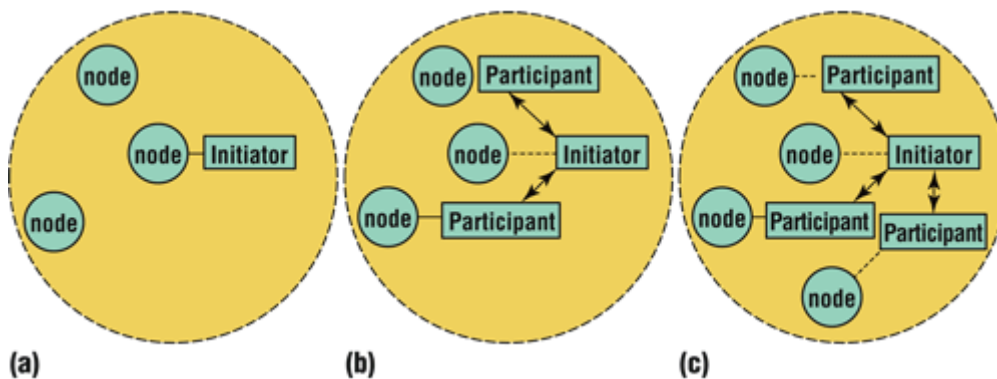


Figure 2. Initiating an interaction session. (a) An initiator role is activated. The circle denotes the group definition. (b) ObjectPlaces instantiates the participant roles, and the roles begin executing the protocol. (c) A new node enters the group definition zone. ObjectPlaces activates a participant role on the new node and notifies the initiator of this new participant, which can then take part in the protocol.

Group membership guarantees. ObjectPlaces offers two group membership guarantees:

- An initiator receives messages only from group participants, guaranteeing that initiators don’t receive messages from another interaction session or from removed participants.

- ObjectPlaces notifies the initiator of node arrival and removal with a best-effort guarantee.

Two factors determine group update granularity: the frequency of node property updates and the delay that the underlying communication medium imposes. The application controls the update frequency, taking into account that more updates cause more overhead (as we describe later). For example, if position is updated every second, ObjectPlaces updates groups defined by a constraint on position approximately every second as well (taking into account jitter on communication delay).

Groups that ObjectPlaces supports are different from groups that Group Communication Systems supports. Researchers have studied GCS in fixed distributed systems^{4,5} and, more recently, in mobile ad hoc networks.^{6,7} Their main goal is to provide a consistent (that is, identical) view on the group's members at each member and to make sure that all group members can communicate. However, an ObjectPlaces group is more fine-grained than a GCS group—the latter is long-lived and doesn't differentiate between different interaction sessions, while ObjectPlaces' groups are short-lived and specific to one particular interaction session. This lets ObjectPlaces encapsulate tedious session-management tasks, such as generating session IDs and routing incoming messages to the appropriate role.

A protocol session is always started by a component that contacts one or more other components. ObjectPlaces thus ensures that the initiator sees all participants, but participants only see the initiator. So, we can view a GCS group as a combination of multiple ObjectPlaces groups.

Furthermore, ObjectPlaces allows temporary deviations from consistency over several groups. Suppose node X has a participant in an interaction with a node Y initiator. If an application component then starts an initiator on X with the same group definition as Y's initiator, ObjectPlaces doesn't guarantee that the X initiator has node Y as a participant at the same time that node Y has node X as a participant.

Some protocols, such as the one used for task assignment in the AGV application,^{8,9} don't require consistency; others, such as the collision avoidance protocol, do. In the latter case, we can enforce consistency at the protocol level.

AGV application. In the AGV application, ObjectPlaces updates the position and hull length on each AGV every second. When a new AGV enters collision range during a collision avoidance interaction session, ObjectPlaces detects this and activates a new voter role on that AGV. Next, it notifies the requester and sends a request to the new voter, and the requester awaits that AGV's vote. Similarly, when an AGV moves out of collision range, ObjectPlaces notifies the requester, which then disregards that voter's vote. The first guarantee—that initiators receive only votes from participants in the group—ensures that no initiator receives allow votes from voters no longer in the group.

The second guarantee addresses safety. Given that the update interval t_{update} for position and hull length is one second, every initiator takes a minimal safe time, $t_{update} + \text{delay}$, into account before closing a session and locking a hull. In practice, we set this safe time to $2 \times t_{update}$ to ensure that ObjectPlaces has time to exchange the initiator's new position and hull length with other nodes and thus keep each node's view up to date. To enforce consistency, a voter can reply `allow` only if the requester's node is in the voter node's collision avoidance group.

To ensure that AGVs drive smoothly in the face of locking delays, AGVs request hull projections in advance, before they run out of locked hull projections.

Application programming interface

Figure 3 shows the main classes of ObjectPlaces' API. `RoleActivator` is the main access point, and `startInteraction` and `stopInteraction` set up and stop interaction sessions. To update node properties, the application can use `updateNodeProperty` and `registerNodePropertyObserver`. The application can ask to resolve a group definition without setting up an interaction session using `viewGroup`, which is

instrumental for enforcing consistency (as in the earlier AGV example). `Requester` and `Voter` are examples of application-specific roles.



Figure 3. UML diagram of ObjectPlaces' API. RoleActivator is the main access point to ObjectPlaces' services, offering methods to start and stop interaction sessions. Roles, such as Requester and Voter, should be implemented as subclasses of either Initiator or Participant.

For each participant role that an application component is capable of playing, the application component should register a `ParticipantFactory` with a participant name using `RoleActivator.setParticipantFactory`. `ObjectPlaces` can then use the factory to instantiate a new `Participant` of the appropriate type.

As figure 3 also shows, roles can communicate using send and receive methods. Initiators get to know participants through `inGroup`, and participants get to know their initiator through `inGroupOf`.

Application components and roles

In addition to interaction between roles on different nodes, `ObjectPlaces` must support interaction between roles on the same node. For example, an AGV's voter role can't vote `allow` on a request if the same AGV has an initiator role that has locked an overlapping hull. Similarly, components involved in the interaction might need to influence a session while it's in progress, and role instances might need to influence the

application components' behavior. For example, once the collision avoidance protocol is finished, the requester role locks the hull so the component driving the AGV sees that it can continue on.

To this end, application components share state with the roles they play in one or more shared tuplespaces.¹⁰ The application components use the tuplespace to share state relevant for their roles' interactions, and roles use the tuplespace to give the application components feedback regarding the interaction's outcome. Typically, related roles and components interact in one or more tuplespaces separated from other components and roles that deal with different interactions.

Two properties of a tuplespace justify its use:

- It hides communicating partners' identities from each other, facilitating the dynamic addition and removal of roles without "bothering" the decoupled application components.
- It provides 1–n communication. Typically, one application component simultaneously participates in several interactions, so several role instances must observe the same state.

Architecture and implementation

Figure 4 shows an overview of the ObjectPlaces architecture. The figure shows two mirror images: the top image reflects the entities relevant for a node with an initiator, while the bottom image reflects the entities relevant for a node with a participant. Because each node can be initiator and participant simultaneously, we must "fold" the two images together to get a complete overview of a node's processes.

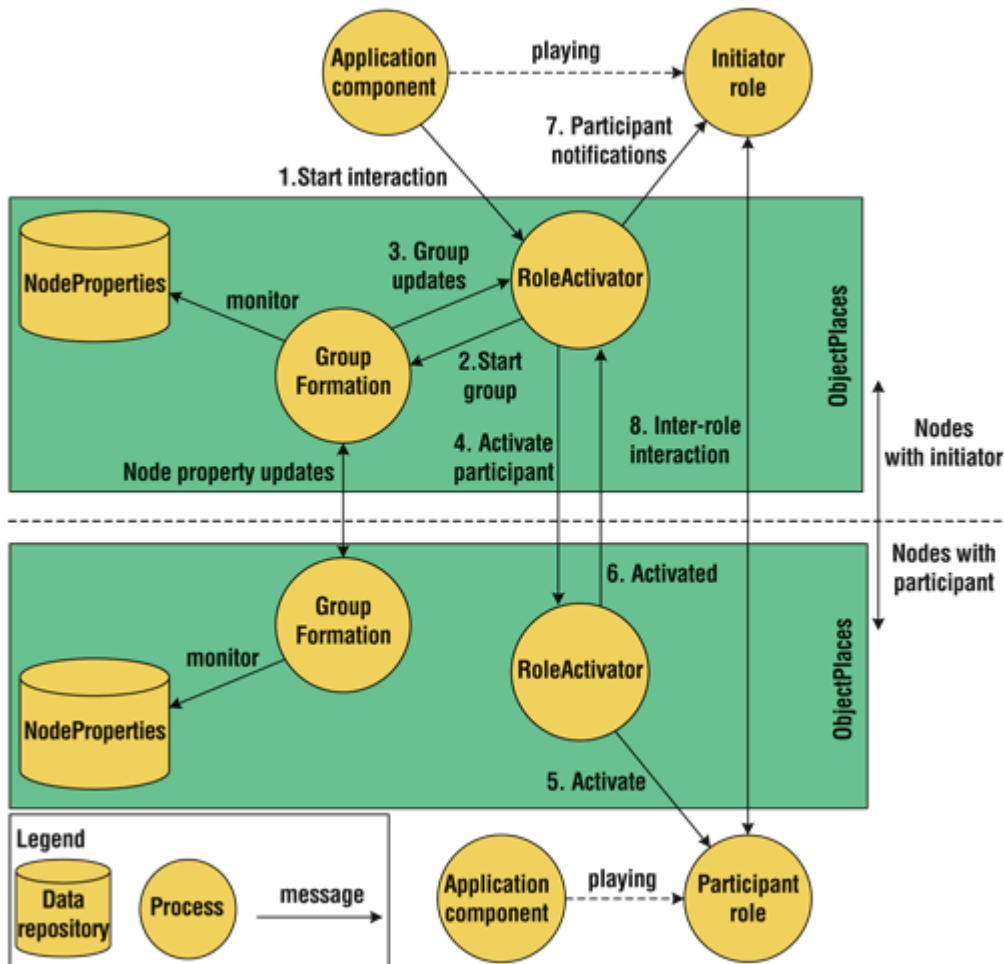


Figure 4. The ObjectPlaces middleware architecture.

The numbered events are successive messages exchanged between `Role Activator` and `Group Formation`, the main processes involved in setting up an interaction session. `Role Activator` activates `Group Formation` to find and monitor the group of nodes satisfying the group definition. It then activates the participant roles on this group of nodes; if activated, `Role Activator` notifies the initiator role, which can then execute the protocol with the participants.

Implementing group formation poses the greatest challenge. To accomplish it, `ObjectPlaces` must resolve the group definition into a specific group of nodes, maintain the group as properties change, and guarantee the two group membership properties we discussed earlier.

We implemented an `ObjectPlaces` prototype for the AGV application using .NET (demonstration movies and more information are available at <http://emc2.egemin.com>). In the AGV application, the network is fully connected, and all messages arrive within a known time bound. These assumptions are justified because AGV systems exist in a controlled environment. For the prototype, we didn't take node failure into account. We discuss options that scale more favorably to larger networks, particularly using content-based routing, elsewhere.¹¹

The prototype disseminates node properties by multicasting updates to nodes that can start interactions on the basis of those properties. On each node, `ObjectPlaces` can then maintain, per node ID, the current value of each node's properties. Given this information, an initiator node resolves group definitions and contacts the participant nodes to activate participant roles. The initiator node's `ObjectPlaces` middleware is responsible for the group: if node properties indicate that a participant node should leave the group, the middleware on the initiator notifies the participant node that it is no longer part of the group.

Evaluation

We tested `ObjectPlaces`' proposed abstractions by applying them to the AGV application's coordination problems.

Interaction management

`ObjectPlaces`' role abstractions let the middleware encapsulate management of both interaction partners and interaction sessions. This simplifies protocol implementation: the application doesn't have to deal with tedious work, such as session IDs to determine which role should handle an incoming message, or managing constantly changing interaction partners.

Separation of concerns

`ObjectPlaces` lets developers separate two concerns: interaction partner selection and interaction protocol execution. In our AGV case, this separation proved useful during system evolution. In the first implementation of collision avoidance, we didn't account for deadlocks (as when two AGVs stand head-on on a bidirectional path). Because AGV hull length was constant, our hull projection was fixed. We could thus write the collision avoidance group definition function solely in terms of AGV positions.

We later added deadlock avoidance by extending the length of requested hull projections (for example, the collision avoidance protocol would request and lock an entire bidirectional path). This didn't require changing the collision avoidance protocol's code; we simply added the hull length to the node properties and changed the group definition function.

Applicability

`ObjectPlaces`' concepts are useful in a wide range of mobile applications. Using `ObjectPlaces`, we implemented protocols ranging from mutual exclusion¹ to deadlock detection¹² to task assignment⁹ in the AGV application. Our success indicates that developers can use the `ObjectPlaces` model to support a wide range of protocols.

We have also implemented `ObjectPlaces`' group formation in other types of mobile environments^{13,14}—

such as mobile ad hoc networks—albeit with weaker guarantees than those we described earlier.

General overhead

ObjectPlaces' overhead depends on several application-specific factors. ObjectPlaces needs at least one multicast message for each node property update. Depending on network reliability, ObjectPlaces might need slightly more messages to achieve reliable multicast. The primary influence on bandwidth overhead is the frequency of the node property updates. Furthermore, the node property value's size determines the update message's size.

For example, an AGV system with 20 vehicles—a large system for Egemin AGV applications—updates positions and hull lengths every second. These updates have a message size of 16 bytes and include receiver ID, x and y position, and hull length. This uses slightly more than 320 bits per second, depending on transmission errors, presenting no problems when compared to the bandwidth available in an IEEE 802.11 network (maximally 5.9 Mbits per second over Transmission Control Protocol (TCP) and 7.1 Mbps over User Datagram Protocol (UDP)). In addition, the AGV application most frequently updates position, while other node properties are updated only as they change. AGV status, for example, changes only two to three times per minute.

Decreasing bandwidth

ObjectPlaces supports a system's division into subgroups of interacting components. In addition to supporting interaction session setup, such a division can decrease the bandwidth usage, earning back the overhead needed for group formation. Because it has extra information about node properties, a particular interaction typically requires fewer interaction partners. For example, extra information about AGV position lets the collision avoidance protocol narrow down the number of voters.

Whether or not dividing a system into subgroups of interacting components decreases bandwidth depends on

- the average number of nodes in interacting groups relative to the overall number of nodes in the system,
- the relative sizes of node property messages and protocol messages,
- how frequently interaction sessions start, and
- how frequently the application updates the node properties required for the interaction.

The collision avoidance protocol uses lower bandwidth with ObjectPlaces—when it executes each collision avoidance interaction session with a subset of all AGVs—than without it. Using ObjectPlaces narrows the number of AGVs in subgroups to two or three per session. Also, the protocol messages are large compared to the node property messages. Request messages contain a hull projection—a convex polygon (possibly containing curves), whose representation is two to 15 times larger than the position message (containing only a coordinate and a hull length). Finally, other protocols can also use the exchanged node properties to reduce the number of interaction partners in their interaction sessions. In the AGV application, for example, we also use position to determine which AGVs to negotiate with for task assignment.

Although ObjectPlaces can detect changes in interaction partners, it offers little support to deal with such changes on the protocol level. We plan to add stronger guarantees—such as transactional semantics—to make it easier for application developers to build correct protocols with a changing number of participants.¹⁵ We also plan to add fault-handling mechanisms.

Acknowledgments

This research is supported by the Flemish Institute for Advancement of Research in Industry. Many thanks

also to Egemin, in particular to Tom Lefever and Jan Wielemans, Jan Vercammen, Jan Peirsman, Wim Van Betsbrugge, Rudi Vanhoutte, and Walter De Feyter. We also thank the anonymous reviewers for their constructive comments.

References

1. G. Ricart and A.K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," *Comm. ACM*, vol. 24, no. 1, 1981, pp. 9–17.
2. R. Cooper and K. Marzullo, "Consistent Detection of Global Predicates," *Proc. Workshop on Parallel and Distributed Debugging*, ACM Press, 1991, pp. 167–174.
3. V.K. Garg and Brian Waldecker, "Detection of Strong Unstable Predicates in Distributed Programs," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 12, 1996, pp. 1323–1333.
4. G.V. Chockler, I. Keidar, and R. Vitenberg, "Group Communication Specifications: A Comprehensive Study," *ACM Computing Surveys*, vol. 33, no. 4, 2001, pp. 427–469.
5. G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design*, 3rd ed., Addison-Wesley Longman, 2001.
6. G.C. Roman, Q. Huang, and A. Hazemi, "Consistent Group Membership in ad hoc Networks," *Proc. 23rd Int'l Conf. Software Eng.*, IEEE CS Press, 2001, pp. 381–388.
7. J. Liu et al., "Group Management for Mobile ad hoc Networks: Design, Implementation and Experiment," *Proc. 6th Int'l Conf. Mobile Data Management*, ACM Press, 2005, pp. 192–199.
8. D. Weyns, N. Boucké, and T. Holvoet, "Gradient Field-Based Task Assignment in an AGV Transportation System," P. Stone and G Weiss, eds., *Proc. 5th Int'l Joint Conf. Autonomous Agents and Multiagent Systems*, ACM Press, 2006, pp. 842–849.
9. R.G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Trans. Computers*, vol. C-29, no. 12, 1980, pp. 1104–1113.
10. N. Carriero, D. Gelernter, and J. Leichter, "Distributed Data Structures in Linda," *Proc. 13th ACM Symp. Principles of Programming Languages*, ACM Press, 1986, pp. 236–242.
11. K. Schelfhout, D. Weyns, and T. Holvoet, "Middleware for Protocol-Based Coordination in Dynamic Networks," *Proc. 3rd Int'l Workshop on Middleware for Pervasive and ad-hoc Computing (MPAC 05)*, ACM Press, 2005, pp. 1–8.
12. K.M. Chandy, J. Misra, and L.M. Haas, "Distributed Deadlock Detection," *ACM Trans. Computer Systems*, vol. 1, no. 2, 1983, pp. 143–156.
13. G.C. Roman, C. Julien, and Q. Huang, "Network Abstractions for Context-Aware Mobile Computing," *Proc. 24th Int'l Conf. Software Eng.*, ACM Press, 2002, pp. 363–373.
14. K. Schelfhout, T. Holvoet, and Y. Berbers, "Views: Customizable Abstractions for Context-Aware Applications in Manets," *Proc. 4th Int'l Workshop on Software Eng. for Large-Scale Multi-Agent Systems*, ACM Press, 2005, pp. 1–8.
15. G. Cugola and H.A. Jacobsen, "Using Publish/Subscribe Middleware for Mobile Systems," *ACM Sigmobile Mobile Computing and Comm. Rev.*, vol. 6, no. 4, 2002, pp. 25–33.



Kurt Schelfthout is a PhD student in the Department of Computer Science, Katholieke Universiteit Leuven. His research interests include coordination, middleware for mobile networks, and multiagent systems. He received his MSc in engineering from the Katholieke Universiteit Leuven. Contact him at DistriNet, Dept. of Computer Science, K.U. Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium; kurt.schelfthout@cs.kuleuven.be.






Danny Weyns is a PhD student in the Department of Computer Science, Katholieke Universiteit Leuven. His research interests include software engineering of multiagent systems and software architecture. He received his MSc in computer science from the Katholieke Universiteit Leuven. Contact him at DistriNet, Dept. of Computer Science, K.U. Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium; danny.weyns@cs.kuleuven.be.



Tom Holvoet is a professor in the Department of Computer Science, Katholieke Universiteit Leuven. His research interests include software engineering of decentralized and multi-agent systems, software architecture, autonomic computing, and aspect-oriented software development. He received his PhD in computer science from the Katholieke Universiteit Leuven. Contact him at DistriNet, Dept. of Computer Science, K.U. Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium; tom.holvoet@cs.kuleuven.be.

Related Links

-  [DS Online's Middleware Community](#)
-  ["A Coordination Middleware for Wireless Sensor Networks," Proc. 2005 Systems Communications](#)
-  ["Engineering Mobile Agent Applications via Context-Dependent Coordination," IEEE Trans. Software Eng.](#)

Cite this article:

Kurt Schelfthout, Danny Weyns, and Tom Holvoet, "Middleware for Protocol-Based Coordination in Mobile Applications," *IEEE Distributed Systems Online*, vol. 7, no. 8, 2006, art. no. 0608-o8001.

Sidebar: Related Work

Our use of roles in ObjectPlaces is inspired by ongoing work in object-oriented software engineering, where developers use roles to model object collaborations in framework design, implementation support, patterns, and languages. Multiagent systems developers also use roles in protocol-based interaction, letting them describe interagent interaction separately from agent behavior. ObjectPlaces focuses more on runtime support for activation and deactivation of roles in mobile networks. It therefore relates to work in three primary areas: mobile network middleware, location services, and role assignment in wireless sensor networks.

Mobile network middleware

Most middleware systems for mobile networks use a publish/subscribe paradigm, in which publishers send notifications of state changes to a list of subscribers. Subscribers aren't known in advance; they use subscriptions to inform the middleware of their interest in certain notifications. Typically, these subscriptions are defined as a predicate on the notifications' contents.

Publish/subscribe model extensions typically deal with frequent disconnections by providing a dispatching server that buffers events and republishes lost events on reconnection. Examples include JEDI (Java Event-Based Distributed Infrastructure),¹ Elvin,² and work by Mariano Cilia and his colleagues.³ Other approaches^{4,5} introduce location-dependent subscriptions, refining a subscription to accept events related to the mobile publisher or the subscriber's current location.

REDS (reconfigurable dispatching system) is a publish/subscribe middleware that offers possible one-to-one subscriber-publisher communication, which is instrumental for supporting protocols.

Other types of middleware for mobile applications are based on a tuplespaces approach. In EgoSpaces,⁶ which builds on the work of LIME (Linda In a Mobile Environment),⁷ application components specify *views* to manipulate a set of tuplespaces on connected nodes. A view declaratively describes a set of tuplespaces on remote nodes by specifying a constraint on the node properties. An application component can then manipulate the view's tuplespaces as if they were one shared tuplespace.

In TOTA (Tuples on the Air),⁸ application components inject a tuple in a local tuplespace, together with propagation rules. The TOTA middleware transparently propagates the tuples to neighboring nodes and maintains the distributed tuple as the network topology changes.

In contrast to these approaches, ObjectPlaces explicitly supports protocol-based interaction for mobile networks. Although applications can use the above methods to discover interaction partners, the middleware methods don't offer first-class abstractions to represent an ensuing interaction protocol. So, at the protocol level, the application developer loses the middleware's support for dealing with dynamics.

Location service

A location service aims to maintain tangible objects' geographical locations to support location awareness. Developers typically use a location service in ubiquitous computing applications.⁹⁻¹¹ Usually, the location service gathers object locations at a fixed server that supports location-based lookup queries. This lets the application determine whether objects are in symbolic locations (such as "in the room"), as well as in locations based on a coordinate system.

ObjectPlaces is more flexible in that it doesn't require a fixed server. It can also select interaction partners on the basis of any node property, not just location.

Role assignment in wireless sensor networks

Christian Frank and Kay Römer describe a middleware that automatically assigns roles to nodes in a wireless sensor network.¹² Their approach is based on evaluating a constraint on node properties to determine whether a node should assume a particular role. TinyCubus¹³ uses this approach to perform efficient code deployment. The approach assumes that node properties change infrequently, whereas ObjectPlaces is specifically targeted toward applications with frequently changing node properties.

Furthermore, Frank and Römer represent roles simply as names that trigger the node's application to assume some functionality. In ObjectPlaces, roles encapsulate component behavior in a protocol so the middleware can manage activation and deactivation directly.

References

1. G. Cugola and H.A. Jacobsen, "Using Publish/Subscribe Middleware for Mobile Systems," *ACM Sigmobility Mobile Computing and Comm. Rev.*, vol. 6, no. 4, 2002, pp. 25–33.
2. P. Sutton, R. Arkins, and B. Segall, "Supporting Disconnectedness—Transparent Information Delivery for Mobile and Invisible Computing," *Proc. 1st Int'l Symp. Cluster Computing and the Grid (CCGRID 01)*, IEEE CS Press, 2001, pp. 277–287.
3. M. Cilia et al., "Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware," *Proc. 2nd Int'l Workshop on Distributed Event-Based Systems (DEBS 03)*, ACM Press, 2003, pp. 1–8.
4. L. Fiege et al., "Supporting Mobility in Content-Based Publish/Subscribe Middleware," *Proc. Int'l Middleware Conf.*, LNCS 2672, Springer, 2003, pp. 103–122.
5. R. Meier and V. Cahill, "Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications," *Proc. 4th IFIP Int'l Conf. Distributed Applications and Interoperable Systems (DAIS 03)*, LNCS 2893, Springer, 2003, pp. 285–296.
6. C. Julien and G.C. Roman, "Supporting Context-Aware Interaction in Dynamic Multi-Agent Systems," *Proc. 1st Int'l Workshop in Environments for Multi-Agent Systems*, LNCS 3374, Springer, 2004, pp. 168–189.
7. A. Murphy, G.P. Picco, and G.C. Roman, "LIME: A Middleware for Physical and Logical Mobility," *Proc. 21st Int'l Conf. Distributed Computing Systems (ICDCS 21)*, IEEE CS Press, 2001, pp. 524–533.
8. M. Mamei and F. Zambonelli, "Self-Maintained Distributed Tuples for Field-Based Coordination in Dynamic Networks," *Proc. 2004 ACM Symp. Applied Computing*, ACM Press, 2004, pp. 479–486.
9. R. Want et al., "The Active Badge Location System," *ACM Trans. Information Systems*, vol. 10, no. 1, 1992, pp. 91–102.
10. A. Ward, A. Jones, and A. Hopper, "A New Location Technique for the Active Office," *IEEE Personal Comm.*, vol. 4, no. 5, 1997, pp. 42–47.
11. P. Bahl and V.N. Padmanabhan, "Radar: An In-Building RF-Based User Location and Tracking System," *Proc. Int'l Conf. Computer Comm. (Infocom 00)*, IEEE CS Press, 2000, pp. 775–784.
12. C. Frank and Kay Römer, "Algorithms for Generic Role Assignment in Wireless Sensor Networks," *Proc. 3rd Int'l Conf. Embedded Networked Sensor Systems*, ACM Press, 2005, pp. 230–242.
13. W. Heinzelman et al., "Middleware to Support Sensor Network Applications," *IEEE Network*, vol. 18, no. 1, 2004, pp. 6–14.