# A Distributed Stand-in Agent based Algorithm for Opportunistic Resource Allocation

Petr Benda, Pavel Jisl, Michal Pěchouček
Gerstner Laboratory, Czech Technical University
166 27 Prague 6, Czech Republic
{bendap1|jisl|pechouc}@labe.felk.cvut.cz

Niranjan Suri, Marco Carvalho
Institute for Human and Machine Cognition
40 South Alcaniz St. Penascola, FL, USA
{nsuri|mcarvalho}@ihmc.us

## ABSTRACT

Mobile ad-hoc networks (MANET) are expected to form the basis of future mission critical applications such as combat and rescue operations. In this context, communication and computational tasks required by overlying applications will rely on the combined capabilities and resources provided by the underlying network nodes. This paper introduces an integrated **FlexFeed**/$\mathcal{A}$-**globe** technology and distributed algorithm for opportunistic resource allocation in resource- and policy-constrained mobile ad-hoc networks. The algorithm is based on agent negotiation for the bidding, contract and reservation of resources, relying primarily on the concept of remote presence. In the proposed algorithm, stand-in Agents technology is used to create a virtual, distributed coordination component for opportunistic resource allocation in mobile ad-hoc networks.

## Categories and Subject Descriptors

[**Agents, Interactions, Mobility, and Systems (AIMS)**]: Multi-Agent systems

## General Terms

Coordination, Performance

## Keywords

agents, multi-agent systems, ad-hoc networks, inaccessibility, defence applications

## 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 environments 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.

In this paper, we introduce a novel agent-based communications framework designed to help address this issue in these types of environments. The goal is to provide a framework 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. INTEGRATED ARCHITECTURE

The **FlexFeed**/$\mathcal{A}$-**globe** integrated infrastructure provides a powerful environment supporting point-to-point messaging, publish-subscribe data streaming, on-demand, opportunistic resource allocation in the dynamics of the environment that is given by (i) mobility of the communication infrastructure (ad-hoc networking) and (ii) changing operational priorities and optimization criteria.

For simulation purposes we use simulator, based on $\mathcal{A}$-**globe** platform [13] (see section 2.3). It simulates the be-


*SAC'06* April 23-27, 2006, Dijon, France

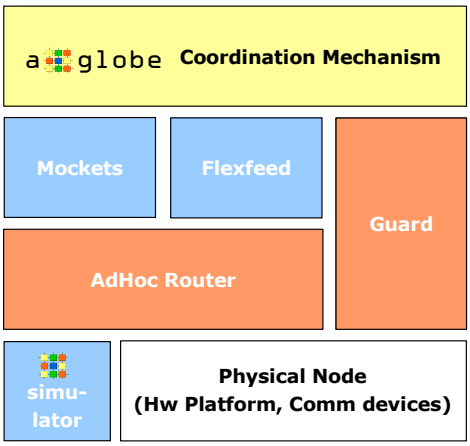


**Figure 1: FlexFeed/$\mathcal{A}$-globe architecture design**

havior of real hardware and is extended with geographical and environment simulators. We can simulate the physics of units, their movements and dynamically add and remove the units.

The architecture has been designed to operate with the real-life hardware (robots, unmanned vehicles or communication devices) and it consists of several mutually linked components (see Figure 1):

- **$\mathcal{A}$-globe simulator** – Due to the fact that the real-life experiments can be costly and in some cases it can be impossible to implement, an inseparable component of this architecture is the agent-based simulator (presented in Section 2.3 that models the communication and interaction environment.

- **ad-hoc Router** – The simulator and/or real hardware is connected with highest levels of architecture over ad-hoc Router. Ad-hoc router supports multiple routing algorithms and is implemented at application level of ISO/OSI specification [8]. For connected applications the ad-hoc router acts as normal UDP sockets. This extension provides restrictions in connection between units and components depending on visibility or policies.

- **Mockets** – Mobile Sockets are an application level transport layer, designed to transparently provide resource redirection and cross-layer interaction in mobile ad-hoc network environments. For details see Section 2.2.

- **FlexFeed** – This technology is used for stream-oriented communication between agents. While **FlexFeed** is designed for stream planning and controlling, Mockets provide an infrastructure used for stream transmission. For thorough presentation see Section 2.1

- **Guard** – This component is used for controlling the resource allocation and policy checking. Extended with KAoS [2, 3], the policy framework for **FlexFeed** and NOMADS, it can provide the mechanism for definition, verification, distribution and enforcement of policies restricting agent access to sensor data, bounding agent resources and governing the mode of notification to users.

- **$\mathcal{A}$-globe coordination** – The above listed components are closely linked to **$\mathcal{A}$-globe** coordination mechanism. This is a critical component which provides both central and distributed high-level planning and resource allocation algorithms for coordinating the communication flows. The coordination mechanisms represent the main contribution of this paper and presented in Sections 2.4 and 3.1.

The **FlexFeed**/**$\mathcal{A}$-globe** integrated architecture was designed on background knowledge of complex frameworks, designed in IHMC (eg. **FlexFeed** , KAoS, NOMADS) and the **$\mathcal{A}$-globe** project, designed in Gerstner Laboratory at Czech Technical University. Both technologies have its advantages and disadvantages and their connection can provide easy implementable architecture for mobile agents in environments with constrained accessibility.

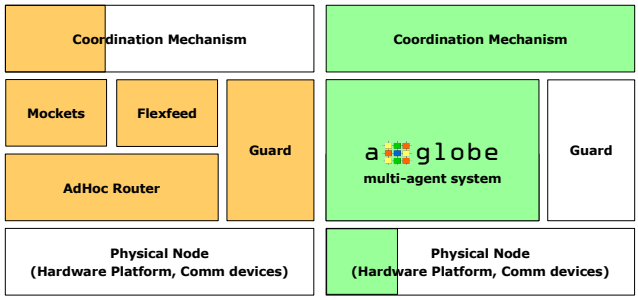


**Figure 2: IHMC provided (left) and Gerstner Laboratory provided (right) components**

The provided technologies are complementary and in part overlapping (see Figure 2). The GL provided **$\mathcal{A}$-globe** multi-agent platform that implements its own messaging and routing services and supports full agent migration. This is why it can replace the infrastructure provided by **FlexFeed**, Mockets and the ad-hoc router. The policy enforcement is provided only by Guard and is not implemented in **$\mathcal{A}$-globe**. As the **$\mathcal{A}$-globe** component is in charge of the coordination mechanisms, in **FlexFeed** only a centralized coordination is implemented, using the ULM algorithm [4]. This centralized coordination is implemented in **$\mathcal{A}$-globe** as well, however **$\mathcal{A}$-globe** also provides partially distributed and fully distributed coordination algorithms and motion oriented coordination planning. The physical nodes can be used in both architectures, but **$\mathcal{A}$-globe** provides the support for simulation of these real-life physical nodes (discussed in Section 2.3).

## 2.1 FlexFeed

**FlexFeed** [5], 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. 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 frame-

work level, hidden from data producers and consumers. For example, the Unmanned Underwater Vehicle (UUV) is the source of the data, as it carries a high resolution video camera. Then, a human crew at the base requests a high resolution data-stream from the UUV to identify an object while at the same time, another UUV requests visual data to monitor the movement of this object. The goal of the **FlexFeed** framework in this case is to identify the best data distribution tree that would minimize the 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 is found.

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 UUV, for instance, can be moved to a sub-optimal position to re-allocate or release resources.

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. 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 [10] 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 an approximate global optimization of joint or disparate streams.

## 2.2 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 are used by **FlexFeed** framework as a message transport layer. During the movements of agents in environment, the Mockets provide **FlexFeed** with a stable channel for streaming information from source to sink node. The problem with redirecting the streams and messages is then solved by Mockets.

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 [6, 7].

## 2.3 $\mathcal{A}$-globe Simulation

**$\mathcal{A}$-globe** is Java-based, fast, scalable and lightweight agent development platform with environmental simulation and mobility support. Beside the functions common to most agent platforms (such as JADE, COUGAAR, FIPA-OS or JACK agent platform) it provides a position-based messaging service, so it can be used for experiments with extensive **environment simulation** and **communication inaccessibility**. Communication in **$\mathcal{A}$-globe** is very fast and the platform is relatively lightweight [13].

The main focus of the **$\mathcal{A}$-globe** developers has been given to the following applications domains:

- **simulation**, especially simulation of the multi-agent environment and collective behavior of large communities
- **scalability and lightweightness**, high-number of fully fledged and fully autonomous agents, that are loosely coupled with lightweight infrastructure
- **weak agent migration** persistence and code and state migration within the communication network as much as physical reallocation of the computational host and thus modeling of partial and non-permanent communication inaccessibility [9].

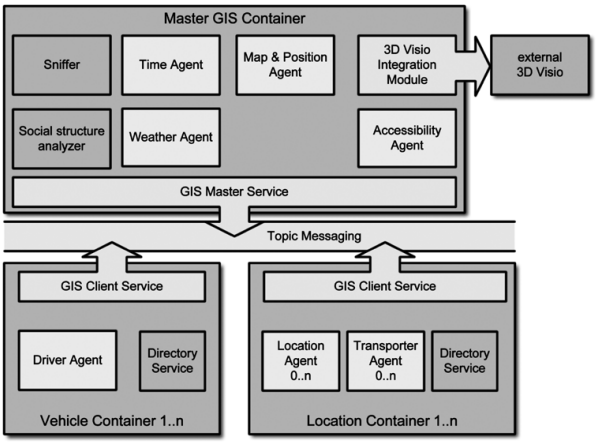


**Figure 3: System architecture of $\mathcal{A}$-globe**

**$\mathcal{A}$-globe** is suitable for real-world simulations including both static and mobile units (e.g. logistics, ad-hoc networking simulation), where the core platform is extended by a set of services provided by a Master GIS (Geographical Information System) container hosting various environment simulator (ES) agents. The ES agents simulate geographical position and time dynamics of each unit. Simulation scenario is defined by a set of actors represented by agents residing in the agent containers. All agent containers are connected together to one system by the GIS services. Beside the simulation of dynamics the ES agents can also control communication accessibility among all agent containers. The GIS service applies accessibility restrictions in the message transport layer of the agent container.

## 2.4 $\mathcal{A}$-globe/FlexFeed Coordination

The proposed integrated architecture proposes four different approaches to coordination:

- **centralized** – A single process (coordinator) searches *F-Graph* – a centralized knowledge structure that represents communication accessibility in the network. By means of Dijkstra-like algorithm, the coordinator finds the optimal feed. This centralized coordination is implemented in **FlexFeed** and uses the ULM algorithm [4].
- **partially decentralized** – The centralized approach can be partially distributed among regions, where the coordinators maintain their own *F-Graphs* representing communication accessibility particular region. For planning feed and coordination, the region coordinators need to negotiate the path for routing to another region.
- **fully decentralized** – In fully decentralized coordination approach each agent maintains a local *F-Graphs* representing communication accessibility in its direct neighborhood. This information is used for planning and coordinating the feed in peer-to-peer manner.
- **motion enforcement coordination** – Yet another approach to coordination is used in the situations with higher level of communication inaccessibility among the agents. This approach plans and controls not only the communication traffic but also movement and change of location of the particular nodes (robots).

At this moment the **FlexFeed** framework implements only the centralized coordination. In $\mathcal{A}$-**globe** simulation scenarios the partially and fully decentralized coordination as well as the motion enforcement coordination were implemented in the NAIMT [11] project.

In the remaining part of this document we will be discussing the fully decentralized approach to coordination. In principle, there are two fundamental approaches how such coordination can be implemented – by means of the *remote awareness* or *remote presence* agent concepts. While in the former case, each node is expected to autonomously maintain the information about the accessible nodes and perhaps their accessibility in the *acquaintance model*, in the latter case each individual node creates so called *stand-in agents*. The stand-in agents represent its owner and are disseminated across the network in order to represent its owner accessibility.

# 3. FULLY DISTRIBUTED COORDINATION

In the next section, the stand-in agents based approach is going to be presented.

## 3.1 Stand-in Agents

In standard situations, agents in a multi-agent community need to communicate with each other to accomplish cooperation and coordination of joint activities. Normally it is expected that each agent can contact any other using a reliable communication infrastructure. Often one has to think about situations when a communication subsystem becomes disintegrated and some agents become isolated from the rest of the community. Stand-in agents solve inaccessibility in two ways: the first is the routing communication protocol based on swarming and micropayments within agent community and the second is distribution of social knowledge. In this paper we talk about stand-ins without social knowledge functionality [12], because we want to build only a 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 4, we present a sample stand-in agent network: nodes represent fixed locations, lines between nodes mean that there exists a communication link between them and a residing stand-in agent is represented by a white point near the node.

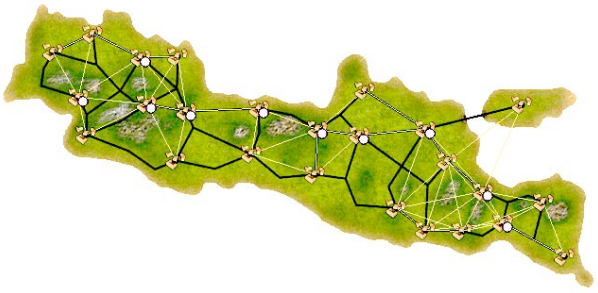

**Figure 4: Stand-in agent network in a system with inaccessibility.**

We will address the integration of the algorithm with the generic stand-in agent architecture. Unlike classical middle agent architectures [14] where the prime functionality is devoted towards matchmaking and negotiation, we would like to extend the concept of the middle agent by its capability to autonomously migrate in the network, clone and destruct copies. In Figure 5 we present an abstract architecture of the stand-in agent, that is composed of 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 information propagator 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**, 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. 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** – 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 accessibility of 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.

In the next section we will discuss the details of swarming controller module that has placed the stand-in agents in the system.

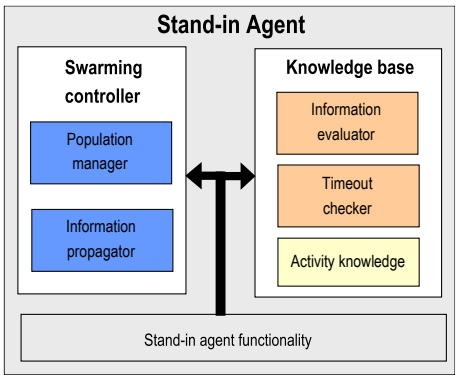


**Figure 5: Architecture of Stand-in agent**

## 3.2 Stand-in based Coordination

As mentioned above one of the key issues in stand-in operation is their proper location in the network. The distributed stand-in agent allocation mechanism uses only locally accessible information. It does so not only to minimize the network maintenance communication, but it also enables operation in the disruptive or partially inaccessible environment. Locally accessible information is obtained by monitoring stand-ins neighborhood – identifying currently visible targets and other stand-in agents. The algorithm needs to be lightweight and computationally simple, as the stand-in instances can be constrained by the devices they run on. Scalability in space and density shall also be an important property of the targeted solution.

In principle there are two key approaches to controlling the efficiency of the stand-in agents allocation:

1. **forward swarming control** – where the stand-in agent migrates its clone only to the locations with higher possibility of future inaccessibility and higher interaction expectancy and
2. **backward swarming control** – where the stand-in agents dispatch their clones to every reachable destination and the useless ones are eliminated in the future, reflecting the actual state of inaccessibility.

Each of the approaches has its advantages and disadvantages. The forward swarming control is computationally efficient, as it tries to minimize the number of stand-in agents in the system and prevent the possible swarming explosion. This is why this approach seems to be particularly suitable for domains with high scalability and operational efficiency requirements. On the other hand, the backward swarming control has an important advantage. This approach is substantially more domain independent, demands less knowledge about the environment nature and is more robust, as it doesn't explicitly use any prediction about the future of the community.

We have opted for the use of the backward swarming, as this approach is more robust and domain independent. Abstract criteria of the system quality defined in the introduction were also formulated in a precise manner, with descending priority:

– Provides connection between any two system elements through the minimum number of stand-in agents.

– Minimize the number of stand-in agents in the system.

– Minimize the number of messages for system operation and/or knowledge maintenance.

Population manager (see Figure 5) is driven by a biology inspired algorithm. Social dominance and altruism models [15] were successfully used to partition the group of agents into those who work for the good of the community and the others, who profit from the altruism of the first group. During the experiments with rats, it was determined that exactly the sufficient number of individuals behaves in an altruistic manner to optimize the whole group fitness. They bring the food and share it with the others, who only consume. This behavior is formalized by a simple mathematical model formulated in [15]. To ensure the target coverage, stand-in agents can be reproduced in the system using two main propagation strategies:

– **full flood fill** – any stand-in agent initiates full flood filling reproduction strategy when it identifies a new unserved knowledge target in its reach. To decide whether the target is really new, all agents keep a set of served targets, that includes both the other stand-in agents and the knowledge about final users. A target is removed from the set when it is not used for a specified period – forget time. In practice, the stand-in agent is cloned to every visible node where it is not running yet if the new knowledge target is not reachable from current position of the agent. Created clones further clone themselves to new locations without existing stand-in agents using the same cloning termination condition: target reachability. Only this simple strategy can ensure that the stand-in agent network will reach the target. Using random walk instead of flood fill is possible, but not advisable, because the random walk does not guarantee finding the target, as known from Polya's random walk theorem [1],

– **bounded flood fill** – this is depth-limited version of the previous reproduction strategy. After the initiation, the stand-in agents are successively cloned only up to the depth specified by FloodFillDepth constant. This reproduction strategy is triggered by a local accessibility change when the source agent holds relevant, non-expired knowledge. Application of this mechanism can identify shorter paths enabled by the accessibility change or can deliver the knowledge to the isolated cluster by the stand-in agent on the mobile node.

Both flooding strategies are time limited. There is a specified constant flood duration during which the stand-in agent retains reproduction intention. When this period expires, the agent no longer reproduces until the new reproduction is started by the agent itself or the others.

To keep the number of stand-in agents close to optimum, the population manager contains also the methods that decrease the number of stand-in agents in the system:

In random duels the attacking stand-in agent randomly selects an adversary between accessible agents and launches an attack with force proportional to its profit during specific period, as determined by information propagator (see bellow). Besides the attack force, the attack also includes the information about its active target set at the time of the attack. The attacked agent evaluates the attack and decides whether it won or lost. If the attacked agent loses, it removes itself from the system; losing attacker is not penalized for the attack. Attack evaluation compares the active targets first and when one is a subset of the other, its owner loses the fight. When the sets are identical, force of the attack decides the fight – the stronger agent wins. Active target set size is evaluated differently for new and old agents. For the young agents that are not yet completely adapted to the environment, the set contains all directly accessible targets, while it contains only the really used ones. Besides this advantage, the youngest agents benefit from the immunity period, during which they can not lose a fight while attacked.

In contrast to the previous case, the uselessness detection is an individual process. The stand-in agent can remove itself while it is isolated from the rest of the community and no access to relevant knowledge anymore.

Information propagator manages knowledge propagation and use in the system. This component uses virtual payments to reward the other agents for the knowledge, receives payments from the others for the information provided and generates the profit also from acting on behalf of the represented agent. Each agent optimizes its profit, ensuring the overall information flow efficiency. More specifically, stand-in agents reward the information received from the others in function of information usefulness and redundancy – the first agent from which they receive the information receives significant payment, while the subsequent information is rewarded less. On the other hand, the transmitting the information to other agents is not free of charge for the agent – it must carefully decide to which agents it propagates which information. The decision is taken in function of the previous experience (and the current network status) with similar knowledge, and the knowledge source and potential target are important, domain independent similarity criteria. Besides these criteria, we may enhance the knowledge with meta-data specifying to which agent it has been already provided. To make the system more robust, the decision to which nodes we send the information is probabilistic – agents may therefore send the knowledge to the less rated directions to find better paths in the system. Payment for the information is transmitted as a reply to the knowledge update message.

Historical data (represented as probabilities assigned to knowledge characteristics, origins and targets) that are used to identify the targets to which we send the information are periodically updated and the old data is discarded. When a new target appears, the initial message transmission probability is set to the level derived automatically from the current network state of the evaluating agent. The domain specific functions that evaluate the usefulness of the knowledge, indexable knowledge characteristics and rewards for actions are provided by the domain-specific knowledge base. We shall keep in mind that the knowledge is not only propagated by messaging in the network of stand-in agents, but also carried by the agents created during the reproduction process. This is especially important for the communities where the accessibility is relatively low, or where the agents are clustered.

# 4. CONCLUSIONS

The main contribution of this paper is in reporting on the integration of the stand-in technology into the **FlexFeed** framework. This integration provides all stand-in behavior and many changes in NOMADS and **FlexFeed** core were applied.

The stand-ins implementation was divided into stand-ins Core and platform-dependent implementation of underlying infrastructure. The stand-in Core is platform independent and provides all the above mentioned behavior. For easy implementation, the interfaces for stand-in required infrastructure were deployed. Platform-dependent part of stand-ins provides infrastructure for stand-in creation and cloning, messages propagation and solving visibility of containers and environments.

A set of experiments was carried out in order to study the stand-in behavior in environment with limited communication ranges.

At first (Fig. 6, top) we were slowly changing visibility ranges on the network with fixed nodes. In the second setup (Fig. 6, middle), we have added one mobile node into the network. The movement of the mobile node through the whole network introduces local accessibility changes. In the third setup (Fig. 6, bottom), two mobile nodes were moving faster through the community, causing more important disturbances in the network topology.

In the experiments, we clearly demonstrate that the agents are able to organize themselves efficiently and to approach the optimal number as determined by the reference algorithm. However, after the steep initial decrease, we can observe the peaks that correspond to agent propagation in response to the topology changes. In the mobile scenarios, we have failed to match the reference solution perfectly, as the adaptation time is somewhat higher than the average change period, but the results remain comparable and the robustness and distribution of the algorithm provides enough of the incentive for its application. Note, the network of stand-is creates top-level reliable message transport layer in our experiments.

We measured the evolution of the number of stand-in agents at three levels of inaccessibility dynamics. To determine the optimal number of stand-in agents in each moment, we have implemented an efficient centralized algorithm [12]. In our domain, where the accessibility is distance-based, this algorithm behaves optimally.

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.

# 5. ACKNOWLEDGMENTS

This work has been prepared through IHMC participation in the Advanced Decision Architectures Collaborative Technology Alliance sponsored by the U.S. Army Research Lab-

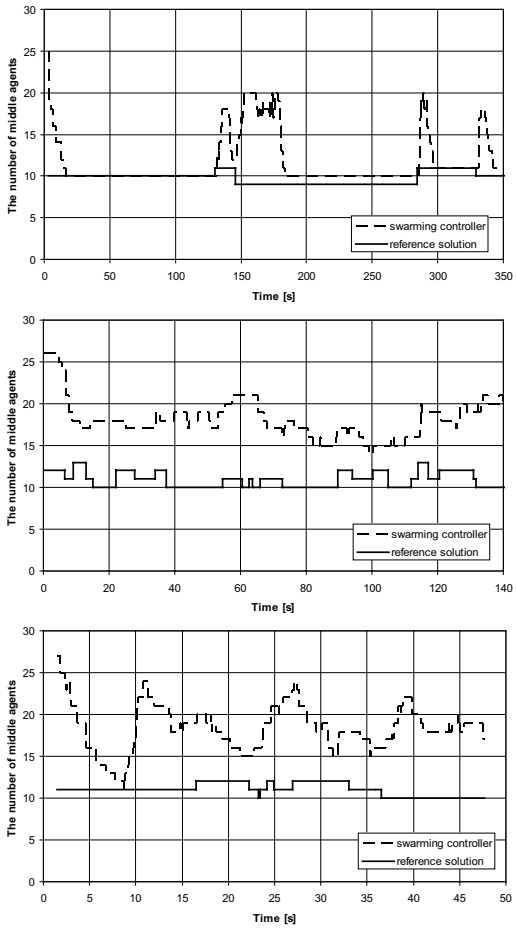


**Figure 6: The figure presents adaptation of stand-in agent network to the changing environment with different speed of changes: from infrequent changes (top chart) to the fast, frequent changes (bottom chart).**

oratory under Cooperative Agreement DAAD19-01-2-0009, and was supported in part by the DARPA Control of Agent-based Systems (CoABS) Program.

This work has been also facilitated by the ATG research grant N00014-03-1-0292, granted by Office of Naval Research and grant FA8655-04-1-3044, granted by European Office for Aerospace Research and Development (EOARD), US Air Force Research Laboratory, Rome (NY).

## 6. REFERENCES

[1] G. L. Alexanderson. *The random walks of George Pólya*. The Mathematical Association of America, 2000.

[2] J. M. Bradshaw. Kaos: Toward an industrial-strength generic agent architecture. In *Software Agents, AAAI Press/MIT Press*, pages 375–418, 1997.

[3] J. M. Bradshaw, M. Greaves, H. Holmback, T. Karygiannis, W. Jansen, B. G. Silverman, N. Suri, and A. Wong. Agents for the masses? *IEEE Intelligent Systems (March/April)*, 14(2):53–63, 1999.

[4] M. Carvalho, F. Bertele, and N. Suri. The ulm algorithm for centralized coordination in flexfeed. In *Proceedings of the 9th World Multi-Conference on Systemivcs, Cybernetics and Informatics - Orlando, USA*, 2005.

[5] M. Carvalho and M. R. Breedy. Supporting flexible data feeds in dynamic sensor grids through mobile agents. In *MA '02: Proceedings of the 6th International Conference on Mobile Agents*, pages 171–185, London, UK, 2002. Springer-Verlag.

[6] C. Cordeiro, S. Das, and D. Agrawal. Copas: Dynamic contention-balancing to enhance the performance of tcp over multi-hop wireless net-works. In *Proceedings of IC3N'02, Miami, FL*, 2002.

[7] A. Gupta, I. Wormsbecker, and C. Williamson. Experimental evaluation of tcp performance in multi-hop wireless ad hoc networks. In *The IEEE Computer Societys 12th Annual International Symposium on Modelin, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS04)*, pages 3–11, 2004.

[8] ISO. Open Systems Interconnection Reference Model. `http://en.wikipedia.org/wiki/OSI_model`, 2005.

[9] M. Pěchouček, V. Mařík, D. Šišlák, M. Rehák, J. Lažanský, and J. Tožička. Inaccessibility in multi-agent systems. final report to Air Force Research Laboratory AFRL/EORD research contract (FA8655-02-M-4057), 2004.

[10] C. Perkins. Ad hoc on-demand distance vector (aodv) routing. *IETF - IETF Request for Comments (RFC3561)*, 2003.

[11] M. Rollo, P. Novák, and P. Jisl. Simulation of underwater surveillance by a team of autonomous robots. In Mařík, Brennan, and Pěchouček, editors, *Holonic and Multi-Agent Systems for Manufacturing*, number 3593 in LNAI, pages 25–34. Springer-Verlag, Heidelberg, 2005.

[12] D. Šišlák, M. Rehák, M. Pěchouček, and P. Benda. Optimizing agents operation in partially inaccessible and disruptive environment. In *Intelligent Agent Technology, 2005 IEEE/WIC/ACM International Conference*, number P2416 in IEEE, 2005.

[13] D. Šišlák, M. Rollo, and M. Pěchouček. A-globe: Agent platform with inaccessibility and mobility support. In M. Klusch, S. Ossowski, V. Kashyap, and R. Unland, editors, *Cooperative Information Agents VIII*, number 3191 in LNAI. Springer-Verlag, Heidelberg, September 2004.

[14] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOID Record*, 28(1):211–246, 1999.

[15] V. Thomas, C. Bourjot, V. Chevrier, and D. Desor. Hamelin : A model for collective adaptation based on internal stimuli. In S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer, editors, *From animal to animats 8 - Eighth International Conference on the Simulation of Adaptive Behaviour 2004 - SAB'04, Los Angeles, USA*, pages 425–434, Jul 2004.