A Mobile Agent-Based Middleware for Opportunistic Resource Allocation and Communications

Marco Carvalho[1], Michal Pechoucek[2], and Niranjan Suri[1]

[1] Florida Institute for Human and Machine Cognition,
40 South Alcaniz St. Pensacola, FL, USA
{mcarvalho, nsuri}@ihmc.us
[2] Gerstner Laboratory, Czech Technical University,
166 27 Prague 6, Czech Republic
pechouc@labe.felk.cvut.cz

Abstract. Dependable communication capabilities are amongst the most important technical requirements for mission success in military combat operations. This paper introduces a mobile agent-based middleware that supports both point-to-point messaging and hierarchical data-streaming. Two infrastructure technologies (Mockets and FlexFeed) are introduced as service providers for messaging and publish-subscriber models for data streaming. Opportunistic resource allocation and monitoring are handled by distributed coordination algorithms, implemented here through two complementary technologies: Stand-In Agents and Acquaintance models.

1 Introduction

Communications in military battlefield operations are currently one of the most critical technical capabilities for mission success. From a network perspective, tactical military operations are often characterized by highly dynamic ad hoc wireless environments and include heterogeneous nodes under resource and security constraints. Furthermore, the communications infrastructure is expected to change its behavior and optimization criteria to adapt to changes in goals and priorities.

In recent years, a number of research efforts have focused their attention on this problem, looking for better routing or transport algorithms that would correct the deficiencies observed in the use of traditional wired network protocols.

Despite the invaluable progress these efforts have brought to the field, the reality is that in practice today networks are still deployed and configured in a customized fashion, to address specific needs or missions, with very specialized capabilities.

The problem is often associated with the notion that the communications infrastructure is expected to be completely isolated from the semantics of the data. Traditionally, this has been a very fundamental concept that provided portability and standardization of different network and communication protocols.

It seems clear, however, that mission-critical applications require not only the generality and flexibility inherent from context-free protocols, but also the efficiency provided by data-aware protocols, better able to create and maintain specialized data distribution trees in the network.

S.G. Thompson and R. Ghanea-Hercock (Eds.): DAMAS 2005, LNAI 3890, pp. 121 – 134, 2006.

In this paper, we introduce a novel agent-based communications framework designed to help address the issue in these types of environments. The goal is to provide a middleware[1] that will overlay the physical network and transparently provide both services, with minimal changes in current software applications and systems.

In our framework, intelligent software agents are used to enable, on demand, data-aware capabilities in the network. The agents are mobile, so code and computation can be moved as necessary to opportunistically create capabilities and to react to changes in topology, resource availability, and policies.

The framework proposed in this paper leverages from a set of core technologies that have been designed and developed by our research teams for similar types of scenarios. Extensively tested in numerous proof-of-concept applications and demonstrations, these technologies have matured to a point where they complement each other to enable the framework.

2 Capabilities and Components

The communications middleware proposed in this work provides three core capabilities: support to point-to-point messaging; publish-subscribe oriented data streaming, and on-demand, opportunistic resource allocation. The idea is to combine the context-specific and context-free communication requirements supported by two APIs (Mockets and FlexFeed) into a common communications middleware, so they can leverage from a common resource management infrastructure.

The Mockets API provides a TCP-like interface to the applications. The API can be used to exchange general purpose message or streaming data. The FlexFeed API provides a publish-subscribe interface to applications. FlexFeed is responsible for handling data-aware data-streams, leveraging from the opportunistic allocation of resources provided by the coordination components. Both FlexFeed and Mockets are regulated through distributed (or centralized) algorithms, implemented as coordination components that identify, configure, and allocate resources in the network.

The coordination components constitute the “intelligent” part of the framework. They are responsible for sharing state between neighbor nodes and for negotiating the temporary allocation of resources necessary to support FlexFeed and Mockets.

Furthermore, the coordination components are also responsible for determining the appropriate course of action or recovery measures to be taken when changes in network topology or resource availability interfere with the communication tasks.

As a distributed component, the issues involved in resource coordination tasks are critical and complex. In the next section the basic technologies involved in the coordination components and the communication APIs will be discussed in greater detail.

3 Design and Implementation

The framework is implemented in Java. Each of the components integrated in the framework have their own access API which is used by external applications and between components to exchange services and state.

[1] The terms “Framework” and “Middleware” are used interchangeably in the context of this publication.

In order to support the capabilities required by the framework, it is important to have access and control of the underlying ad hoc routing protocol. Currently, an application-level implementation of a customized version of AODV [1] is integrated with the framework.

The coordination components use routing information and costs from multiple paths (from the underlying routing component) to identify possible data distribution trees that will lead to approximate global optimization of joint or disparate streams.

Figure 1 shows a diagram of the main components of the framework. In the figure, the physical node block represents the actual hardware platform, including the radios and the data-link layer.

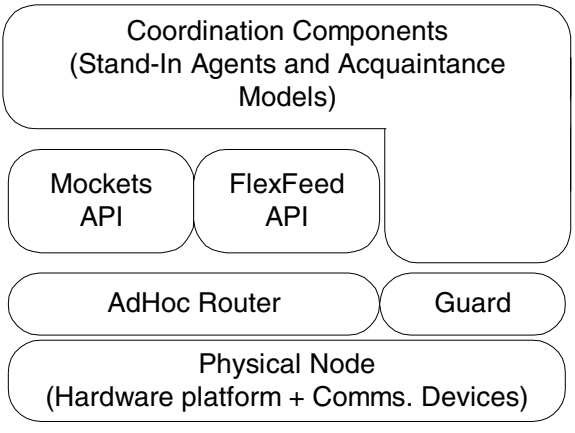

Fig. 1. Framework Components

The Guard is a software component with direct hardware access to provide local enforcement of policies and resource utilization. The middleware is integrated with a policy framework (KAoS) [2, 3] that provides the mechanisms for policy definition, verification, distribution, and enforcement, done at each node by the Guards.

The Guard components are responsible for maintaining and enforcing constraints and obligations in the local host. In the proposed framework, the Mockets and FlexFeed components work as local enforcers on data-streaming and message related policies. When necessary, these components will query the local Guards for policy information so the coordination components can take the constraints into account when building a specific solution. Resource utilization by all other processes in the local host (like applications and agents) is directly enforced, at a lower level, by the Guards. Together, Mockets, FlexFeed and the Guard components ensure that all host level operations are within policy constraints, even if the policy repository is temporarily inaccessible to the node.

The coordination components are the entities responsible for negotiating and determining resource distribution both at the local and global levels. These components have direct access to the guard interface (for policy querying) and to underlying routing and transport protocols for parameter estimation.

During resource negotiation with peers, the coordination components also check policy constraints with the KAoS framework (through Guards interface). Both at the level of resource utilization and information release, constrains are enforced by the selective deployment of agents. To a certain extent coordination components are responsible for policy enforcement at the framework level in the same way that guards, FlexFee and Mockets are responsible for local policy enforcement.

3.1 FlexFeed

FlexFeed [4], in the context of this framework, provides the mechanisms to configure and task network resources in order to deploy the allocation schemes determined by the coordination components. FlexFeed was initially designed and tested as a middleware for data distribution in military operations (as part of a previous research effort sponsored by the Army Research Laboratory).

The framework relies on mobile software agents to transparently enable capabilities in network nodes for data processing and distribution between multiple (possibly disparate) applications.

The FlexFeed framework provides a Java interface for stream-oriented communications between applications. Sensors and clients use the FlexFeed API to provide and access information feeds. The transport mechanism, the message distribution, and filtering are handled at the framework level, hidden from data producers and consumers.

Consider the case illustrated in figure 2. In this example, the Unmanned Aerial Vehicle (UAV) is the source of the data, as it carries a high resolution video camera aimed at enemy positions.

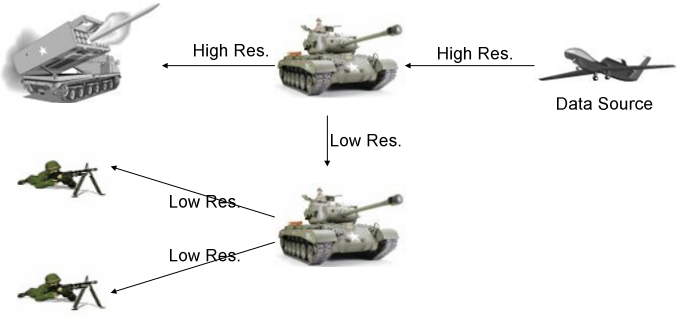

Fig. 2. Providing customized data-aware streams

In this example, a missile launcher requests a high resolution data-stream from the UAV to identify potential targets while at the same time, soldiers in the battlefield guarding the perimeter request visual data to monitor the movement of enemy troops.

The goal of the FlexFeed framework in this case is to identify the best data-distribution tree that would minimize overall transmission and data processing costs to support client requests.

Specialized agents, capable of transforming video data in this case, are injected in the framework at run-time (either by clients or by the source of the data). These agents can be positioned at any node in the network to establish the data distribution

tree. FlexFeed interacts with the coordination components to identify the best resources available for the task.

With that information, FlexFeed then positions the agents accordingly and initiates the data-streams. From a minimum cost perspective, the best solution in the example shown in figure 2 is to send a high-resolution video stream to a nearby tank (with spare processing capabilities) that would then be responsible to clone and transform the stream, relaying the high resolution stream to the missile launcher and the low resolution equivalent to another tank near the soldiers for final distribution.

Once established, coordination components will continuously monitor the state of the network to react to changes in topology, resource availability, or policies. If necessary, the data-processing elements deployed in the first tank, for instance, can be moved to a sub-optimal position to re-allocate or release resources.

3.2 Mockets

Mobile Sockets (or Mockets) are an application-level transport layer specially designed to transparently provide resource redirection and cross-layer interaction in mobile ad hoc network environments. Mockets exposes a TCP-like interface implemented over UDP messaging and provides an extended API to support the exchange of state information and control messages between applications in the upper layers and underlying network protocols. Mockets were extensively used in a recent field exercise coordinated by ARL (Quantum Leap 2) to abstract complex underlying communication behaviors from applications that were originally designed to operate reliably over TCP.

The mockets communication framework provides a comprehensive communications API that addresses the limitation of TCP while at the same time offering new primitives for applications. To support existing TCP-style semantics, mockets provides a straightforward stream-oriented interface.

Applications written to use TCP can be easily modified to operate on top of the Mockets API. In addition, applications can take advantage of new capabilities such as keep-alive and connection statistics (bytes and packets sent as well as retransmission counts). The stream mockets implementation is designed to work on top of wireless and ad-hoc networks and does not exhibit the problems observed with TCP in such environments [8][9].

Mockets also provides a message-oriented interface. The message mockets support four different types of service: unreliable/unsequenced, reliable/unsequenced, unreliable/sequenced, and reliable/sequenced. The semantics of the unreliable/unsequenced service are similar to UDP and those of the reliable/sequenced service are similar to R-UDP. While there is no existing equivalent to unreliable/sequenced, this service is useful for situations such as video or audio streaming where packet dropout is preferred over out of sequence packets. Finally, the reliable/unsequenced service rounds out the set of capabilities.

When using the message-oriented paradigm, the mockets API provides additional features such as in-queue message replacement. This capability is particularly important for wireless networks and mobile ad-hoc networks where connectivity may be intermittent. It allows an application to replace a previous message with a new one,

provided the previous message is still awaiting transmission in the mocket's outgoing queue. Messages may be tagged in order to classify them into different traffic categories. For example, if an application is sending periodic GPS position updates along with other traffic, the application can tag all the GPS position updates with a unique tag. When a new update is available, the application can simply ask mockets to replace any previous GPS update messages with the new one, which invalidates all the previous messages. This prevents large outgoing queues from accumulating when a node temporarily loses connectivity.

3.3 Coordination Components

The coordination components in the framework are the entities responsible for gathering state information to determine a cost-effective data distribution graph that will satisfy (within policies and resource availability constraints) the requests placed by each client.

In the types of environments considered in this work, global optimum allocation of resources is a very complex task, even for centralized algorithms with complete global information.

The complexity of the problem lies essentially in the fact that for each in-stream data processing configuration, network resources are affected, which leads to changes in the costs originally considered for the initial configuration as well as for all other active streams. The problem can be solved iteratively, with the risk of finding local optimal solutions and often 'hot-spots' in the data distribution paths.

There are, however, a number of heuristics and approximate cost functions that can be used to obtain a reasonable solution within the time scales required to make the framework practical. An example of a centralized approach to the problem is shown in [5], where a dijkstra-inspired algorithm is used to find an approximate solution.

Resource allocation in the framework is done through negotiation protocols, which requires p2p interactions between each host (and representative agent) involved in the transaction. This approach, however, is impractical for large scale systems due to communications overhead (for inter-agent negotiation) and transient partitions of the network caused by local link failures and changes in topology.

Furthermore, some of the candidate resources for data processing or communications relay might not be in communication range during the coordination process. There are many practical situations where this might occur, for instance if potential relays are mobile with a predictable path (like surveillance UAVs or ground vehicles), or in cases where nodes in strict passive mode (for instance, for security reasons) can be activated by framework if selected, or even cases where the framework is allowed to request nodes to physically move within communications range of other nodes to support temporary data-streams.

In cases like these, it is important for the negotiation procedures between coordination components to take into consideration temporary communications inaccessibility with potential candidates for the client requests.

The agent negotiation approach adopted in our framework is based on the notions of remote presence and remote awareness.

4 Remote Presence and Remote Awareness

In this section we discuss the techniques used in the framework to support the distributed negotiation process carried out by the coordination components.

The most critical cases include situations where coordination must account for environments that are temporarily unavailable (either by physical constraints, policies or choice), but the techniques can be easily extended to reduce communications overhead associated with the negotiation process.

The goal of the suggested techniques is to provide limited mutual awareness without the need for direct and continuous communication. The main applicability potentials are in the problems with a planning or a longer term coordination component. In such problem domains, the entities need to plan sharing resources and coordinate responsibilities, so that the joint goal can be effectively achieved.

Example 1. Let us have a computationally intensive task that needs to be decomposed and allocated to specific number of processors in a distributed, partially inaccessible environment. Without any knowledge of the services, current and planned load of the inaccessible processors, as much as the expected time when the connectivity can be established, the task would be delegated among the processors that are currently accessible. Awareness of the inaccessible processors allows planning the resource allocation on top of the inaccessible processors as well.

Example 2. Let us have a community of mine-sweeping underwater robots that are searching through the specific areas. The communication reach is restricted. Once a suspicious object is detected the robots need to position themselves into a communication feed so that the picture of the object can be transmitted successfully to the human operator. With the limited number of available robots, forming the communication feed is only possible if the agents maintain their approximate knowledge about each other location.

We distinguish among two types of techniques that can be used in the situations of communication inaccessibility:

- **remote awareness**, a set of techniques for representing the agents knowledge about the other inaccessible agents.
- **remote presence**, the concept of migrating the computational representative before the inaccessibility situation happens.

4.1 Acquaintance Models

The acquaintance models and computational models of the mutual awareness of the network elements (hosts, devices, routers, etc.) have been thoroughly studied and investigated in the area of agent technologies and agent-based computing. Each computational agent in the interaction environment is supposed to form (either collaboratively or individually) models of the other computational agents in terms of their location, load, plans and resource commitments, interaction accessibility, etc. This knowledge can be used for distributed coordination, planning for cooperation and resource commitments of the individual computational agents (of any kind).

The concept of acquaintance models have been exploited in modeling coordination in the semi-trusted environment of the OOTW domain within an AFRL funded project and for underwater mine-sweeping simulation project funded by ONR.

The acquaintance model does not need to be precise and up-to-date. Agents may use different methods and techniques for maintenance and exploitation of the acquaintance model. There has been various acquaintance models studied and developed in the multi-agent community, e.g. tri-base acquaintance model [6] and twin-base acquaintance model [7]. In principle, each acquaintance model is split into two parts: self-knowledge containing information about an agent itself and social-knowledge containing knowledge about other members of the multi-agent system. While the former part of the model is maintained by the social knowledge provider (an owner), the latter is maintained by the social knowledge requestor (a client). Social knowledge can be used for making operation of the multi-agent system more efficient. The acquaintance model is an important source of information that would have to be repeatedly communicated otherwise. Social knowledge and acquaintance models can also be used in the situations of agents' short term inaccessibility. However, the acquaintance models provides rather 'shallow' knowledge, that does not represent complicated dynamics of the agent's decision making, future course of intentions, resource allocation, or negotiation preferences. This type of information is needed for inter-agent coordination in situations with longer-term inaccessibility.

4.2 Stand-in Agents

An alternative option is to integrate the agent self-knowledge into a mobile computational entity that is constructed and maintained by the social knowledge provider. We will refer to this computational entity as a stand-in agent. The stand-in agent resides either on the same host where the social knowledge requestor operates or in the permanently accessible location. While using stand-in, the social knowledge requestor does not create an acquaintance model of its own. Instead of communicating with the provider or middle agent, it interacts with the stand-in agent. Therefore, the client agent is relieved from the relatively complex task of building and keeping up-to-date detailed acquaintance model and both provider and requestor may benefit from the full-fledged remote presence. Factoring the acquaintance model out of the each requestor agent's internal memory allows it to be shared between all locally accessible agents, further minimizing the traffic and computational resources necessary for model maintenance. The stand-in agents operate in three phases:

- **stand-in swarming**, when stand-ins propagate through the system to reach the locations that may become inaccessible in the future. First, an existing stand-in agent or knowledge provider determines the set of currently accessible locations using a broadcast-like mechanism of the underlying communication infrastructure. Then, it may decide to create and deploy its clones on one or more of these accessible locations. After its creation, each deployed stand-in agent chooses the type of functionality it will provide in its location and repeats the evaluate/deploy process.
- **information propagation**, sophisticated mechanisms for propagating the agent state updates (in terms of location, available resources, updated commitments) to its stand-in agents and synchronization within the community of the stand-ins. In

trusted communities synchronization is based solely on the freshness of the mode, while in non-trusted communities trust values also need to be considered.
- **conflict resolution**, when the stand-in agents synchronize their status (mainly in terms of the agreed commitments) with the formerly inaccessible agent. It often happens that the several stand-ins may have agreed on different commitments, which need to be resolved.

All three processes need to be tuned to the specific needs and properties of the given domain, such as the cost of communication, average accessibility among two agents, etc. The concept of stand-in agents is currently advantageous in the two very specific situations:

- in a very **dynamic environment**, with relatively low path accessibility (this can be in situations where a small number of unmanned vehicles are collaboratively inspecting large areas), or
- in a **non-trusted environment** with at least some communication inaccessibility (in these cases the agents do not want to provide sensitive knowledge for sharing while off-line).

Stand-in agents solve inaccessibility in two ways: the first is routing communication protocol based on swarming and micro-payments within agent community and the second is distribution of social knowledge. In this paper the authors talk about stand-ins without social knowledge functionality [10], because we want build only message passing system there. These stand-ins provide top-level communication API in mobile network.

An important attribute of the stand-ins is their passive role in the network. These agents are meant to be carried on a physical device and they aren't able to affect position of the device in mobile network anyway.

On the Figure 3, we present a sample stand-in agent network: nodes represent fixed locations, lines between nodes means that there exists a communication link between them and residing stand-in agent is represented by point near the node.

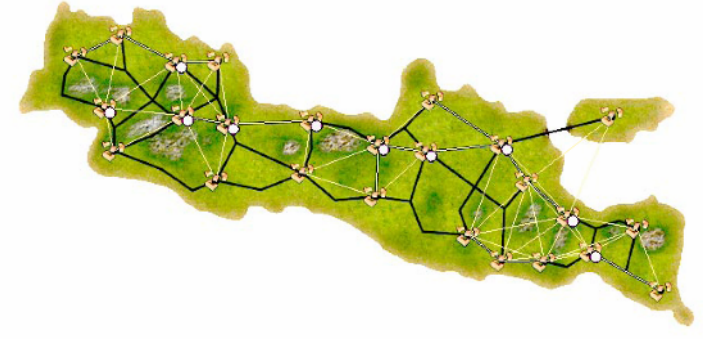

Fig. 3. An Illustrative Scenario

Unlike classical middle agent architectures [11] where the prime functionality is devoted towards matchmaking and negotiation, we would like to extend the concept

of middle agent by its capability to autonomously migrate in the network, clone and destruct copies. Such extensions would allow us to better integrate the generic stand-in agent architecture with the coordination algorithms for the proposed framework. In Figure 4 we present an abstract architecture of the stand-in agent. The architecture presents the following components:

– **Swarming controller** consists of two modules: population manager ensures cloning; migration and destruction of stand-in agents in the system while the informationpropagator manages information flows through the agent, more specifically the messages or knowledge to transfer or actions to take. The module must balance between two extreme cases of knowledge handling: propagation to all visible targets or no propagation at all. Even if both modules are domain independent, they depend on the domain specific functions included in the knowledge base algorithms.
– **Knowledge base** is a domain specific knowledge structure of the stand-in agent, consists of three parts: activity knowledge, information evaluator and timeout checker. While the activity knowledge contains the domain specific knowledge and the meta-data provided by the propagator, the information evaluator and timeout checker are the algorithms working on this knowledge.

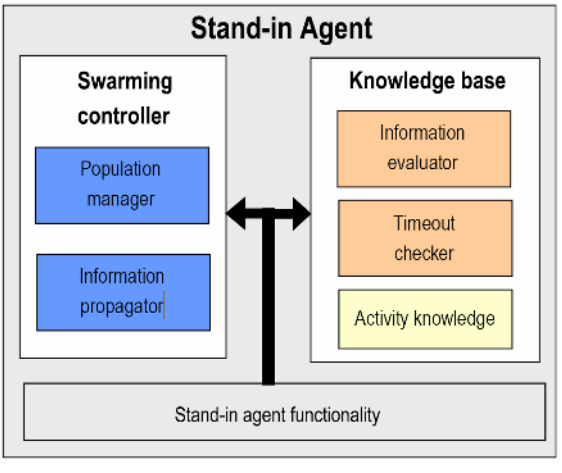

Fig. 4. Architecture of Stand-in agent

The information evaluator classifies and indexes the knowledge, so that the index values can be used by information propagator to manage its activity and further propagation. It also evaluates the knowledge usefulness. The timeout checker module implements forgetting of the activity knowledge.
– **Stand-in agent functionality** is the universal interface between modules and agent platform. It provides fundamental agent functions (clone, migrate and die), message interface and monitoring listeners, as well as original stand-in agent code. This code depends on the actual type of the stand-in agent. Via monitoring listeners it notifies modules about visibility of the other nodes, information about accessible other stand-in agents and also about presence of potential message receiver. Only this part of relay agent needs to be changed to work properly with another agent platform.

Combined, these capabilities allow the stand-in agent to provide the basic infrastructure for a distributed negotiation-based coordination component for the framework. In the following illustrative example, we illustrate the necessary steps involved for resource negation and allocation for a single data stream between a provider (source) and two consumers (sinks).

5 Illustrative Scenario

An illustrative application scenario is shown in figure 5, where a schematic view of a combat operation is presented. The figure illustrates a scenario with nine interconnected hosts. In the battlefield, each of these hosts might represent tanks, robotic vehicles, or soldiers. Each computational host (or environment) can execute a number of applications (or agents) represented in the figure as circles connected to the environment where they are executing.

Consider the case illustrated in figure 5 where clients (A) and (B) residing respectively in hosts H3 and H8 place a request for a specific data-stream from the same sensor application (S) executing in host H1.

Furthermore, consider the situation where the requests are different (but derivable from) each other. This would be the case, for instance, of client (A) requesting continuous GPS position information from sensor (S – presumed to be a UAV), while client (B) requests only low rate information for visualization purposes.

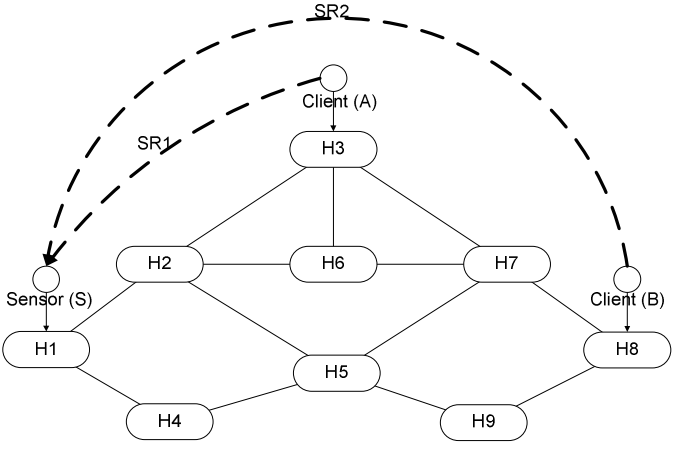

Fig. 5. An Illustrative Scenario

In this example, the order in which each request is place should not (in theory) change the final data distribution graph, as regardless of order, a final optimal solution will always be sought. However, in practice, the order does matter because changes in data distribution trees are based on changes in cost thresholds. That is, unless there are significant cost gains in moving from one solution to another, the system tends to preserve the current configuration.

Let us assume, for the purpose of this illustration, that both requests are placed simultaneously, represented in figure 5 by the dotted edges oriented from the "re-

quester" to the "source" node. Conversely, the actual data streams in this illustration will be represented by solid arrows from the source node to the sinks (original requesters).

When receiving the requests, the sensor node (represented here as the agent – S) will start the policy verification and negotiation process to determine best allocation of resources.

First, the agent will query the local guard for policy constraints that might restrict communications or information release between the source and sink agents. The Guard will relay the query to the policy framework, and will then parse, cache, and return the results. Cached information will be used by the Guard to answer future policy queries in situations when the policy framework is temporarily inaccessible.

If there are policies constraining communications, for example between the sensor agent (S) and the client (A), these are automatically enforced in the form of changes in request parameters or automatic insertion of policy enforcement agents (or filters) in the data path. To simplify the example, let us consider the case where there are no policy constraints in place.

In this case, the sensor (S) will start negotiating resource allocation with other environments. Due to the incomplete connectivity inherent to these types of networks a protocol that would actually support agent negotiation between peers would be very expensive (in terms of messages exchanged) and possibly highly inefficient. Furthermore, during the negotiation process, there might be potential candidate nodes that are temporarily out of range, or in passive mode.

As discussed in section 4, the negotiation of resources in the framework can be done through two different mechanisms: Stand-In agents or Acquaintance Models. Different scales and types of scenarios might benefit from utilizing one approach or the other.

Consider for instance the case where Stand-In agents are used for resource negotiation. In this approach, a representative of each environment is either present at host H1, ready for negotiation, or are directly accessible (at low communication costs) by other representatives resident at H1. Information about data transmission and processing costs can be queried, on demand, by agent (S) to representatives of candidate nodes. Stand-In agents can also make resource commitments on behalf of their environments.

Armed with that information, agent (S) can identify resources in the network that can be tasked for data transmission or processing. Selected environments are then notified and configured for the task.

Figure 6 shows a schematic view of the data distribution stream established by agent (S) in coordination with its peers. It is important to note that in cases where (S) is a node highly constrained in computational resources, the negotiation process can be moved to a neighbor node (for instance H4).

Once established, the data-stream (or streams) will persist until canceled by the clients, or blocked by policies. In the case where changes in policy or topology might temporarily break sections of the stream the local agent in the affected node is responsible for corrective actions to quickly reestablish the data flow. For instance, if the communications edge between hosts H5 and H7 is dropped, agent R1 immediately starts a negotiation process to reestablish the link with R2.

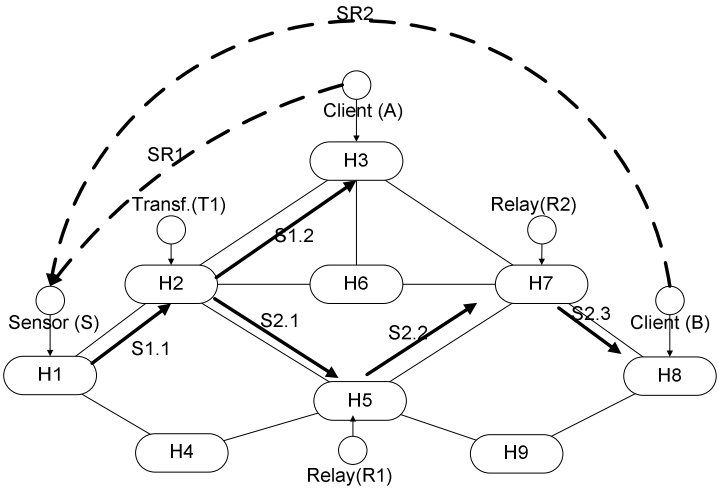

Fig. 6. A data distribution example

Periodically, agents re-evaluate local decisions with their peers, to find new local optima in cases where differences in cost are high enough to justify the changes. In this example, there is no global optimization between multiple sources.

6 Conclusions and Future Work

In this paper, a conceptual description of a mobile agent-based framework was presented for military battlefield environments. The framework provides three core capabilities: a) point-to-point messaging between applications; b) a publish-subscribe oriented model for data streaming, and c) on-demand, opportunistic resource allocation for communications and in-stream data processing.

Our approach relies on four core technologies that combined provide the features necessary to address the requirements demanded by the environment. Mockets and FlexFeed provide the application level access to the middleware, exposing the APIs for messaging and access to data-streams. Stand-In agents and Acquaintance models are presented as instances of remote presence and remote awareness capabilities (respectively), fundamental for a robust and truly distributed coordination model.

Although we currently don't have a fully integrated version of the framework, based on several examples of actual deployment and on a number of experiments conducted with each component independently (and some combined in ad hoc tests), we are confident of the capabilities that such middleware can provide to the types of scenario addressed here. Future work will involve the integration of coordination components with the rest of the framework to build a prototype for tests and demonstrations. We also plan an extensive set of tests and experiments to validate the concept and characterize the prototype.

References

1. Perkins, C. (2003). Ad hoc On-Demand Distance Vector (AODV) Routing. IETF – IETF Request for Comments (*RFC3561*).
2. Bradshaw, J.M. et al.: KAoS: Toward an Industrial-Strength Generic Agent Architecture. Software Agents, AAAI Press/MIT Press, Cambridge, Mass. 1997, pp. 375-418.
3. Bradshaw, J. M., et al. (1999). Agents for the masses: Is it possible to make development of sophisticated agents simple enough to be practical? IEEE Intelligent Systems(March-April), 53-63.
4. Carvalho, M. and Breedy, M. (2002) Supporting Flexible Data Feeds in Dynamic Sensor Grids Through Mobile Agents. Proceedings of the 6th International Conference on Mobile Agents (MA 2002). Barcelona, Spain. Berlin: Springer-Verlag
5. Carvalho, M. et al. (2005) The ULM Algorithm for Centralized Coordination in FlexFeed. To appear in the proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics – Orlando (USA) - 2005
6. Pechoucek, M., Marık, V. and Stepankova, O. Towards reducing communication traffic in multi-agent systems. Journal of Applied Systems, 2(1):152-174, 2001. ISSN 1466-7738.
7. Cao, W.,. Bian, C.G., and Hartvigsen. G. Achieving e cient cooperation in a multi-agent system: The twin-base modeling. In P. Kandzia and M. Klusch, editors, Cooperative Information Agents, Number 1202 in LNAI, pages 210-221. Springer-Verlag, Heidelberg, 1997.
8. Cordeiro, C., Das, S. and Agrawal, D. COPAS: Dynamic Contention-Balancing to Enhance the Performance of TCP over Multi-hop Wireless Net-works", Proceedings of IC3N'02, Miami, FL, Oc-tober 2002
9. Gupta, A., Wormbecker, I, Williamson, C., Experimental Evaluation of TCP Performance in Multi-Hop Wireless Ad Hoc Networks. The IEEE Computer Society's 12th Annual International Symposium on Modelin, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04). October 2004, The Netherlands.
10. K. Sycara, J. Lu, M. Klusch, and S. Widoff. Dynamicservice matchmaking among agents in open information environments. ACM SIGMOID
11. D. Šišlák, M. Rehák, M. Pěchoucěk, and P. Benda. Optimizing agents operation in partially inaccessible and disruptive environment. In Intelligent Agent Technology, 2005 IEEE/WIC/ACM International Conference, number PR2416 in IEEE, 200