Third International Workshop on Agent Technology for Sensor Networks (ATSN-09)

Dr. W. T. Luke Teacy
School of Electronics and Computer Science
University of Southampton, UK
wtlt@ecs.soton.ac.uk

Dr. Alex Rogers
School of Electronics and Computer Science
University of Southampton, UK
acr@ecs.soton.ac.uk

Dr. Daniel D. Corkhill
Computer Science Department
University of Massachusetts, USA
corkill@cs.umass.edu

Dr. Paul Scerri
The Robotics Institute
Carnegie Mellon University, USA
pscerri@cs.cmu.edu

Prof. Sandip Sen
Department of Mathematical & Computer Sciences
University of Tulsa, USA
sandip-sen@utulsa.edu

Program Committee

Prof. Gul Agha
University of Illinois at Urbana-Champaign, USA

Prof. Giuseppe Anastasi
University of Pisa, Italy

Dr. Maxim Batalin
University of California at Los Angeles, USA

Dr. Jesús Cerquides
Universitat de Barcelona, Spain

Mr. Archie Chapman
University of Southampton, UK

Dr. Alessandro Farinelli
University of Southampton, UK

Prof. Nicholas R. Jennings
University of Southampton, UK

Prof. Victor Lesser
University of Massachusetts, USA

Dr. Bruce Moulton
University of Technology Sydney, Australia

Prof. Alun Preece
University of Cardiff, UK

Dr. Juan Antonio Rodriguez
IIIA, CSIC, Spain

Dr. Rónán Mac Ruairí
Dundalk Institute of Technology, Republic of Ireland

Dr. Steve Reece
University of Oxford, UK

Dr. Dimitris K. Tasoulis
Imperial College London, UK

Dr. Niki Trigoni
University of Oxford, UK

Table of Contents

Preface

Sensor networks are increasingly seen as a solution to the problem of performing wide-area monitoring and surveillance within environmental, security and military scenarios. Such networks consist of multiple sensors, deployed over a wide area, connected through a communication network (wired or otherwise). To ensure minimal human intervention the sensors within these networks should be able to self-organise, autonomously manage their own resources, and co-ordinate their behaviour to achieve system wide goals.

The distributed nature of these networks, and the autonomous behaviour expected of them, naturally lend themselves to a multi-agent methodology, and many of the technical challenges posed by these systems (e.g. decentralised control, co-ordination, resource allocation) form the basis of main-stream research within the agent community. However, such systems pose many additional challenges, not least how to manage limited computation and energy resources, constrained communication, and unreliable or fault prone network components within a dynamic and uncertain environment. Furthermore, the increasing availability of sensor network data, and the need to make use of it in real-time for informed decision making, requires the development of intelligent agents that can autonomously acquire data from these networks, and perform information processing tasks such as fusion, inference and prediction.

Against this background, the International Workshops on Agent-Technology for Sensor Networks have from 2007 to 2009 provided a forum where work and results in this area are discussed.

The impact of localization errors on the performance of the Ants exploration algorithm

Ettore Ferranti Niki Trigoni
Computing Laboratory
University of Oxford, Oxford, UK
{Ettore.Ferranti|Niki.Trigoni}@comlab.ox.ac.uk

ABSTRACT

When an emergency occurs within a building, it is safer to send autonomous mobile agents instead of human responders, to explore the area and identify hazards and victims. Existing exploration algorithms [11, 4] allow mobile agents to make distributed navigation decisions by communicating with nearby fixed sensors embedded in the environment. These algorithms are very efficient in terms of exploration time, but they have mainly been evaluated in simulation environments, where idealized assumptions were made regarding the ability of mobile agents to detect and localize fixed sensors in their vicinity. To address this problem, recent work [3] has focused on practical mechanisms for detecting and localizing sensors, implemented them in a real testbed, and derived realistic models of localization errors.

The objective of this work is to investigate the impact of these realistic errors [3] on the performance of the Ants exploration algorithm [11]. In particular, we simulate the performance of Ants with and without realistic errors, and show that introducing small errors can have a significant effect on the total exploration time.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Autonomous vehicles*

General Terms

Design, Measurement, Experimentation

Keywords

Sensors, Autonomous Agents, Robots

1. INTRODUCTION

When an emergency occurs within a building, the area is typically off-limits for anyone not wearing garments to protect themselves from exposure to hazards. In such adverse conditions, it is safer to deploy a group of autonomous robots, (*mobile agents*) to explore the area as fast as possible. Agents should overcome three important limitations: 1) lack of location information in indoor environments; 2) lack of direct connectivity between agents and 3) lack of map information. In order to address these challenges, recent work has proposed instrumenting the emergency area with tiny fixed sensors [11, 4]. By using the instrumented environment, mobile agents are able to explore the environment without map or location information, and to communicate with each other indirectly by using the sensors to leave and retrieve messages.

For simplicity, consider an area instrumented with fixed sensors lying in a grid topology. Wall cells, i.e. cells that are occupied by some obstacle, are the only ones without fixed sensors. We assume that a mobile agent is able to communicate with the fixed sensor on the current cell, as well as with at most eight fixed sensors in the surrounding cells. We also assume that the mobile agent is able to detect hazards and victims within the current cell. Exploration algorithms that use the above model [11, 4] typically follow four steps: 1) Sensor localisation: the mobile agent identifies the fixed sensors lying in the current and eight surrounding cells; 2) Sensor querying: the mobile agent queries the state of the previously localized sensors; 3) Sensor updating: the mobile agent updates the state of the fixed sensor in the current cell; 4) Navigation: the mobile agent selects one of the surrounding fixed sensors and navigates towards it. Note that exploration decisions are made in a completely distributed manner, by simply relying on the local state of the instrumented environment.

The weakness of previous studies [11, 4] is that they have only focused on the sensor tasking and marking steps, and have largely ignored the practical issues pertaining to sensor localization and navigation. They make unrealistic assumptions about the ability of an agent to accurately localize sensors in its vicinity, and move towards a selected sensor without odometry errors. In order to address these challenges, recent work [3] has proposed realistic localization and odometry error models based on experiments in a real testbed. The objective of this paper is to investigate the effects of applying the proposed error models [3] to the Ants exploration algorithm [11]. In particular, we integrate the error models into an existing simulation environment, and assess how the performance of Ants degrades as a result of introducing realistic errors.

The paper is organized as follows: Section 2 provides an overview of existing localization techniques, and summarizes the error models derived from applying one of them in a

real testbed. Section 3 briefly describes the Ants algorithm, which is one of the most popular and simple approaches to exploring a sensor-instrumented environment. Section 4 assesses the performance of the Ants exploration algorithm in a simulation environment with and without realistic errors.

2. BACKGROUND

In this section, we first give an overview of existing technologies for localizing sensor nodes. We then focus on a practical localization technique in which mobile agents equipped with cameras detect fixed sensors lying in their vicinity and localize them [3]. We provide a summary of detection and localization errors reported in [3], which are based on experiments run in a real testbed.

2.1 Localization technologies

Radio Signals: Radio signal strength is a not reliable way of identifying the robot relative position with respect to tags deployed in an environment. In fact, it heavily depends on factors like the relative orientation of the deployed motes, their height from the floor, the material of the floor, and the obstacles in the environment. Batalin et al. [1] create an algorithm called Adaptive Delta Percent, which takes into account the signal strength of the messages received from the various tags while the robot is moving in order to guide it toward one of them. A strong limitation of this approach is that the authors consider an experiment to be successful if the robot is able to reach a tag in the environment within a distance of 3m, an accuracy which is unreasonable for our scenario.

Infrared Signals: Several systems have been created to define mobile robot localisation in indoor environments. Some of them use ultrasonic and infrared technologies simultaneously [5], others radio frequency (RF) and infrared together [7], and some just infrared techniques [8]. However, infrared signals are not completely suitable for our scenario because they have a particularly limited transmission range (i.e. ∼20-30cm), thus the robot risks not being able to identify the deployed tag if the dimension of the cell is bigger than the allowed range. Moreover, interference from the IR component of other light sources could compromise the localisation process [6].

Ultrasonic Signals: Ultrasonic sensors [9] alone could be used to avoid obstacles, but not to identify specific tags in the environment due to the poor resolution of their readings. Therefore, we argue that IR or sonar are not suitable technologies for localizing sensors around an agent (avoiding *localisation errors*), or for guiding the agent to one of the sensors *odometry errors*.

Cameras and image processing: Since the previous approaches are not suitable for our scenario, we decided to explore sensor localisation using camera technologies. Several approaches investigated this area adopting feature cluster recognition [2]. In particular, some of them use image processing techniques to recognize landmarks in the environment [10]. However, most of the approaches are very sophisticated, and cannot run in resource-constrained mobile agents. A simple approach to localizing sensor nodes using cameras is proposed in [3]. In the next subsection, we summarize the error model derived by applying this approach in a real testbed.

2.2 Localization errors

In previous work [3], we proposed practical techniques that allow agents to use their on-board camera to localize sensors lying in their vicinity. In this section, we summarize the localization errors that were observed when we applied these techniques in a real testbed. Our system consisted of three different platforms: 1) mobile agent: Surveyor SRV-1 robot connected with a Tmote Sky mote; 2) fixed sensor: Tmote Sky mote with external bright LED and 3) gateway: laptop connected with a Tmote Sky mote (via its USB interface) used primarily for visualisation of experimental results. The sensors were deployed on the ground in a grid topology as shown in Figure 1, and the agent was placed in the middle of the central cell. The size of each cell was set to 48 cm.

The main results regarding detection and localization errors, are reported in [3], and summarized below: The percentage of undetected sensors, due to adverse light conditions, is not negligible and amounts to 5.56% of all sensors. Sensors that are correctly detected are then localized relative to the position of the mobile agent. Figure 1 shows estimated (circles) and real (squares) positions of sensors surrounding a given agent. In this case one can notice how, even if the sensors were not always correctly localised, the errors are always small enough, so that a sensor can not be thought to be in another cell from its own.

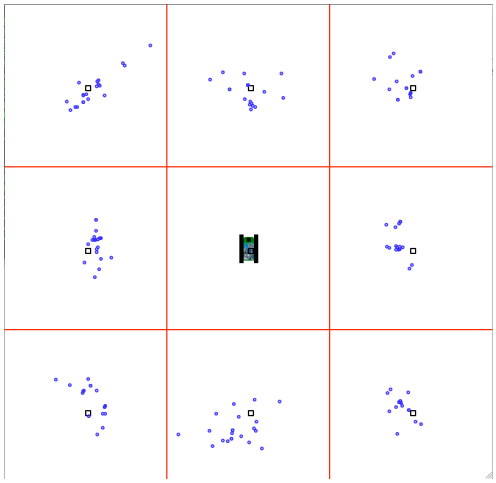


Figure 1: Localisation of sensors around a mobile agent.

3. THE ANTS ALGORITHM

In this section, we briefly describe the Ants algorithm proposed by Svennebring and Koenig in [11]. This is a distributed algorithm that simulates a colony of ants leaving pheromone traces as they move in their environment. Initially, all cells are marked with value 0 to denote that they are unexplored. At each step, an agent reads the values of the four cells around it and chooses to step onto the least traversed cell (the one with the minimum value). Before moving there, it updates the value of the current cell, for example by incrementing its value by one. The authors discuss a few other rules that could be used instead to mark a cell and navigate to the next one, but they all exhibit similar

performance in terms of exploration time. Hence, we select the above variant of the Ants algorithm (move to the least visited cell) as a basis for comparison. The authors provide a proof that the agents will eventually cover the entire terrain (provided that it is not disconnected by wall cells).

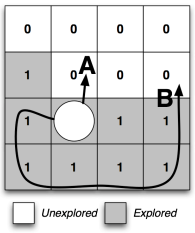


Figure 2: Impact of a small error.

The first advantage of the algorithm is its simplicity: agents do not require memory or radio communication, but only one-cell lookahead. Since they are easy to build, many of them can be used to shorten the coverage process. Secondly, there is no map stored inside the agents: if one of them is relocated (accidentally or on purpose) it will not even realise it and it will continue to do its work as if nothing happened. This means that the whole system is flexible and fault tolerant, and the area can be covered even if some markings or agents are lost. At the sensor of each cell, we only need to store an integer counting the number of times that agents have visited the cell. When the number of times exceeds a threshold, the counter is reset to 0.

The main limitation of the Ants algorithm is that agents do not know when the exploration is terminated, and they continue the exploration phase until they run out of energy. Thus, this approach is not suitable in an emergency scenario, in which the primary consideration is to cover the overall area as soon as possible, and be notified immediately after the task is completed. A further drawback of the algorithm is the limited collaboration among agents. For example, in scenarios with many rooms most of the agents tend to sweep the first few rooms repeatedly, while only a few of them venture to explore new areas.

4. EVALUATION OF ANTS

In this section we illustrate how localization errors impact the behavior of the Ants algorithm and evaluate the algorithm's performance, with and without errors, in a variety of scenarios. In Section 2, we summarized two types of localization errors reported in previous work [3]: 1) Agents tend to introduce small errors in the locations of sensors they identify in their vicinity; these errors are not big enough to impact the behavior of Ants. The reason is that agents see sensors in slightly different locations, but in the correct cells where sensors are actually placed. 2) Agents sometimes completely fail to identify some of the sensors in their vicinity - this type of localization error is referred to as sensor detection error. Although the percentage of missed sensors is reported to be low (5.56%), it significantly affects the performance of Ants. This is illustrated via an example, and quantitatively measured with simulation experiments.

Figure 2 shows the impact of a sensor detection error. In the absence of errors, the agent at the center of the area would choose to explore the cell immediately north of it, by choosing path A. But if it wrongly believes that the cell north of it is not occupied by a sensor and thus is an obstacle or a wall, it will choose to explore first the cell on its left, by choosing path B. This error will bring the agent away from the main front of exploration, causing it to follow a long path of already explored cells, before it can get back to exploring new cells. Note that path B, which is the effect of one detection error, is seven times longer than the regular path A that would be followed in the absence of errors.

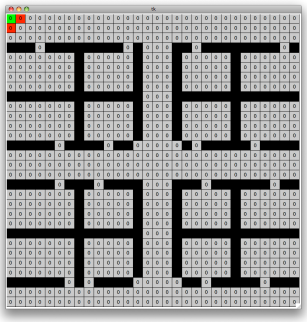


Figure 3: Example of area used during the simulations.

Our next step is to quantify the impact of sensor detection errors on the performance of Ants in a variety of scenarios. To this end, we simulated the Ants algorithm with and without errors and ran a number of simulations varying the number of agents, the size of the area and the number of rooms. The simulations were performed on automatically generated environments representing office-like scenarios (see Figure 3 for an example) with a default area size of 30x30 cells and 4x4 rooms in them. The positions of doors and walls were changed randomly during the experiments, while the default number of agents was 20. For each experiment, we computed the total time necessary to explore the whole area (every cell was traversed at least once by one of the agents). Each point of the graph is the average time of 100 different runs, and is plotted with the corresponding standard deviation bar.

Figure 4 shows the performance of Ants with and without localization errors as we increase the number of agents. Note that with one agent, sensor detection errors actually double the exploration time. Increasing the number of agents helps in reducing the negative effect of these errors, but even with the maximum number of agents the difference remains noticeable. These findings show that simplifying assumptions about the ability of agents to perfectly detect sensors in their vicinity lead to results that are very different from reality.

Figure 5 shows the performance of Ants with and without sensor detection errors as we increase the size of the area. Observe that the negative impact of these errors becomes more pronounced in larger areas. We believe that this is due to the fact that in large areas, a sensor detection error can lead agents to take much long detour paths before returning to the exploration front.

Figure 6 shows the impact of sensor detection errors as we increase the number of rooms, whilst keeping the size of the area constant. Observe that the impact of these errors decreases as we increase the number of rooms. This is due to

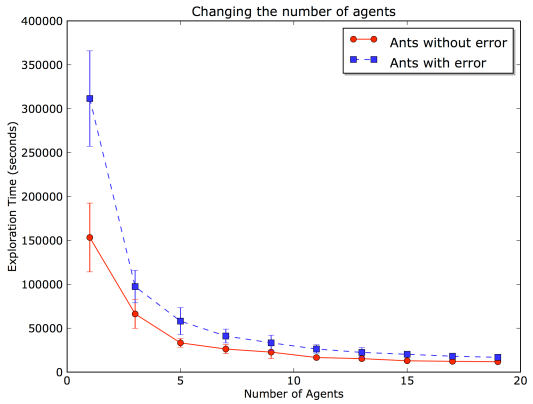


Figure 4: Effect of changing the number of agents.

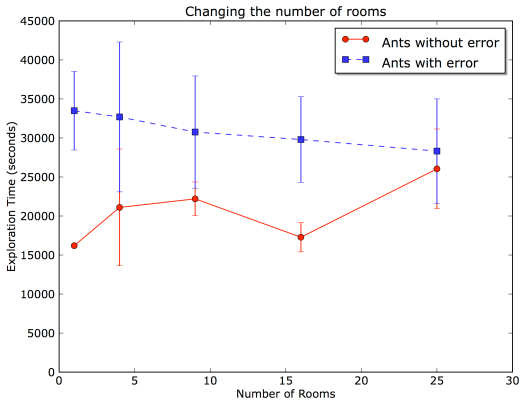


Figure 6: Effect of changing the number of rooms.

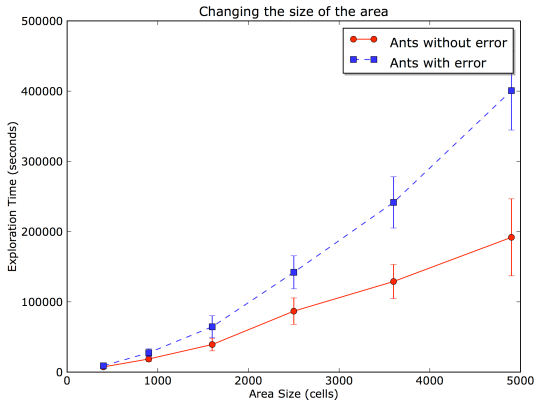


Figure 5: Effect of changing the size of the area.

the fact that with more rooms, accessible areas of the map are narrower, agents are more constrained in their movements, and have smaller chances of following long detour paths as a result of a sensor detection error.

5. CONCLUSIONS

In this paper, we studied the impact of localization errors on the performance of the Ants algorithm. We distinguished two types of errors: i) inaccuracies in determining the exact location of detected sensors wrt the agent's current position, and ii) complete failure to detect and localize sensons. We showed that small errors in locating sensors are not critical, but completely failing to detect sensors can significantly slow down the exploration process. The impact of failing to detect sensors is more pronounced in scenarios where few agents are used to explore large areas with few rooms.

Acknowledgments

Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3003. The U.S. Government is authorized to reproduce and distribute reprints for Government purpose notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

6. REFERENCES

[1] M. Batalin, G. Sukhatme, and M. Hattig. Mobile Robot Navigation using a Sensor Network. In *ICRA04*, pages 636–642. IEEE Press, April 2004.

[2] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Video-rate recognition and localization for wearable cameras. In *BMVC07*, pages 1100–1109, September 2007.

[3] E. Ferranti and N. Trigoni. Practical issues in deploying mobile agents to explore a sensor-instrumented environment. Technical Report RR-09-02, Oxford University Computing Laboratory, February 2009.

[4] E. Ferranti, N. Trigoni, and M. Levene. Brick&Mortar: An On-Line Multi-Agent Exploration Algorithm. In *ICRA07*, pages 761–767. IEEE Press, April 2007.

[5] S. S. Ghidary, T. Tani, T. Takamori, and M. Hattori. A new Home Robot Positioning System (HRPS) using IR switched multi ultrasonic sensors. In *SMC99*. IEEE Press, October 1999.

[6] S. Kataoka and K. Atagi. Preventing IR interference between infrared waves emitted by high-frequency fluorescent lighting systems and infrared remote controls. *IEEE transactions on industry applications*, 33(1):239–245, January/February 1997.

[7] I. Kelly and A. Martinoli. A scalable, on-board localisation and communication system for indoor multi-robot experiments. *Sensor Review*, 24(2):167–179, January 2004.

[8] N. Kirchner and T. Furukawa. Infrared Localisation for Indoor UAVs. In *ICST05*, pages 60–65, November 2005.

[9] J. H. Lim and J. J. Leonard. Mobile Robot Relocation from Echolocation Constraints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):1035–1041, September 2000.

[10] A. C. Rice. Dependable Systems for Sentient Computing. *PhD Thesis, University of Cambridge*, 2007.

[11] J. Svennebring and S. Koenig. Building Terrain-Covering Ant Robots: A Feasibility Study. *Autonomous Robots*, 16(3):313–332, May 2004.

A multi-layered semantics-ready sensor architecture

Thomas Harman
Department of Computer Science
University of Bath
Bath, BA2 7AY, UK
t.p.harman@bath.ac.uk

Julian Padget
Department of Computer Science
University of Bath
Bath, BA2 7AY, UK
jap@cs.bath.ac.uk

Martijn Warnier
Department of Computer Science
Vrije Universiteit
Amsterdam, NL
warnier@cs.vu.nl

ABSTRACT

There is an intrinsic tension between sensor systems and multi-agent systems that comes down to the trade-off between cost and value:[1] the agents want as much knowledge of their environment as possible, while the sensors are rightly protective of their often very limited resources that enable sensing and transmission. The architecture and implementation that we present here aims to provide sufficient flexibility for the cohabitation of both classes—where class is a relative term—of components through a policy-aware framework that permits the construction of "sensors" at whatever level of abstraction is regarded as appropriate by the designer. There are many sensor architectures available, nevertheless we believe there is some novelty in the approach we present here in terms of systems engineering, deriving mainly from the principled design of Agentscape upon which we are building, such that the notable features are modularity—there is a high degree of separation of concerns—extensibility—leading to relative ease of integration of different sensor infrastructures—and scalability—as a result of the distributed architecture that Agentscape provides. In addition, our choice of RDF as the initial database format has positive practical implications for the integration of supported sensor networks with semantic processing mechanisms.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Multiagent systems*

General Terms

Design, Management, Measurement

Keywords

multi-agent systems, sensor networks, middleware

1. INTRODUCTION

[1]Some readers may recall the criticism of Lisp programmers knowing the value of everything and the cost of nothing: the same might be said of MAS.

Typical middleware systems for sensor networks assume that sensors are homogeneous and have limited resources to consume. As others have observed [2], this assumption seems more and more misplaced. New types of sensors that use different technologies appear all the time and combination and aggregation of data from different sensors can be very useful.

Multi-Agent Systems (MAS) form one of the most promising contributions to the sensor networks domain. The ability to reason about their environment and use sensor data in a autonomous self-aware manner makes the agents especially suitable for sensor based applications. However, most of the current research in this area focuses either on agents embedded in sensors [1] or on more high-level applications such as routing information between sensors or sensor sensing strategies [5]—typically using some out-of-the-box sensor middleware and building the agent-based application on top.

This paper proposes a new multi-layered semantics-ready sensor architecture that addresses both issues. The main idea is to extend a multi-agent platform, AgentScape, with generic services for accessing sensors and a generic database service for storing the sensor data. Both services form part of the middleware and support multiple sensors types and multiple database types, that are in turn freely accessible to user agents. The architecture as a whole provides a uniform agent-based sensor middleware to support a wide range of sensor based research. In particular, it allows the programmer to focus on the detail programming of particular sensor types, but also the high level programming of sensor based applications [8], including the creation of virtual sensors, that synthesize signals from arbitrary combinations of other (stored) sensor data. AgentScape's connection to web-services and thus grid computing forms an additional motivation for this work, enabling a single platform to span the gamut of computing applications from sensor data-collection through to the processing of large data-sets.

The remainder of this paper is organized as follows: the next section introduces AgentScape, Section 3 describes the architecture of the sensor support components and Section 4 outlines our initial demonstrator and application domains. The paper ends with a discussion and conclusions.

2. AGENTSCAPE

The multi-agent platform AgentScape supports agents as autonomous processes. A uniform middleware layer provides an agent run-time that is available at numerous heterogeneous platforms.

Within AgentScape, *agents* are active entities that reside within *locations*, and *services* are external software systems accessed by

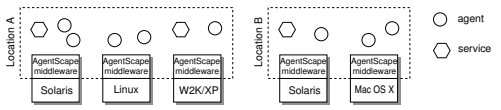


Figure 1: Conceptual model of AgentScape middleware

agents hosted by the AgentScape middleware (see Figure 1). Agents in AgentScape can communicate with other agents and can access services. Agents may migrate from one location to another.

All agent operations are modulo authorization and security precautions. For example, an agent may have to have the appropriate credentials (ownership, authorization, access to resources, and so on) to access a specific service, possibly for a limited time period.

The guiding principle in the design of the AgentScape middleware has been to develop a minimal but sufficient open agent platform that can be extended to incorporate new functionality or adopt (new) standards into the platform. This design principle has resulted in a multi-layered architecture with (i) a small *middleware kernel*, called the AgentScape Operating System (AOS) kernel, that implements basic mechanisms, (ii) high-level *middleware services* that implement agent platform specific functionality and policies (see Figure 2) and (iii) external directory services. This approach simplified the design of the kernel and has made it less vulnerable to errors or improper functioning. The current set of middleware services includes agent servers, host managers, location managers, a look-up service and a web service gateway.

AgentScape's middleware services implement the agent specific functionality. The current set of middleware services include:

- **Location Manager:** Every *location* has a Location Manager, which runs on one of the hosts within that location. This process manages that location's hosts. Locations typically are formed by hosts that belong to one single administrative domain.
- **Host Manager:** Every host (typically, one physical machine) runs a Host Manager. This process is responsible for managing the middleware components running on that host. It also regulates and guards access to its resources.
- **Agent Server:** An Agent-Server provides a run-time environment for agents. Each host can run one or more Agent-Servers to host agents supporting e.g. different programming languages.
- **Web Service Gateway:** The Web Service Gateway enables agents to communicate with web services using the SOAP/ XML protocol [6].
- **Look-up Server:** This external (to the middleware) service keeps track of the current location of agents. Strictly speaking, this service is not part of the AgentScape middleware as it can be run as a stand-alone application. Two versions exist, a centralized, unsecured version and a decentralized secured one.

Agent servers provide agent access to the AgentScape middleware (see Figure 2). AgentScape supports multiple (simultaneous) code bases through the provision of multiple agent servers, at least one per code base. From a security perspective, it is important to note that agent servers 'sandbox' agents.

The location manager is the coordinating entity in an AgentScape location (managing one or more hosts in its location). Note that for

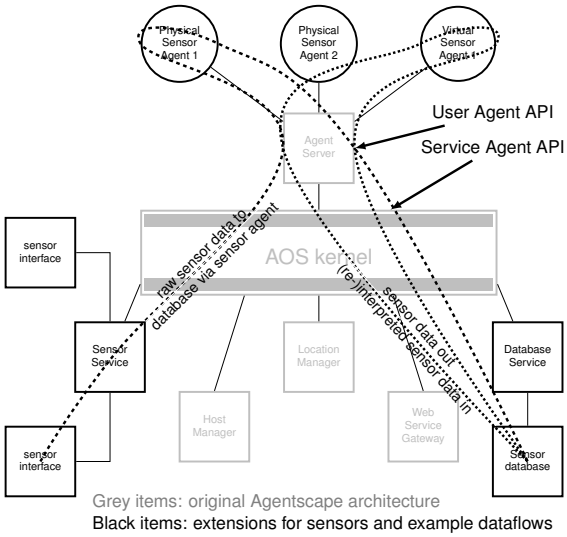


Figure 2: Agentscape sensor architecture

fault tolerance a location manager may be replicated. Agent creation, migration, and all policy related issues relevant in the context of a location, are managed or coordinated by the location manager. The host manager manages and coordinates activities on a host. The host manager acts as the local representative of the location manager, but is also responsible for local (at the host) resource access and management. The policies and mechanisms of the location and host manager infrastructure are based on negotiation and service level agreements [4].

3. SENSOR ARCHITECTURE

Scalability is the fundamental contribution that the Agentscape middleware has the potential to bring to sensor architectures. This may not sound entirely novel, so some justification is due. As Section 2 has described, Agentscape is a middleware and one which has been designed for modularity and extensibility, so that the designer can choose how sophisticated to make the agents: they can be ordinary Java programs communicating through the Agentscape API or (for example) BDI agents implemented via the Jason library [9]. Other agent architectures may be interfaced in a similar way. More important, for the purpose of this article, is the capacity for adding interfaces—in effect stretching the middleware both upwards, towards grid resources (not covered further here) and downwards, towards mobile platforms (not covered further here) and sensors.

Our goals in extending Agentscape to work with sensors are:

1. To provide a generic sensor interface for agents to access, for example, bluetooth, ZigBee, RFID
2. To provide a generic database interface for agents to store sensor data, fine-tuned by sensor policies that control how much and what data is stored.
3. To allow agents to use the sensor data in a uniform manner and publish it or use it in other ways, for example in combination with web-services.

3.1 Sensor and Database Services

AgentScape's modular design enables the straightforward addition of new services. To support the construction of with sensor networks two services have been added: (i) a generic sensor service and (ii) a database service. Together, these services provide a uniform interface for agents.

The sensor service acts as a proxy and its purpose is to provide access to multiple sensor infrastructures, so that subsequently different types of physical sensors can be registered with the sensor service. The sensor service provides a minimal uniform interface to agents with the possibility of additional functionality on a per sensor type basis. This makes it relatively straightforward to re-use agents with different sensor types.

Similarly, the database service provides a uniform interface that can access different database back-ends. The database service also enforces the policies that agents can define per sensor type or, if necessary, per sensor. Policies are further detailed in Section 3.2 below. Figure 2 shows how the sensor architecture extensions are integrated into AgentScape.

3.2 Sensor Agents

Agents can access individual sensors through the sensor service. After the agent provides the service with a URI of the sensor, an interface belonging to the specific sensor type, including the generic sensor interface, is returned. Thus, sensors are individually accessed on a per URI basis.

The agent can at this point chose to use the sensor data directly, for example by publishing it on a web-site, or it can store the sensor data in a database (see the example data-flows in Figure 2. If the data is stored in a database, using the database service, an additional *sensor policy* is required. This policy states how much data of one sensor instance is stored and/or for what time period. Such policies might be specified, for example, as the most recent 10MB of a stream, the last 100 samples, or all data generated over a week by one sensor. In addition it is possible to store all data generated by a sensor indefinitely, though, depending on the sensor type, this can be a large amount of data. The *specification* of this policy is the responsibility of the physical or logical sensor agent (see Figure 2), while the *implementation* of the policy is the responsibility of the relevant database.

In other circumstances, it may be desirable to collect samples over a period of time, in which case it is possible to set up a direct connection between the sensor service and database service. This speeds-up the data storing process by circumventing the agent, once the connection between both services is established. At some later time the agent can send a 'stop' message to the sensor service to halt data-collection for a specific sensor.

3.3 Scalable Data Collection and Usage

A key attraction for extending AgentScape with sensor-network middleware is the scalability that is an intrinsic aspect of AgentScape. The database service makes this point especially clear. This service acts as a front-end to various databases and typically, one database service is instantiated per AgentScape location. However, one AgentScape location can also run multiple databases, both by type and instance. Agents can also access the database services that run at other locations. In particular, this means that one database can be used for multiple sensor networks, deployed around the world. Or, conversely, each location may use its own database service, possibly interfacing with multiple databases. Additionally, data may be aggregated with some delay at one location, combining data from numerous (physical) locations.

3.4 Technical details

At this stage, entirely because of device availability, we have implemented only a bluetooth interface, although the two devices chosen have different characteristics, in that one generates a stream of data, while the other supplies data on request, thus exercising two of the standard modes in which sensors typically supply data. The two physical devices in question are a Wii-mote and a GPS.

Likewise, at this point in development, we had to make a choice for sensor data storage and have adopted the JRDF package[3] that provides an API to a triple store, deriving features from Jena and Sesame, amongst others. The primary motivations for this choice are (i) the flexibility afforded by the RDF triple structure and (ii) the fact that a triple store naturally accommodates semantic annotation. In this way we believe we are putting minimal constraints on downstream consumers (agents) of the data collected.

Of course, it is to be expected that the platform can support the connection of more than one sensor of the same kind, so for this reason we identify each data source by an unique URI. Consequently, the data that is stored in the triple store takes the form of:

```
# Wii-mote triples
(uri:wiimote1, hasWiiData, uri:data1)
(uri:data1, hasDate, <date>)
(uri:data1, hasButtonPressed, buttonX)
# GPS triples
(uri:gpsSensor1, hasGpsData, uri:data1)
(uri:data1, hasLatitude,<degrees+minutes+seconds>)
```

As illustrated in Figure 2, one dataflow passes from the sensor to the generic sensor interface, through the Agentscape kernel and the agent server to be delivered to the user-level sensor agent that is responsible for the particular sensor. A standard behaviour is then for the sensor agent to store that data, via the Agentscape kernel and the generic database interface to the triple store. In this way, raw sensor data is captured in the short term for subsequent processing. Clearly in the case of a source like the Wii-mote, the data needs cleaning in order to identify a smooth gestural path (for example). Whether this kind of task is the responsibility of the sensor agent itself or is delegated to a downstream "smoothing" agent is an issue for the programmer to decide: the mechanisms are available either to augment the sensor agent and only store smoothed data, or alternatively another user agent—see the virtual sensor data path in Figure 2—may subscribe to the Wii-mote feed and then publish a smoothed feed to a database—perhaps the one from which it obtained the data or another, as desired.

It is often impractical to keep data for an unlimited period in the triple store. If long-term preservation is required, then alternative measures must be taken, but in many circumstances, and almost certainly in the case of the two devices with which we are currently working, data need only be retained in the short-term. This raises the question of where that decision is made and where that decision is implemented. We regard data retention as a policy issue as far as the sensor agent is concerned: it is responsible for specifying for how long (time period), or how much of (sample size) the data shall be retained. But policy implementation is a matter for the storage mechanism and so it is the particular database interface that carries out the necessary deletion operation.

3.5 Data semantics

A relational database would have been an obvious choice for data storage and has indeed been selected by several of the published sensor architectures. However, we felt that although it would be straightforward to develop (or replicate) a schema to fit the purpose of sensor data collection and querying, it would almost certainly compromise consumers of the data, whose intentions we cannot foresee. Clearly such consumers could create new schema for their needs in the same database or extract data and store it another database structured for their purpose. None of these scenarios is ruled out, but the one-size-fits-all aspect of RDF means that consumer agents need do nothing more than query using languages that are seeing increasing up-take (in this case SPARQL) and assert new triples, perhaps defining their own predicates and new object datatypes. Furthermore, by choosing this representation, arbitrary semantic annotations are facilitated, as well as enabling interaction with external semantic web tools.

4. USING SENSOR DATA

Using AgentScape as a uniform means to *collect* sensor data provides some clear advantages, as we have outlined above. A further advantage is that (other) agents can directly *access* the collected data. Where the previous section focused mostly on the underlying middleware infrastructure, this section explores the possibilities for agent-based usage of sensor data.

Agents can access the database service to obtain sensor data. This sensor data can then be used directly in an agent-based application, published via a web service, combined with other web-services to form a new web based application [10], or combined with other sensor data to realize outputs from a new virtual sensor.

This is in particular relevant for the ALIVE project [7]. ALIVE aims to apply organizational theory to the design and implementation of software systems. The main focus of the project is to create complex systems based on the composition of (existing) services, through the addition of levels of abstraction. The advantage of added levels of abstraction to the design process of systems is two-fold: (i) it is often more intuitive to think in organizational structures and interactions when designing complex interactions for services, and the addition of the layers of abstraction allows for a gradual (fluid) transition from the system as foreseen to the actual implementation; (ii) when changes happen in the environment (for example, specific services become unavailable) the added levels of abstraction act as an explicit representation of the conceptual steps made at design, thus giving additional information on why certain interactions are as they are, that enables the system to dynamically cope with the changes. A sensor network enabled AgentScape can be regarded as a (simplified) version of such a system. In this view the low-level sensor framework can be seen as a first abstraction layer, on top of which reside the sensor and database services. The agents form the next level of abstraction and the actual, possibly web service based, application forms the top level in this view.

5. DISCUSSION AND CONCLUSIONS

This paper describes a multi-layered semantics-ready sensor architecture based on the AgentScape middleware. The main benefits of the proposed system are (i) a generic sensor interface for agents to access, (ii) a generic database interface through which agents may store sensor data, (iii) means for agents to access and add sensor data in a uniform manner and (iv) a scalable framework for accessing large-scale sensor networks, over different physical locations.

The system is currently in the implementation stage. Basic support for two bluetooth type sensors: a GPS and a Nintendo Wii-mote (`www.nintendo.com`) and one database back-end, a RDF store, has been completed. In the short term we will be developing a ZigBee interface for the purpose of interacting with energy monitoring sensors and an IEEE802.15.4 interface to work with high frequency structural monitoring sensors. RFID is in our medium term plans, which in conjunction with the J2ME Agentscape deployment currently under development, will enable us to collect data via a mobile device and either store locally or propagate to other platforms over networks, when/where connectivity permits. On the storage side we will be adding a conventional relational database to the database service agent.

Acknowledgements

This work is partially supported by the ALIVE project (FP7-IST-215890, `http://www.ist-alive.eu`) and the NLnet Foundation (`http://www.nlnet.nl`).

6. REFERENCES

[1] C. Fok, G. Roman, and C. Lu. Mobile agent middleware for sensor networks: an application case study. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 382–387, 2005.

[2] K. Henricksen and R. Robinson. A survey of middleware for sensor networks: State-of-the-art and future directions. *Proceedings of the international workshop on Middleware for sensor networks.*, November 28-28, 2006.

[3] Jrdf. `jrdf.sourceforge.net`, retrieved 20090215. Java Resource Description Framework API for graphs, object persistence and querying.

[4] D. G. A. Mobach, B. J. Overeinder, and F. M. T. Brazier. WS-Agreement based resource negotiation framework for mobile agents. *Scalable Computing: Practice and Experience*, 7(1):23–36, 2006.

[5] M.Vinyals, J. Rodriguez-Aguilar, and J. Cerquides. A survey on sensor networks from a multi agent perspective. *2th International Workshop on Agent Technology for Sensor Networks (ATSN-08)*, 2008.

[6] B. J. Overeinder, P. D. Verkaik, and F. M. T. Brazier. Web service access management for integration with agent systems. In *Proceedings of 23rd Annual ACM Symposium on Applied Computing, Mobile Agents and Systems Track*, 2008.

[7] T. B. Quillinan, F. M. T. Brazier, H. M. Aldewereld, F. Dignum, V. Dignum, L. Penserini, and N. J. E. Wijngaards. Developing Agent-based Organizational Models for Crisis Management. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (Industrial Track)*, May 2009.

[8] R. Sugihara and R. K. Gupta. Programming models for sensor networks: A survey. *ACM Trans. Sen. Netw.*, 4(2):1–29, 2008.

[9] R. C. van het Schip. Integrating Jason into AgentScape - Joining BDI-theory with Agent Technology practice. Master's thesis, Vrije Universiteit Amsterdam, October 2008.

[10] S. van Splunter, F. M. T. Brazier, J. Padget, and O. Rana. Dynamic service reconfiguration and enactment using an open matching architecure. In *Proceedings of the International Conference on Agents and Artificial Intelligence, Porto, Portugal*, January 2009.

Information Agents for Autonomous Acquisition of Sensor Network Data

A. Rogers and N. R. Jennings
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{acr,nrj}@ecs.soton.ac.uk

M. A. Osborne and S. J. Roberts
Department of Engineering Science
University of Oxford
Oxford, OX1 3PJ, UK
{mosb,sjrob}@robots.ox.ac.uk

ABSTRACT

In this paper, we describe an information agent that can autonomously acquire sensor readings from environmental sensor networks (deciding when and which sensor to acquire readings from at any time). Moreover, this agent can perform a range of information processing tasks including modelling the accuracy of the sensor readings, predicting the value of missing sensor readings, and predicting how the monitored environmental parameters will evolve into the future. We describe how our agent uses an iterative formulation of a multi-output Gaussian process to build a probabilistic model of the environmental parameters being measured by local sensors, and the correlations and delays that exist between them. We validate our approach using data collected from a network of weather sensors located on the south coast of England.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence*

General Terms

Algorithms, Experimentation

Keywords

information agent, sensor networks, Gaussian processes

1. INTRODUCTION

Sensor networks have recently generated a great deal of research interest within the computer and physical sciences. Their use for the scientific monitoring of remote and hostile environments is increasingly common-place, and recent research has addressed how the information from such sensor networks can be made available to multiple users directly through standard web interfaces (see [4] for a review of such environmental sensor networks). Such systems pose a number of novel challenges, not least the need for self-describing data formats, and standard protocols such that sensors can advertise their existence and capabilities to potential users of the network.

However, more significantly for us, many of the information processing tasks that would previously have been performed by the owner or single user of an environmental sensor network (such as detecting faulty sensors, fusing noisy measurements from several sensors, and deciding how frequently readings should be taken) are now delegated to the multiple different users of the system, all of whom may have different goals and may be using sensor readings for very different tasks. Furthermore, the open nature of the network (in which additional sensors may be deployed at any time, and existing sensors may be removed, repositioned or updated) means that these users may have only limited knowledge of the precise location, capabilities, reliability, and accuracy of each sensor.

Thus, there is a clear need for *information agents* that are capable of autonomously performing the acquisition and processing of information from such sensor networks. Given this, in this paper, we describe our work developing just such an agent. This agent uses a novel iterative formulation of a multi-output Gaussian process (described in more detail in [6]) to build a probabilistic model of the environmental parameters being measured by local sensors, and then uses this model to perform a number of information processing tasks including: modelling the accuracy of the sensor readings, predicting the value of missing sensor readings, predicting how the monitored environmental parameters will evolve in the near future, and performing active sampling by deciding when and which sensor to acquire readings from. We use a network of weather sensors on the south coast of England to validate this approach, and we illustrate its effectiveness by benchmarking against the more conventional single-output Gaussian processes that models each sensor independently.

2. INFORMATION PROCESSING

As discussed above, we require that our information agent be able to autonomously perform data acquisition and information processing despite having only limited specific knowledge of each sensor (e.g. their precise location, reliability, and accuracy). To this end, we require that it explicitly represent:

1. The noise in the sensor readings, and hence, the uncertainty in the environmental parameter being measured; sensor readings will always include measurement noise, and thus there will always be uncertainty in the agent's world picture.

2. The correlations or delays that exist between sensor readings; sensors that are close to one another, or in similar environments, will tend to make similar readings, while many physical processes involving moving fields (such as the movement of weather fronts) will induce delays and correlations between sensors.

We then require that the information agent use this explicit representation in order to perform:

1. Efficient active sampling by selecting when to take a reading, and which sensor to read from, such that the minimum number of sensor readings are used to maintain an agent's world uncertainty below a specified threshold (or minimising uncertainty given a constrained number of sensor readings).

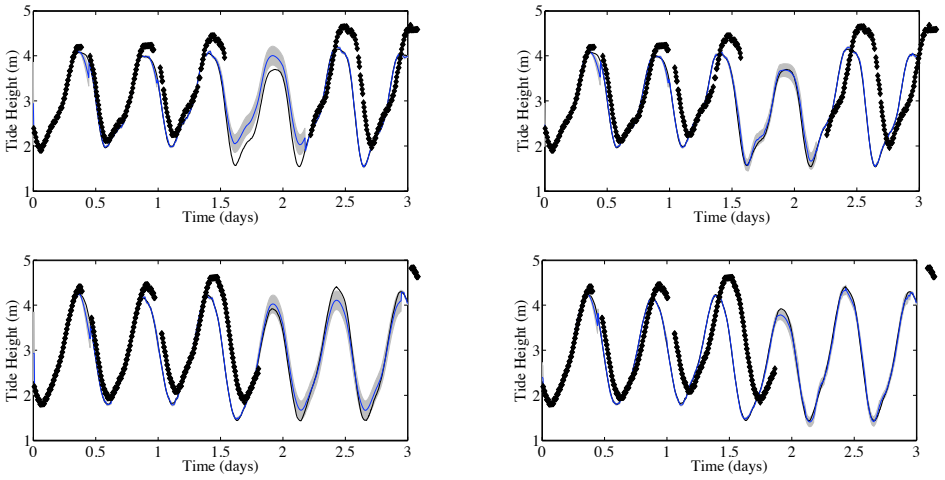


Figure 1: Prediction and regression of tide height data for (a) independent and (b) multi-output Gaussian processes.

2. Regression and prediction of sensor readings; interpolating between sensor readings to predict the value of missing sensors (i.e. sensors that have failed or are unavailable through network outages), and performing short term prediction of sensor readings in order to support decision making.

More precisely, we consider a multivariate regression problem in which we have $m = 1 \dots M$ environmental parameters of interest (such as air temperature, wind speed or direction specified at different sensor locations) represented by the space $\mathcal{Y} = \mathbb{R}^M$. Given a set of N sensor readings, $D = \{(t_1, \mathbf{y}_1), \dots, (t_N, \mathbf{y}_N)\}$, where $\mathbf{y}_i$ may be fully or partially specified (corresponding to observation of all, or some subset of the environmental parameters), we attempt to infer the value of $\mathbf{y} = \{y^1, \dots, y^M\} \in \mathcal{Y}$ at any time t.

3. GAUSSIAN PROCESSES

Gaussian processes (GPs) present a principled approach to addressing multivariate regression problems of the form described above [9]. When using a GP, we assign a multivariate Gaussian prior distribution over the outputs of the regression problem and then produce analytic posterior distributions for outputs of interest, conditional on whatever sensor readings have been collected. Crucially, the posterior distributions are also Gaussian, with a predictive mean, and a variance that explicitly represents uncertainty.

GP regression has a long history of use within geophysics and geospatial statistics (where the process is known as kriging [2]), but has only recently been applied within sensor networks. Examples here include the use of GPs to represent spatial correlations between sensors in order that they may be positioned to maximise mutual information [5], and the use of multi-variate Gaussians to represent correlations between different sensors and sensor types for energy efficient querying of a sensor network [3].

Our work differs from this earlier work in that we use a novel iterative formalism of a multi-output GP to represent both temporal correlations in readings from a single sensor, and correlations and delays between multiple sensors. Space precludes a full description of this algorithm (see [6] for the full details), however we describe the intuition behind this algorithm here.

3.1 Covariance Functions

The covariance matrix of the GP informs it of how different outputs are related to one another. To generate this matrix, we use *covariance functions*. Fortunately, there exist a wide variety of functions that can serve in this purpose [1], all of which can then be combined and modified in a multitude of ways. This gives us a great deal of flexibility in our modelling of functions, and covariance functions can be found to model periodicity, delay, noise and long-term drifts.

More specifically, here we represent the covariance matrix by the Hadamard product of a covariance function over time alone, and a covariance function over environmental parameter labels alone, such that:

$$K([m,t],[m',t']) = C(m,m')K(t - d_m, t' - d_{m'}) \quad (1)$$

where $\mathbf{d}$ represent delays between environmental parameters. Assuming no prior knowledge of what the correlations over environmental parameters are, we use the completely general *spherical parameterisation*, $\mathbf{s}$, such that:

$$C(m,m') = \operatorname{diag}(\mathbf{l})\mathbf{s}^T\mathbf{s}\operatorname{diag}(\mathbf{l}) \quad (2)$$

where $\mathbf{l}$ gives represents an intuitive length scale for each environmental parameter, and $\mathbf{s}^T\mathbf{s}$ is the correlation matrix [7]. Similarly, we can represent correlations over time with a wide variety of covariance functions, incorporating as much domain knowledge as we have. However, in general, we find that the additive combination of a periodic term and a disturbance term performs well on a wide range of data sets, and we represent both using the standard Matérn class (with $\nu = 5/2$), given by:

$$K(t,t') = h^2\left(1 + \sqrt{5}r + \frac{5r^2}{3}\right)\exp\left(-\sqrt{5}r\right) \quad (3)$$

where $r = \left|\frac{t-t'}{w}\right|$e for non-periodic terms, and $r = \sin\pi\left|\frac{t-t'}{w}\right|$ for periodic ones.

3.2 Marginalisation

In order to use the GP for regression or prediction, the correlation hyperparameters (i.e. $\mathbf{l}$, $\mathbf{s}$ and $\mathbf{d}$), along with others such as the periods and amplitudes of each covariance term (i.e. h and w), must

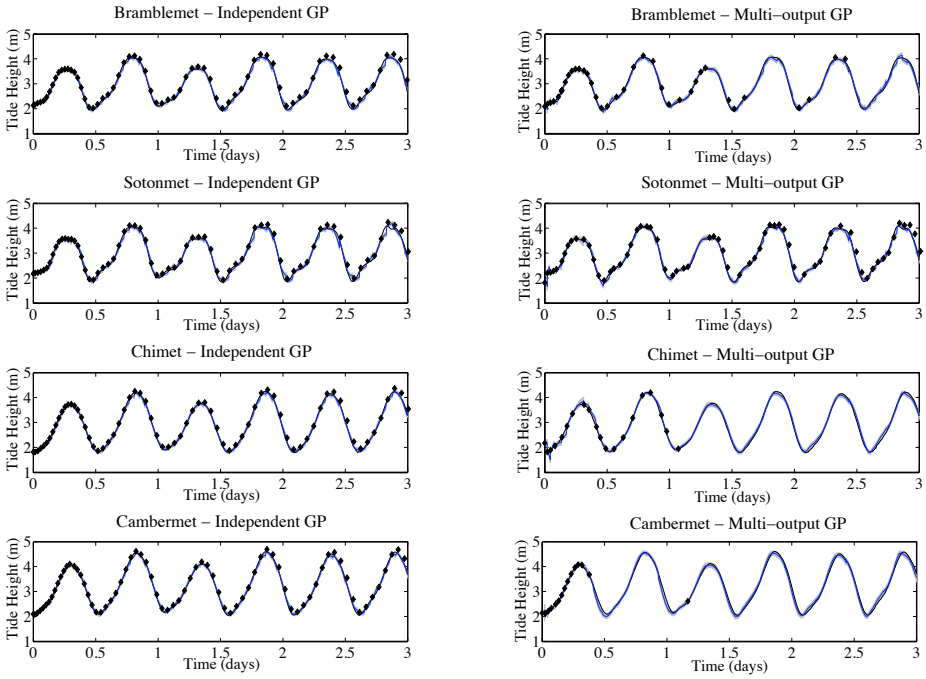


Figure 2: Comparison of active sampling of tide data using (a) independent and (b) multi-output Gaussian processes.

be marginalised from our model. To each we assign an independent Gaussian or log Gaussian prior distribution (if the hyperparameter is strictly positive). We then use *Bayesian Monte Carlo* to numerically resolve the non-analytic marginalisation integrals [8]. This essentially involves the assignation of another GP to the likelihood of the data as a function of the covariance hyperparameters. We evaluate predictions for a set of sample hyperparameters, and use this second GP to infer what predictions for other possible hyperparameters, producing a posterior for our marginalised predictions.

3.3 Iterative Formulation

Gaussian processes have traditionally been used largely for regression, producing predictions for a fixed set of data. However, in our setting both the environmental parameters of interest and the data available are constantly updated. In order to manage this situation, we employ a novel iterative formulation of a GP, which allows us to efficiently update our predictions upon the receipt of new data.

Similarly, we allow the GP to discard old data once it judges it sufficiently uninformative, hence reducing memory usage and computational requirements. In this, it is guided by the uncertainty in its predictions; the GP will retain only as much data as necessary to achieve a pre-specified degree of accuracy (a principled form of 'windowing'). These features give us an efficient on-line algorithm.

3.4 Active Data Selection

Our algorithm is also able to perform active data selection, whereby the GP decides for itself which observations it should take. In this, we use once again the uncertainty in our predictions as a measure of utility. For a GP, this uncertainty increases monotonically in the absence of new data – once it grows to our pre-specified threshold, our algorithm takes a sample in order to reduce it once again. The algorithm can also decide which observation to make at this time, by determining which sensor will allow it the longest period of grace until it would be forced to observe again. Hence we maintain our uncertainty below a specified threshold, while taking as few observations as possible.

4. ILLUSTRATIVE EVALUATION

To illustrate the effectiveness of our GP formalism, we have used a network of tide and weather sensors located on the south coast of England (see `www.bramblemet.co.uk`). Such weather sensors are attractive since they exhibit challenging correlations and delays, and they are subject to network outages that generate real instances of missing sensor readings on which we can evaluate our information agent. We compare our multi-output GP formalism against conventional independent GPs in which each environmental parameter is modeled separately (i.e. correlations between these parameters are ignored). We present results using tide height measurements since this data set demonstrates the ability of the GP to learn and predict periodic behaviour, and more importantly, because this particular data set contains an interesting period in which extreme weather conditions cause both an unexpectedly low tide and a failure of the wireless connection between the sensor and the shore that prevents our information agent acquiring sensor readings (see [6] for more results using air temperature measurements).

4.1 Regression and Prediction

Figure 1 illustrates the efficacy of our GP formalism in this setting. We plot the sensor readings acquired by the information agent (shown as markers), the mean and standard deviation of the GP prediction (shown as a solid line with the standard deviation shown as shading), and the true fine-grained sensor readings (shown as bold) that were downloaded directly from the sensor (rather than through the web site) after the event. Note that we present just two sensors

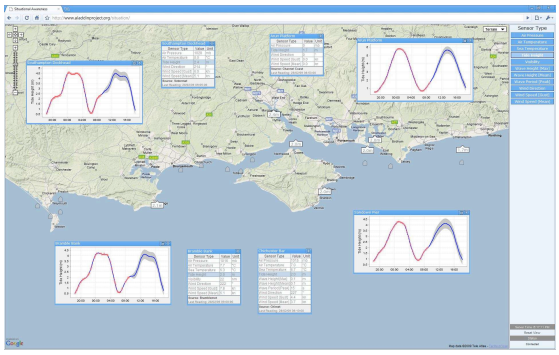


Figure 3: Live implementation of our information agent available at www.aladdinproject.org/situation/.

for reasons of space, but we use readings from all four sensors in order to perform regression.

We consider the performance of our multi-output GP formalism when the Bramblemet sensor drops out at $t = 1.45$ days. In this case, note that the independent GP predictions quite reasonably predicts that the tide will continue to do more or less what it has seen before, and predicts the same periodicity it has observed in the past. However, the GP can achieve better results if it is allowed to benefit from the knowledge of the other sensor's readings during this interval of missing data. Thus, in the case of the multi-output GP, by $t = 1.45$ days, the GP has successfully determined that the sensors are all very strongly correlated. Hence, when it sees an unexpected low tide in the Chimet sensor data (caused by the strong Northerly wind), these correlations lead it to infer a similarly low tide in the Bramblemet reading, and produces significantly more accurate predictions. Exactly the same effect is seen in the later predictions of the Chimet tide height, where the multi-output GP predictions use observations from the other sensors to better predict the high tide height at $t = 2.45$ days.

4.2 Active Data Selection

We now demonstrate our active data selection algorithm. Using the fine-grained data (downloaded directly from the sensors), we can simulate how our GP would have chosen its observations had it been in control. Results from the active selection of observations from all the four tide sensors, are displayed in figure 2. Again, these plots depict dynamic choices; at time t, the GP must decide when next to observe, and from which sensor, given knowledge only of the observations recorded prior to t, in an attempt to maintain the uncertainty in tide height below 10cm.

Consider first the independent case shown in figure 2(a), in which separate GPs are used to represent each sensor. Note that a large number of observations are taken initially as the dynamics of the sensor readings are learnt, and then later, a low but constant rate of observation is chosen. In contrast, for the independent case shown in figure 2(b), the GP is allowed to explicitly represent correlations and delays between the sensors. This data set is notable for the tide heights at the Chimet and Cambermet sensors, which due to tidal flows in the area are slightly delayed relative to the Sotonmet and Bramblemet sensors. Note that after an initial learning phase as the dynamics, correlations, and delays are inferred, the GP chooses to sample predominantly from the undelayed Sotonmet and Bramblemet sensors[1]. Despite no observations at all subsequently being made of the Chimet sensor, the resulting predictions remain remarkably accurate. Consequently only 66 observations are required to keep the uncertainty below the specified tolerance, whereas 127 observations were required in the independent case.

[1]The dynamics of the tide height at the Sotonmet sensor are more complex than the other sensors due to the existence of a 'young flood stand' and a 'double high tide' in Southampton. For this reason the GP selects Sotonmet as the most informative sensor and samples it most often.

5. CONCLUSIONS

In this paper we have demonstrated the use of a novel iterative formalism of a multi-output Gaussian process to perform information processing on sensor readings acquired from a sensor network, and shown that with minimal domain knowledge we can perform effective prediction, regression, and active sampling. A live implementation of our information agent is currently available online at `www.aladdinproject.org/situation/`. This agent uses the Gaussian process to predict several measured environmental parameters and makes these predictions available through an interactive Web-based map (see figure 3).

Acknowledgments

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Networks) project and is jointly funded by a BAE Systems and EPSRC strategic partnership (EP/C548051/1).

6. REFERENCES

[1] P. Abrahamsen. A review of Gaussian random fields and correlation functions. Technical Report 917, Norwegian Computing Center, Norway, 1997. 2nd edition.

[2] N. A. C. Cressie. *Statistics for spatial data*. John Wiley & Sons, 1991.

[3] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB 2004)*, pages 588–599, 2004.

[4] J. K. Hart and K. Martinez. Environmental Sensor Networks: A revolution in the earth system science? *Earth-Science Reviews*, 78:177–191, 2006.

[5] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN'06)*, pages 2–10, 2006.

[6] M. A. Osborne, A. Rogers, S. Ramchurn, S. J. Roberts, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In *International Conference on Information Processing in Sensor Networks (IPSN'08)*, pages 109–120, 2008.

[7] J. Pinheiro and D. Bates. Unconstrained parameterizations for variance-covariance matrices. *Statistics and Computing*, 6:289–296, 1996.

[8] C. E. Rasmussen and Z. Ghahramani. Bayesian Monte Carlo. In *Advances in Neural Information Processing Systems 15*, pages 489–496. The MIT Press, 2003.

[9] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Collaborative Sensing by Unmanned Aerial Vehicles

W. T. Luke Teacy, Jing Nie,
Sally McClean and Gerard Parr
School of Computing & Information Engineering
University of Ulster
Cromore Road, Coleraine, BT52 1SA, UK
{l.teacy,j.nie,gp.parr,s.mcclean}@ulster.ac.uk

Stephen Hailes and Simon Julier
Department of Computer Science
University College of London
London WC1E 6BT, UK
{s.hailes, s.julier}@cs.ucl.ac.uk

Niki Trigoni and Stephen Cameron
Computing Laboratory
University of Oxford
Oxford OX1 3QD, UK
{niki.trigoni,stephen.cameron}@comlab.ox.ac.uk

ABSTRACT

In many military and civilian applications, Unmanned Aerial Vehicles (UAVs) provide an indispensable platform for gathering information about the situation on the ground. In particular, they have the potential to revolutionize the way in which information is collected, fused and disseminated. These advantages are greatly enhanced if swarms of multiple UAVs are used, since this enables the collection of data from multiple vantage points using multiple sensors. However, enhancements to overall operational performance can be realised only if the platforms have a high degree of autonomy, which is achieved through machine intelligence.

With this in mind, we report on our recently launched project, SUAAVE (Sensing, Unmanned, Autonomous, Aerial VEhicles), which seeks to develop and evaluate a fully automated sensing platform consisting of multiple UAVs. To achieve this goal, we will take a multiply disciplinary approach, focusing on the complex dependencies that exist between tasks such as data fusion, ad-hoc wireless networking, and multi-agent co-ordination. In this position paper, we highlight the related work in this area and outline our agenda for future work.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

General Terms

Theory, Algorithms, Experimentation

Keywords

UAV, UAV swarms, sensor networks, decentralized control

1. INTRODUCTION

In many military and civilian applications, an aerial view is invaluable to gain information about the situation on the ground [13]. Such applications include search and rescue, perimeter surveillance, crowd control and situation awareness in natural disasters. In manned flight, such scenarios place a heavy burden on pilots, requiring long hours of monotonous flight at high-levels of concentration. Moreover, in combat situations, there may be a significant risk to the pilot if a mission must be carried out over hostile territory.

Increasingly, however, advances in airframe design and control technology mean that using Unmanned Aerial Vehicles (UAVs) for such tasks is becoming a viable option. Small, inexpensive aircraft are now commercially available, and using GPS technology, exhibit a high degree of stability in the air (see Section 6). Existing applications of these systems typically involve small numbers of UAVs working in isolation, where each one is under the constant control of a single user on the ground. Thus, operating current UAVs can still require a significant number of man hours, and there is limited technological support to co-ordinate the actions of multiple UAVs effectively.

For this reason, there are many potential benefits to be gained by increased autonomy and co-ordination in the control of multiple UAVs. In particular, we identify the following three as significant.

1. Fully autonomous UAVs require less human intervention, and therefore can increase the number of UAVs that can be operated by a single user.
2. By working together, autonomous UAVs can split up to reduce redundancy when performing a search task, and so can cover a large area efficiently with minimal resources.
3. Multiple UAVs can increase accuracy in sensing tasks by fusing information gathered from different viewpoints.

Realizing this potential is the aim behind our recently launched project, SUAAVE (Sensing, Unmanned, Autonomous, Aerial, VEhicles). The SUAAVE consortium is an interdisciplinary group in the fields of computer science and engineering. Its focus is to investigate and elucidate the principles underlying the control of clouds of networked resource-limited UAVs acting as sensor platforms, that are targeted towards achieving a global objective in an efficient manner. We consider

a number of application scenarios, for example, disaster relief after an earthquake, flood prediction and monitoring, homeland security and surveillance, and search and rescue.

Although extensive research relevant to this vision has been done before (See [13] for a full review), published works suffer from at least one of three limitations:

1. they focus on a small part of the sensing problem in isolation, such as image processing [6] or communication [9];
2. they are developed and evaluated in simulated environments [10], which make simplifying assumptions that may not hold in real-life applications; or
3. they are not directly applicable to UAV control, because they are aimed either at stationary or ground based mobile sensor networks [8].

In SUAAVE, our intention is to address these limitations by adopting a practical engineering approach, building fully integrated hardware and software systems, which we shall evaluate in a real environment. In this way, we not only intend to make novel contributions to individual areas of research, but shall investigate the complex interplay that exists between tasks such as data fusion, wireless communication, and distributed search strategies. Moreover, by evaluating our solutions as fully deployed systems, we aim to establish results on performance characteristics that are directly applicable to real world scenarios.

In the following sections, we shall outline in more detail the key issues we wish to investigate with an emphasis on the interplay between the problems and solutions they entail. More specifically, in the remainder of this paper, we highlight our proposed research in the areas of wireless ad-hoc networking (Section 2), distributed data fusion (Section 3), decentralized optimization (Section 4) and safety management (Section 5). In Section 6, we discuss the hardware platform that we shall use to conduct experiments. Finally, we conclude in Section 7, by summarising the main research issues that we aim to address in SUAAVE, and how we propose to do so.

2. COMMUNICATION

In most communication networks, routing and transmission protocols are loosely coupled with the applications that they support. This approach makes sense in networks, such as the Internet, that must support a wide variety of applications, and so should not be optimized to suit one purpose at the expense of another. However, in mobile sensor networks, there is generally a case for tighter integration between routing and application processes. There are two reasons for this:

1. Sensor networks are generally designed with a specific set of tasks in mind. We therefore have more information that can be used to optimize communication algorithms to suit the application.
2. Radio Frequency (RF) characteristics can be greatly affected by the current position of a UAV relative to other UAVs and features on the ground. This can lead to trade-offs between moving UAVs into a good location for sensing, and moving into a good location for communication.

Together, these two features mean that there is both the need and opportunity to develop tightly coupled sensing and communication protocols, which can optimize the overall performance of the sensor network by taking into account the dependencies between communication and sensing tasks.

In SUAAVE, we plan to achieve this by following three lines of investigation. First, using the hardware platform outlined in Section 6, we shall collect data about how factors such as atmospheric conditions, and UAV position relative to the ground affect RF characteristics. Although some relevant data about RF characteristics does exist [9, 4], the number of variables involved mean that results do not always generalize well, and so there is no substitute for performing platform specific observations. Second, using this data, we shall develop probability models of RF characteristics as a function of space and time. These will be used inform optimization algorithms when deciding how a UAV should behave to maximise its overall performance. Third, we shall develop both unicast and multicast routing protocols, which will take into account the needs and behaviour of high-level processes to optimize communication performance.

3. DISTRIBUTED DATA FUSION

Although UAVs provide several advantages for gathering information about various phenomena, the nature of this platform poses unique challenges for data analysis and fusion. In particular, although it is in the nature of all sensors to generate noisy output, by mounting them on a moving object whose position is difficult to determine with accuracy introduces a extra level of uncertainty. For example, even small changes to the pitch of the craft can make significant changes to the angle that it makes with the horizon, thus making it difficult to associate images with specific regions on the ground.

Another important issue is the location where data fusion takes place. The conventional approach is to use centralised data fusion — information is sent to a central site to be fused. However, such architectures require significant bandwidth and are potentially fragile: a failure in the link with the central site means that no fusion can take place. An alternative approach is to use distributed data fusion (DDF) [1], in which fusion occurs in nodes throughout the network. Such networks can be inherently robust (failures lead to gradual degradation), scalable (nodes need only know local network topology) and modular (new nodes can be introduced to enhance sensing capabilities). However, distributed fusion introduces a number of challenges. One of the most significant problems is double counting [5]: the system overestimates the amount of information available, leading to implausibly accurate estimates.

In the SUAAVE project, we shall investigate DDF algorithms that overcome these issues, and can also optimize the location where data is fused and processed based on factors such as the available processing capacity of each node,

and their proximity to the data. We will then implement field versions of these algorithms that take into account positional information and RF characteristics, as discussed in the previous section.

4. MULTI-OBJECTIVE DECENTRALIZED OPTIMIZATION

For most non-trivial applications, a UAV's actions must be guided by multiple objectives and are subject to multiple constraints. For example, in a search and rescue scenario, the optimal action for a UAV is not necessarily to cover the search area using the shortest possible path. Instead, it may also need to avoid obstacles, return to base to refuel, or move to a different position to acquire a reliable communication link with a ground control unit or other UAVs.

The complexity of this issue is further increased when UAVs act as part of a collective to achieve a common set of goals. In this case, actions of multiple UAVs must be co-ordinated to achieve the best results. For instance, by flying in a specific formation, UAVs may cover a search area more efficiently, achieve more reliable communication links or minimise the chance of collision. More significantly, by taking into account all of these aspects in parallel the collective may be able to achieve the best trade-off between all three.

In such cases, it may be possible to achieve trade-offs in a centralised way be sending all information to be processed at a single location (for example, a control station on the ground). However, as with data fusion, adopting a centralised approach can incur an unacceptable communication overhead and leave the system vulnerable if the single point of control fails or becomes unavailable. For this reason, we plan to investigate the use of decentralized control and optimization algorithms for co-ordinating multiple UAVs.

To achieve this, we plan to exploit and build on existing work on decentralized co-ordination in multi-agent systems. In particular, agent co-ordination problems can often be framed as Distributed Constraint Optimization Problems (DCOPs) for which a number of optimal solutions exist [11] [12]. However, existing optimal algorithms for DCOPs scale poorly as the number of agents in a system increases and are prone to failure if message between agents are lost or the topology of the network chances. However, more robust results have been shown using the max-sum algorithm [7]. Although this is not guaranteed to produce optimal solutions in all cases, it has demonstrated near optimal performance empirically in a wide variety of cases, scales well in large systems containing many agents, and is robust against message loss and failure of individual agents.

Although these approaches are promising, one of their draw backs is that they do not explicitly deal with cases in which agents must co-ordinate their future actions in an uncertain environment. For example, during search and surveillance, UAVs may need to plan their future joint actions based on current information, and then re-plan should an unexpected obstacle turn up in their flight path. Problems such as these are addressed by multi-agent reinforcement learning algorithms. However, existing approaches can suffer from limitations, such as lack of convergence guarantees. In our work, we shall attempt to address these issues by looking at how DCOP algorithms can be used to share information between UAVs to achieve more effective results.

Figure 1: Ascending Technologies Hummingbird UAV

5. SAFETY AND COLLISION AVOIDANCE

As discussed in previous sections, we aim to develop sophisticated autonomous UAVs that can take into account multiple objectives when deliberating over their actions. However, increased autonomy and sophistication comes with increased risk. This is particularly true if UAVs are to operate in or near to populated areas where there is the potential for collision resulting in personal injury of damage to property, or legal issues regarding flight in prohibited areas. Thus, to ensure public acceptance of such technology, it is essential that we have appropriate safety mechanisms in place to minimise both the probability of a malfunction occurring and the damage caused if one does occur.

For this reason, we shall develop a safety protocol and associated architecture that will fulfill three roles. First, it will provide real time information and predictive models to inform high-level decision processes about the current and future state of critical resources. Most significantly, this will include information about a UAV's current energy supply and how this will change in response to future actions. Second, it will verify that each planned action does not move the UAV into a position of adverse risk, such as into the flight plan of other UAVs, or close to known obstacles or no flight zones. Third, it will continually monitor the current state of the UAV to detect malfunctions if and when they occur. If a fault is detected the UAV will attempt a set pre-programmed behaviours to recover from the failure if possible (for example, by invoking collision avoidance algorithms [2, 3]), or abort the mission in the safest possible way.

6. IMPLEMENTATION AND EXPERIMENTS

Although software simulation can be a useful tool for exploring the properties of proposed algorithms, only so much can be learnt without applying a system to its intended purpose. This is particularly true for UAV applications, since simplifying assumptions are often made during algorithm or simulation development, which may give an inaccurate portrayal of how a system will perform in a real environment.

For this reason, we are adopting a practical approach on the SUAAVE project, developing fully integrated systems deployed on real UAVs. Initially, this will be based around As-

cending Technologies[1] Hummingbird platform, which consists of a four rotor airframe with on-board battery, GPS receiver, and wireless communication capabilities (see Figure 1).

At an approximate diameter of 53cm, this device is relatively small and inexpensive and can carry a payload of up to 200 grams. Out-of-the-box, it is capable of maintaining its position using its GPS receiver; and accepting instructions by remote control, or from on board software developed using the vendor's APIs. In each case, commands can either be issued in terms of low level maneuver (for example, by varying thrust to alter the yaw, pitch or roll or the aircraft) or as instructions to visit GPS way-points.

From a research perspective, these features give us two advantages. First, the available APIs and potential payload allow us to add our own devices, such as additional processors and sensors, to interface with the main system to support additional functionality. Second, since most low level control issues are taken care of by the platform, we can concentrate on adding high-level functionality, such as co-ordination and data fusion algorithms.

In our initial trails, we plan to use four of these UAVs with on board cameras, to gather realistic data for image processing tasks and radio transmission characterization. This will be used to carry out initial experiments and analysis in the lab, which will inform the development of our communication protocols and high-level control algorithms. Finally, we aim to evaluate a complete system consisting of up to ten UAVs, including high-level co-ordination, safety and sensing mechanisms.

7. CONCLUSIONS

In many applications, an aerial view is indispensable for improving situation awareness about various phenomena on the ground. Until recently, this could only be achieved by manned flight, which is expensive and places a high burden on the pilot, with long hours of potentially dangerous activity at high levels of concentration. UAVs have the potential to elevate these problems, but their maximum benefit can only be achieved through a high level of automation and co-ordination between multiple UAVs.

In SUAAVE, we aim to achieve this vision by developing fully integrated hardware and software systems. We shall investigate the complex dependencies that exist between tasks such as ad-hoc wireless communication, data fusion and multi-agent co-ordination; and so produce mechanisms that optimize overall system performance.

Moreover, to establish performance results that are directly applicable to the real world, we shall evaluate our solutions using field trails of multiple UAVs operating together to achieve a common set of goals.

Acknowledgments

The SUAAVE project is a collaboration involving academic partners from University College London, the University of Oxford, and the University of Ulster. This work is funded by the UK Engineering and Physical Sciences Research Council (EPSRC), grant reference EP/F06358X/1.

[1] `http://www.asctec.de`

8. REFERENCES

[1] D. Akselrod, A. Sinha, and T. Kirubarajan. Hierarchical markov decision processes based distributed data fusion and collaborative sensor management for multitarget multisensor tracking applications. In *the IEEE International Conference on Systems, Man and Cybernetics*, pages 157 – 164, 2007.

[2] E. Boivin, A. Desbiens, and E. Gagnon. Uav collision avoidance using cooperative predictive control. In *the 16th Mediterranean Conference on Control and Automation*, pages 682 – 688, 2008.

[3] F. Borrelli, T. Keviczky, and G. Balas. Collision-free uav formation flight using decentralized optimization and invariant sets. In *the 43rd IEEE Conference on Decision and Control*, volume 1, pages 1099 – 1104, 2004.

[4] C. Cheng, P. Hsiao, H. Kung, and D. Vlah. Performance measurement of 802.11 a wireless links from uav to ground nodes. In *the 15th International Conference on Computer Communications and Networks*, 2006.

[5] C.-Y. Chong, S. Mori, W. Barker, and K.-C. Chang;. Architectures and algorithms for track association and fusion. *IEEE Aerospace and Electronic Systems Magazine*, 15(1):5 – 13, Jan 2000.

[6] G. Conte and P. Doherty. An integrated uav navigation system based on aerial image matching. In *the IEEE Aerospace Conference*, pages 1 – 10, 2008.

[7] A. Farinelli, A. Rogers, A. Petcu, and N. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *the 7th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 639–646, 2008.

[8] E. Ferranti, N. Trigoni, and M. Levene. Brick&mortar: an on-line multi-agent exploration algorithm. In *the IEEE International Conference on Robotics and Automation*, 2007.

[9] D. Hague, H. Kung, and B. Suter. Field experimentation of cots-based uav networking. In *the Military Communications Conference*, Jan 2006.

[10] R. Henriques, F. Bacao, and V. Lobo. Uav path planning based on event density detection. In *International Conference on Advanced Geographic Information Systems & Web Services*, pages 112 – 116, 2009.

[11] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 161:149–180, 2005.

[12] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *the 19th International Joint Conference on Artificial Intelligence*, pages 266—271, 2005.

[13] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. Hedrick. An overview of emerging results in cooperative uav control. In *the 43rd IEEE Conference on Decision and Control*, volume 1, pages 602 – 607, 2004.

DCOPs Meet the Real World: Exploring Unknown Reward Matrices with Applications to Mobile Sensor Networks

Manish Jain*, Matthew Taylor*, Makoto Yokoo+, Milind Tambe*
* University of Southern California, Los Angeles, CA 90089
{manish.jain,taylorm,tambe}@usc.edu
+ Kyushu University, Fukuoka 812-8581, Japan
yokoo@is.kyushu-u.ac.jp

ABSTRACT

Buoyed by the recent successes in the area of *distributed constraint optimization problems* (DCOPs), this paper addresses challenges faced when applying DCOPs to real-world domains. This expedition reveals that three fundamental challenges must be addressed for a large class of real-world domains, requiring design of novel DCOP algorithms. First, in many domains, agents do not know the initial payoff matrix and must explore the environment to determine rewards associated with different variable settings. Second, the agents have a goal to maximize the total accumulated reward rather than the instantaneous reward at the end of the run. Third, limited task-time horizons disallow agents the luxury of full exploration of their environment and payoff matrices. We propose and implement a set of novel algorithms and provide positive experimental results. At their core, these new algorithms interweave decision-theoretic approaches to explore the environment within the limited time horizon with the DCOP-mandated coordination. In addition to simulation results, we implement these algorithms on robots, deploying DCOPs on a distributed mobile sensor network – illustrating the benefits of DCOPs in the real world.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

DCOPs, Unknown Reward Function, Balanced Exploration

Keywords

DCOP, Multiagent Exploration and Exploitation, Mobile Sensor Networks

1. INTRODUCTION

Distributed constraint optimization problems (DCOPs) [10, 12, 20] are a class of problems where cooperative agents must coordinate to maximize some reward. Examples include multiagent plan coordination [4], sensor networks [8, 20], meeting scheduling [16] and RoboCup soccer [18]. A team of agents coordinate their individual actions within a DCOP to achieve joint goals, but the utility of an agent's action depends on the action choices of a subset of the other agents; DCOPs are thus ideally suited when agents must coordinate via local interactions. Significant progress has been achieved in design and analysis of globally optimal DCOPs algorithms (c.f., Adopt [12], DPOP [16], and OptAPO [10]). However, given that DCOPs are NP-Hard [12], significant communication and computation overheads result in attempting to solve them optimally, which motivates the need for locally optimal algorithms. Such locally optimal algorithms that have been shown to scale to much larger tasks in practice [20, 15, 19].

Given their recent progress, DCOPs are now ideally poised to tackle real-world applications. Motivated by this objective, we target a large class of real-world distributed sensor network applications, that include tasks such as AUVs (Autonomous Underwater Vehicles) [21] used for surveying underwater structures, UAVs (unmanned air vehicles) to measure environmental phenomena [2], and small mobile robots that establish a communication network. Our study reveals that three novel challenges must be addressed in applying DCOP algorithms to these domains. First, agents in these domains do not know the initial payoff matrix and must explore the environment to determine rewards associated with different variable settings. All payoffs are dependent on agents' joint actions, requiring them to coordinate in their exploration. Second, the agents must maximize the total accumulated reward rather than the instantaneous reward at the end of the run. Third, agents face a limited task-time horizon, requiring efficient exploration. These challenges disallow direct application of current DCOP algorithms which implicitly assume that all agents have knowledge of the full payoff matrix. Furthermore, agents cannot fully explore their environment to learn the full payoff matrices and then run a globally optimal algorithm. The time horizon is much too short for such a sequential phase of exploration followed by optimization; indeed, interleaving these phases may improve accumulated reward during exploration.

To address these challenges, this paper proposes novel DCOP algorithms based on two key ideas. First, as mentioned above, these DCOP algorithms must seamlessly interleave *distributed exploration* and *distributed exploitation* phases. Second, DCOP algorithms may need a range of exploration strategies in their arsenal, each potentially useful in a different setting. For example, in a *one-step* strategy, an agent deciding to explore examines one unknown payoff matrix value at a time and triggers optimization, enabling fine-grained interleaving of exploration and exploitation. In contrast, a *multi-step* exploration strategy allows a single agent to scout out multiple payoff matrix values and then select the best observed dur-

ing the exploration phase. Here a decision-theoretic approach uses knowledge of reward distribution to compute the optimal number of unknown values to explore and the expected reward gain. However, this approach forces a coarser-grained integration where a single agent in a neighborhood of agents explores for multiple steps before triggering distributed optimization. We also introduce a *hybrid* approach which attempts to combine the strengths of the two strategies. In particular, agents compute their expected gain as in a multi-step exploration strategy, but trigger optimization at each time step, enabling fine-grained interleaving of exploration and exploitation. Using these key ideas, we provide a family of five novel DCOP algorithms.

Given this diverse family of algorithms, it is crucial to understand if particular algorithms dominate others in key circumstances. Our empirical tests are based on a novel, concrete, real-world domain in which agents must maximize an accumulated signal strength in a mobile sensor network within a set time limit. Our algorithms enable agents to reason about movement policies in order to improve their signal strength over time and increase the reliability of the network. To perform such optimization efficiently, we model this problem as a DCOP with the novel extensions discussed above.

Our new DCOP algorithms are implemented not only on simulated agents, but on physical robots as well. Apart from early work on distributed constraint reasoning by Lesser et al. [8], this is the first application of DCOPs on physical robots with a demonstrated improvement in performance in a real world problem. Furthermore, the simulations allow us to gather experimental results quickly while changing parameters of the task (such as the number of robots in the network and length of the experiments) to further evaluate our algorithms.

Key results from our experiments include: (i) algorithms based on the hybrid strategy dominate in most circumstances and (ii) one-step strategies are sufficient when networks are fully connected or even superior when few optimization movements are allowed. Furthermore, we compare to an algorithm which is given the DCOP reward matrix at the beginning of each experiment and find that our algorithms are able to achieve signal improvements of up to 80% of this "omniscient" algorithm.

These results combine to show that DCOP algorithms can be modified to work in environments where: on-line reward is critical, rewards are initially unknown, the algorithms are fully distributed, and using physical hardware. Combined, these results strongly suggest that DCOPs are a powerful and appropriate solution technique for complex multi-agent problems in the real world.

2. BACKGROUND

This section of the paper first defines the mobile sensor network optimization task. Section 2.2 discusses the DCOP formulation and why it is useful for this problem domain.

2.1 Problem Domain

This paper focuses on tasks in which agents do not know their initial rewards, there is a fixed time horizon, and agents are evaluated during optimization (i.e., the on-line reward is critical). One such problem with these characteristics is a *wireless sensor network* (c.f., Akyildiz et al. [1]), where a common goal is to monitor a region of the world and report when interesting objects or events are perceived. Many multiagent problems share these characters; problems in this class include aerial surveillance [2] and underwater agent coordination [21].

Sensors in this paper are assumed to have movement abilities. Rather than framing the problem as an ad hoc mobile wireless network, we take the topology of the network as fixed under the assumption that humans have placed the robots in reasonable positions. We also assume that each sensor knows its *neighbors*, the sensors with which it can directly communicate. For example, during natural disasters, rescue personnel may quickly place such mobile sensors around a disaster site to relay information about endangered humans, fires, etc. It will then be critical for the sensors to quickly optimize the network's signal strength to ensure reliable and effective communication.

Radio communication, commonly used in wireless sensor networks, has a predictable signal strength based on the inverse square of the distance between transmitter and receiver.[1] However, in urban or indoor settings, obstacles often interrupt line of sight communication, creating a *multi-path* setting. Scattering, reflection, and diffraction of radio waves make it very difficult to predict the optimal placement of sensors in a network. Constructive and destructive interference of the radio waves, known as the *small scale fading effect*, results in significant signal strength differences over small distances.

This paper concentrates on settings where the sensors are not line of sight. Due to small scale fading, the signal strength between two locations separated by a distance of at least $\frac{1}{2}$ of a wavelength is uncorrelated. Put another way, if a sensor moves $\frac{1}{2}$ of a wavelength, it will measure a new signal strength from each of its neighbors, where each signal can be modeled as an independent random number drawn from some distribution [7]. Our initial experiments suggest a Normal distribution, but our algorithms are distribution-independent and other distributions can easily be substituted.

We assume in this study that sensors have been placed in reasonable locations so that no sensor is disconnected from the network. Further, we assume that nodes do not fail, they are not malicious, and communication between neighbors is reliable. Given a network topology and length of the experiment (time T), our goal is to maximize the signal strength in the network over this time. Using l as our index over network links, we want to maximize: $\sum_{0 \leq t \leq T} \sum_{l \in network} \mathit{SignalStrength}(l, t)$ by allowing agents to take small movements without changing the overall network topology. We discretize the experiment and the agents' decision making into synchronized *rounds*. A round ends after all agents perform the required computation, finish communication, and move to a new location (if desired). The length of a round in our distributed setting is dominated by the robot movement time, which is much longer than either the computation of any algorithms in this paper or the communication time.[2]

Experiments in this paper use results from a set of Create robots

[1] For an in-depth discussion of signal strength propagation, we refer interested readers to more complete treatments elsewhere [13].

[2] If agents converge upon a final configuration before the test ends, no robots will move within a round. However, the length of the round does not vary, but remains the time necessary for a robot to move, calculate, and communicate. The length of a round determines how continuous time is discretized to measure signal strength in experiments but is not critical for measuring the relative performance of algorithms.

Figure 1: This photo shows a team of iRobot Creates, the platform used for physical tests of our algorithms. A more detailed view of the robot is shown in the inset.

from iRobot and from a custom simulator built to mimic properties of the Create. The Creates have 2 actuators which can be controlled. They are capable of moving accurately in a straight line, while rotation introduces significant error. Each of these Creates have a wireless radio card provided by CenGen mounted on them. A fully charged battery is capable of running the Creates for approximately 3 hours. Figure 1 shows a team of the Create robots.

As the real world is continuous[3] and Create robots have limited sensors, agents in our physical implementation are unable to determine their absolute location during experiments, but must instead rely on odometry to estimate relative location. Due to the high variance in signal strength over small distances and accumulated errors in odometry measurements, agents in some tasks may not be able to return to a previously observed signal strength. Thus, in this paper, we consider two distinct cases, as both may be valid, depending on the particular implementation of agent movement and available sensors. In the first case, each agent may either `stay` where it is, or `explore` by moving to a new location. In the second case, we assume that odometry errors can be ignored, which enables an agent to additionally execute the action `backtrack`, returning it to a previously explored location. In our physical implementation, the robots are able to `backtrack`, but we also run experiments assuming they cannot. Note that in this work we assume that agents are always able to explore a new state and *never* return to a previously visited state by selecting the `explore` action.

Mobile sensors must efficiently explore their surroundings to maximize the network signal strength as quickly as possible. If the goal was to maximize final signal strength and the agents had a small number of possible values, agents could explore the entire state space to populate the reward matrix and then use a traditional DCOP method to find an optimal setting of values (as discussed in the following section). However, because the goal is to maximize signal strength accumulated during the entire episode, fully exploring the state space would cause the agents to spend much of their time in highly sub-optimal configurations. Furthermore, in this work we assume that the number of states each robot can visit is large, making it impossible for a robot to explore all possible locations within the time of a single experiment, let alone the cross product of all possible locations.

[3]As discussed in Section 4.1, we discretize the continuous world based on the wavelength of radio waves.

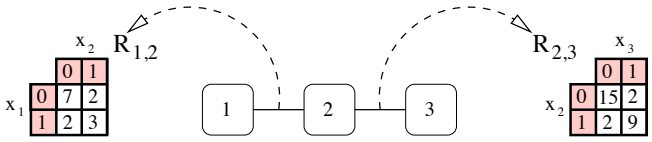


Figure 2: This figure depicts a three agent DCOP. Agents 1–3 each select a value for their corresponding variable (x_1 – x_3) from the domain $\{0, 1\}$. There are two sets of constraints which each define one of the two reward matrices.

2.2 DCOP Background

A DCOP consists of a set V of n variables, $\{x_1, x_2, \ldots, x_n\}$, assigned to a set of agents, where each agent controls one variable's assignment. Variable x_i can take on any value from the discrete finite domain D_i. The goal is to choose values for the variables such that the sum over a set of binary constraints and associated payoff or reward functions, $f_{ij} : D_i \times D_j \rightarrow N$, is maximized. More specifically, find an assignment, A, s.t. F(A) is maximized: $F(A) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j)$, where $d_i \in D_i, d_j \in D_j$ and $x_i \leftarrow d_i, x_j \leftarrow d_j \in A$. Take the constraint graph in Figure 2 as an example. x_1, x_2, and x_3 are variables, each with a domain of {0,1} and the reward function as shown. If agents 2 and 3 choose the value 1, the agent pair gets a reward of 9. If agent 1 now chooses value 1 as well, the total solution quality of this complete assignment is 12, which is locally optimal as no single agent can change its value to improve its own reward (and that of the entire DCOP). $F((x_1 \leftarrow 0), (x_2 \leftarrow 0), (x_3 \leftarrow 0)) = 22$ and is globally optimal.

We use the mobile sensor network domain introduced in Section 2.1 as an experimental domain for our algorithms. The different robots in the network are the DCOP-aware agents. Communication links between robots represent constraints between agents and the signal strength obtained measures the reward of an assignment. The different physical positions of a robot constitute the domain of values possible for agents. An agent can accurately sense the signal strength between its current location and the current location of each of its neighbors only when it explores that particular location.

3. SOLUTION TECHNIQUES

This section describes novel extensions to DCOP methods to tackle the class of problems outlined in the Introduction, using the mobile sensor network problem as a concrete example of one such problem. The distributed and multiagent nature of the problem makes DCOP an appropriate solution technique. The primary difference between this domain and traditional DCOP domains are: (1) firstly, the agents' goal is to maximize the cumulative reward, (2) secondly, the agents do not know the reward matrix R, and (3) thirdly, the agents do not have time to explore all states. Thus, standard DCOP solvers cannot be directly applied to the type of problem discussed in this paper.

Given the inapplicability of globally optimal algorithms, we build on an existing locally optimal DCOP algorithms. The *Maximal Gain Messaging* (MGM) algorithm [15] and DSA [5] are natural candidates, but DSA has an additional probability parameter that must be set which has a significant impact on its performance [9]. While all the algorithms presented are in the framework of MGM, the key ideas of the paper can be embedded in any locally optimal DCOP framework. However, we keep the framework constant to ensure a fair comparison of the algorithms.

MGM-Omniscient: We first implement the *Maximal Gain Messaging* (MGM) algorithm [15] and artificially provide agents with all possible signal strengths. This represents an upper bound for us to compare with as the agents do not need to explore. Given such a matrix, any standard DCOP algorithm may quickly find a locally optimal solution for all agents. MGM-Omniscient defines a *round* as involving multiple broadcasts of *messages*. Every agent will broadcast its current value to all its neighbors at the beginning of the round. After the receipt of these messages, each agent broadcasts a gain message to all its neighbors that represents the maximum change in its local utility if it is allowed to act under the current context (i.e., values of neighboring agents). An agent is then allowed to act if its gain message is larger than all the gain messages it receives from all its neighbors (ties can be broken through variable ordering or another method) [19]. MGM-Omniscient belongs to the class of 1-optimal algorithms which have the property that no one agent can deviate from the proposed assignment and increase the net reward [15].

3.1 Static Estimation (SE) Algorithms

Our first set of solution techniques relies on selecting an action based on the probability of improving the signal strength on the next round. All are one-step approaches.

3.1.1 SE-Optimistic

In *Static Estimation with Optimistic Exploration* (SE-Optimistic), each agent assumes that if it moves to a new location, the signal strength between it and every neighbor will be maximized. On every round, each agent bids its expected gain based on its current sum of signal strengths (R_c): *NumLinks*$\times$*MaxSignalStrength*$-R_c$. In each neighborhood, the agent with the highest bid is allowed to `explore` for the current round and other agents execute the `stay` action. Agents which have the lowest signal strengths will have the highest bid, similar to a 1-step greedy optimization algorithm. Because it will be rare to achieve maximal signal strengths to all neighbors, agents will typically continue to bid to explore on every round until the test concludes.

3.1.2 SE-Mean

Static Estimation with Mean-Aware Exploration (SE-Mean) modifies the previous algorithm to assume that visiting an unexplored state will result in the average signal strength to all neighbors (denoted μ) instead of the maximum. Agents therefore have an expected gain of: *NumLinks* $\times\ \mu - R_c$. This modification to the previous algorithm causes agents to continue to greedily explore, but now agents will stop bidding to move once they achieve the average signal strength (averaged over all neighbors), allowing them to balance exploration with exploitation. Note that *MaximumReward* and μ can be defined initially as the reward distribution is known.

3.2 Balanced Exploration (BE) Algorithms

The objective of the *Balanced Exploration* (BE) approach, our second set of algorithms, is to allow each agent to explicitly estimate the value of exploration, which will depend on the following three things: (1) the number of timesteps left in the trial, (2) the distribution of the signal strengths (rather than just the maximum or the mean), and (3) the current signal strength of the agent, or the best explored signal strength if the agent can backtrack to a previously explored state.

The primary motivation for this class of algorithms is to allow the agents to more accurately estimate their gains. After agents calculate their expected gain from exploring or exploiting, each will decide whether to exploit its current signal strength or to bid to explore. As in MGM, the agent with the highest bid per neighborhood wins the ability to move.

3.2.1 BE-Backtrack

When an agent can execute the `backtrack` action, it will keep track of the location in which it has received the highest total signal strength (R_{best}). At any point, the agent may return to this location if the agent's neighbors have not moved. If one or more of the agent's neighbors have moved, return to this previous location will likely give a different total signal strength. Note that R_{best} will always be well defined, as an agent may always set $R_{best} = R_c$, its current total signal strength. The state of the agent can thus be defined as a 2-tuple of (R_{best}, T) consisting of the reward R_{best}, and the T time steps remaining in the current test.

The *Balanced Exploration with Backtracking* (BE-Backtrack) algorithm takes a multi-step approach, where an agent calculates $V(R_{best}, T)$, the expected utility of the agent if the current best backtrack location has a reward of R_{best} and T time steps remain in the current test.[4] If the agent backtracked immediately to the location with reward R_{best}, again assuming that no neighbors move, the agent would accrue the utility R_{best} for the remainder of the experiment. Therefore, the value of backtracking will be

$$V_{back}(R_{best}, T) = R_{best}T.$$

If the agent explores, it receives a reward based on the best location explored. Let the number of rounds for which the agent explores be t_e. An exploration policy would be in the form "explore for t_e rounds, `backtrack` to the best location found on round $t_e + 1$, and then `stay` in that location for the remainder of the experiment for t_s rounds, where $t_s = T - (t_e + 1)$.

$V_{explore}(R_b, T)$ can be calculated by summing three separate components: the expected utility accrued while exploring for t_e steps, the utility accrued after exploration multiplied by the probability of finding a reward better than R_b, and finally the utility accrued after exploration multiplied by the probability of failing to find a reward better than R_b.

The first component will simply be $t_e \times \mu(n)$, where $\mu(n)$ is the average expected signal strength over n neighbors. The second component will depend on the probability of finding locations with a total signal strength higher than R_{best}, multiplied by the number of steps left in the trial. The expected best signal strength in this case will be described by the probability distribution:

$$\int_{x > R_{best}} x \times P(x, n, t_e)dx$$

where $P(x, n, t_e)$ gives the probability of x being the maximum sample among the t_e samples drawn when the agent has n neighbors and is defined as:

$$P(x, n, t_e) = t_e \times f(x, n) \times F(x, n)^{t_e - 1}$$

[4]Note that throughout the discussion of the balanced exploration, there is a notion of *state*, which is different from an agent's *location*. For instance, the value of taking the `backtrack` action depends on the best reward seen (R_{best}) and the number of steps remaining in the episode, which describe the state. It does not depend on the current location, or the current signal strength. Thus $V_{back}(\textit{state}) = V_{back}(R_{best}, T)$.

This n^{th} order statistics calculates the probability that x will be the maximum reward found in t_e values. n is the number of neighbors, $f(x,n)$ is the probability of drawing x as a sample, and $F(x,n)$ is the cumulative probability of drawing a sample less than or equal to x, defined as $\int_{y \leq x} f(y,n)dy$. Informally, $P(x,n,t_e)$ is calculated by drawing a sample x from any of the t_e samples with a probability $f(x,n)$, and drawing the rest of the $t_e - 1$ samples, such that their values are less than x, with a probability of $F(x,n)^{t_e-1}$.

The third component will depend on how likely it is that we fail to discover a location better than R_{best}, multiplied by the number of steps left in the trial. After the agent explores, it will backtrack to the location that receives a total signal strength of R_{best} and the agent will receive this reward for the remaining t_s rounds. Again, the cumulative probability of drawing a sample less than or equal to R_{best} in t_e samples is defined as $F(R_{best})^{t_e}$, where $F(x)$ is defined as before.

Summing the three components, we find that $V_{explore}(R_{best}, T) =$

$$\max_{0 \leq t_e \leq T} \left\{ t_e \mu(n) + t_s \int_{x > R_b} x P(x,n,t_e) dx + t_s R_b F(R_b, n)^{t_e} \right\} \quad (1)$$

The value of t_e that maximizes $V_{explore}$ gives the number of exploration steps. The expected return of being in a location with T steps left, having previously seen a location with a total signal strength of R_{best}, is: $\max \left\{ V_{back}(R_{best}, T), V_{explore}(R_{best}, T) \right\}$. Having calculated the expected utility of `backtrack` and `explore`, the agent will select the action with the highest utility. Unless the action selected is `backtrack` and the current signal strength equals R_{best}, the agent will bid its expected utility, and the agent with the highest utility within each neighborhood will be allowed to move for t_e rounds, after which it backtracks to the location with the best found signal strength. BE-Backtrack dictates that the agent will not move after backtracking in the $(t_e + 1)^{th}$ round. However, if the agent's neighbors later move, the agent may choose to explore rather than staying at its backtracked value.

Notice that when an agent's neighbors explore and then backtrack, they could not have reduced the overall DCOP reward. In particular, the signal strength of an agent that has backtracked after exploring cannot be lower than its signal strength at the time it started exploring (although it may be lower during exploration). This is because only this agent was allowed to move in its neighborhood, and the agent could have backtracked to its initial location (and, thus initial signal strength) if it were unable to find a better configuration.

3.2.2 BE-Rebid

The BE-Backtrack approach allows the agents to return to a previously explored state. The SE-Optimistic and SE-Mean approaches allow agents to rebid after every round. Prior work in different decision making contexts [14] has shown that such reevaluation at each timestep can lead to better performance in practice. *Balanced Exploration with Backtrack and Rebidding* (BE-Rebid) combines both of the above algorithms, representing a type of hybrid approach. The agents calculate their gain using the same equations as in BE-Backtrack, but all agents re-calculate and rebid their most favorable action in each time step.

Equation 1 calculates the expected gain of exploring for t_e steps, but if agents rebid on each round, the number of exploratory steps an agent executes may be different from t_e, potentially invalidating the agent's initial estimate. However, if an agent wins the bid to move and then moves for fewer than t_e rounds, it will be due to signal strengths received after moving: either the moving agent has found itself in a favorable position and no more exploration is needed, or the cumulative signal strength of one of its neighbors has significantly decreased. As an example, consider three agents connected in a chain, as in Figure 2. Suppose that agents 1 and 3 explore and the signal strength between 1–2 and 2–3 drops. It may be more efficient for agent 2 to explore in order to improve both of its signal strengths, but this on-the-fly reasoning is disallowed in BE-Backtrack when $t_e > 1$.

Since the agent is allowed to `backtrack`, the reward enjoyed by the agent after it `backtrack`s depends on the maximum reward that it encounters during the exploration phase. In the next section, we consider a similar algorithm, but assume that the agent cannot `backtrack`, removing the ability for an agent to retrace its steps back to a previous location. The expected utility of moving to a new location is given by the mean μ of the distribution, and the immediate reward is known to the agent.

3.2.3 BE-Stay

The *Balanced Exploration with Stay* algorithm applies when agents are unable to backtrack. Based on initial experiments which suggested that BE-Rebid outperformed BE-Backtrack, BE-Stay was designed as another one-step approach where agents make a decision during every round. The intuition behind this approach is that at each round, every agent considers its current total signal strength, R_c, and compares the expected value it would get by staying in the same location (V_{stay}) with the expected value of exploring ($V_{explore}$). Because only one agent in a neighborhood can move at a time, we again assume that no neighbors move when calculating these expected values. The value of exploring can be formulated recursively, where the reward will be zero at time $T = 0$, but the agent can choose to either perform the `stay` or `explore` action at time $T > 0$. The value of exploring is calculated as follows:

$$V(R_c, T) = \begin{cases} V_{stay}(R_c, 0) = V_{explore}(R_c, 0) = 0 & \text{for } T = 0 \\ \max(V_{stay}(R_c, T), V_{explore}(R_c, T)) & \text{for } T > 0 \end{cases} \quad (2)$$

The expected value from stay will be the current signal strength multiplied by the time left in the trial:

$$V_{stay}(R_c, T) = R_c T.$$

The expected value of moving will depend on the probability of achieving a given signal strength in the next state, the reward received for that signal strength on one timestep, and the expected value of the rest of the trial:

$$V_{explore}(R_c, T) = \int_{-\infty}^{\infty} P(x)(V(x, T-1) + x) dx$$

where $P(x)$ is the probability of receiving the total signal strength x in an unexplored location.

In each round, agents calculate V_{stay} and $V_{explore}$ using Equation 2. If `explore` has the higher expected value, an agent will bid to move for one round, and the agent with the highest bid in each neighborhood will move. Note that BE-Stay differs from BE-Rebid even when the backtrack state is the current state: BE-Rebid assumes the agent may backtrack to this state in the future, which BE-Stay does not.

4. EXPERIMENTAL RESULTS

This section of the paper discusses the empirical validation of the five novel DCOP algorithms in both simulated and physical agents.

4.1 Experimental Setup

Experiments in this section compare the performance of our DCOP algorithms in multiple settings by measuring the cumulative signal strength achieved. Signal strength values were all non-negative integers (which conforms to the CenGen interface of the physical robots), which allowed our implementations to use summations rather than integrations.

In simulation, every experiment is run for 30 independent trials with each of the five algorithms (and one additional time for MGM-Omniscient). Each of 30 trials begin with the same initial configuration to reduce the impact of randomness. In each experiment, lower and upper bounds are determined by disallowing all sensor movement and using MGM-Omniscient, respectively. Results are reported as a scaled gain, scaled uniformly between 0 and 1. Any gain greater than zero represents an improvement directly due to the controlling DCOP algorithm. Such a metric helps isolate the improvement due to sensor movement and scales across tasks with different numbers of links, agents, and horizons. Signal strengths are drawn from a normal distribution with a mean of 100 and a standard deviation of 16.[5]

When experimenting with the physical robots, we do not compare with MGM-Omniscient because the reward matrix is unknown in the real world and instead report the absolute gain. Experiments are conducted with three Create robots running the DCOP algorithm and a fixed radio controller, forming the set of agents in the DCOP. Due to the wavelength used by our robots (5 GHz corresponding to wireless 802.11a specifications), we discretize the state space so that agents move by 2.5cm at a time, ($\frac{1}{2}\lambda$).

On each round, robots wait one second for signals to stabilize and then calculate their signal strengths by querying the CenGen interface. 10 samples, collected $\frac{1}{4}$ second apart, are averaged to define the agent's current signal strength. Robots were placed at a distance of 5–10 meters apart in a non-line of sight configuration. Since the objective of the agents was to maximize the cumulative reward in the given time horizon, we did not analyze the runtime of the experiments either in simulation or on physical robots.

Results in simulation (Section 4.2) show that the performance of the hybrid BE-Rebid approach is statistically the same, or better than, all the other algorithms (except for very short experiments). Results on hardware (Section 4.3) show that these algorithms are able to provide significantly improvements over no optimization. Additionally experiments show that the balanced exploration techniques work better than SE-Mean and SE-Optimistic for the majority of settings.

4.2 Simulation Results

This section presents three sets of results, each of which varies a different component of the problem domain: the number of agents, the time horizon, or the network topology. First, however, consider the learning curve in Figure 3, which shows the total reward per timestep of our five algorithms, along with MGM-Omniscient

[5]The range $\mu - 6\sigma$ to $\mu + 6\sigma$ covers 99.999% of the samples for a normal distribution. We therefore considered signals within the range [0,200].

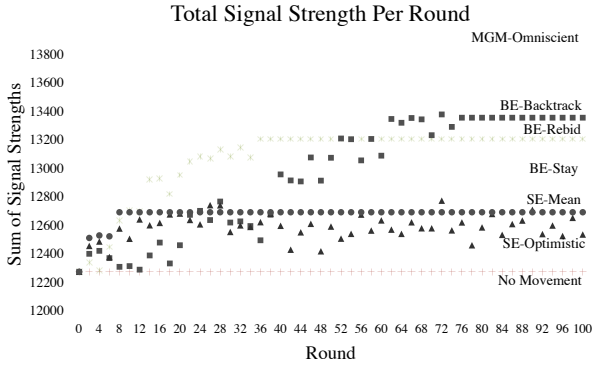


Figure 3: A representative learning curve for 20 agents in a chain topology where $T = 100$.

(topmost) and NoMovement (bottom most). This learning curve depicts the algorithms' rewards over time. For each algorithm, the total cumulative signal strength will be the area under the curve, and the gain will be the area between the curve and the NoMovement line. The y-axis shows the total signal strength and the x-axis shows the time horizon. SE-Mean converges quickly to a comparatively low value while SE-Optimistic continually explores, attempting to achieve the maximal signal strength. BE-Stay can not backtrack and thus must be more cautious; it converges the fastest of all three BE methods. BE-Backtrack attains the highest final reward, but it takes much longer to converge than BE-Rebid and does not achieve the highest cumulative signal strength (BE-Rebid explored for only 36 rounds and received a final reward of 13,198, whereas BE-Backtrack explored for 76 rounds and received a final reward of 13,347; the cumulative rewards for BE-Rebid and BE-Backtrack were 666,062 and 659,925 respectively).

Having examined a single trial, consider the first set of results in Figure 4(a) which examine how the algorithms' relative performance changes as the number of rounds is increased. The y-axis measures the scaled gain, where $y = 0$ is equivalent to no sensor movement and $y = 1$ on the y-axis is the cumulative reward from MGM-Omniscient. The x-axis shows the five values of T, the total number of rounds in a trial, used in the experiment. All trials used random sparse graphs with 15–20 links and 10 agents. Each result is averaged over 30 independent trials and the error bars show the standard error. The difference between scaled gain for each pair of algorithms are statistically significant within a single value of T (paired Student's t-tests calculate $p < 0.05$), except for $T = 5$. When the time horizon is very small, SE-Mean and the BE algorithms all performed roughly the same because all four algorithms performed very little exploration. As the number of rounds per experiment increases, BE algorithms outperform SE algorithms, and BE-Rebid consistently achieves the highest scaled gain.

The second set of experiments, summarized in Figure 4(b), varies the number of agents. The y-axis again depicts the normalized gain. The x axis shows the number of agents, varied from 5 to 50. Paired Student's t-tests determine that the results are statistically significantly different ($p < 0.05$) within all sets of tests of the same number of agents. The performance of BE-Rebid was significantly better than the performance of other algorithms in all cases.

The third set of results shown in Figure 4(c) compares the performance on different graph topologies: a chain structure, random structures (with $\frac{1}{3}$ or $\frac{2}{3}$ of all possible links enabled), and a fully

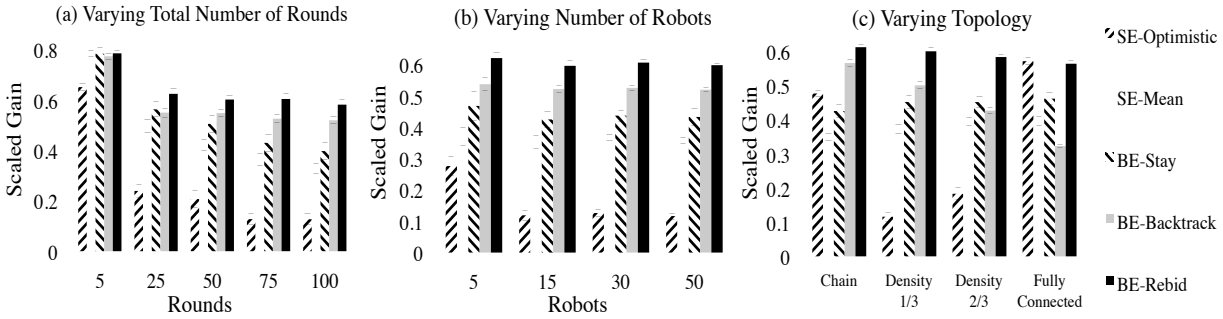


Figure 4: The performance of different algorithms is shown where the y-axis is the scaled gain (0 represents No-Movement and 1 represents the gain of MGM-Omniscient), the x-axis describes the setting, and the error bars show standard error.

connected topology. Each test uses 20 agents and 100 rounds. All results within a single topology are again statistically different ($p < 0.05$). The y-axis again measures scaled gain and error bars show standard error. Results are again averaged over 30 trials, all with random payoff matrices. All results within a single topology are statistically different ($p < 0.05$).

Three trends in Figure 4(c) are worth noting. First, BE-Rebid statistically significantly ($p < 0.05$) outperforms all other algorithms in all topologies tested, except in the fully connected graph (where it is roughly equivalent to SE-Optimistic as only one agent can move per round). Fully connected graphs are thus one setting where the relatively simple static estimation algorithms can perform just as well as the more complex BE algorithms.

Second, as the link density of the graph is increased, the relative performance of BE-Backtrack *decreases*, with statistical significance ($p < 0.05$), due to the aggressive nature of the algorithm. A BE-Backtrack agent will explore for t_e steps, preventing all neighbors from moving during this time. Thus, as the link density increases, higher number of agents are not allowed to move until after t_e steps.

Third, SE-Mean outperforms SE-Optimistic in randomly generated graphs, but not in chain and fully connected graphs. Unlike in chain and fully connected graphs, agents in random graphs can have a high variance in their degrees of network connectivity. We analyze the number of agents that were able to optimize their rewards. While 40% of the robots moved when running SE-Mean, only 18.5% robots could do so when running SE-Optimistic in random graphs with density $\frac{1}{3}$. SE-Optimistic agents with a high degree of connectivity monopolize movement opportunities because they bid unrealistically high rewards. Their bid is relatively large when compared to the bids of agents with lower degrees. There exists a large correlation (Pearson's coefficient of $\rho > 0.5$) between the degree of the agent and the number of moves made by the agent in SE-Optimistic. In contrast, SE-Mean agents allow others to win bids once the reward of an agent reaches the average over all neighbors. There exists only a weak correlation with $\rho < 0.05$ between the degree of the agent and the number of moves made by the agent for SE-Mean, explaining this difference in performance.

The results also show that BE-Stay is statistically significantly dominated by BE-Rebid, demonstrating that the ability to backtrack can lead to significantly better performance.

4.3 Physical Robots Results

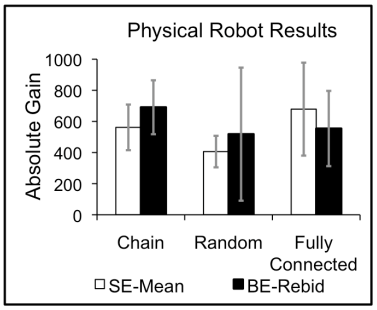


Figure 5: This graph shows the performance of SE-Mean and BE-Rebid for different topologies, as described in Section 4.3, averaged over Each experiment summarizes results from 5 independent trials, the y-axis shows the absolute gain in signal strength over the initial configuration, and in all cases our DCOP methods significantly improve over the initial configuration. Error bars show the standard error.

The previous section showed that our novel DCOP algorithms were able to significantly improve the accumulated signal strength in a simulated environment. This section takes the evaluation one step further by demonstrating that two of our algorithms significantly improve performance on physical hardware, using measured signal strength data as the reward.

Three topologies were tested: chain, fully connected, and random graphs. In the random topology tests, the robots were randomly placed and the CenGen API automatically defined the neighbors, whereas in the chain and fully connected tests the agents had a fixed set of neighbors over all trials. All tests were conducted with a time horizon of 20 time steps and used four agents (three robots and one fixed radio controller). SE-Mean was chosen because it performed best with a small number of agents in simulation and BE-Rebid was chosen because it almost always outperformed all other algorithms.

Figure 5 shows the results of running BE-Rebid and SE-Mean on the robots. The y-axis shows the gain of the algorithms. Note that the y-axis hasn't been normalized in this figure as MGM-Omniscient could not be run on the real robots (the actual signal strength cannot be determined *a-priori*). The values are signal strengths in decibels (dB). Each value is averaged over five independent trials. For example, when the topology is chain, the average gain of BE-Rebid is 691 units where as the average gain of SE-Mean is 561. BE-Rebid performed better than SE-Mean in the chain and random graphs,

but SE-Mean performed better in the fully connected graph. While too few trials were conducted for statistical significance, it is important to note that in all the cases, there is a significant improvement over the initial configuration in the robots. Additionally, because decibels are a log-scale metric, the gains are *even more significant than one may think on first glance.*

Taken as a whole, experiments in this paper show that our DCOP algorithms are able to significantly improve mobile sensor networks in both simulation and hardware. They also demonstrate the superiority of the hybrid approach of BE-Rebid in most situations, except for fully connected graphs or when very few movements are allowed. In such cases, BE-Rebid doesn't have any additional advantage over simpler approaches like SE-Optimistic.

5. RELATED WORK AND CONCLUSIONS

Related work in DCOPs has been discussed in earlier sections. Specifically, previous work in distributed constraint reasoning in sensor networks [8, 20] does not use a DCOP formulation or handle unknown reward matrices. A number of other works on mobile sensor networks for communications (c.f., Cheng. et al [3] and Marden et al. [11]) are based on other techniques (e.g., swarm intelligence, potential games, or other robotic approaches). Instead, we extended DCOPs as they can scale to large tasks using local interactions. *Reinforcement learning* [17], a popular approach in multiagent learning, does not directly apply in this domain as agents must quickly discover good variable settings, not a control policy. *Optimal Stopping Problems* (c.f., the *Secretary Problem* [6]) optimize the final rank of the selected instance, not on-line metrics, and are exclusively single agent.

This paper focuses on a class of problems that DCOPs could not address before. Apart from early work on distributed constraint reasoning by Lesser et al. [8], this is the first application of DCOPs on physical robots with a demonstrated improvement in performance in a real world problem. We show that such real world domains raise new challenges: (1) agents do not know the initial payoff matrices, (2) the goal is to maximize the total reward instead of the final reward, and (3) agents have insufficient time to fully explore the environment. These challenges open up a new area for DCOP research, as current DCOP algorithms cannot be directly applied. We present and empirically compare five novel DCOP algorithms addressing these challenges. We also present results from two algorithms implemented on physical robots. Our results show significant improvement in the reward in mobile sensor networks. Our experiments demonstrate the superiority of decision theoretic approaches, but static estimation strategies perform well on fully connected graphs or when task time horizon is small. In the future, we anticipate scaling up our evaluation to include additional robots, verifying our algorithms in other domains, and examining alternate reward metrics, such as minimizing battery drain.

6. ACKNOWLEDGEMENTS

We would like to thank Rajiv Maheswaran, Chris Kiekintveld, James Pita, Jason Tsai and the reviewers for their helpful comments and suggestions. This work has been sponsored by the DARPA under contract FA8650-08-C-7811 and subcontracted from Lockheed Martin Cooperation.

7. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38, December 2002.

[2] R. W. Beard, T. W. Mclain, D. B. Nelson, D. Kingston, and D. Johanson. Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs. *Proceedings of the IEEE*, 94(7), 2006.

[3] J. Cheng, W. Cheng, and R. Nagpal. Robust and self-repairing formation control for swarms of mobile agents. *AAAI*, 2005.

[4] J. Cox, E. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *AAMAS*, 2005.

[5] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks*. Kluwer Academic Publishers, 2003.

[6] P. R. Freeman. The secretary problem and its extensions: A review. *International Statistical Review*, 51, 1983.

[7] S. Kozono. Received signal-level characteristics in a wide-band mobile radio channel. In *IEEE Transactions on Vehicular Technology*, 1994.

[8] V. Lesser, C. Ortiz, and M. Tambe. *Distributed sensor nets: A multiagent perspective*. Kluwer Academic Publishers, 2003.

[9] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *PDCS*, 2004.

[10] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, 2004.

[11] J. Marden, G. Arslan, and J. Shamma. Connections between cooperative control and potential games illustrated on the consensus problem. In *European Control Conference*, 2007.

[12] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[13] A. F. Molisch. *Wireless Communications*. IEEE Press, 2005.

[14] R. Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS*, 2004.

[15] J. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization. In *IJCAI*, 2007.

[16] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.

[17] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[18] N. Vlassis, R. Elhorst, and J. R. Kok. Anytime algorithms for multiagent decision making using coordination graphs. In *SMC*, 2004.

[19] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction and optimization problems. In *ICMAS*, 1996.

[20] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS*, 2003.

[21] Y. Zhang, J. G. Bellingham, R. E. Davis, and Y. Chao. Optimizing autonomous underwater vehicles' survey for reconstruction of an ocean field that varies in space and time. In *AGS, Fall Meeting*, 2005.

Multi-Agents Supporting Reflection in a Middleware for Mission-Driven Heterogeneous Sensor Networks

Edison Pignaton de Freitas
IDE – Halmstad University – Sweden - PPGC UFRGS - Brazil
PO Box 823. SE - 301 18
Halmstad - Sweden
+46 35 16 78 47
edison.pignaton@hh.se

Marco Aurélio Wehrmeister
PPGC UFRGS - Brazil
Caixa Postal 15.064 - 91.501-970 - Porto Alegre/RS
Brazil
+ 55 51 3316 3561
mawehrmeister@inf.ufrgs.br

Carlos Eduardo Pereira
PPGC UFRGS - Brazil
Caixa Postal 15.064 – 91.501-970 - Porto Alegre/ RS
Brazil
+ 55 51 3316 3561
cpereira@ece.ufrgs.br

Armando Morado Ferreira
Military Institute of Engineering
22290-270 – Rio de Janeiro/RJ – Brazil
+55 21 3873 2298
armando@ime.eb.br

Tony Larsson
IDE – Halmstad University – Sweden
PO Box 823. SE - 301 18 Halmstad - Sweden
+46 35 16 71 68
tony.larsson@hh.se

ABSTRACT

The emerging applications using sensor networks technologies constitute a new trend requiring several different devices to work together and this partly autonomously. However, the integration and coordination of heterogeneous sensors in these emerging systems is still a challenge, especially when the target application scenario is susceptible to constant changes. Such systems must adapt themselves in order to fulfill requirements that can also change during the system runtime. Due to the dynamicity of this context, system adaptations must take place very quickly, requiring system autonomous decisions to perform them without any human operator intervention, besides the first directions to the system. Thus a reflective behavior must be provided. This paper presents a reflective middleware that supports reflective behaviors to address adaptation needs of heterogeneous sensor networks deployed in dynamic scenarios. This middleware presents specific handling of users' requirements by representing them as missions that the network must accomplish with. These missions are then translated to network parameters and distributed over the network by means of the reasoning about network nodes capabilities and environment conditions. A multi-agent approach is proposed to perform this initial reasoning as well as the adaptations needed during the system runtime.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architectures and Patterns*.

I.2.9 [**Artificial Intelligence**]: Robotics – *Autonomous vehicles, Sensors*.

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems*.

General Terms

Management, Performance, Design.

Keywords

Sensor Networks, Heterogeneous Sensors, Dynamic Scenarios, Self-adaptation, Reflective Middleware, Multi-agents Reasoning.

1. INTRODUCTION

Sensor network applications are becoming more complex due to the use of different kinds of mobile and sophisticated sensors, which provide advanced functionalities [1] and also are deployed in dynamic scenarios where context-awareness is needed [2]. To support those emerging applications, an adaptable underlying infrastructure is necessary. Current proposals suggest the use of a middleware, for example TinyDB [3]. However, this kind of state-of-the-art middleware have important non-negligible drawbacks that make them useless in the context of such new emerging applications, because: (i) the assumption that the network is composed only by a homogeneous set of basic or very constrained low-end sensors; (ii) the lack of intelligence in such network compromises the adaptability required to deal with changing operation conditions, e.g. lack of QoS management and control.

Adaptability is a major concern that must be addressed in sensor networks due to two main factors: (a) long usage life time; and (b) deployment in highly dynamic environments. The first reason increases the probability of changes in user requirements through systems life time, which requires flexibility in order to comply with the changing demands. The second reason implies that applications have to be flexible enough in order to cope with drastic changes in the operation scenarios. In such environments, services are required in different places at different times; resources must be reallocated in order to fulfill specific requirements and also assure compliance with different constraints; and nodes that satisfy specific constraints during a certain period of time can become unable to continue working properly after changes. In addition, there are real-time requirements that are especially hard to be met. Thus, QoS management must be flexible, allowing renegotiation of required/ provided QoS among nodes during the system runtime [4].

This paper presents a reflective middleware aimed to support sophisticated sensor network applications that need to adapt its behavior according to changes in the environment and in the application demands. The idea is that the users specify missions to be accomplished by the network using a high-level Mission Description Language (MDL) in which they describe the desired data and constraints related to the gathering of them, for example

space and time limits, representing mission goals. In order to promote the missions accomplishment, the concept of multi-agents is used to provide the reasoning about the network and, among other things, to decide about service, resource allocation, time-related requirements and QoS control. The reasoning by the agents, i.e. the self-reflection of middleware agents, decides about what adaptations that must take place based on the mission goals. These adaptations are tuned through the use of aspects, which weave the desired behaviors into the middleware (e.g. jitter monitoring), and also through the movement of mobile-agents that change their location in the network in order to provide different services in different places as required in a context specific moment. In this paper, the focus is to present the mission-driven approach and the multi-agent reasoning based on this approach.

The remaining text is organized as follows: Section 2 presents the ideas of nodes' heterogeneity and dynamicity of operation conditions that motivate the present work. Section 3 provides an overview of the middleware structure, while Section 4 gives a summary description of the Mission Description Language. Section 5 presents the mission parameters representation. Section 6 presents the planning-agent intern model, while the multi-agents reasoning is described in Section 7. In Section 8 some related work are shortly presented, and Section 9 concludes the paper with some final remarks and directions of future work.

2. HETEROGENEOUS AND DYNAMIC

The intention of this work is to develop a flexible middleware that can be used to support applications in heterogeneous sensor networks deployed in dynamic environments. In the context of this work, heterogeneity means that nodes in the network may have different sensing capabilities, computation power, and communication abilities, running on different hardware and operating system platforms.

Low-end sensors are those with constrained capabilities, such as piezoelectric resistive tilt sensors, with limited processing support and communication resource capabilities. Rich sensors comprehend powerful devices like radar, visible light cameras or infrared sensors that are supported by moderate to high computing and communication resources. Thus, in order to deal with these very distinct capabilities, the proposed middleware must be lightweight, while being scalable and customizable. The mobility characteristic is also related to the heterogeneity addressed by the middleware. Sensor nodes can be static placed on the ground or can move themselves on the ground or fly at some height over the target area in which observed phenomenon is occurring. Figure 1 illustrates the heterogeneity dimensions considered in this work.

The proposed middleware is aimed to support applications that deal with dynamic and changing scenarios. Consequently, the set of sensors chosen in the beginning of a mission may not be the most adequate one during the whole mission. For example, an area surveillance system receives the mission to survey an area that may not allow traffic of certain kinds of vehicles. Ground sensors are set to trigger and send an alarm in the presence of unauthorized vehicles. Then Unmanned Aerial Vehicles (UAVs) equipped with visible-light cameras are set to fly over the area where the ground sensor has issued an alarm to verify the occurrence. However, a sudden change in the weather, e.g. the area becomes foggy or cloudy, turns visible-light cameras useless. However the detection mission must still be accomplished. This type of change in the operational conditions during a mission must be handled by the middleware. It must be able to choose the best alternative of employable sensors among the set of available options. In the described situation it may choose, for instance, an infrared camera instead of the visible-light one.

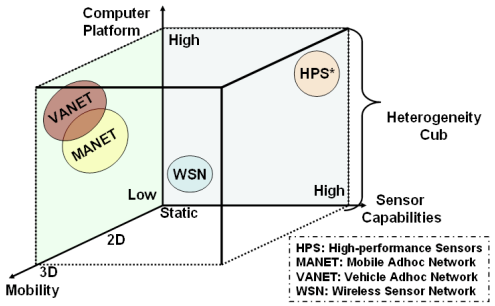


Figure 1. Heterogeneity Dimensions

According to the taxonomy presented in [5], the sensor network described above can be classified as: a mix of static and dynamic configurable sensors with full self-awareness; a heterogeneous dynamic ad-hoc network with a large number of nodes; deployed in a high dynamic environment partially observable; and which achieve its goals by collectively coordinated actions with a non-local environment dependency. A sensor network with this classification requires a great flexibility in its behavior and at the same time "in-network intelligence", which is represented by the spread of intelligent capabilities over its nodes. These two features, flexibility and in-network intelligence, enables reflection about network status and environment conditions in order to adapt the network for the mission and to new demands from end-users.

3. MIDDLEWARE STRUCTURE

The main goal is that the proposed middleware fits both resource constrained and rich sensors. In order to achieve this goal, aspect- and component-oriented techniques are used in a way similar to the approaches discussed in [6], and [7] and the mobile and multi-agents approach presented in [8].

The proposed middleware is divided in three parts or layers, see also Figure 2:

Infrastructure Layer. It is responsible for the interaction with the underlying operating system and for the management of the sensor node resources, such as available communication and sensing capabilities, remaining energy, etc. This layer also helps to coordinate resource sharing according to application needs passed through the upper layers. Additionally, services provided by upper layers may also need some resource sharing support.

Common Services Layer. It provides services that are common to different kinds of applications, such as QoS negotiation and control, quality of data assurance, data compression, and the handling of real-time requirements. Other concerns such as deadline expiration alarms, timeouts for data transmissions, number of retries and delivery failure announcements, resource reservation negotiation among applications (based on priorities established by missions and operation conditions), bindings, synchronous and asynchronous concurrent requests are also handled within this layer. Readers specially interested in those concerns are referred to [9] for more details about the mechanism used in the middleware to provide these features.

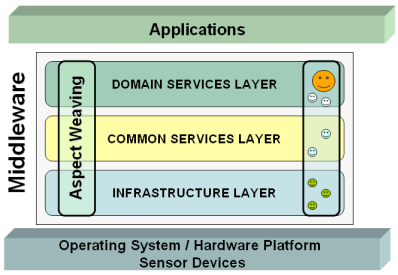


Figure 2. Overview of the Middleware Layers

Domain-Services Layer. It supports domain specific needs, such as data fusion and specific data semantic support, in order to allow the production of application-related information from raw data processing. Fuzzy classifiers, special kinds of mathematical filters (e.g. Kalman Filter) and functions that can be reused among different applications in the same domain are found in this layer.

Multiple applications performing different missions can run concurrently in the network. As stated before, the middleware handles resource and data sharing among applications, which need the same type of data, allowing a better energy management in resource constrained nodes. In powerful nodes, with more energy available, the middleware can provide more complex services aiming at the handling of rich data, such as those related to image processing, and pattern matching. This also means that such nodes can take some of the burden from low-end nodes.

"Smile faces" in the Figure 2 represent agents that can provide specific services in a certain node at a certain moment of system runtime. A special region (called *agents-space*) links agents throughout layers, allowing information exchange. The *Domain-Services Layer* hosts a special agent (bigger smiling face), which is responsible receiving the mission directions from the application layer, and based on that, plan and reason about the activities related to the sensing missions. This agent is called *planning-agent*, and details about it will be presented later in the following sections. The other agents (small smiling faces) are used to provide specific services that support applications.

Non-functional concerns that affect elements in more than one layer of the middleware, such as security, are represented as cross-layer features, which are addressed, at least partly, with the aspect-oriented approach presented in [6].

4. MISSION DESCRIPTION LANGUAGE

The Mission Description Language (MDL) provides means to describe the information requested or to be monitored about certain detectable phenomena in a given time-space domains interesting to the end-users. For instance, the user may want to know about the different kinds of vehicles that pass through a given area during a certain time, or the environmental conditions during the occurrence of a pre-defined event. By using the MDL to setup a mission to the network the user "tests" the environment in order to achieve the desired information about a phenomenon or an event of interest. The idea of "test" the environment is based on the C/ATLAS test language [10], in which high-level test commands are specified in order to retrieve information about devices in a system. In a similar way, the MDL provides high-level commands to retrieve specific information about matters or changes that occur in the environment. Based on this idea, the MDL uses patterns and definitions to test the environment in order to gather information that matches with those patterns and definitions that describe the user's need for information. However, it is important to highlight that in this proposal the MDL uses just this conceptual idea behind C/ATLAS, it does not use the terminology presented by the test language, neither the same taxonomy.

By using the MDL, the user defines high-level statements, which define and describe the events of his/her interest, as well as the constraints that are linked with that specific sensing mission. For instance, the maximum tolerated delay to receive an alert or the maximum amount of energy that can be used for that mission, among others. Another important concept in the MDL is the mission priority ordination, which allows several missions to run at the same time in the network, but prioritizing those which are more important, according to the user's definitions. Linked with the former idea and the constraints enforcement is the usage of policies to govern the performance of missions. The user can order the mission accomplishment according to their priorities, selecting some constraints and also link a policy that will dictate how persistent the nodes in the network will be in order to gather the requested information. For instance, in an aggressive policy, nodes can deplete their batteries in order to assure that the requested data will be delivered to the end user (i.e., by performing several retransmissions until the end-user receives the information or the battery is depleted). On the other hand, in a less aggressive policy, nodes may preserve their batteries in spite of that they cannot assure the data delivery. The user can also use a policy and "tune" it by means of specifying specific constraints that override the general policy for those specific parameters.

The mission described as a set of MDL statements is then translated to parameters that will configure the system as to retrieve the information desired by the user. The definition of these parameters is done by the interpretation of the MDL statements, together with the analyses of the characteristics of the deployed network and the chosen policy, if any, and the priority level. A configuration console (Mission Specification Console) enables the user to enter this information, which will be translated in a tuple of parameters (the content of this tuple will be explained in Section 5) representing the mission that will be injected into the network. Figure 3 illustrates the configuration console and its components.

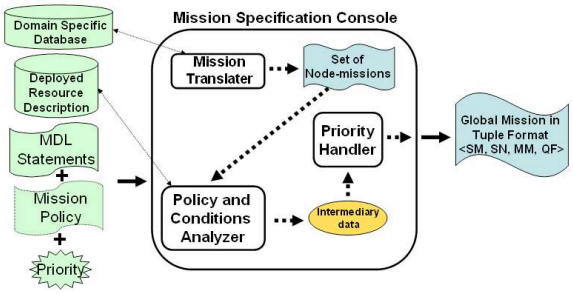


Figure 3. Mission Specification Console

The MDL statements are the essential elements used to define the overall mission (Global Mission) of the network. The Global Mission will be divided into node-missions (sub-missions), which will be executed by specific nodes or a group of nodes in the network. The subdivision of a mission in node-missions is an

important part of the Global Mission translation in system parameters. It is done by a component called Mission Interpreter, which takes the MDL statements as input, consult a domain-specific database (for information about a particular domain), and translate these statements into node-missions, see Figure 4.

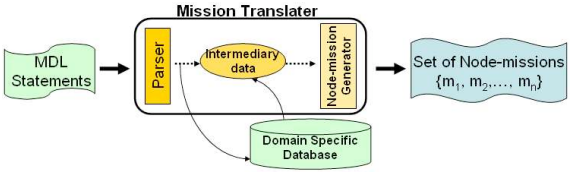


Figure 4. MDL Interpretation and Node-missions generation

The MDL is divided in two parts: (1) the kernel of the language, which defines events of interest, space and time parameters, as well as the priority of the sensing mission; and (2) the extensions, which define advanced parameters, such as accuracy, precision and constraints. It is composed by: imperative commands (i.e. SCAN and DETECT); keywords (i.e. pattern and object), which are the parameters of the commands; connectors to link commands and keywords (i.e. IF and WITH); and pre-defined patterns, which are in fact a kind of keyword that are stored in a domain library (i.e. FOG and LINEAR_MOVEMENT). As an example, the following represents a conditional MDL statement:

IF **DETECT** <DECREASE_OF <temperature>>
WITH GRANULARITY<3> **MONITOR** <FOG>
WITH ACQUISITION <period = yy>

This example statement means: if a 3ºC decrease in temperature is detected, then provide monitoring of the pre-defined pattern "FOG" with data acquisition by the sensors each "yy" time units.

5. MISSION PARAMETERIZATION

The input to the sensor network system, coordinated by the proposed middleware, is seen as a "mission" that the whole network must to accomplish in cooperation. By using a language such as MDL the user can specify necessary "data" for such a mission at a high level of abstraction. The user thus specifies the goals and priorities for the missions' directions in an MDL file, which after translation will drive the mission implementation based on reflection about the network, in order to accomplish with the users' requirements. The reflection consists of analysis and reasoning performed inside autonomous network nodes in order to allow the adaption required to face the changes in dynamic scenarios and users' requirements. Node-mission responsible agents, called **planning-agents**, reason about the network adaptations based on the mission directions and the network actual state. The former gives the requirements for data from users' point of view, while the latter is mainly characterized by nodes availability and environmental conditions. Their reasoning is made by a construction of believes about the network and its environment, which will help to achieve their goals and like this, comply with the network global mission.

The representation of the mission is provided by a set of goals that each planning-agent desires to achieve, supported by a set of "known facts" that they have about the network and the environment. Based on these facts and their goals, the planning-agents establish activity plans to achieve their goals, negotiating the best distribution of the work that must be done in order to accomplish with the global mission.

The network global mission is divided in sub-missions, called node-missions, which are assigned to planning-agents present in each individual node. Each node has just one planning-agent (placed in the *Domain Services Layer* of the middleware), thus for comprehension of the remaining text, a node-mission is assigned to a node or to the planning-agent installed on it. The accomplishment of each node-mission will corroborate for the success of the global-mission. In order to complete their node-missions, planning-agents break the node-mission into minor tasks that are related to the individual devices inside the node. A hierarchy among these concepts can be drawn: at the top is the global mission, followed by the node-missions, which is divided in several tasks in the node abstraction level. In the following, a formalization of these concepts is presented.

A **Global Mission** is represented by a tuple composed by the sets called SM and SN, and also the functions MM and QF. SM is a sub-set of all possible node-missions that could be assigned to a node; SN is a sub-set of all the nodes in the network; the mapping function MM maps elements of SM into elements of SN; while the quality function QF evaluates the mapping provided by MM. It is represented by:

$$GM = \langle SM, SN, MM, QF \rangle$$

M is the set of all possible node-missions;

SM is a set of node-missions (sub-set of ***M***) that can be assigned to the nodes members of the set ***SN***:

$$SM = \{m_i \mid m_i \in M\},\ i \in \{1,...,I\},$$

where ***I*** is the total number of all possible missions, i.e. the number of elements of set *M* or *SM*⊂*M*; Each **node-mission** m_i is represented by a tuple composed by a set of **measurements** that must be provided (***SME***), a set of **conditions** to the measurements (***SMC***), and a relation ***C*** that maps the set of conditions in the set of measurements:

$$m_i = \langle SME, SMC, C \rangle,$$

where ***SME*** is sub-set all possible measurements (***ME***); and ***SMC*** is the sub-set of all possible measurement conditions (***MC***),such as those related to periodicity, accuracy, time interval, range, among other. Such that SME ⊂ ME, and SMC ⊂ MC. ***C*** is the relation that maps conditions into measurements, where one measurement can be linked to none or several conditions. The opposite is also valid, i.e. one condition can be linked with none or several measurements:

$$C = \{r(mc_k) = me_j \mid mc_k \in SMC, me_j \in SME\},$$
$$k \in \{1,...,K\},\ j = \{1,...,J\},$$

where ***K*** is the total number of possible measurements conditions in the network (number of elements of the set ***MC***); and ***J*** is the total number of measurements in the network (number of elements of the set ***ME***).

N is a set of all nodes that compose the network.

SN is a sub-set of nodes in the network (a sub-set of N):

$$SN = \{n_v \mid n_v \in N\}, v \in \{1,...,V\},$$

where ***V*** is the total number of nodes in the network (the number of elements of the set N) or SN ⊂ N.

MM is the **mission-mapping function** that maps each node-mission to a certain node. A node in ***SN*** can perform one or more node-missions, but each node-mission is atomic (from the entire network point of view), i.e. it can be assigned to only one node:

$$MM = \{f(m_i) = n_v \mid m_i \in SM, n_v \in SN\},$$
$$i \in \{1,...,I\}, v \in \{1,...,V\}$$

QF is a function that evaluates the mapping provided by ***MM***, given a grade between 0 and 10 for each par (m_i, n_i):

$$QF = \sum x_i$$

where $g(m_i, n_v) = x_i \mid m_i \in SM, n_v \in SN, x_i \in [0,10]$.

In order to achieve the goals of an assigned node-mission, a node must perform several different smaller tasks, called **node-tasks**. To read a value from the sensor device, or to turn a sensor device on/off are examples of **node-tasks**. At the node abstraction level, a specific **node-mission** is a sub-set of all **node-tasks** that a given node can perform, represented formally by:

$$nm = \{t_w \mid t_w \in T\}, w \in \{1,...,W\}$$

where ***W*** is the total number of possible node-tasks that any node can perform (the number of elements of the set ***T***) or $nm \subset T$ where ***T*** is the set of all node-tasks that can be performed by any node.

6. PLANNING-AGENT MODEL

The proposed approach uses different kinds of agents; both cognitive and reactive ones, in order to perform different activities in the middleware, from the provisioning of simple services to complex reasoning about the network setup. To keep attention on the focus of this paper, only the model of the planning-agent, which is a cognitive agent, will be presented.

The model used in the present approach for the cognitive agents is based on the model of mental attitudes, known as BDI model (Believes-Desires-Intentions) presented in [11]. The BDI approach appears to suite well to the problem addressed by the current work, as some decisions that must be taken by the agents in the proposed approach require cognitive skills to "wonder" if certain actions are adequate to achieve a desired result, based on knowledge about conditions that may interfere on the performance of those actions. In the current problem formulation, what is desired is to obtain information by means of sensing activities, which are the goals of a sensing mission. Such knowledge is the "believe" that the node has about the relevant conditions and the intentions are translated into the actions need to retrieve the desired information. It thus seams that this model fits well to the goals of the proposed approach.

However, it is important to highlight that the approach used in this work is slightly different from the traditional BDI frameworks, such as [12] and [13], or more complex teamwork models, such as those presented in [14]. The major difference is that the model presented in this paper is focused on sensor networks activities, in which the network nodes do not perform any action that changes the world around them, what simplifies the model by eliminating the assumptions about this aspect. Besides, the proposal herein is simpler than those presented in the works mentioned above, as one can see in the remaining text.

The planning-agent has a complex "mental" activity, being responsible for different kinds of reasoning related to the mission accomplishment. It communicates with all other kinds of agents in the system. Besides, it negotiates with other planning-agents installed in the other nodes about the distribution of the node-missions. During these negotiations, it gathers information about the other nodes in order to achieve necessary knowledge about the network. It also has to maintain and update information about its own state, in order to inform other planning-agents and be capable to take right decisions. Environment conditions are also important in some of the deliberations taken by this agent. So this kind of information also constitutes its mental state, more precisely, as a part of its beliefs. In the following, a description about the beliefs, desires and intentions of the planning-agent, as well as a description of its plan and actions is provided.

Beliefs: Basically consisting of four groups of information: 1) background information, such as maps of the region; 2) the planning-agent's own conditions, translated in terms of the actions that it can perform and the node status (energy level, devices status, location, installed services, agents hosted in the node, etc); 3) other nodes status; 4) environment conditions.

Desires: The planning-agent has two types of desires: 1) General-Desires: which correspond to "built-in" goals, such as: distribute the node-missions in order to achieve the best overall result efficiently, and cooperate with other nodes; and 2) Specific Goals: which are related to the assumed node-missions and that come to its desires' set when it assumes the responsibility of a given node-mission ($\mathbf{m_i}$). These goals are ranked according to the related node-mission priority. It will be used to drive the construction of the plans that governs the execution of the agent's actions.

Intentions: Following the same idea of the desires, the planning-agent has two types of intentions: 1) General Intentions: which are directly related to the built-in goals, such as: to have an agreement about which that node will take the responsibility of a given node-mission after a negotiation with other nodes; to have provided the required resources to a requesting node; to have provided the correct information to other agents about data of interest; and 2) Specific Intentions: which specify intentions related to actions needed to accomplish a given node-mission, such as: to have sent the samplings with the correct accuracy within the timing constraints, which comes to the agent via the sets *SME* and *SMC* to the corresponding $\mathbf{m_i}$ that it assumes to accomplish.

Actions: Operating System or direct device drivers calls to perform commands on the underlying software and hardware platform; send and receive messages to and from other agents (request, inform, reply, notify, subscribe, publish, propose, reject, accept). Particularly in the negotiations occurring during the reasoning, the types of messages used are: inform, propose, accept or reject.

Plan: A plan is described in terms of a sequence of actions that an agent perform in order to achieve a sub-goal, decided by its deliberation and related to its intentions, and ultimately to achieve a motivational goal related to its desires. In order to accomplish with a given node-mission, the agent choose specific tasks ($\mathbf{t_w}$) such as they form a set that fulfill that node-mission. A plan will be a list of tasks that have to be done in order to accomplish with the node-missions allocated for that node. If a new node-mission is assumed and some of the tasks that are required to accomplish it are already in the plan, there is no need to insert them again in the plan, as their results are reused for this new node-mission. The

planning-agent needs to construct a new plan, which can be totally new or at least a reviewed version of the current one, each time at least one of the following events occurs: it assumes a new node-mission; the conditions of the environment changes; a change in the network or in the user requirements occurs. This reflects the flexibility of the network to adapt itself according to the dynamicity of the network operation.

6.1 Architectural Structure

After the presentation of the cognitive planning-agents' internal model, its architectural structure can be described, based on the BDI architecture presented in [15], and is shown in Figure 5.

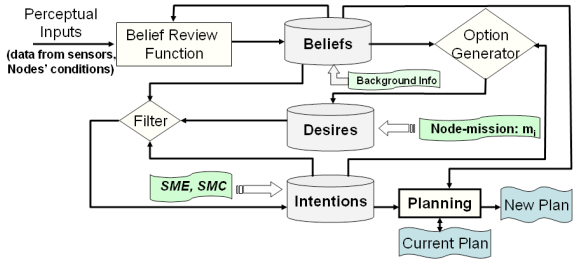


Figure 5. Planning-agent Internal Architectural Structure

In Figure 5 is shown that the agent takes the perceptual inputs (changes in the network or in the environment, data from its sensors, etc) and its current beliefs, and performs an update of their beliefs by means of the **Belief Renew Function**. After analysis of the updated beliefs and current intentions, the **Option Generator** function selects a sub-set of the desires representing the next possible goals to perform. Its beliefs, desires and intentions are then used as inputs to a **Filter** function that represents the deliberation of the agent and will provide the update of its intentions. The planning-agent constructs its own plans by reasoning about its current intentions and its beliefs. Ultimately the generated plans will fulfill with goals defined in its desire base, as the current intentions were decided based on the analysis of the set of the possible goals, by the use of the Option Generator. The result of the **Planning** is the selection of actions that are needed to perform the current intentions. It considers the current plan and beliefs, in order to select and order the execution of the actions. The current plan is used to reuse some previous decisions, and the current beliefs to evaluate which actions are more adequate to take in relation to the current conditions.

7. MULTI-AGENT REASONING

As stated above, the planning-agents construct believes that will guide their decisions based on the mission needs (the goals in their desires), which are characterized by node-missions. As the network receives a global-mission, the nodes will try to find a best fit to accomplish this mission, which characterizes the mission setup reasoning. Their decisions will influence the mission mapping function (***MM***), which may change in case of adaptation during the system runtime, due the adaptation reasoning. The mechanisms for the network setup and adaptation are described in the following.

7.1 Mission Setup

When a mission is received, the reasoning required to perform the network setup is divided in four steps, which are explained more in detail in the following.

Step One: each node performs an analysis of the elements of the set ***SM***, as well as its capabilities and the surrounding environment. If a node can provide the measurements of the set ***SME*** for a certain node-mission $\mathbf{m_i}$, satisfying the respective relation ***C***, it "declares" itself as "candidate" to perform $\mathbf{m_i}$. At this time, each node constructs a partial belief in relation to $\mathbf{m_i}$, only based on its own knowledge about the network, which is composed by the mission needs and its own capabilities.

Step Two: if a node considers itself as "candidate" to accomplish $\mathbf{m_i}$, it informs its "candidacy" to the other nodes, using an "inform message". However if the node does not consider itself as "candidate", it will just listen for the "candidacy" of other nodes.

Step Three: after a pre-established time-out, if no one considers itself as "candidate", no message will be exchanged. Thus, all nodes that can provide the data required by the measurements described in the set ***SME***, but that cannot satisfy the relation ***C***, communicate with the other nodes informing about the conditions that it can satisfy. The node, which provides the assurance closest to the desired one (specified by ***C***), takes the node-mission. This characterizes a best-effort way to solve the problem.

Step Four: nodes analyze their own conditions as well as conditions of the others, deciding which one must take $\mathbf{m_i}$. Such analysis uses the quality function ***QF*** and the node-tasks needed to perform $\mathbf{m_i}$. By maximizing $\mathbf{g(m_i,n_v)}$, nodes know which one ($\mathbf{n_v}$) will be in charge of the node-mission $\mathbf{m_i}$. If two nodes are capable to accomplish the node-mission, the one that has best conditions, e.g. remaining energy and/or other influencing parameters, takes the responsibility for that node-mission. In other words, the function **g** has a higher value for that node in comparison with the others. With this information, the nodes construct a common belief. If two nodes have the same value for the function **g** for the same node-mission, one of them is then randomly chosen.

Communication in wireless sensor networks can face problems that compromise message delivery. In case, any message of the coordination protocol is not received by any node in any of these steps, the node acts according to its belief from the last received message, if any. If it does not receive any message, it will act according to its initial partial belief. When the communication is reestablished, nodes will "listen" to the passing messages related to the same node-mission ($\mathbf{m_i}$) measurements, and then they will redo the above steps in order to achieve a new global-belief.

If compared with a centralized task distribution, the advantage in performing this reasoning in a distributed way is to avoid communication to and from the base station, which would consume more energy if compared with the local computation of the task allocation. As presented in [22], communication is the main source of energy consumption in sensor nodes, so communicating requires more energy than computation. In order to achieve a centralized task distribution that has the same quality as a distributed one can achieve; data about the current status of the nodes have to be sent often to the base station, which would increase the energy consumption. On the other hand, by the use of the distributed approach as presented, the nodes decide locally how to divide the new job, according to their status, without the need to send information through the network to the base station. This same argument holds to support the distributed way in which the adaptation is done, which is presented in the next sub section.

7.2 Mission Adaptation

During the system runtime, mission requirements and/or operational conditions can change. Nodes can perceive these changes, which induces node believes to be updated. If a change makes a node unable to proceed in the mission accomplishment, the network must adapt itself to solve the problem. The reasoning performed by the planning-agents will try to find another node that can perform node-mission $\mathbf{m_i}$ in the place of the previous node. This reasoning is similar to the one presented above, but there are two different circumstances that also must be taken into account: (1) the node simply fails; (2) the node continues to work, but is aware that it cannot continue performing the mission.

Considering the first case, faulty nodes are perceived by other healthy nodes, which have participated with the faulty one in the initial mission establishment reasoning. As nodes can perceive that the node responsible by the node-mission $\mathbf{m_i}$ is not responding during an established time-out period, they redo the reasoning to decide which one must perform $\mathbf{m_i}$. Information about the failure is added to the belief of these healthy nodes.

In the second case, the node that becomes unable to accomplish $\mathbf{m_i}$ informs this situation to the nodes that participated in the mission establishment reasoning. Further they decide which one will take the node-mission previously assigned to that node.

Another situation that requires adaptation is when changes make other nodes (more) capable to accomplish a certain node-mission $\mathbf{m_i}$. An adaptation can be triggered if the node-mission was previously assigned in a best effort way, as explained in the step three in section 3.2. The need for a best service can also trigger the adaptation, foreseeing a possible increase in the users' requirements priority. The mechanism of these changes is implemented by an exchange of "proposal" and "accept" or "reject" messages.

Adaptations decisions are also based on the quality function ***QF***. It is done during the establishment of the best mapping of node-missions to nodes as explained in the section 3.2. The target is always to maximize the value of ***QF*****,** what can be achieved by maximizing the function $\boldsymbol{g(m_i,n_v)}$ for each node-mission $\mathbf{m_i}$: i.e. $\boldsymbol{max(g(m_i,n_v))}$.

As result of the procedure explained above, all nodes know which node $\mathbf{n_v}$ has taken the responsibility for the node-mission $\mathbf{m_i}$, (similarly to the establishment of the node-missions mapping). Therefore, nodes' beliefs are updated with this knowledge. In the same way as explained before, if two nodes have an equal value of function **g** for a given node-mission, one of them is chosen randomly.

7.3 Considerations about Complexity

As the middleware is intended to run in a variety of nodes, from resource constrained nodes to resource rich ones, the mechanisms presented above have to be customized for each type of node.

Considering resource rich nodes, function **g** used to evaluate the quality of a given mapping may be an elaborated and computing intensive algorithm. However, when it comes to the low-end nodes, simpler functions may take place in order to perform the evaluation of the part of the mission related to them. The same way that there is a tradeoff between energy consumption and communication resource usage, there is also a tradeoff in the amount of resources that should be used and the quality of a solution provided by the used algorithm. It is possible that an optimal solution is not achieved by a simpler algorithm, but considering the resource constraints, a sub-optimal can be better than an optimal one that depletes the available resources.

The same kind of variation is present in the internal components of the planning-agent that inhabit different types of sensor nodes. The above explained Option Generator Function, Filter Function provides for that the Planning can be much richer, considering much more parameters and having more complex algorithms in the resource rich nodes, if compared with the same functions in the low-end nodes. However, it is important to highlight that every node of a given kind use the same set of functions, so the coherence is maintained.

8. RELATED WORKS

Agilla [16] is one of the precursors in the use of mobile agents in middleware for WSN. Its approach is to use agents that can move from one node to another in the network. It also allows multiple agents to run in the same node. These characteristics provide the desired features of energy saving, as the agents can run near to the data avoiding unnecessary communication. In comparison with the proposed approach, the use of agents is not restricted to moving and using services around the network but also to help in the network reflection and decision for adaptability using multi-agents.

In [17] a proposal to use a distributed mechanism to control adaptive sampling to support energy-constrained network operations is presented. In the proposal, each sensor is considered an autonomous agent, enabling decentralized control of the sampling rate of sensor nodes in the application domain of flood monitoring. Besides the contribution in the increase the efficiency of the energy consumption, the goal of this approach is also to maximize the information value of the data collected to the base-station. The major differences between our work and the one mentioned above are respectively: the consideration of a heterogeneous sensor network instead of a homogenous one; the domain independent instead of a domain-specific approach; and the direct cooperation among the agents instead of just the decentralization of the problem.

AWARE [18] is a project that proposes a middleware whose goal is to provide integration of the information gathered by different type of sensors, including WSN and mobile robots. Our proposal aims also at addressing heterogeneous sensors, but also concerns like QoS, as presented in [9], and runtime reflection to address changes in the environment and in the network. Moreover, our approach provides the capability of autonomy to the network nodes, by using an agent-orient approach. In the referred middleware, the nodes do not have the same capability.

In [19] an approach that uses Artificial Intelligence to configure an underlying middleware is presented. This approach uses the concepts of missions and goals to plan the allocation of tasks in a network of homogeneous nodes. The handling of heterogeneous nodes is one of the differences between the referred work and the one presented in this paper. Additionally, in that work, the intelligence is outside the middleware by means of just sending "commands" or adjusting its parameters. In our presented approach, agents make part of the middleware, spreading intelligence over the network.

In [20] an information processing paradigm for intelligent sensor networks is presented. Nodes in sensor networks have different levels of autonomy in terms of the signal processing, information fusion and situation assessment in order to contribute with the overall system decision making. This approach is based on the use of a genetic algorithm to provide learning features to the sensor nodes, and fuzzy cognitive maps to perform situation assessment. The sensor networks aimed by this work are those composed only by rich nodes, as the architecture and the techniques used are quite heavy to fit in low-end nodes. On the other hand, the proposal of the present paper is to address heterogeneous sensor networks that are composed by both low-end and rich nodes, allowing them cooperate in order to achieve the overall mission goals.

9. CONCLUSION AND FUTURE WORK

This paper presented the concepts of a middleware needed to address mission-driven heterogeneous sensor networks deployed in highly dynamic scenarios. These scenarios require middleware reflection to support adaptations to face constant changing conditions during runtime. Multi-agents reasoning is used in order to setup, configure and reconfigure the network. Besides, a formal definition of the mission statements and conditions (described in MDL) was presented, as well as its mapping to elements of a BDI approach that supports the proposed network wide reasoning.

The direction of the ongoing and future work includes enrichment of the Mission Description Language specification, adding abstractions that can help the user specify missions. Another ongoing work is the implementation of the simulation to provide results that validate the presented ideas. In order to do that, the adaptation of a simulator for wireless networks called Shox [21] is being done. This adaptation consists of the inclusion of the agents concepts in the simulator framework, and an interface with the Mission Specification Console.

10. ACKNOWLEDGMENTS

E. P. Freitas thanks the Brazilian Army for the given grant to follow the PhD program in Embedded Real-time Systems in Halmstad University in cooperation with UFRGS in Brazil.

11. REFERENCES

[1] Culler, D., Estrin, D. and Srivastava, M. Overview of sensor networks. *IEEE Computer*, vol. 37, no. 8, pp. 41–49, 2004.

[2] Henricksen K. and Indulska, J. A software engineering framework for context-aware pervasive computing. In *Proceedings of PerCom*, pages 77–86. IEEE Computer Society, March 2004.

[3] Madden, S., Franklin, M. J., Hellerstein, J. M. and Hong, W. TinyDB: An acquisitional query processing system for sensor networks. ACM *Transactions on Database Systems*, 30(1):122–173, 2005.

[4] Liberatore, V. Implementation challenges in real-time middleware for distributed autonomous systems. In *Proceedings of Second IEEE SMC-IT*, 2006.

[5] Vinyals, M., Rodríguez-Aguilar, J.A. and Cerquides, J. A Survey on Sensor Networks from a Multi-Agent perspective. In *Proceedings of 2nd International Workshop on Agent Technology for Sensor Networks (ATSN-08),* 2008.

[6] Freitas, E. P., Wehrmeister, M. A., Pereira, C. E., Wagner, F. R., Silva Jr., E. T., Carvalho, F. C. DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design. In *Proceedings of 10th International Workshop on Early Aspects*, Springer, 2007, pp. 55-74.

[7] Tesanovic, A. et al, Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software. *Journal of Embedded Computing*, IOS Press, v.1, n.1, 2005.

[8] Freitas, E. P., Wehrmeister, M. A., Pereira, C. E. and Larsson, T. Reflective middleware for heterogeneous sensor networks. In *Proceedings of 7th Workshop on Adaptive and Reflective Middleware (ARM'08)*, ACM. 2008. pp. 49-50.

[9] Freitas, E. P., Wehrmeister, M. A., Pereira, C. E. and Larsson, T. Real-time support in adaptable middleware for heterogeneous sensor networks. In *Proceedings of International Workshop on Real Time Software (RTS'08)*, IEEE. 2008. pp. 593-600.

[10] IEEE Std 716-1995, 1995. IEEE standard test language for all systems-Common/Abbreviated Test Language for All Systems (C/ATLAS), IEEE, Inc.

[11] Bratman, M. E. *Intention, Plans, and Practical Reason*. Cambridge, MA, 1987.

[12] Cohen, P. R. and Levesque, H. J. Teamwork. *Nous*, 25(4):487–512, 1991.

[13] Grosz, B. and Kraus, S. Collaborative plans for complex group actions. *AIJ*, 86:269–358, 1996.

[14] Pynadath, D. V. and Tambe, M. Multiagent teamwork: analyzing the optimality and complexity of key theories and models. *Proceedings of 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*. ACM. 2002. pp. 873-880.

[15] Weiss, G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

[16] Fok, C.-L., Roman, G.-C. and Lu, C. Rapid development and flexible deployment of adaptive wireless sensor network applications. *Proceedings of the 24th ICDCS'05*, 2005.

[17] Kho, J., Rogers, A. and Jennings, N. R. Decentralised Adaptive Sampling of Wireless Sensor Networks. *Proceedings of 1st International Workshop on Agent Technology for Sensor Networks (ATSN-07)*, 2007.

[18] Gil P. et al. Data centric middleware for the integration of wireless sensor networks and mobile robots. In *Proceedings of 7th ROBOTICA'07*. 2007.

[19] Schmidt D. C. et al. A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-Time Applications. *Proceedings of 8th ISADS'07*. 2007. pp.461-472.

[20] Leung, H., Chandana, S. and Wei, S. Distributed sensing based on intelligent sensor networks. *IEEE Circuits and Systems Magazine*, 8(2). pp. 38-52, 2008.

[21] Lessmann, J., Heimfarth T. and Janacik, P. ShoX: An Easy to Use Simulation Platform for Wireless Networks. In *Proceedings of Tenth International Conference on Computer Modeling and Simulation*, 2008. pp. 410-415.

[22] Akyildiz, I. F., Weilian S., Sankarasubramaniam, Y., Cayirci, E. A survey on sensor networks. *IEEE Communications Magazine*, 40(8). pp. 102-114, 2002.

Agent-based Sensor-Mission Assignment for Tasks Sharing Assets*

Thao Le
Department of Computing Science
University of Aberdeen
AB24 3UE, UK
thao.le@abdn.ac.uk

Timothy J. Norman
Department of Computing Science
University of Aberdeen
AB24 3UE, UK
t.j.norman@abdn.ac.uk

Wamberto Vasconcelos
Department of Computing Science
University of Aberdeen
AB24 3UE, UK
wvasconcelos@acm.org

ABSTRACT

A sensor network may be required to support multiple mission to be accomplished simultaneously. Furthermore, the environment may change at any time; i.e. a new mission may arrive at any time. In solving this many-mission, many-sensor problem in dynamic environments, conflicts between missions may occur for the use of sensor resources. A mechanism to match sensor resources to mission demands thus becomes necessary. In this paper, motivated by the conservation of resources, we consider the problem of sensor-mission assignment, in which sensors may be shared and reassigned between tasks. To achieve this, sensors are represented by agents, which coordinate to establish virtual organizations to meet mission requirements. The agent coordinating the achievement of a mission utilises a novel multi-round, Knapsack based algorithm, GAP-E, to allocate sensor agents to tasks based on bids received. Through simulations, we empirically demonstrate that this model provides a significant improvement in the number of completed missions as well as execution time.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Sensors*

General Terms

Design, Measurement, Experimentation

Keywords

Sensors, Dynamic allocation, Resources sharing

1. INTRODUCTION

When a sensor network is deployed it is typically required to support multiple simultaneous missions. A given sensor may be beneficial to some missions, providing varying amounts of information to each one. Missions, on the other hand, can appear at any time and may place varying demands on sensors. In such multiple sensors and multiple missions problems in dynamic environments, conflicts between missions may occur for the use of the same sensor resources. Thus, we need efficient mechanisms to assign individual sensors to appropriate missions on the basis of information need.

An additional pragmatic problem arises in this domain. Due to the energy limitation and also to prolong the lifetime of the sensor network, conservation of energy consumed is an important consideration in managing wireless micro-sensor networks. However, to the best of our knowledge, in existing sensor-mission assignment approaches, each asset may be assigned to only one mission at any one time. The fact that assets may not be shared can waste energy since there might be more than one mission that requires the same kind of information which can be provided by a single sensor. Making decisions on how best to utilize limited sensor resources in order to satisfy mission demands without conflict and without wasting resources due to redundant assignment is, therefore, the key issue in sensor-mission assignment

Motivated by such necessity, we find that allowing sensors to be shared and to be reassigned between multiple tasks may provide substantial savings in sensor battery, often leading to improvements to the network's ability to meet its global objectives. In order to realize the idea, we need to decompose each mission to a set of specific tasks so that we can identify which tasks can share assets and which assets should or could be reassigned as circumstances change.

Within the model presented here, sensors are represented in the system by agents. These "sensor-agents" decide autonomously whether to offer to become involved in a mission depending on both their available resources and, in a broader sense, the environment that they are situated within. They communicate with each other, passing and sharing information when needed. In our context, they operate in a cooperative manner in which their resources are contributed toward achieving the global objective: the successful allocation of the most appropriate sensors to a mission according to its information requirements.

For each task, the sensor-agent located closest to its centre will act as the coordinator for that task; each task represents an information need within a geographical area at a certain

*The first author is a PhD student

time. The coordinator agent initiates interaction with other sensor agents within the scope of the task using a variant of the well-known contract net protocol [13]. The coordinator issues a call for bids for the delivery of information pertaining to the needs of the task. Each sensor agent receiving this call will analyze the information requirements of the task and make a bid only if it decides that it can satisfy the request based on its type and current workload. On the basis of the bids received, the coordinator agent utilises a novel multi-round, Knapsack-based algorithm, GAP-E, to allocate sensor agents to that task.

In this paper we make the following contributions to the state of the art. First, our solution allows the sensors to be shared between tasks, significantly improving the percentage of successfully allocated missions. Second, the search space of the problem and, consequently, the time consumed to find a solution, are greatly reduced. This increase the reliability of the system in practical situations.

The remainder of this paper is structured as follows: Section 2 details our approach including the novel GAP-E algorithm and how it is employed to provide our solution to the sensor-mission assignment problem in dynamic environments. In Section 3 we present a rigorous evaluation of our approach with varying sensor asset availability and varying mission arrival rates against existing solutions and an estimate of the optimal solution. Section 4 introduces the assignment problem within the broader context of the identification of information needs, sensor asset management, deployment and information delivery. We relate our model to existing research in this area, discuss the shortcomings of our model and point towards avenues for future research in Section 5, and, finally, we present our conclusions in Section 6.

2. PROPOSED APPROACH

Our proposed approach provides a solution for allocating a collection of intelligence, surveillance and reconnaissance (ISR) assets to a number of missions in order to satisfy the information requirements of that mission. These ISR assets are composed of various sensors, each with its own location and sensing range, and each sensor is able to provide different utilities to different tasks. We equate these sensors with agents, as each sensor is wrapped by an autonomous computational entity, that is, a software agent, which communicates with others by means of message-passing. A mission consists of a number of tasks, each task having a specific type, which, in turn, requires a number of sensor types and each task has a specified own location and operational range and has its own sensing demand. A task can only be satisfied if its demand is met (within a threshold) and all of its sensor types are present in its allocation. If a task is not satisfied, the mission requiring that task is not successful.

More formally, a sensor s_i is defined as the tuple $\langle \gamma_i, l_i, r_i, u_i \rangle$ where $\gamma_i \in \Gamma$ specifies s_i's type, l_i and r_i are the location and sensing range of s_i, u_i is the maximum utility s_i can provide in a single time unit, Γ is the set of all sensor types. A mission M is defined as a tuple $\langle T, l_m, r_m, time_m \rangle$ where T is a set of required tasks, l_m and r_m are M's location and operational range, $time_m$ is the time when M is active. Each task $t_j \in T$ is defined as the tuple $\langle \delta_j, l_j, r_j, d_j \rangle$ where $\delta_j \in \Delta, \delta_j = \{\gamma_k | \gamma_k \in \Gamma' \subseteq \Gamma\}$ denotes t_j's type, Δ is the set

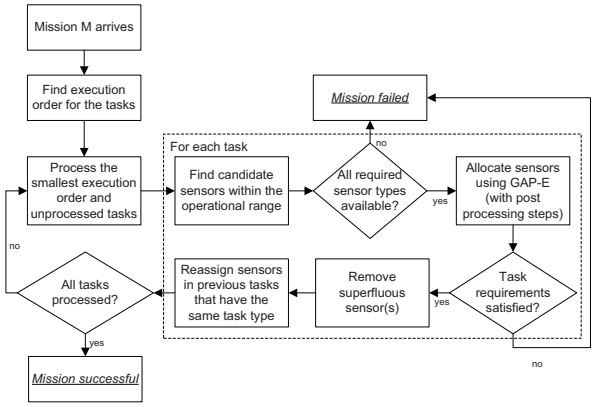


Figure 1: *Our proposed approach as a flowchart.*

of all task types, l_j and r_j specifies t_j's operational range (l_j is located inside (l_m, r_m)) and d_j is the sensing demand that t_j requires. The active time for t_j is the same as $time_m$ We denote u_{ij} as the utility that s_i can provide to t_j, which is defined as a percentage of u_i calculated by the ratio between the overlap of the ranges of s_i and t_j and the range of s_i. If the operational areas of s_i and t_j do not intersect, the value of u_{ij} will be 0.

Here we assume that the sensors cooperate with each other and they know both their locations as well as the location of other sensors. Moreover, we also allow a sensor to provide its service to multiple same-type tasks (for example, an audio sensor can provide the same information to all the detecting tasks that require its service). The sensor agents communicate with each other based on the message exchange protocol detailed in [7]. A mission can arrive at any time and there may be more than one mission active at any given time.

When a mission M arrives with its tasks, our approach will attempt to allocate all the sensors to the tasks as follows (see Figure 1):

1. The execution order of the tasks is established. Two tasks t_i and t_j belong to the same execution set (i.e. they can be executed at the same time) if their operational ranges do not intersect or their sensor type requirements do not overlap. If, however, two tasks have the same type, both will be in the same execution set. Initially, the execution set containing t_0 will be processed first followed by the set containing the next unprocessed task until all the tasks have been handled.

   It can be observed that the outcome of this algorithm depends on the order in which the tasks are executed. Obviously, it is impossible to determine in advance the execution order of the tasks in order to obtain the optimal outcome. However, to compliment the lack of available sensors for the tasks that are executed in subsequent orders, we allow the sensors to be shared and reassigned between these tasks. By having this feature, the tasks which are executed later can grab previously assigned sensors which, otherwise, will be unavailable.

2. For each task t_j:

   (a) Identify the available sensors within t_j's operation range and add to its sensor list. This is done by querying each sensor agent with a message containing the task information (task type, location) and waiting for an answer from that sensor[1]. Each sensor will analyze the task and decide to bid for the task, providing the amount of utility it can provide. Moreover, if a sensor has already been allocated to one or more tasks of the type δ_j, it can also provide a service to t_j. This will enable more tasks to be successfully allocated without reducing the practicability of the approach or putting more constraints on the sensors. After all the sensors within the operation range have been queried, if the available sensors do not cover the sensor requirements of t_j (i.e. a required sensor type cannot be found), then t_j cannot be allocated and the mission M fails.

   (b) If all required sensor types are available for the mission, the agent coordinating the tasks uses our multi-round GAP-E algorithm to find a potential allocation of sensors for t_j. The details of the GAP-E algorithm are presented in Section 2.1

   (c) After all the rounds are completed and all task requirements are satisfied, there is a final post processing step to release all the superfluous sensors (the one that can be released without violating t_j's requirements - both in terms of utility and sensor type). If there is more than one sensor which can be released, select the one with the smallest utility. This is to ensure that t_j will never be allocated more sensor agents than needed.

   (d) When the GAP-E finishes, if the final allocation does not satisfy t_j's requirements, M is failed. However, if t_j is successfully processed, the next step will attempt to reassign these allocated sensor to other tasks (not necessary from M) that are active.

3. When tasks in T have been processed, mission M is completed successfully.

2.1 The GAP-E Algorithm

In this section, we detail our algorithm, GAP-E, to allocate sensors to a particular task t_j. In order to realize the idea of stricter governing of selected sensors round-by-round, we introduce two additional matrices. P_j is the priority matrix that indicates the importance relationship amongst sensor types with regarding to t_j. C_j is the cost matrix that specifies the cost that will need to be met if t_j requires the service of a certain sensor. In addition, a budget b_j acts as a constraint that governs the number of sensors that can be allocated to t_j.

Figure 2 summarizes the steps that GAP-E takes to find an allocation to t_j, the details of which as follows:

[1]We do not consider failure cases in which, for example, sensor agents do not respond to such requests. There are, however, well known mechanisms for handling such situations such as "setting a deadline for receipt of responses" or simply add "with a deadline".

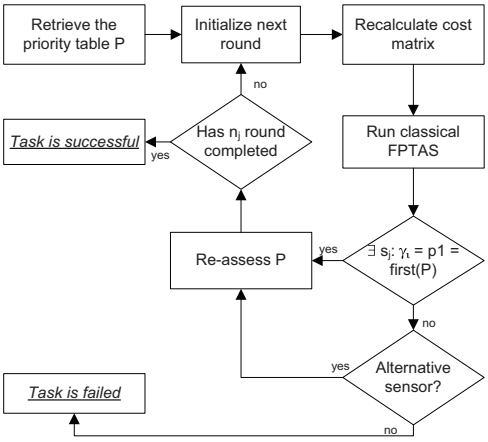


Figure 2: *GAP-E algorithm as a flow chart.*

1. Initially, we assume that t_j requires n_j sensor types ($|\delta_j| = n_j$), the priority matrix $P_j = \{p_i | i = 1...n_j\}$ where p_i is the sensor type that has the i^{th} importance with respect to t_j. This information is provided by the mission M. For example if t_j has the set of required sensor types $\delta_j = \{1, 2, 3\}$ and $P_j = \{3, 2, 1\}$ meaning that 3 is the most important sensor type and 1 is the least important one. Nonetheless, all these types must be presented in the final allocation otherwise the task will fail.

2. The utility matrix of the candidate sensors to t_j is updated. Based on the value of P_j, GAP-E introduces the concept of the cost matrix $C_j = \{c_{ij}\}$ for all s_i that has $u_{ij} > 0$. (c_{ij} is the cost of t_j using s_i's service as given in the bid received from s_i) and b_j as the overall budget of task t_j. Here, C_j has a similar objective as P_j; it is used to specify the relation between the sensor types and a particular task. The budget, b_j, is used to control the number of sensors allocated to t_j.

3. Next, there will be n_j rounds, with each round $r = 1..n_j$ composed of the following steps:

   (a) Reconstruct the matrix C_j so that if $k = p_i, l = p_j$ with $p_i, p_j \in P, i < j$ then $c_{k'j} < c_{l'j}\ \forall s_{k'}, s_{l'}$: $\gamma_{k'} = k$,$\gamma_{l'} = l$. If sensor type k is more important than type l then all the sensors of type k will have a lower cost than those of type l.

   (b) Run the FPTAS (Fully Polynomial Time Approximation Scheme) algorithm [14] with the input of C_j, $U = \{u_{ij}\}$ and budget $b = b_j * r/n_j - cost(A')$ with A' being the allocation from the last round. The obtained solution is then merged with A' to form the temporary allocation A.

   (c) If A does not contain at least one sensor of type p_1, we need to replace a sensor in A with a sensor of type p_1. Providing there exists some available sensors of type p_1 for t_j, the one with a minimum cost is selected as the target sensor. If, however, there are some sensors of type p_1 within t_j's range but they are all allocated to other tasks, GAP-E

will try to reassign sensors for one of such task in order to obtain a sensor of type p_1. In more details, if GAP-E can identify a task t_k that is currently holding sensor s_k of type p_1 and t_k can find a replacement sensor s_l without violating t_k's allocation requirements, GAP-E will reassign s_l to t_k and s_k is selected as the target sensor. If no target sensor can be found then t_j fails and consequently, M fails.

If adding this target sensor to A causes the allocation to go over-budget then we need to remove other sensors from A to accommodate this new sensor. This is done by repeatedly removing a sensor, which is (1) a sensor of type k that has more than two members within A or (2) a sensor of type l that has never been the first member of P in this or previous rounds. If more than one removable sensors can be identified, the one which provides the lowest utility is removed first. If the budget constraint is still violated but no removable sensor can be identified then t_j fails and consequently, M fails.

(d) Reassess P_j so that p_1 is now the sensor type that has the highest importance and does not appear in A. The order of $p_i|i = 2..n_j$ is kept as the initial version of P_j. If all the required sensor types are presented in A, P_j will revert back to the initial constructed version.

4. After n_j rounds has been completed, the GAP-E algorithm terminates.

In the following section, we present an example to illustrate the operations of this algorithm.

2.2 GAP-E Example

The following example demonstrates the workings of our approach. Let us assume that $t_1 = \{\delta_1, l_1, r_1, d_1\}$ where $\delta_1 = \{1, 2, 3, 4\}$ (sensor types 1, 2, 3 and 4 are required for task t_1) and $d_1 = 1.5$ (the task has a demand of 1.5). Additionally, there are the following candidate sensors within t_1's operational range: $S' = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}, s_{41}, s_{42}\}$ with s_{ij} denoting sensor type i and index j. Here we have $U = [u_{111}, u_{121}, u_{211}, u_{221}, u_{311}, u_{411}, u_{421}] = [0.5, 0.25, 0.4, 0.7, 0.5, 0.15, 0.75]$ where u_{ij1} is the utility that s_{ij} can provide for t_1.

Initially, the budget for the task is set to $b_1 = 2$, and the initial priority matrix is $P_1 = [1, 2, 3, 4]$ (i.e sensor type 1 is most important, followed by type 2, etc.). The initial cost matrix is generated from the bids received from the candidate sensors as $C = [[c_{11}, c_{12}], [c_{21}, c_{22}], [c_{31}], [c_{41}, c_{42}]] = [[0.1, 0.15], [0.03, 0.17], [0.2], [0.05, 0.16]]$. There will be 4 rounds as follows:

- Round 1:
  - The matrix C_1 is recalculated based on C as $C_1 = [[c_{11}, c_{12}], [c_{21}, c_{22}], [c_{31}], [c_{41}, c_{42}]] = [[0.1, 0.15], [0.28, 0.42], [0.7], [0.8, 0.91]]$[2]
  - $b = 0.5$ (the total budget is initially split equally between the four sensor types required for this task), FPTAS returns $A = \{s_{11}, s_{12}\}$
  - Now we have all sensors of type 1, the priorities among tasks are now revised such that $P_1 = [2, 1, 3, 4]$. The allocation is within budget and so we do not consider sensors to be removed from the allocation. $cost(A) = 0.25$, $U(A) = 0.75$
- Round 2:
  - The matrix C_1 is recalculated as $C_1 = [[c_{21}, c_{22}], [c_{31}], [c_{41}, c_{42}]] = [[0.03, 0.17], [0.7], [0.8, 0.91]]$
  - $b = 1.0 - 0.25 = 0.75$, FPTAS returns $A = \{s_{21}, s_{22}\}$, combined with A' (the allocation from the previous round), we have $A = \{s_{11}, s_{12}, s_{21}, s_{22}\}$
  - Now we have sensors of type 1 and 2, thus $P_1 = [3, 1, 2, 4]$. The removal of sensors from the allocation need not be considered. $cost(A) = 0.45$, $U(A) = 1.85$
- Round 3:
  - The matrix C_1 is recalculated as $C_1 = [[c_{31}], [c_{41}, c_{42}]] = [[0.2], [0.8, 0.91]]$
  - $b = 1.5 - 0.45 = 1.05$, FPTAS returns $A = \{s_{42}\}$, combined with A' we have $A = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{42}\}$
  - Now we have sensors of type 1, 2 and 4 but since $p_1 = 3$ and there is no type 3 in A, we will need to add a sensor of type 3 to A. Since there is only one sensor of such type, s_{31} needs to be added. However, adding s_{31} will cause the budget to be exceeded and we, therefore, will need to remove sensors from the allocation. Any of the sensors in A can be removed and thus, s_{12} is removed first since it has the lowest utility (u_{121}). By doing so, the cost of the solution falls below the budget and we have a valid allocation $A = \{s_{11}, s_{21}, s_{22}, s_{31}, s_{42}\}$ with $cost(A) = 1.41$ and $U(A) = 2.85$. Additionally, we have all the required sensor types, $P_1 = [1, 2, 3, 4]$
- Round 4:
  - The matrix C_1 is recalculated as $C_1 = [[c_{12}], [c_{41}]] = [[0.15], [0.8]]$
  - $b = 2.0 - 1.41 = 0.59$, FPTAS returns $A = \{s_{12}\}$, combined with A' we have $A = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}, s_{42}\}$
  - As we have all the required sensor types, $P_1 = [1, 2, 3, 4]$ $cost(A) = 1.56$, $U(A) = 3.1$

Thus, GAP-E returns a valid allocation of $A = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}, s_{42}\}$. However, since $U(A) = 3.1$ is greater than $d_1 = 1.5$, we will need to release superfluous sensors. Of the sensors in A, we can release any sensor in the set $\{s_{11}|s_{12}, s_{21}|s_{22}\}$. Thus, s_{12} will be released first because

[2]Since we have 4 sensor types and $P_1 = [1, 2, 3, 4]$, c_{11} and c_{12} remains the same, c_{21} and c_{22} are increased by $\frac{1}{4}$, c_{31} is increased by $\frac{1}{2}$, c_{41} and c_{42} are increased by $\frac{3}{4}$.

u_{211} is smallest of all (0.25). This brings the $U(A)$ down to 1.6. After removing s_{12}, $U(A) = 2.85$ and still greater than d_1. This time we can only release either s_{21} or s_{22} otherwise the sensor type requirements of t_1 will be violated. Thus, s_{21} will be released since $u_{211} < u_{221}$. As a result, $A = \{s_{11}, s_{22}, s_{31}, s_{42}\}$ is the final allocation to t_1.

Having defined our model, next section will details the results of our experiments.

3. EVALUATION

This section evaluates our approach in a range of different environments and assesses its performance in terms of the number of successfully completed missions and the amount of processing time required. There are a number of internal variables which control the behavior of our model as well as external variables which define the environment in which our model is being used. The system developed in Java allows us to manipulate these variables, conduct experiments and analyze the results.

3.1 Simulation Setup

Our approach is evaluated using randomly generated problems. A set of data is generated for each run. We compare the performance of our model (Multiple-Sensor-Mode or MSM) with the following alternatives:

1. Exclusive sensor mode (ESM): here each sensor can only be assigned to one task at a time. For each task, we test all combinations of sensors within the available candidates with the restriction that each sensor type only has one member[3]. The combination providing the highest utility value will be checked against the demand of the task, and, if acceptable, it will be selected as the final allocation for that task.

2. Shared sensor mode (SSM): this control operates in a similar way to ESM. The difference being that it allows sensors to be shared between two same-type tasks, which is exactly the feature that our approach also possesses. Again, the combination providing the highest utility will be selected as the final allocation if it satisfies the demand of the task in question. This is our implementation for the work presented in [9] (see section 5.

3. Shared sensor mode but without demand checking (SSM-NC): this control operates in a similar fashion to SSM. However, there is no evaluation against the task's demand. Instead, the combination providing the highest utility will be selected as the final allocation for that task. This will give us an idea of what the optimal success rate of the whole mission might be[4]

[3]Since the assignment problem is NP-Hard it is time- and memory-consuming to go through all possible combinations. Thus one single sensor per type is chosen to reduce the running time of this case.

[4]This cannot be considered to be the optimal result since it does not involve checking all the combinations of all the potential allocations for each individual task and then measuring their requirements. There are cases where optimal allocation involves non-local maxima allocation. Given the set of experiments need to be carried out, however, it is impractical to do a complete exhaustive search to find the optimal solution.

We recall from Section 2 that a mission is composed of a set of individual tasks and can it only be satisfied if all of its tasks can be allocated. Here, the mission arrival rates are controlled by the *rate_per_hour* parameter, which ranges from 2 to 8, and *number_of_days* parameter, which is kept at 2 days. Each mission can last for an arbitrary amount of time, ranging from 5 minutes to 4 hours.

There are *total_sensor_types* different sensor types, which will vary between 4 and 8 and, for each sensor type, there will be *total_sensors_per_types* sensors. For each mission, the number of tasks will be varied between 3 and 8. There will be *total_task_types* different task types, which will vary between 4 and 8. Each task type will require a number of different sensor types, which is varied between tasks and has the figure randomized between 1 and 4. These individual sensor type requirements are generated randomly and have the value between 1 and *total_sensor_types*.

The battlefield has the size of 400m x 400m. This is where the sensors and missions are deployed in uniformly random locations. Each sensor range (r_i) is randomized between 20m and 40m and their maximum utility is calculated as $(r_i/40)^2$, which ensure the values between 0.25 and 1. The operation range of the tasks are set to be randomized between 40 and 80m. We now turn to the specific results.

3.2 Results

HYPOTHESIS 1. *MSM performs well in comparison to the estimated optimum.*

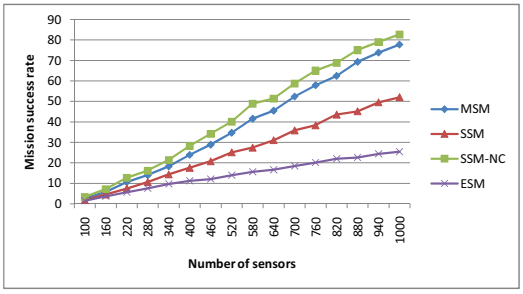


Figure 3: *Mission success rate with 4 sensor types and 4 missions arriving per hour.*

EVALUATION. Figure 3 shows the mission success rate of the four mechanisms with *total_sensor_types* = 4, *rate_per_hour* = 4 and *total_ sensors_per_types* between 25 and 250. Figure 4 shows the mission success rate with *total_sensor_ types* = 8, *rate_per_hour* = 8 and *total_sensors _per_types* also between 25 and 250. As the overall objective is to increase the number of successful missions, this control variable strongly indicate the performance of each mechanism.

As can be seen from both Figures 3 and 4, SSM provides better results than ESM. This is because in SSM, multiple same-type tasks can share a single sensor and, thus, a task in SSM has a greater chance of being successfully allocated.

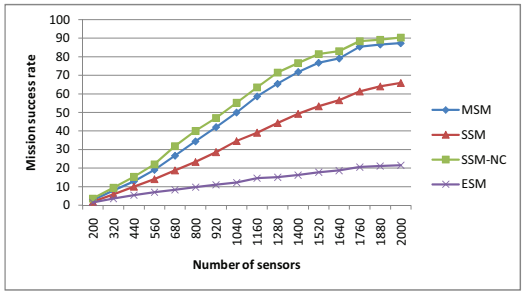


Figure 4: *Mission success rate with 8 sensor types and 8 missions arriving per hour.*

However, in both SSM and ESM, only one sensor per sensor type can be presented in the allocation and in many cases this is not sufficient to satisfy the sensing demand of a task. This explains why MSM has a significantly better mission success rate than SSM and ESM. With SSM-NC, since the demand requirement is not checked, the number of successful missions is the largest. However, this does not reflect the optimum allocation since there are situations in which the sensor type requirements of a task can be met but the sensing demand cannot be achieved. Nonetheless, the assignment problem being identified in this paper is NP-hard and it is impractical to do an exhaustive search of all combinations to find the optimal solution. Thus, the SSM-NC control is introduced here to give an idea of where the optimal solution might lie.

HYPOTHESIS 2. *The computational complexity (running time) of MSM is much less than that of other mechanisms.*

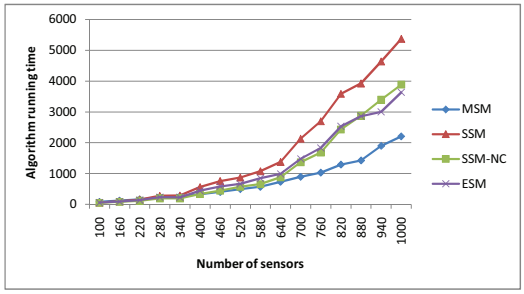


Figure 5: *Running time (ms) with 4 sensor types and 4 missions arriving per hour.*

EVALUATION. Similar to hypothesis 1, figure 5 shows the running time of the four mechanisms with *total_sensor_types* = 4, *rate_per_hour* = 4 and *total_ sensors_per_types* between 25 and 250. Figure 6 shows the running time with *total_sensor_ types* = 8, *rate_per_hour* = 8 and *total_sensors _per_types* also between 25 and 250. Of course, the running time of each mechanism depends largely on the machine that the experiments are run on. However, putting them all together can give a picture of their overall complexity.

As can be seen from both figures, MSM running time is always the smallest. Both SSM and SSM-NC has to exhaustively search through all the potential combinations of sensors to find the best allocation, their complexity will increase exponentially when the number of sensors increased. Thus, their running time are always highest and in many cases can be 2.5 times larger than that of our MSM. On the other hand, MSM employs the FPTAS algorithm (see section 2.1) which has the polynomial complexity and therefore, its running time only increases steadily when the number of sensors increases.

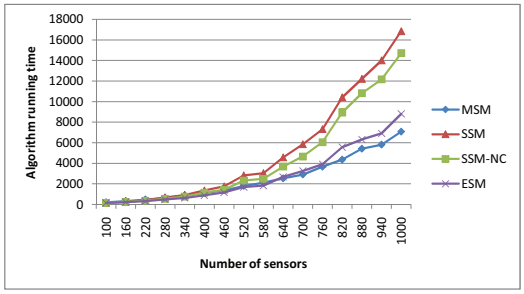


Figure 6: *Running time (ms) with 8 sensor types and 8 missions arriving per hour.*

HYPOTHESIS 3. *The computational complexity of MSM is increased in a steadily fashion with the number of missions (or tasks).*

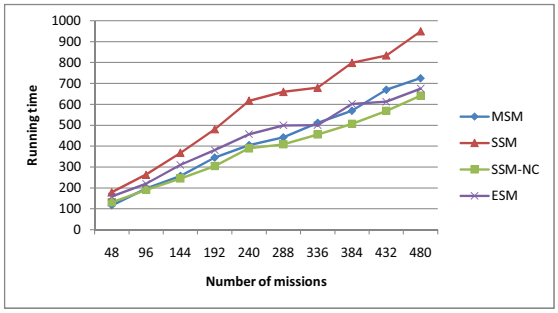


Figure 7: *Running time (ms) with 4 sensor types and 25 sensors per type.*

EVALUATION. To evaluate this hypothesis, we measure the running time of the mechanisms with *total_sensor_types* = 4, *total_sensors _per_types* = 25 and varies *rate_per_hour* between 1 and 10. The result is displayed in figure 7. It is clearly that as the number of mission increases from 48 to 480, the running time of MSM also increases steadily from 100 to just over 700ms. As our GAP-E algorithm has a polynomial complexity depending on the number of sensors and missions, once we keep the former variable unchanged, therefore, MSM running time increases inline with the number of missions.

4. SENSOR ASSIGNMENT IN CONTEXT

This section discusses the allocation problem addressed in this paper in a specialized environment setting. In particular, the problem of sensor-mission assignment is defined as

that of allocating a collection of intelligence, surveillance and reconnaissance (ISR) assets (both sensors and platforms) to one or more missions. Missions are composed of various tasks focused on satisfying their information requirements (IRs). These IRs will be identified as part of the process of mission planning. IRs are derived from questions such as "is there suspicious activity on the main supply road?" (see Figure 8). Each IR is then broken down to a set of scenario-specific informal requirements (SSIRs) such as "are there suspicious vehicles on the road?" or "is there suspicious pedestrian activity along the road side?". Before being able to match these to sensing types, decision-makers identify the interpretation tasks (ITs) which indicate what kinds of things need to detected, identified, distinguished, etc. The results of this further breakdown resemble a set of database queries like "detect vehicles where vehicle type or behaviours is suspicious", "detect people where person type or behaviours is suspicious". Furthermore, information is typically available that details the ISR assets (platforms and sensors), characterised in terms of their types, locations, readiness status, etc., that can be deployed to meet the information requirements.

Once having identified the ITs, given the informal of ISR assets available in the theatre, semantic matchmaking mechanism (such as those used by SAM [5]) may be employed to identify appropriate types of assets for the interpretation tasks specified. These approaches provide decision-makers with an at-a-glance view of feasible solutions.

It is this kind of information regarding information needs that we assume is available for the mission-sensor assignment problem. We are, therefore, concerned with the problem of moving from a "type level" fitting to an "instance level" allocation. When a mission needs to be accomplished, we consider a set of real assets available in the field that are compatible with the corresponding sets of sensor types required for each mission that arrives.

Allocated assets will then be configured for deployment in the operating environment. As the sensor network operates, information will be disseminated and delivered to users and the operational status of the assets will be monitored, thus closing the loop between the specification of information requirements and sensor information delivery. Both the ongoing monitors and the appearance of new tasks and ISR requirements can cause the decision-makers to reassess the sensor-mission assignment solution.

5. DISCUSSION & RELATED WORK

Sensor-task assignment problems in wireless sensor networks have been studied mainly using simplified models in which only a single sensor type is introduced and the exclusive of resources is required. For example, in [4, 8] the authors propose distributed approaches to solve the assignment problem assuming that the same type of sensors are deployed in the battlefield and that there is no competition for the sensing resources between tasks. On the other hand, the decentralised approaches considered in [12, 1, 7] introduce the competition between sensing tasks of same type. However, this approach set constraints which prevent more than one sensor from being assigned to any one task.

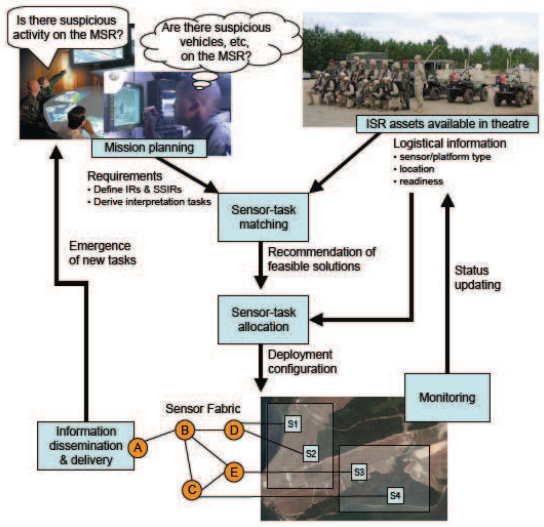


Figure 8: *The sensor-mission assignment problem in context.*

Our proposed E-GAP algorithm presented in Section 2 is an adaptation of the MRGAP algorithm proposed in [7]. The MRGAP algorithm aims to solve the static assignment problem, which is a common generalisation of the problems presented in [3, 1, 11], incorporating both budgets and a profit thresholds. The idea of that algorithm is to consider missions as knapsacks that together form an instance of the Generalized Assignment Problem (GAP). The author of [3] give an approximation algorithm for GAP which takes a knapsack algorithm as a parameter. There, the standard knapsack algorithm FPTAS [14] offers an approximation guarantee of $2 + \epsilon$. MRGAP, however, still lacks the ability to consider multiple sensor types. Moreover, it does not take into account the trade-off between communication cost and utilities gained. Nonetheless, the battery life-time of individual sensor is typically limited by the power required to transmit their data. As a result, power conservation issues in wireless sensor networks are essential and attract the interest of many researchers. In [10], for example, the authors selected a sensor acting as a mediator to relay data for other sensors. It saves the battery life of other sensors to the detriment of its own battery. To choose such mediator, the authors proposed a payment scheme in which the power p to reliably transmit over a distance d (p is proportional to the square of d) is considered as the decision value.

A number of protocols to locally make decisions for a sensor in a wireless sensor network have begun to tackle the challenge of coordinating between the network's interconnected nodes given the absence of a central coordinator. Protocols in [10, 6] allow sensors to request other sensors to forward data, which provides sensors with a broader range of information, often leading to an improvement in the network's ability to meet its global objectives. However, it requires extra communication that imposes an energy cost on the network. Therefore, the authors of [2] observed that the proper level of coordination (the degree of hops a sensor

broadcast message to) leads to a significant increase in the performance of the network.

Protocols have also been developed in [12, 9] where each task leader runs a local protocol to match sensors within two hops to the requirement of the task. Additionally, these works are the most important related to ours. They consider the many-sensor type, many-task type assignment problem. They also use the results of matching sensor type to mission to reduce the search space in order to find the allocation. However, they need to generate and check all the instances of feasible solutions of the matching sensor type problem. This is not good in dense sensor network. The main difference with our work is that the authors do not allow assets to be shared between tasks.

There are several shortcomings of our model that may restrict its applicability. First, we do not take into account the fact that some of the sensors can be mounted on mobile platforms. This is typical in many situations; for example, an image sensor can be mounted on an unmanned aircraft. Thus, such sensor will have a greater operational area compared to a static one. We plan to address this problem by incorporating new utility prediction model that will include the cost of relocating sensors within the battlefield.

Second, the tasks comprising a mission are independent of each other. In practice, there are situations in which there exist inter-dependencies between themselves. For example, there might be tasks that can only be allocated based on the results of some other tasks, which mean they can only be processed afterward. Furthermore, in such cases, the time-line for these tasks will be different from each other (rather than derived from the mission time-line). Future work will need to include a suitable planning model in order to be able to handle such scenarios.

Finally, only static sensors are considered in this model. Our next target research will consider mobile sensors that can be mounted on a platform. In order to achieve this goal, we need to incorporate a joint utility model instead of the current additive one and consequently, complete the resource sharing feature. Specifically, we aim to be able to handle situations in which a task only needs a percentage of the capacity of a sensor, the rest will be still available for other tasks.

6. CONCLUSION

In this paper we have introduced an agent-based (and hence decentralised) approach to solving the sensor-mission assignment problem for tasks sharing assets. We transformed the global sensor-mission assignment problem into a collection of sub-problems of sensor-task assignment. These sub-problems are then solved by task coordinator agents employing our GAP-E algorithm, which utilizes the multi-round knapsack algorithm to provide concrete solution consisting of the required sensor types together with the specific sensors belonging to these types. Our approach has been evaluated in a number of different scenarios, and we have demonstrated empirically that good results can be achieved in a considerably less time compared to the traditional exhaustive search counterpart.

7. REFERENCES

[1] A. Bar-Noy, T. Brown, M. P. Johnson, T. F. L. Porta, O. Liu, and H. Rowaihy. Assigning sensors to missions with demands. In M. Kutylowski, J. Cichon, and P. Kubiak, editors, *ALGOSENSORS*, volume 4837 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2007.

[2] E. Bulut, J. Zheng, Z. Wang, and B. K. Szymansk. Analysis of cost-quality tradeoff in cooperative ad hoc sensor networks. In *ACITA 2008*, 2008.

[3] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100(4):162–166, 2006.

[4] C. Frank. Algorithms for generic role assignment in wireless sensor networks. In *in SenSys Š05: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 230–242, 2005.

[5] M. Gomez, A. Preece, M. P. Johnson, G. Mel, W. Vasconcelos, C. Gibson, A. Bar-Noy, K. Borowiecki, T. Porta, D. Pizzocaro, H. Rowaihy, G. Pearson, and T. Pham. An ontology-centric approach to sensor-mission assignment. In *EKAW '08: Proceedings of the 16th international conference on Knowledge Engineering*, pages 347–363, 2008.

[6] D. B. Johnson. Scalable and robust internetwork routing for mobile hosts. In *In Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 2–11, 1994.

[7] M. P. Johnson, H. Rowaihy, D. Pizzocaro, A. Bar-Noy, S. Chalmers, T. F. L. Porta, and A. Preece. Frugal sensor assignment. In S. E. Nikoletseas, B. S. Chlebus, D. B. Johnson, and B. Krishnamachari, editors, *DCOSS*, volume 5067 of *Lecture Notes in Computer Science*, pages 219–236. Springer, 2008.

[8] H. Park and M. B. Srivastava. Energy-efficient task assignment framework for wireless sensor networks. Technical report, 2003.

[9] A. Preece, D. Pizzocaro, K. Borowiecki, G. de Mel, M. Gomez, M. Vasconcelos, A. Bar-Noy, M. P. Johnson, T. L. La Porta, H. Rowaihy, G. Pearson, and T. Pham. Reasoning and resource allocation for sensor-mission assignment in a coalition context. In *MILCOM 2008*, 2008.

[10] A. Rogers, E. David, and N. R. Jennings. Self-organized routing for wireless microsensor networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 35(3):349–359, 2005.

[11] H. Rowaihy, M. Johnson, T. Brown, A. Bar-Noy, and T. L. Porta. Assigning sensors to competing missions. In *Technical Report NASTR-0080-2007*, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, 2007.

[12] H. Rowaihy, M. Johnson, D. Pizzocaro, A. Bar-Noy, T. L. Porta, and A. Preece. Sensor-task assignment protocols. In *International Technology Alliance, Tech. Rep. (submitted for publication)*, 2008.

[13] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. pages 61–70, 1988.

[14] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

MAPS: A Mobile Agent Platform for WSNs based on Java Sun Spots

Francesco Aiello, Giancarlo Fortino*, Antonio Guerrieri, Raffaele Gravina
Department of Electronics, Informatics and Systems (DEIS), University of Calabria
Via P. Bucci, cubo 41c, 87036 Rende (CS), Italy
*phone: +39.0984.494063, fax: +39.0984.494713
faiello@si.deis.unical.it, aguerrieri@deis.unical.it, rgravina@si.deis.unical.it, g.fortino@unical.it

ABSTRACT
Wireless Sensor Networks (WSNs) are emerging as powerful platforms for distributed embedded computing supporting a variety of high-impact applications. However programming WSN applications is a complex task which requires suitable paradigms and technologies capable of supporting the specific characteristics of such networks which uniquely integrate distributed sensing, computation and communication. Mobile agents are a distributed computing paradigm based on code mobility that has already demonstrated high effectiveness and efficiency in IP-based highly dynamic distributed environments. Due to their intrinsic characteristics, mobile agents may provide more benefits in the context of WSNs than in conventional distributed environments. In this paper we present the design, implementation and experimentation of MAPS (Mobile Agent Platform for Sun SPOTs), an innovative Java-based framework for wireless sensor networks based on Sun SPOT technology which enables agent-oriented programming of WSN applications. The MAPS architecture is based on components which interact through events. Each component offers a minimal set of services to mobile agents which are modeled as multi-plane state machines driven by ECA rules. In particular, the offered services include message transmission, agent creation, agent cloning, agent migration, timer handling, and easy access to the sensor node resources. Agent programming with MAPS is presented through a simple example related to agent-based monitoring of a sensor node. Finally a performance evaluation of MAPS carried out by computing micro-benchmarks, related to agent communication, creation and migration, is presented.

Categories and Subject Descriptors
D.2.11 [Software Architectures]. I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – Multiagent systems.

General Terms
Design, Measurement, Experimentation.

Keywords
Mobile agent systems, event- and state-based programming, wireless sensor networks.

1. INTRODUCTION
Due to recent advances in electronics and communication technologies, Wireless Sensor Networks (WSNs) have been introduced and are currently emerging as one of the most disruptive technologies enabling and supporting next generation ubiquitous and pervasive computing scenarios. WSNs have a high potential to support a variety of high-impact applications such as disaster/crime prevention and military applications, environmental applications, health care applications, and smart spaces. However programming WSNs is a complex task due to the limited capabilities (processing, memory and transmission range) and energy resources of each sensor node as well as the lack of reliability of the radio channel. Moreover, WSN programming is usually application-specific (or more generally domain-specific) and requires tradeoffs in terms of task complexity, resource usage, and communication patterns. Therefore the developed software which usually integrates routing mechanisms, time synchronization, node localization and data aggregation is tightly dependent on the specific application and scarcely reusable. Thus to support rapid development and deployment of WSN applications flexible, WSN-aware programming paradigms are needed which directly provide proactive and on-demand code deployment at run-time as well as ease software programming at application, middleware and network layer.

Among the programming paradigms proposed for the development of WSN applications [20, 3], the mobile agent-based paradigm [14], which has already demonstrated its effectiveness in conventional distributed systems as well as in highly dynamic distributed environments, can effectively deal with the programming issues that WSNs have posed. In particular, a mobile agent is a software entity encapsulating dynamic behavior and able to migrate from one computing node to another to fulfill distributed tasks. We believe that mobile agents can provide more benefits in the context of WSNs than in conventional distributed environments. In particular, mobile agents can support the programming of WSNs at application, middleware and network levels. At application level, mobile agents can be used as design and programming abstractions through which WSN applications can be effectively designed and implemented. At middleware level, mobile agents can be used for implementing WSN core services such as data aggregation/fusion/dissemination and query-based information retrieval, and for dynamically deploying new services through efficient code dissemination. At network level, mobile agents can be used as the mobile capsules in Active Networks for smart multi-hop routing and other network services. Few trials have been to date devoted to the development of

mobile agent systems for wireless sensor networks (Agilla [7], actorNet [11], SensorWare [2]); however none of them has been developed for the Sun SPOT sensor platform [18] which is completely programmable in Java, supported by the SquawkVM [15] and compatible with JavaME.

In this paper, we propose MAPS (Mobile Agent Platform for Sun SPOTs), an innovative Java-based framework for wireless sensor networks based on Sun SPOT technology which enables agent-oriented programming of WSN applications. The architecture of MAPS is component-based and offers a minimal set of services to mobile agents, including message transmission, agent creation, agent cloning, agent migration, timer handling, and easy access to the sensor node resources. The dynamic behavior of mobile agents is modeled as multi-plane ECA-based state machines. MAPS therefore enables an effective application programming through an integration of three of the most important paradigms for WSN programming: agent-oriented, event-based and state-based programming.

The rest of the paper is organized as follows. Section 2 discusses related work and, in particular, currently available mobile agent systems for WSNs. Section 3 presents the requirements, architecture and the agent programming model of MAPS. Section 4 describes the implementation of MAPS based on the Java Sun SPOT library. In section 5, a simple example is provided for exemplifying the agent-based application programming with MAPS. Section 6 shows the performance evaluation of MAPS carried out through micro-benchmarks. Finally, conclusions are drawn and future work delineated.

2. RELATED WORK

Mobile agents are supported by mobile agent systems (MASs) which basically provide an agent server, an API for mobile agent programming, and, sometimes, supporting programming and administration tools. In particular, the agent server is able to execute agents by providing them with basic services such as migration, communication, and resource access. In the last decade, a significant number of MASs for IP-based distributed computing systems have been developed [14]. The majority of them are Java-based (e.g. Aglets, Voyager, Ajanta, etc) and few others rely on other languages (D'Agents, ARA, etc).

In the context of WSNs it is challenging to develop MASs for supporting mobile agent-based programming [1]. Due to the currently available resource-constrained sensor nodes and related operating systems, building flexible and efficient MASs is a very complex task. Very few MASs for WSNs have been to date proposed and actually implemented. The most significant ones are: Agilla [7], SensorWare [2] and actorNet [11]. A general mobile-agent-oriented sensor node architecture to which such MASs adhere is shown in Figure 1. The MAS relies on the services offered by the OS and the mobile agents are executed within the MAS, which supports their inter-node migrations, sensing capabilities and resource access, and inter-agent communications.

Agilla [7] is an agent-based middleware where each node supports multiple agents and maintains a tuple space and neighbor list. The tuple space is local and shared by the agents residing on the node. Special instructions allow agents to remotely access another node's tuple space. The neighbor list contains the address of all one-hop nodes. Agents can migrate carrying their code and state, but do not carry their own tuple spaces. Agilla is currently implemented on MICA2, MICAZ and TelosB motes.

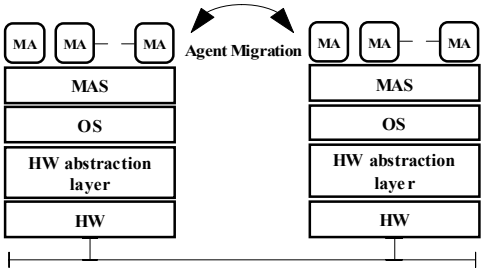


Figure 1. A general mobile-agent-oriented sensor node architecture

SensorWare [2] is a general middleware framework based on agent technology, where the mobile agent concept is exploited. Mobile control scripts in Tcl model network participants' functionalities and behaviors, and routing mechanisms to destination areas. Agents migrate to destination areas performing data aggregation reliably. The script can be very complex and diffusion gets slower when it reaches destination areas. The replication and migration of such scripts in several sensor nodes allows the dynamic deployment of distributed algorithms into the network. SensorWare, defines, creates, dynamically deploys, and supports such scripts. SensorWare is designed for iPAQ devices with megabytes of RAM. The verbose program representation and on-node Tcl interpreter can be acceptable overheads, however they are not yet on a sensor node. While both Agilla and SensorWare rely on mobile agents they employ a different communication model: Agilla's agent interaction is based on local tuple spaces whereas SensorWare's agent interaction is based on direct communication based on network messages. In [17] another mobile agent framework is proposed. The framework is implemented on Crossbow MICA2DOT motes. In particular, it provides agent migration and agent interaction based both on local shared memory and network messages. In [5] a specification language centered on Statecharts-based agents for programming WSN applications is proposed and exemplified. Moreover, a high-level run-time architecture supporting agent execution is defined. However, an implementation of this proposal on sensor nodes is not yet available. In [16] the authors propose an extension of Agilla to support direct communication based on messages. In particular, to establish direct communications, agents are mediated by a middle component (named landmark) that interact with agents through zone-based registration and discovery protocols. In [11] actorNet, a mobile agent platform for WSNs based on the Actor model is proposed. In particular, it provides services such as virtual memory, context switching and multi-tasking to support a highly-expressive yet efficient agent functional language. Currently, the sensor node actorNet platform is specifically designed for TinyOS on Mica2 sensors. Finally, in [13] the Agent Factory Micro Edition (AFME), a lightweight version of the Agent Factory framework purposely designed for wireless pervasive systems and implemented in J2ME, is used for exemplifying communication and migration in WSNs. However, AFME was not specifically designed for WSNs and, particularly, for Java Sun SPOTs.

3. MAPS ARCHITECTURE AND PROGRAMMING MODEL

3.1 Requirements

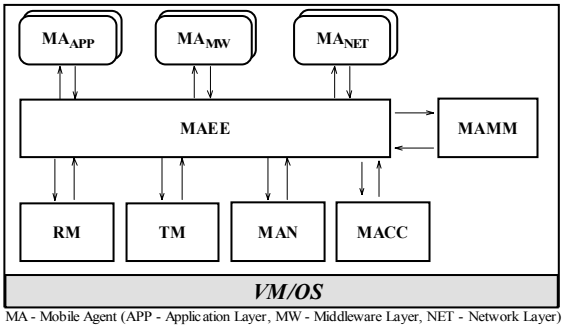
The MAPS framework has been appositely defined for resource-constrained sensor nodes; in particular its requirements are the following:

- *Lightweight agent server architecture*. The agent server architecture must be lightweight which implies the avoidance of heavy concurrency models and, therefore, the exploitation of cooperative concurrency to run agents.
- *Lightweight agent architecture*. The agent architecture must be also lightweight so that agents can be efficiently executed and migrated.
- *Minimal core services*. The main core services must be: agent migration, sensing capability access, agent naming, agent communication, and timing. The agent migration service allows an agent to be moved from one sensor node to another by retaining code, data and execution state. The sensing capability access service allows agents to access to the sensing capabilities of the sensor node, and, more generally, to its resources (actuators, input signalers, flash memory). The agent naming service provides a region-based namespace for agent identifiers and agent locations. The agent communication service which allows local and remote one-hop message-based communications among agents. The timing service allows agents to set timers for timing their actions.
- *Plug-in-based architecture extensions*. Any other service must be defined in terms of one or more dynamically installable components (or plug-ins) implemented as single mobile agent or cooperating mobile agents.
- *Layer-oriented mobile agents*. Mobile agents may be natively characterized on the basis of the layer to which they belong: application, middleware and network layer. They should be also able to locally interact to enable cross-layering.

3.2 Agent server architecture

The designed sensor node architecture is shown in Figure 2. The architecture is based on components that interact through events. The choice to design the architecture according to a component- and event-based approach is motivated by the effectiveness that such a kind of architecture has demonstrated for sensor node programming. In fact, the TinyOS operating system [8], the de facto standard for motes, relies on this kind of architecture. In particular, the main components are:

- *Mobile Agent (MA)*. The MAs are computing components which are differentiated on the basis of the layer (application, middleware and network) at which they perform tasks. Application layer MAs (or MA_{APP}) incorporate application-level logic performing sensor monitoring, actuator control, data filtering/aggregation, high-level event detection, application-level protocols, etc. Middleware layer MAs (or MA_{MW}) perform middleware-level tasks such as distributed data fusion, discovery protocols for agents, data and sensors, scope management, etc. Network layer MAs (or MA_{NET}) mainly implement transport (e.g. data dissemination) and network (e.g. multi-hop routing) protocols. MA_{APP}, MA_{MW}, and MA_{NET} can locally interact to implement cross-layering.
- *Mobile agent execution engine (MAEE)*. The MAEE is the component which supports the execution of agents by means of an event-based scheduler enabling cooperative concurrency. The MAEE handles each event emitted by or to be delivered at MAs through decoupling event queues. The MAEE interacts with the other core components to fulfill service requests (message transmission, sensor reading, timer setting, etc) issued by the MAs.
- *Mobile agent migration manager (MAMM)*. The MAMM component supports the migration of agents from one sensor node to another. In particular, the MAMM is able to: (i) serialize an MA into a message and send it to the target sensor node; (ii) receive a message containing a serialized MA, deserialize and activate it. The agent serialization format includes code, data and execution state.
- *Mobile agent communication channel (MACC)*. The MACC component enables inter-agent communications based on asynchronous messages. Messages can be unicast, multicast or broadcast.
- *Mobile agent naming (MAN)*. The MAN component provides agent naming based on proxies and regions [19] to support the MAMM and MACC components in their operations. The MAN also manages the (dynamic) list of the neighbor sensor nodes.
- *Timer manager (TM)*. The TM component provides the timer service which allows for the management of timers to be used for timing MA operations.
- *Resource manager (RM)*. The RM component provides access to the sensor node resources: sensors/actuators, battery, and flash memory.



MA - Mobile Agent (APP - Application Layer, MW - Middleware Layer, NET - Network Layer)
MAEE - Mobile Agent Execution Engine
MAMM - Mobile Agent Migration Manager
MACC - Mobile Agent Communication Channel
MAN - Mobile Agent Naming
RM - Resource Manager
TM - Timer Manager

Figure 2. The sensor node architecture.

3.3 Agent programming model

The architecture of an MA is modeled as a multi-plane state machine communicating through events (see Figure 3). This architecture allows exploiting the benefits deriving from three paradigms for WSN programming: event-driven programming [8], state-based programming [10] and agent-based programming [7]. Moreover it enables role-based programming, an important

paradigm for agents, as agents behave differently according to the role they can assume during their lifecycle [21].

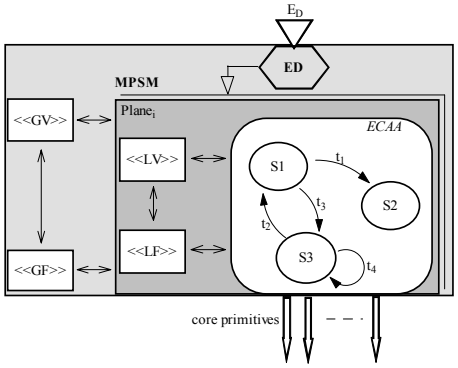


Figure 3. The mobile agent architecture.

In particular the architecture consists of:

- *Global variables (GV)*. The GV component represents the data of the MA including the MA identity.
- *Global functions (GF)*. The GF component consists of a set of supporting functions which can access GV but cannot invoke neither core primitives nor other functions.
- *Multi-plane State Machine (MPSM)*. The MPSM component consists of a set of planes. Each plane may represent the behavior of the MA in a specific role. In particular a plane is composed of:
  - o *Local variables (LV)*. The LV component represents the local data of a plane.
  - o *Local functions (LF)*. The LF component consists of a set of local plane supporting functions which can access LV but cannot invoke neither core primitives nor other functions.
  - o *ECA-based Automata (ECAA)*. The ECAA component which represents the dynamic behavior of the MA in that plane and is composed of states and mutually exclusive transitions among states. Transitions are labeled by ECA rules: E[C]/A, where E is the event name, [C] is a boolean expression based on the GV and LV variables, and A is the atomic action. A transition t is triggered if t originates from the current state (i.e. the state in which the ECAA is), the event with the event name E occurs and [C] holds. When the transition fires, A is first executed and, then, the state transition takes place. In particular, the atomic action can use GV, GF, LV, and LF for performing computations, and, particularly, invoking the core primitives (see Figure 4) to asynchronously emit one or more events. The delivery of an event is asynchronous and can occur only when the ECAA is idle, i.e. the handling of the last delivered event (ED) is completed.
- *Event dispatcher (ED)*. The ED component dispatches the event delivered by the MAEE to one or more planes according to the events the planes are able to handle. In particular, if an event must be dispatched to more than one plane, the event dispatching is appositely serialized.

```
send(SourceMA, TargetMA, EventName, Params, Local)
SourceMA  = id of the invoking MA
TargetMA  = id of the MA target |
            id of the Group target |
            ALL for event broadcast to neighbors
EventName = name of the event to be sent
Params    = set of event parameters encoded
            as pairs <attribute, value>
Local     = local (true) or remote (false) scoped event

create(SourceMA, MAId, MAType, Params, NodeLoc)
MAId    = id of the MA to be created
MAType  = type of the MA to be created
Params  = agent creation parameters
NodeLoc = node location of the created agent

clone(SourceMA, MAId, NodeLoc)
MAId    = id of the cloned MA
NodeLoc = node location of the cloned agent

migrate(SourceMA, NodeLoc)
NodeLoc = target location of the MA | ALL neighbors

sense(SourceMA, IdSensor, Params, BackEvent)
IdSensor  = id of the sensor
Params    = parameters for sensor readings
BackEvent = notifying event containing the readings

actuate(SourceMA, IdActuator, Params)
IdActuator = id of the actuator
Params     = parameters for actuator writings

input(SourceMA, BackEvent)
BackEvent  = event notifying the input captured from the
switch

flash(SourceMA, Params, BackEvent)
Params     = flash memory access parameters
BackEvent  = event notifying the completion of the flash
memory operation (if it is a read operation, it contains
the read data)

setTimer(SourceMA, Params, BackEvent)
Params     = timer parameters
BackEvent  = event notifying the timer firing

resetTimer(SourceMA, IdTimer)
IdTimer    = id of the timer to reset
```

Figure 4. The prototypal core primitives.

4. THE MAPS FRAMEWORK

The implementation of MAPS is a real challenge due to the constrained resources of the current sensor nodes. Nevertheless, due to recent advances in operating systems and virtual machines as well as sensor technologies, an actual implementation could be done in nesC/TinyOS on TelosB motes or in Java on Sun SPOT nodes [18]. Although the implementations of the currently available mobile agent frameworks for WSN (see section 2) have been to date carried out in nesC/TinyOS, by also using the Maté virtual machine, we believe that the object-oriented features offered by the Sun SPOT technology could provide more flexibility and extendibility as well as easiness of development for an efficient implementation of the proposed framework. The Sun SPOT sensor nodes are based on the Squawk VM [15] which is fully Java compliant and CLDC 1.1-compatible. In particular, the offered features are the following:

- *Java programming language*. Sensor node software is programmed in the Java language by using Java standard libraries and specific Sun SPOT libraries such as main Sun SPOT board classes, sensor board transducer classes, and Squawk operating environment classes.

- *NetBeans IDE for software development*. The IDE fully supports code editing, compilation, deployment and execution for Sun SPOTs. This enables a more rapid software prototyping.
- *Single-hop/multi-hop and reliable/unreliable communications*. The current version of the Sun SPOT SDK uses the GCF (Generic Connection Framework) to provide radio communication between SPOTs, routed via multiple hops if necessary. Two protocols are available: the radiostream protocol and the radiogram protocol. The radiostream protocol provides reliable, buffered, stream-based communication between two devices. The radiogram protocol provides datagram-based communication between two devices and broadcast communications. This protocol provides no guarantees about delivery or ordering. Datagrams sent over more than one hop could be silently lost, be delivered more than once, and be delivered out of sequence. Datagrams sent over a single hop will not be silently lost or delivered out of sequence, but they could be delivered more than once. The protocols are implemented on top of the MAC layer of the 802.15.4 implementation.
- *Easy access to the sensor node devices* (sensors, flash memory, timer, power). The Sun SPOT device libraries contains drivers to easily access and use: the on-board LED, the PIO, AIC, USART and Timer-Counter devices in the AT91 package, the CC2420 radio chip (in the form of an IEEE 802.15.4 Physical interface), an IEEE 802.15.4 MAC layer, an SPI interface (used for communication with the CC2420 and off-board SPI devices), an interface to the flash memory.
- *Code migration support*. An Isolate is a mechanism by which an application is represented as an object. In Squawk, one or more applications can run in the single JVM. Conceptually, each application is completely isolated from all other applications. The Squawk implementation has the interesting feature of Isolate migration, i.e. an Isolate running on one Squawk VM instance can be paused, serialized to a file or over a network connection and restarted in another Squawk VM instance.

MAPS is implemented on the basis of the aforementioned Java Sun Spots features which provide fully support to the implementation of each component introduced in section 3.2. In the following subsections the main MAPS classes (see Figure 5) and related functionalities are described (more details can be found in [12]).

The sensor node components are threads which can be instantiated through a Factory class based on the Singleton pattern. Such components are actually created at the node bootstrap when the MobileAgentServer is instantiated by the main application MIDlet. The MobileAgentServer creates the MobileAgentExecutionEngine which, in turn, creates all the other components. As soon as the MobileAgentExecutionEngine starts, it activates an InterIsolateServer to communicate with mobile agent components and broadcasts a *discovery publish* event to announce itself to the neighbor agent-based sensor nodes. After the creation of the MobileAgentServer, mobile agent components can be added to it by the *addAgent* method.

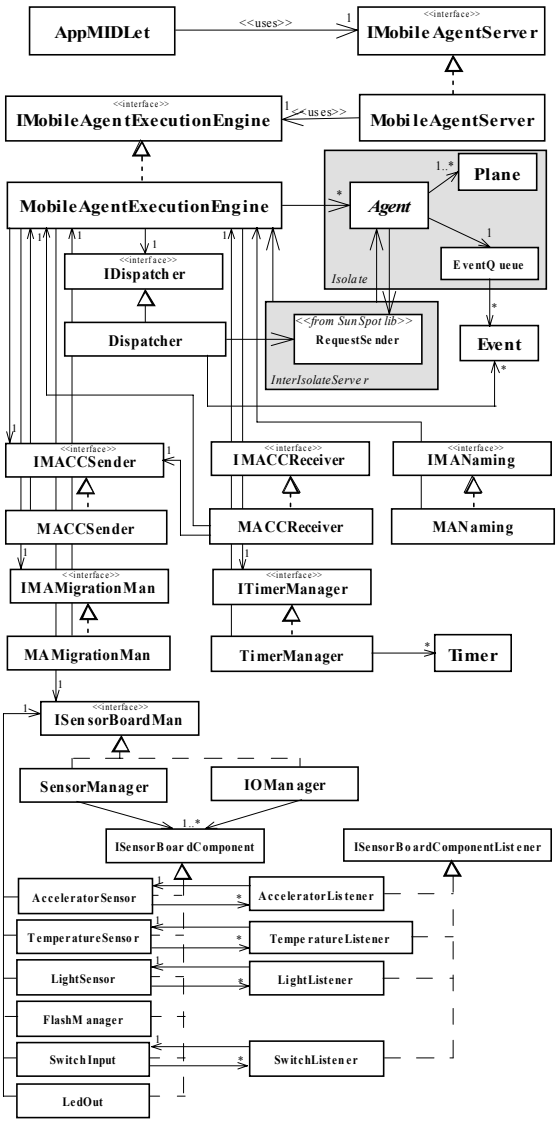


Figure 5. A simplified class diagram of the MAPS framework.

The MobileAgentExecutionEngine is the core component which exposes the interface for supporting all the primitives defined in section 3.3. The communication among agents, between agents and system components, and, sometimes, among components are based on Event objects. An Event object is composed of:

- *sourceID*, which is the agent/component identifier of the event source;
- *targetID*, which is the agent/component identifier of the event target;
- *typeName*, which represents the name of the event types which are grouped according to their specific function;
- *params*, which include the event data organized as a chain of pairs <key, value>;

- *durationType*, which specifies the event duration. It can assume the following three values:
  - NOW, for instantaneous events;
  - FIRST_OCCURRENCE, for events which wait for a specific value to occur;
  - PERMANENT. In this case, the event is sent every time that values set in the event parameter are reached.

A mobile agent lives in an Isolate which is instantiated at agent creation time. It is composed of an event queue which contains the Event objects delivered to the agent by the Dispatcher but not yet processed, and the multi-plane state machine containing the dynamic agent behavior. Interaction between mobile agents and the MobileAgentExecutionEngine is made possible by the InterIsolate server and enabled by its RequestSender component.

Remote inter-agent communication is enabled by MACCSender and MACCReceiver component which respectively allows to transmit and receive network messages according to the radiogram protocol. The MANaming component allows managing the list of neighbor sensor nodes and agents by means of a lightweight beaconing-based announcement protocol based on broadcast messages supported by the radiogram protocol. Moreover, agent proxy components are used to route network messages to migrated mobile agents.

The MAMigrationMan component manages the migration process of a mobile agent from one sensor node to another. To this purpose, it uses the methods provided by the SquawkVM to hibernate/dehibernate and serialize/deserialize an isolate. However, as dynamic class loading is not yet supported by the current version of the SUNSpot libraries (v4.0 blue), the agent code should reside at the destination node. In particular, the migration process, which is one-hop and reliable, is implemented as follows: (i) the agent destination node is contacted through a specific message which causes the opening of a socket waiting for an incoming request based on the Radiostream protocol; (ii) the agent destination node sends back an ack to the agent source node; (iii) the source node therefore establishes a radiostream connection with the destination node; (iv) the mobile agent is paused, hibernated, serialized into a byte array and sent over the connection to the destination; (v) at the destination node, the mobile agent is received, deserialized, dehibernated and reactivated.

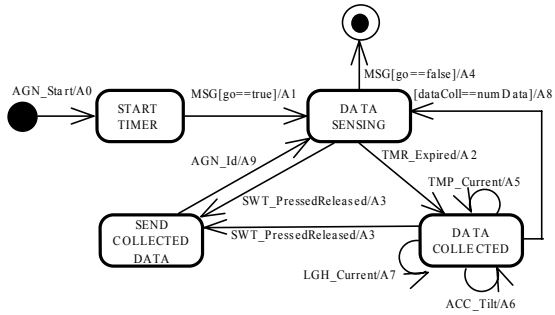
The TimeManager component handles Timer objects which can be requested by mobile agents to time their operations. Timers can be one-shot or periodic.

Finally the SensorManager component manages available sensors (accelerometer, light and temperature) and actuators (e.g. leds) whereas the IOManager component manages input from switches and the flash memory.

5. A PROGRAMMING EXAMPLE

The developed example application is structured in three agents:

- DataCollector, which collects data sensed from the temperature, light and accelerometer sensors of the SunSPOT node;
- DataMessenger, which carries collected sensed data from the sensing node to the base station;
- DataViewer, which displays the received collected data.




```
A0: byte [] fls = new byte[]{12,13,14,15,16};
    Event l = new Event(agent.getId(), agent.getId(),
                        Event.FLS_ADD, Event.NOW);
    agent.flash(l, fls);
A1: Event timer =new Event(agent.getId(), agent.getId(),
                           Event.TMR_EXPIRED, Event.NOW );
    timerID = agent.setTimer(true, 3000, timer);
    Event blink = new Event(agent.getId(), agent.getId(),
                            Event.LED_BLINK, Event.NOW);
    blink.setParam(ParamsLabel.LED_INDEX, "0");
    blink.setParam(ParamsLabel.LED_COLOR, "blue");
    agent.actuate(blink);
    Event swtPressed = new Event(agent.getId(),agent.getId(),
                  Event.SWT_PRESSED_RELEASED,Event.PERMANENT);
    swtPressed.setParam(ParamsLabel.SWT_PRESSED, "false");
    swtPressed.setParam(ParamsLabel.SWT_RELEASED, "true");
    swtPressed.setParam(ParamsLabel.SWT_INDEX, "2");
    agent.input(swtPressed);
A2: Event temp = new Event(agent.getId(), agent.getId(),
                           Event.TMP_CURRENT, Event.NOW);
    temp.setParam(ParamsLabel.TMP_CELSIUS, "true");
    agent.sense(temp);
    Event accel = new Event(agent.getId(), agent.getId(),
                            Event.ACC_TILT, Event.NOW);
    agent.sense(accel);
    Event light = new Event(agent.getId(), agent.getId(),
                            Event.LGH_CURRENT, Event.NOW);
    agent.sense(light);
A3: agent.create("test.Messenger", null,
                  agent.getMyIEEEAddress().asDottedHex());
A4: this.terminateAgent();
A5: data+=event.getParam(ParamsLabel.TMP_TEMPERATURE_VALUE)+"-";
    dataColl++
A6: data+=event.getParam(ParamsLabel.ACC_TILT_X_VALUE) + "-";
    dataColl ++
A7: data+=event.getParam(ParamsLabel.LGH_LIGHT_VALUE);
    dataColl++
A8: data+="|";
    Event blink = new Event(agent.getId(), agent.getId(),
                            Event.LED_BLINK, Event.NOW);
    blink.setParam(ParamsLabel.LED_INDEX, "0");
    blink.setParam(ParamsLabel.LED_COLOR, "blue");
    agent.actuate(blink);
    dataColl = 0;
A9: Event msg = new Event(agent.getId(), messengerAgentID,
                          Event.MSG, Event.NOW);
    msg.setParam("collectedData", data);
    agent.send(agent.getId(), messengerAgentID, msg, true);
    data = "";
```

Figure 6. The DataCollector plane.

After application deployment and execution, the DataViewer sends a message to the DataCollector to start its activity as soon as the user pushes a switch on the SunSPOT on which the DataViewer is running. The DataCollector therefore begins its collecting activity and as soon as the user pushes a switch on the SunSPOT on which the DataCollector is running, it creates the DataMessenger with the collected data that migrates to the DataViewer node for data visualization. Finally, the monitoring activity terminates when the user presses again a switch of the SunSPOT on which the DataViewer is running. This simple yet effective application, deployed on just two sensor nodes, allows for testing all the most important mechanisms provided by MAPS. The state machine of the DataCollector plane along with the action code, which uses the MAPS library, is shown in Figure 6 and briefly explained in the following. The AGN_Start event causes the transition from the creation state to the StartTime state

and the execution of an example operation on the flash memory (action A0), i.e. adding some data to the flash space of the agent. In the StartTimer state, when the network message sent by the DataViewer arrives and the guard [go==true] holds, a timer timing the sensing operations is set up to fire after 3s, some actuation on the leds is requested and some input from the switches is ready to be read (action A1). When the timer fires (see TMR_Expired event), the sensing operations are requested (action A2). When such operations are completed (see actions A5, A6, A7), guard [dataColl==numData] holds so that data are collected. When the switch is pressed, a DataMessenger agent is created (action A3) and the collected data are passed to it (action A9) when the AGN_Id event, containing the agent id of the created agent, is received. When the event MSG is received and the guard [go==false] holds, the agent is terminated (action A4).

6. PERFORMANCE EVALUATION

The used testbed for testing and evaluation consists of a SunSPOT kit (two sensor nodes and one base station) with the SDK 3.0 version (purple). The MAPS framework has a memory occupation (without any running agent) of about 70 Kbyte in central memory, keeping free a space of 378 Kbyte. Such space can be exploited for agent execution. The agent developed for the agent migration benchmarking (see below) needs 22 Kbyte of central memory. The space occupied by the jar of MAPS on the flash memory is 92 Kbyte out of the 4 MB available. To evaluate the performance of MAPS three micro-kernel benchmarks have been defined according to [6, 4] for the following mechanisms:

- *Agent communication*. The agent communication time is computed for two agents running onto different nodes and communicating in a client/server fashion (request/reply). Two different request/reply schemes are used: (i) *Data B&F*, in which both request and reply contain the same amount of data; (ii) *Data B*, in which only the reply contains data. Results are shown in Figure 7. By increasing the amount of data, communication times linearly increase.
- *Agent creation*. The agent creation time is computed for agents having different number of planes ranging from 1 to 51. Figure 8 reports the results which show that the creation time is linear with respect to the number of planes.
- *Agent migration*. The agent migration time is calculated for an agent ping-pong among two one-hop-distant sensor nodes. It has been computed in two cases: (i) *with MAPS*, which uses the complete functionality of MAPS; (ii) *without MAPS*, which does not uses the MAPS engine and migration manager but just the Java SunSPOT library. This allows highlighting the overhead introduced by the framework for having complete migration reliability. Migration times are computed by varying the data cargo of the ping-pong agent. Although migration performances without MAPS are better, complete reliability of agent migration is not guaranteed. The obtained migration times are high due to the slowness of the SquawkVM operations supporting the migration process. In particular, serialization is a very costly operation: serialization of the ping-pong agent averagely takes 4.5s. Moreover, radiostream connections are very slow to guarantee reliability.

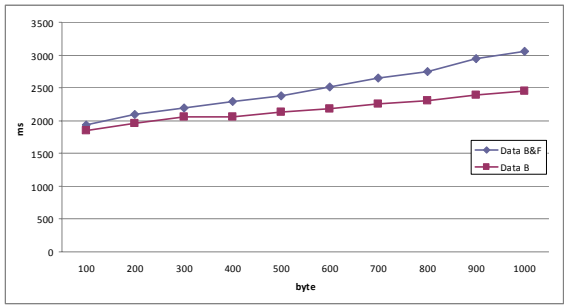


Figure 7. Agent communication: request/reply time.

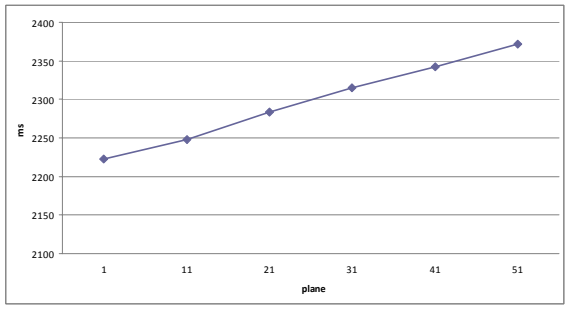


Figure 8. Agent creation time.

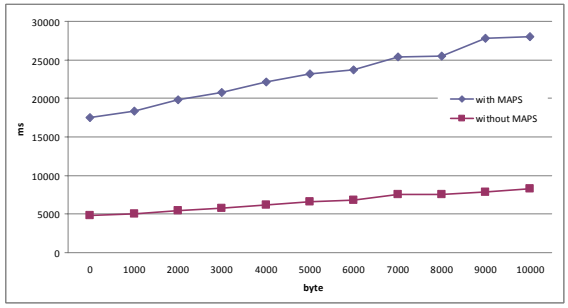


Figure 9. Agent migration: ping-pong time.

7. CONCLUSIONS

Programming WSN applications is a complex task which requires suitable programming paradigms and frameworks coping with the WSN specific characteristics. Several kinds of micro and macroprogramming techniques have been to date proposed. Among them mobile agent-based programming, which has been formerly introduced for conventional distributed systems, can be more effectively exploited in the context of WSNs. In this paper we have therefore proposed mobile agents as an effective paradigm to program WSN applications and, in particular, presented MAPS, a Java-based framework for the development of agent-based applications for SunSPOT sensor platforms. By using MAPS, a WSN application can be structured as a set of stationary and mobile agents distributed on sensor nodes supported by a component-based agent execution engine which provides basic services such as message transmission, agent creation, agent cloning, agent migration, timer handling, and easy access to the

sensor node resources. MAPS programming has been exemplified through a simple yet effective example which shows how to program the dynamic behavior of agents in terms of state machines on the basis of the MAPS library. Finally we have presented an evaluation of MAPS according to micro-kernel benchmarks (agent communication, migration and creation) usually employed for mobile agent systems. Evaluation shows some performance penalty mainly due to very time-consuming operations (serialization and radiostream-based communications) offered by the SunSPOT libraries and SquawkVM. On going research efforts are being devoted to: (i) optimizing the communication and migration mechanisms of MAPS; (ii) developing real applications through MAPS in the context of human activity monitoring for e-health [9].

8. REFERENCES

[1] F. Aiello, G. Fortino, A. Guerrieri, "Using mobile agents as an effective technology for wireless sensor networks," In Proc. of the 2nd IEEE/IARIA Int'l Conference on Sensor Technologies and Applications (SENSORCOMM 2008), Aug 25-31, Cap Esterel, France, 2008.

[2] A. Boulis, C.-C. Han and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks", In Proc. of the 1st Int'l Conference on Mobile systems, applications and services (MobiSys), pp. 187–200, 2003.

[3] M. Chen, S. Gonzalez, V. C. M. Leung, "Applications and Design Issues for Mobile Agents in Wireless Sensor Networks," IEEE Wireless Communications [see also IEEE Personal Communications] 14 (6) (2007) 20–26.

[4] M. Dikaiakos, M. Kyriakou, G. Samaras, "Performance evaluation of mobile-agent middleware: A hierarchical approach," In Proc. of the 5th IEEE Int'l Conference on Mobile Agents, 2–4 December 2001, Atlanta, Georgia, LNCS 2240, Springer Verlag, Berlin, 2005, pp. 244–259.

[5] G. Fortino, A. Garro, S. Mascillaro, W. Russo, "Specifying WSN Applications through Agents Based on Events and States," in Proc. of the IARIA/IEEE Int'l Conference SensorComm'07, Valencia, Spain, October 14-19.

[6] G. Fortino, A. Garro, W. Russo, "Achieving Mobile Agent Systems Interoperability through Software Layering," In Information and Software Technology, Elsevier, 50(4), pp. 322-341, 2008.

[7] C-L Fok, G-C Roman, C Lu, "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications," In Proc. of the 24th Int'l Conference on Distributed Computing Systems (ICDCS'05), Columbus, Ohio, June 6-10, 2005, pp. 653-662.

[8] David Gay, Phil Levis, Robert von Behren, Matt Welsh, Eric Brewer, David Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2003), San Diego (CA), USA, June 9-11, 2003.

[9] R. Gravina, A. Guerrieri, G. Fortino, F. Bellifemine, R. Giannantonio, M. Sgroi, "Development of Body Sensor Network Applications using SPINE," In Proc. of. IEEE Int'l Conference on Systems, Man, and Cybernetics (SMC 2008), Singapore, Oct. 12-15, 2008.

[10] O. Kasten, K. Römer, "Beyond event handlers: programming wireless sensors with attributed state machines," In Proc. of the 4th Int'l symposium on Information processing in sensor networks, April 24-27, 2005, Los Angeles, CA.

[11] Y Kwon, S. Sundresh, K. Mechitov, G. Agha, "ActorNet: An Actor Platform for Wireless Sensor Networks," In Proc. of the 5th Int'l Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 1297-1300, 2006.

[12] The MAPS project: documents and software. http://maps.deis.unical.it

[13] C. Muldoon, G. M.P. O'Hare, M. J. O'Grady, R. Tynan, "Agent Migration and Communication in WSNs," In Proc. of the 9th Int'l Conference on Parallel and Distributed Computing, Applications and Technologies, pp.425-430, 2008.

[14] A.R. Silva, A. Romao, D. Deugo, M. Mira da Silva, "Towards a Reference Model for Surveying Mobile Agent Systems," Autonomous Agent and Multi-Agent Systems, Vol. 4, N. 3, pp.187–231, 2001.

[15] D. Simon, C. Cifuentes, "The Squawk Java Virtual Machine: Java on the Bare Metal," In Proc. of the 20th Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2005), San Diego (CA), USA, October 16-20, 2005.

[16] S. Suenaga and S. Honiden, "Enabling direct communication between mobile agents in Wireless Sensor Networks," 1st Int'l Workshop on Agent Technology for Sensor Networks (ATSN-07), jointly held with 6th Int'l Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-07), Honolulu, Hawaii, 14th May 2007.

[17] L. Szumel, J. LeBrun, and John D. Owens, "Towards a Mobile Agent Framework for Sensor Networks," 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-TT), Sydney, Australia. May 30-31, 2005.

[18] Sun™ Small Programmable Object Technology (Sun SPOT), http://www.sunspotworld.com/

[19] M. Welsh and G. Mainland, "Programming Sensor Networks Using Abstract Regions," In Proc. of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), March 2004.

[20] E. Yoneki and J. Bacon, "A survey of Wireless Sensor Network technologies: research trends and middleware's role," Technical Report UCAM-CL-TR-646, University of Cambridge, UK, Sept. 2005.

[21] H. Zhu and R. Alkins, "Towards Role-Based Programming," In Proc. of the CSCW '06, Banff, Alberta, Canada, November 4-8, 2006.

Towards an Extensible Agent-based Middleware for Sensor Networks and RFID Systems

Dirk Bade
Distributed Systems and Information Systems
Computer Science Department
University of Hamburg, Germany
Vogt-Koelln-Strasse 30, 22527 Hamburg
bade@informatik.uni-hamburg.de

ABSTRACT

Sensor networks as well as RFID systems are among the hyped technologies nowadays. A lot of research efforts have been spent to develop standards, middlewares and applications. The industry already made large investments to foster the adoption of these technologies, consequently pushing the development, and already deployed the resulting technologies in different domains. However, the addressed technologies are still very young, best practices as well as standards are expected to frequently change, as new demands arise when using the technologies in our everyday life. Because of this, middleware systems are expected to undergo frequent redesigns as well, requiring well suited design paradigms to avoid a software engineering nightmare. We therefore propose an agent-based middleware for sensor networks and RFID systems. This middleware will meet the challenges for having a robust, adaptable and flexible middleware, which is moreover easily extensible to cope with expected re-engineerings and changes while maintaining a clear and elaborate design.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design; D.2.11 [**Software Engineering**]: Architectures; H.4.0 [**Information Systems Applications**]: General

General Terms

Sensor Networks, RFID, Middleware

Keywords

Sensor Networks, RFID, Software Agents, Middleware

1. INTRODUCTION

One of the most important milestones towards reaching Mark Weiser's vision of Ubiquitous Computing is the ability for computing systems to be aware of their environment. For this purpose, computing systems are being augmented with lots of different kinds of sensors to monitor certain states of affairs in their environment. *Wireless Sensor Networks* (WSN) and *Radio Frequency Identification* (RFID) are among the most promising research areas as WSN allow monitoring the physical environment and RFID technology enables the tracking of physical objects therein.

Although a lot of research efforts have been made to promote these technologies, the industry as well as the consumer sector are still in an early stage of adoption. High investments, few standards, and missing killer applications are some of the reasons for dilatory deployment. But the hardware evolves, costs of sensors, tags, readers, etc. rapidly decrease and several alliances and organizations are continuously publishing new standards so that strategic investments as proposed by Gartner [15] may finally become profitable.

Regardless the initial difficulties, several pioneering projects in the area of WSN and RFID systems have already been realized. But until now, most current projects are developed for a specific purpose. They do not interoperate and are neither generically designed to fit other purposes nor do they adhere to existing standards. But once WSN and RFID become widespread (and possibly converge in the future [25]) more experiences with the technologies are gained and the need for well designed infrastructure components will become evident as standards are expected to be frequently revisioned and new standards will arise.

These circumstances are very challenging from a software engineering point of view and we address this issue by proposing a unified middleware infrastructure for WSN and RFID based on software agents. Software agents are autonomous entities often employed for the development of complex and distributed systems [4]. They are capable of sensing their environment, may reactively or proactively act therein and thus adapt to changes and they are able to communicate and cooperate. The paradigm of agent-oriented software engineering therefore allows to build interoperable and reusable software components enabling a robust, flexible and extensible infrastructure [4, 16]. Regarding the unpredictable evolution of WSN and RFID systems, this paradigm is thus ideally suited to cope with the aforementioned challenges.

In the next section we will briefly introduce the basic technologies and highlight their progress in standardization. Afterwards, Section 3 discusses the challenges for engineering

future sensor network middlewares in the scope of expected changes and identifies some non-functional design goals to meet these challenges. In Section 4 our proposal for an agent-based middleware obeying these design goals is presented and subsequently discussed in Section 5. Finally, we present some related work in Section 6 and conclude with our prospects of future work in Section 7.

2. FUNDAMENTALS

In this section WSN, RFID, agent and middleware basics are introduced to gain a common understanding of the challenges and concepts described in further sections.

2.1 Wireless Sensor Networks

Wireless Sensor Networks (WSN) are a means for monitoring certain attributes of the physical world (used e.g. in environmental, health and home applications) [2, 24]. Such networks consist of a multitude of autonomous nodes, each equipped with sensors, a processing unit and communication capabilities. Once the nodes are deployed in a certain region they start to sense their environment and build up a kind of ad-hoc network with their neighboring nodes. In most WSN one or more base stations can be found, to which the percepts of each node are transmitted using multi-hop routing. For this purpose the nodes cooperate with each other by forwarding percepts of other nodes. Due to the limited resources of the nodes by means of energy as well as processing and communication capabilities, research in the area of WSN mostly concentrates on how to efficiently manage and distribute the information [25].

2.2 RFID

Although RFID systems also aim at monitoring the physical world, their primary use is the identification and tracking of real-world objects [22]. For this purpose objects, e.g. assets, are required to have a unique digital identity. This is provided by tags, which are comparable to a barcode, but may be read by specialized readers without a line of sight and in bulk over a distance ranging from several centimeters to a hundred meter (depending on the tag). The identity stored on a tag is often referred to as an Electronic Product Code (EPC) and can be used to link the identity with further information about the object stored somewhere in a network. Several application domains already make use of RFID technology, e.g. manufacturing control, asset tracking, warehouse and fleet management [24, 21] and concepts for several other domains are already being developed.

RFID systems are similar to WSN in the sense that data is read by specialized sensors (i.e. RFID reader) and can also be written back to tags in some cases. Hence, we also have streams of raw data which need to be processed and transformed to higher level events. And indeed, it is expected that RFID and WSN technologies will further converge in the future [25].

2.3 Software Agents

There is no definition of software agents in literature that is generally agreed upon. A basic definition states that software agents are able to perceive their environment through sensors and act upon it through effectors. As this definition is applicable to a multitude of software components one would not necessarily call an agent, other definitions specify certain characteristics an agent must have. Regarding these, an agent must be autonomous meaning the ability to process a task with as few guidance by its principal as possible. Moreover, an agent should be able to react to changes in its environment, but also to proactively follow its design goals. Additionally, agents must have the capability to cooperate by means of exchanging messages and must be able to adapt their behavior according to changes in the environment [16].

These definitions and characteristics lead to a very abstract view of what an agent actually is. From a software engineering perspective, agents are similar to objects, but a little bit more abstract. They can be seen as software components, developed to exhibit the above mentioned characteristics. And these agents are normally executed on a special middleware, called agent platform, which manages the lifecycle of agents and offers additional infrastructure services like a message transport system and a directory service. We will further enhance this brief introduction in the subsequent sections, once the context allows an explanation by example.

2.4 Middleware

Middleware in general shall shield the application layer from the details of lower layers in a way that applications can transparently use different shapes of services without the need to know any implementation details. Any changes in the lower layers hence do not affect the applications, but only the middleware. In most cases a middleware also provides a set of additional commonly used services for a specific purpose, so that applications do not need to implement the necessary functionality themselves.

Different middleware architectures have already been proposed for WSN as well as for RFID systems and even some for a combination of both technologies [5, 10, 14, 17, 22, 25, 26]. Regarding WSN, the term middleware often refers to a software layer residing between the application layer and the lower level hardware-oriented layers of sensors. It's main purpose is to support the development, maintenance, deployment and execution of sensing-based applications [24, 21], particularly focusing on power and topology management, data aggregation, transmission protocols, etc. inside a sensor network [1]. But a holistic view on WSN and traditional networks is often not provided, i.e. the connection to infrastructure networks is hardly considered by the middleware [21, 25, 29]. It is important to point out that our proposal coexists with current WSN middlewares as we are focusing on the post-processing of sensor data beyond the sensor network border.

In contrast to WSN, RFID middleware concentrates on efficiently processing and constructing meaningful events. In this case, the term 'meaningful' plays an important role, as the focus of an RFID middleware is not only on distributing data to a specific base station (as in WSN), but on processing and enriching data with contextual information on its way up the processing hierarchy. RFID tags themselves are mostly not capable of processing the data stored on the tag. Instead they are given just enough resources to communicate with a reader [22]. As a consequence, readers are the bottom-most layer of common RFID middleware and simply push data the RFID stack upwards, where the data is

filtered, aggregated, translated, enriched, etc. before meaningful *Application Level Events* (ALE) can finally be sent to applications for further processing [22].

2.5 Standards

Standardization issues play a major role in the adoption of technologies. Software and hardware developers want to be sure that their work gains acceptance by customers and will not be outdated within the near future. Also the customers need to feel confident that their investments in a technology are future-proof and are globally used in order to facilitate the cross-enterprise exchange of information. And finally, standards are the basis for competitive marketplaces where different system components may be traded, and consequently interoperability needs to be assured [8].

WSN is not widely deployed yet and one does not know whether the reasons are a lack of interest by the industry or a lack of standards. There are few standardization efforts for WSN and these are mostly concentrating on processing and communication mechanisms inside the network [1, 25, 29] (e.g. IEEE 1451, ZigBee). Existing standards often rely on standards borrowed from other areas and just add minor changes to adopt them to the special WSN characteristics. But to the best of our knowledge there are no standards for the interface between the data acquisition network (the sensor network itself) and the data distribution network (the backend responsible for post-processing the data), which is in our context the most interesting part. As a result, merging the data of different sensor networks in higher hierarchy levels of processing often relies on proprietary solutions [25].

Regarding RFID technology standardization has made a good progress. Driven by large interest and large investments multiple standards arose during the last years. Besides the *International Organization for Standardization* (ISO) also *EPCglobal*, a consortium of several companies and universities, is engaged in the process. EPCglobal published several standards for data representation and interfaces, among which the *Architecture Framework* [8] is in our context the most important one as it specifies a set of interfaces and roles within an RFID middleware. The main goal of these tasks is to gather, filter, enrich and transform raw sensor data in a way that application level events (ALE) can be forwarded to interested parties. But as this abstract architecture framework does not specify a real system architecture, several concrete architectures taking the infrastructure roles into account have been proposed [5, 10, 22, 26].

Implementing a concrete architecture, for WSN as well as for RFID systems, can be quite challenging, because both technologies are quite young and in some aspects still in an early stage of development. And it gets even more challenging when considering a unified middleware for both WSN and RFID [25]. In the following section we will therefore outline some of the difficulties in realizing such a middleware and list some requirements for future engineering of middlewares for sensor networks in general.

3. ENGINEERING CHALLENGES

Regardless the unpredictable future of RFID and sensor networks, universities as well as several companies are developing infrastructures and applications taking the already existing standards of the according technologies into account and using proprietary solutions if necessary.

Although a lot of standards have already been published for such infrastructures, there are still problems adhering to them. Reasons for this are threefold: First, there exist different standards for different infrastructures. This seems conclusive, but sensor networks in general (including RFID systems) have a least common denominator (e.g. post-processing of percept data) [25], which is not accounted for by the standards. Second, standards for specific aspects of an infrastructure are missing due to the lack of appropriate use cases [8]. As most technologies in these areas are quite young and rarely used, there are few experiences and hence new standards are not proposed until the requirements are fixed. Third, WSN and RFID technologies may further converge in the future [25] and standards will have to be redeveloped or merged in order not to get lost in standardization.

For these reasons, we expect the standards to be subject of frequent changes within the next years. Hence, the development of an infrastructure adhering to the standards (and possibly being compatible to the 'old' ones) may become a software engineering nightmare, because the process of software development needs to be iterated over and over again as new demands and standards arise. These development cycles require a flexible and extensible software architecture, otherwise substantial redesigns will become inevitable. In the following we will thus discuss a set of non-functional design goals for future sensor network middlewares, which take the above mentioned challenges into account.

3.1 Design Goals

One has to distinguish between functional and non-functional design goals. While functional goals define 'what' a system shall do, non-functional goals specify 'how' a system is supposed to be (i.e. quality goals). In order to be prepared for future sensor network developments, we identified some non-functional design goals, specifying evolution qualities, that are of special importance for new generation system architectures (an overview of functional goals can be found e.g. in [1, 10, 22, 23, 26]). Most of these should be obvious and sensor middleware should naturally adhere to them, but in practice this is often not the case.

Robustness and Adaptivity A middleware for sensor networks will need to be robust not in the sense that only the data acquisition network but also the backend, the data distribution network, needs to be tolerant towards failures. This requirement is accompanied with the need to be able to adapt to changing conditions, especially if different participators account for specific services in a network. Therefore a loose coupling of components and the possibility to dynamically choose an appropriate service at runtime is necessary.

Flexibility Someday, handling sensor data will not only be a matter of companies with global-scale processing networks, but will also be managed by individuals within local networks. A middleware must thus be flexible in a way that it must be deployable in different scales, i.e. certain functions an individual does not necessarily needs may be omitted for the sake of simplicity and

more sophisticated functions as required by enterprises must be easy to integrate. From a software engineering perspective, the functions must also be easily exchangeable as requirements and standards change.

Scalability More and more assets will be equipped with RFID tags, more sensor networks be deployed and eventually the data from all of these be joined in global-scale networks. Therefore, a middleware needs to be able to process single percepts as well as thousands or millions percepts.

Extensibility If sensor networks and RFID systems become widely deployed, new use cases accompanied by new requirements will arise. A middleware architecture must thus be extensible and the extensions should not necessarily require the applications built upon that middleware to change, but instead new applications should be able to directly use the extensions.

Having the above mentioned goals in mind, we propose an agent-based system architecture for WSN and RFID systems. In the following we will present the architecture and afterwards discuss our approach with respect to these goals.

4. AGENT-BASED MIDDLEWARE

To face the above mentioned challenges a flexible and adaptable architecture with loosely coupled components is required. Therefore, we propose a middleware infrastructure based on software agents for processing event streams originating from different kinds of sources (e.g. WSN, RFID reader or any other source). To ease understanding we will first present a motivating example and refer to that example in the subsequent sections.

4.1 Motivating Example

A trading company expects to receive a pallet with TV devices. The pallet is shipped within a smart container, equipped with several sensors measuring and logging acceleration, humidity and temperature throughout the whole transport. Once the container is received by the local warehouse, the pallet shall be unpacked and the devices be loaded directly onto a truck to deliver them to the customers, but only in case the sensor values logged during the transport do not exceed a specific limit. In this case, the trading company needs to send a mechanic into the warehouse to check if the devices are damaged.

To be informed about the state of delivery, the trading company registers several event triggers with the warehouses sensor network middleware. States of interest are: a) The pallet does not arrive in time b) the pallet arrived, but sensor values imply a possible damage and c) the pallet arrived in time and devices are going to be loaded onto the truck.

4.2 Overview

Inspired by common event driven architectures [7] and existing RFID systems [5, 10, 22, 26] we also follow a three-layered system architecture comprising the *Application Layer*, an intermediate *Network Layer* and finally the *Network Edge*. The latter connects to the event sources (e.g. WSN base station, RFID reader, etc.) and is responsible for infrastructure management as well as low-level event filtering. The Network Layer's primary role is the higher-level processing of events and the creation of *Application Level Events* (ALE) as required by the Application Layer. In our architecture the Application Layer requires no mandatory base components, hence even resource-constraint devices may make use of the subordinated Network Layer's functionality. Moreover, as our middleware shall be independent of specific kinds of event sources, the sources are not part of the architecture, but one can imagine the sources to reside in an additional layer below the Network Edge (as found in [5, 22, 28]). A detailed introduction of each layer is given in the following.

4.3 Application Layer

The Application Layer (also referred to as the Network Core, cp. [22]) represents the highest layer in the middleware stack. This layer only offers additional services for monitoring and debugging functionalities. It hence contains no mandatory components as to even allow applications running on resource-constrained devices (e.g. mobile phones) to be part of the infrastructure. This layer can be seen as a logical entity as it is not a core in physical means, but may be distributed over several physical locations. It is even possible to have multiple cores, e.g. each organization has its own core, but all of them are operating on the same middleware layers below. In fact, the relationship between Application Layer and Network Layer is an n:m relationship, as it is also possible to have one Application Layer, operating on multiple Network Layers.

Fig. 1 depicts the components residing in this layer as well as the actions a developer has to accomplish in order to connect the Application Layer respectively an application with the Network Layer. The overall goal is to receive meaningful Application Level Events (ALE) from the Network Layer once one or more specific low-level events occur. For this purpose a developer first has to create an event filter for filtering out events of interest. As the event filter is processed by a complex event processing engine (e.g. *Esper* [9]) in the Network Layer, also complex patterns (allowing content-based filtering in contrast to current standards [25]) as well as causal and temporal relationships may be detected in a stream of events [18].

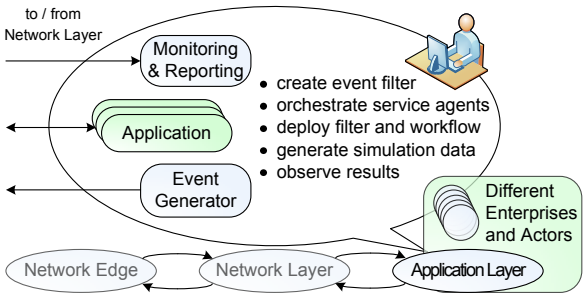


Figure 1: Application Layer

In a second step, the developer may choose how an event passing the filter shall be processed by the lower layers. In RFID systems for example, events may be aggregated, grouped, translated and enriched with additional context information. For this purpose, the Network Layer offers a yellow page service which may be queried by the developer

to find and orchestrate the service agents she needs in order to create a tailored ALE. Such an orchestration is similar to a business process orchestration, in which activities, their processing endpoints and additional parameters may be specified. For example, one may create such a workflow by specifying that events shall be first translated from representation A into representation B and then be enriched by querying a certain context service (e.g. an EPC information service). In case the service agents registered with the yellow page service do not match the needs, a developer may also choose to provide its own service agent for a dedicated task, for example to translate an event into the company's proprietary representation.

After creating the workflow (representing the service agent orchestration) and the corresponding event filter, these need to be registered at the *High-Level Event Filter* (cf. next section) of the Network Layer. As the Network Layer may already be a productive system, testing and debugging the filter as well as the workflow without interfering others may be a problem. For this reason, developers may use the *Event Generator* as well as the *Monitoring & Reporting* components, to create suitable simulation data and monitor the service agents executing their tasks.

In our example scenario the trading company wants to be informed once one of three possible delivery states is achieved. Therefore, three different event filters need to be created. Additionally, at least one workflow description has to be specified, instructing the middleware how the events shall be processed. This way, the trading company may provide e.g. its own aggregation function to be executed on sensor data, specify to call specific external services (e.g. call a mechanic to check the devices) to be executed depending on the aggregated values and enrich an event with additional context data helping the mechanic to bring the right tools. Once event filters and the workflow have been deployed, the company may simulate different scenarios to assure the right operations are performed by using the Event Generator and monitoring the processing of generated events.

4.4 Network Layer

The Network Layer is responsible for mapping low-level events received from the Network Edge to Application Level Events (ALE) that are finally forwarded to the Application Layer. Several tasks may be performed at this stage (cf. [22]):

- Events may be aggregated (e.g. just counted), grouped (e.g. build event sets with respect to a specified attribute) or translated (e.g. change content encoding).
- Additional information based on the event source or a specific event attribute (e.g. EPC) may be retrieved from external sources, e.g. an *Object Naming Service* (ONS) and an *EPC Information Service* (EPCIS) [8]. This task may involve multiple service invocations (e.g. ID resolution, context retrieval, ontology lookup, etc.).
- Every event source may allow to propagate data coming from applications towards a specific sensor. This way applications are able e.g. to write into the memory of an RFID tag or to send instructions to a WSN.
- An application may also specify that an external agent has to be called once a specific condition occurs. This way the processing chain may be easily extended to include arbitrary services.

As already mentioned in Section 3, these tasks may be extended in future middleware generations. Additionally, the interfaces, as standardized by e.g. EPCglobal, may be changed according to further demands [8]. To deal with these circumstances, we propose to encapsulate every task in a dedicated agent type (see Fig. 2). Once an interface changes or new roles are introduced, the corresponding agent types simply need to be refactored or newly created. For backwards compatibility one may decide to additionally keep the 'old' agent working. In order to execute the activities of a workflow (sequentially or in parallel) agents coordinate among each other by exchanging messages in a standardized communication manner.

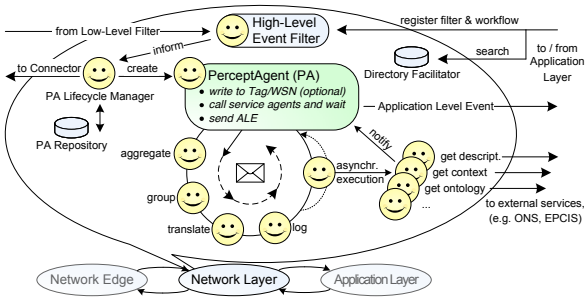


Figure 2: Network Layer

For an application to be informed about specific events, it has to subscribe at the *High-Level Event Filter* (HLEF) providing a (possibly complex) event pattern as well as a workflow description, containing orchestration instructions for the above mentioned service agents (cf. Section 4.3). Events coming from the Network Edge (cf. Section 4.5) are processed by a complex event processing engine in the HLEF and filtered with respect to the registered high-level filters. Once a low-level event passes the filter (meaning an application subscribed for that event) a so called *PerceptAgent* (PA) is instantiated and further on responsible for coordinating the processing of the workflow corresponding to that event. Note, that for a single low-level event, multiple PAs may be instantiated (one for each registered high-level filter). Processing the workflow means sending a message containing event information to a service agent as specified in the workflow and wait for an answer. Processing is finished once either all involved service agents notified the PA of completion or if the event is subject for being aggregated or grouped. In this case, superordinated dedicated *Aggregation-* and *GroupAgents* (not depicted) take over the responsibility for the event.

If a service agent specified in the workflow does not accept a task or does not respond within a specified time interval, the PA may decide to ask the yellow page service (called *Directory Facilitator*) in order to find another agent instance capable of execution. This way dynamic binding and hence adaptation to network failures may be achieved. Once a workflow is completed, an ALE is created and finally sent to the Application Layer for further processing.

Data may not only be read from lower layers, WSN as well as RFID systems also allow to write data. For example, an application may want to send processing instructions, queries, or a new power management configuration to a node inside a WSN or data, e.g. values coming from a sensor network, shall be stored into the memory of an RFID tag. In the case of writing data, two problems arise: 1) how to address a single sensor or tag from an application and 2) what happens if a sensor or tag is currently not in range so that writing temporarily fails? Regarding the first problem, an application shall never address entities in the lowest layer directly, as details of the concrete addressing scheme must be known by the application. Therefore, the middleware needs to abstract from low-level addressing details and must provide a mapping between an abstract and a concrete addressing. In our architecture this is achieved by the PAs and the *PA LifecycleManager* (PALM). An application may send messages to the PALM addressing a PA using a standardized scheme. The PALM in turn creates a new PA instance, which caches the data as long as the corresponding sensor or tag is sensed again, implying that it is also in writing distance and then dispatches the writing request to an appropriate *Connector* (cf. Section 4.5) in the Network Edge, responsible for writing to a tag or sensor. This way, also the second problem stated above may be solved. Similar approaches also exist in other projects, but the involved component roles as well as the naming are slightly different (cf. *virtual counterpart* [20], *virtual sensor* [1], *virtual tag* [10]).

Coming back to our example scenario: as already described, the trading company needs to register event filters and a workflow description with the HLEF. Once the container is being unloaded, the sensor log-files are read and corresponding low-level events from the Network Edge are forwarded to the HLEF. The complex event processing engine, when trying to match a registered filter against a stream of events, caches the relevant sensor data by itself, so that additionally storing the data in the warehouse is not necessary (although reasonable). After reading the sensor logs, an RFID reader scans all tags that are attached to the TV devices and subsequently generates appropriate events, which are again forwarded to the HLEF. Depending on the sensor data either the trading companies event filter for possibly damaged devices or the event filter indicating that everything is fine matches (the third event filter triggers after a specific time interval only if no corresponding TV devices are sensed). As a consequence, the PALM is informed, looking into its repository if corresponding PAs (with pending write instructions) exist, and finally instantiates a (new) PA. The PAs are provided with the trading company's workflow and coordinate the execution of activities by sending messages to the corresponding service agents. After being notified that all agents completed their work, data necessary for constructing an ALE is gathered and the ALE is finally forwarded to the trading company's application.

4.5 Network Edge

The Network Edge (depicted in Fig. 3), as the name indicates, separates the non-IP from the IP segment of the system. Event sources transmit event streams or batches to protocol-specific *Connectors* and further on to a *Low Level Event Filter* (LLEF) before they are finally forwarded to the Network Layer's HLEF.

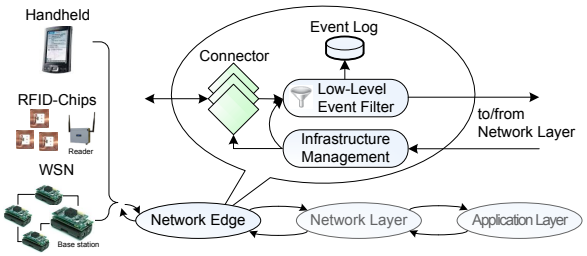


Figure 3: Network Edge

The Connectors are responsible for bridging the protocol gap, as the middleware components communicate over standard Internet protocols, but this must not necessarily hold for sensor networks, RFID readers or other event sources. In the case of RFID for example, low-level protocols for the communication between tags and readers have been specified by EPCglobal, but for the communication between readers and the Network Edge only data formats are standardized. Hence in practice readers communicate using a multitude of interfaces, e.g. Bluetooth, (W)LAN, IrDA or serial RS-232. As a consequence, one or more Connectors (depending on the provided data format) for each technology are required.

Once a Connector receives an event from an event source, contents are extracted and passed further on to the LLEF. This filter corresponds to an event filter as found e.g. in RFID systems and is used to filter out duplicates and incomplete, malformed or unknown events (but it has to be pointed out that the LLEF is no substitute for the filter integrated into RFID reading devices, as these work on an even lower layer). Events passing this stage are handed over to the HLEF residing in the Network Layer (cf. Section 4.4). An *Infrastructure Management* component is used to configure sensors as well as the event filter. At this layer we do not necessarily employ agents as the components effectively processing events in this stage are already in use by several other projects (e.g. *Fosstrack* [3]) and may simply be reused. Although it is possible to additionally wrap the functionality by agents to achieve a coherent addressing.

In our example scenario, the data from the container's sensor logs are read by a specific base station and passed on to a Connector. This Connector in turn forwards the data to the LLEF. Depending on the configuration of the filter, some sensor information may be discarded as it is of no interest. The remaining data is further transmitted to the HLEF. Once the container is unpacked, the TV devices are read by an RFID reader and the information is again passed on to the Connector and further up the stack to be finally processed by the HLEF.

5. DISCUSSION

In Section 3 we discussed challenges for future middleware systems and identified some non-functional design goals. In this section we now want to highlight how our proposed system architecture meets these design goals and where possible problems may arise.

We followed the paradigm of agent-oriented software engineering, which is well suited and approved for developing complex software systems in distributed and dynamic environments [4]. Among the reasons are that agents are autonomous, which means they are able to decide for themselves what they want to achieve, and they are capable of sensing their environment and hence able to adapt their behavior to changing conditions [16]. If for example an agent called an external service, but does not get any answer, it may decide for itself to follow different behavioral strategies, e.g. wait, call service once again or look for another service. In our architecture, *robustness* is achieved by using a yellow-page service for finding required service agents hence the failure of one agent can be compensated by requesting another instance at runtime and moreover by service agents being stateless (a new instance for every event-subscription pair is created) failures do not necessarily affect subsequent processings. *Adaptation* to network and load changes may be achieved by dynamically choosing processing nodes on which appropriate service agents reside or by migrating mobile service agents onto these nodes.

Moreover, agents are able to communicate and cooperate, which allows to divide a complex task into multiple simple tasks being executed by individual agents (cf. our service agents in the Network Layer). Communication is done by exchanging messages, which may be processed asynchronously, therefore tasks may easily be executed in parallel. Standardized infrastructure components for agent platforms, e.g. a yellow page service, may be used to achieve loose coupling as an agent may decide at runtime with whom to communicate based on its current senses. This also fosters the possibility to easily extend our middleware by simply introducing new agents, replacing or cooperating with existing ones. Additionally, agent-oriented programming is even more abstract than e.g. the object-orientation, allowing developers to focus on functionality instead of dealing with low-level details like communication and threading issues for example. All these arguments argue for agent-oriented software engineering achieving the *Flexibility & Extensibility* design goals.

Of course performance issues have to be discussed when processing thousands or millions of events [5]. And by using agents and message-based communication an additional overhead has to be considered. But in general agent-based software scales very well as the agents may be easily distributed among several hosts and by using yellow page services loose coupling and hence a dynamic binding can be achieved. Our middleware is currently implemented using the Jadex V2 agent system [19] which ships with a high-performance agent platform and is capable of executing very lightweight micro agents. Additionally, Jadex also allows BDI (Belief-Desire-Intention) agents to be executed, which may be of interest for designing more complex agents with reasoning capabilities for special tasks (e.g. intelligent adaptive routing). For these reasons we argue that agent-based applications naturally scale very well thus achieving the *Scalability* design goal.

6. RELATED WORK

In the past, different middleware platforms have been proposed for RFID systems, sensor networks and combinations of both. The EPCglobal consortium has published several standards for the processing of RFID data, including the *EPCglobal Architecture Framework* [8]. As this framework only proposes abstract standards, several projects aim at building concrete system architectures adhering to these standards, among these are for example [10, 20, 22, 26].

But the examples lack a flexible design making an adoption of the overall architecture to new demands and standards quite laborious. Moreover, they concentrate on RFID data only and in most cases do not allow multiple applications and organizations to take part in the event processing. To the best of our knowledge, few projects use agent technology as part of the system architecture. For example [27] are using software agents for a manufacturing control system. They embed the functionality of an RFID middleware to a large extent into a single monolithic agent, which is directly interfaced by applications. Another middleware approach using agent technology is [6], which focuses on mobile agents for gaining load balancing in RFID systems.

Although standardization of WSN technology has not made substantial progress, several works propose middleware architectures or guidelines and design issues for the development of such architectures [14, 17, 21, 29, 30]. Some works even make use of software agents [11, 13]. But all approaches mostly concentrate on the ongoings within the sensor network (e.g. data aggregation, routing, etc.) and do not account for the post-processing of sensor data in the backend, which we are focusing on. Sung et al. predict a convergence of RFID and WSN technologies in the future as RFID tags will become more powerful and hence gain the ability of autonomous communication and processing [25]. Additionally, isolated systems may be interconnected in the course of time realizing the vision of the *Real World Web* [15]. Therefore, some research efforts have already been spent in developing an integrated middleware for both WSN and RFID systems [12, 17, 25, 28] as well as for global scale systems (e.g. [1]). As a consequence, a middleware needs to be flexible and generic to abstract from incoming concrete events and outgoing sensor instructions on the one hand, and also should be extensible to easily allow incorporation of new functionality in the future as new demands arise. In our opinion and to the best of our knowledge, none of the systems satisfies these requirements.

All architectural proposals in common lack a future-proof design, rely on centralized infrastructures and/or make use of monolithic building blocks. They mostly concentrate on the state-of-the-art in standardization and apply proprietary solutions if necessary. But as new standards will arise and existing ones be changed, system architectures are confronted with frequent and substantial redesigns and refactorings, putting the architectures to the test.

7. CONCLUSION AND FUTURE WORK

In this paper we argued that the design of future middleware architectures for sensor networks and RFID systems is challenging due to the underlying standards being subject to frequent changes. As a consequence we identified a set of non-functional design goals, which should be considered when developing such middlewares.

By using software agents for engineering the middleware we expect to be able to deal with frequent architectural changes as agents are a means for designing complex software in dynamic and distributed environments. Additionally, agents are from a software engineering perspective the natural answer to scalability, reliability, extensibility and adaptability issues - key concerns for the distributed processing of event streams.

We proposed a three-layered, event-driven architecture following the state of the art middleware designs, in which agents are the main actors responsible for processing the events. From an application's point of view, the processing of events is in this case simply an orchestration of service agents, which collaborate in order to create tailored Application Level Events for the application.

Our prospects for future work are basically finishing the implementation of our proposed architecture to prove its feasibility. Within that scope we will develop exemplary applications of different scale addressing certain aspects of the middleware. Looking further into the future, standardization progresses as well as the evolution of existing middlewares will be closely followed in order to adapt our architecture to changing conditions and demands.

8. REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Intern. Conf. on Mobile Data Management*, pages 198–205, 2007.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38:393–422, 2002.

[3] AutoIDLabs. Fosstrak. www.fosstrak.org, 2009.

[4] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.

[5] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing rfid data. In *Proc. of the 13. Intern. Conf. on Very Large Data Bases*, pages 1189–1195. VLDB Endowment, 2004.

[6] J. F. Cui and H. S. Chae. Mobile agent based load balancing for rfid middlewares. *Advanced Communication Technology, The 9th International Conference on*, 2:973–978, Feb. 2007.

[7] J. Dunkel, A. Eberhart, S. Fischer, C. Kleiner, and A. Koschel. *Systemarchitekturen fuer verteilte Anwendungen*. Hanser Fachbuch, September 2008.

[8] EPCglobal. The epcglobal architecture framework, final version 1.2. Technical report, Sept. 2007.

[9] Esper. Event stream intelligence. esper.codehaus.org.

[10] C. Floerkemeier and M. Lampe. Rfid middleware design: addressing application requirements and rfid constraints. In *Proc of the conf. on Smart objects and ambient intelligence*, pages 219–224, USA, 2005. ACM.

[11] C.-L. Fok, G.-C. Roman, and C. Lu. Agilla: A mobile agent middleware for sensor networks. Technical Report WUCSE-2006-16, Washington University in St. Louis Dept. of Comp.Sc.and Eng., 2006.

[12] M. J. Franklin and et al. Design considerations for high fan-in systems: The hifi approach. In *CIDR*, pages 290–304, 2005.

[13] D. Georgoulas and K. Blow. Intelligent mobile agent middleware for wireless sensor networks: A real time application case study. *Advanced International Conference on Telecommunications*, 0:95–100, 2008.

[14] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, 2004.

[15] G. Inc. Gartner's 2006 emerging technologies hype cycle. www.gartner.com/it/page.jsp?id=495475.

[16] N. R. Jennings and M. Wooldridge. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[17] T. S. Lopez and D. Kim. Wireless sensor networks and rfid integration for context aware services. Technical report, Auto-ID Labs, 2008.

[18] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, May 2002.

[19] A. Pokahr and L. Braubach. From a research to an industrial-strength agent platform: Jadex v2. In to appear, editor, *9. Internationale Tagung Wirtschaftsinformatik*, 2009.

[20] K. Römer, F. Mattern, T. Dübendorfer, and J. Senn. Infrastructure for virtual counterparts of real world objects. Technical report, ETH Zurich, 2002.

[21] K. Roemer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):59–61, 2002.

[22] G. Roussos. *Networked RFID: Systems, Software and Services*. Springer Publishing Company, Inc., 2008.

[23] M. Sgroi and et al. A service-based universal application interface for ad hoc wireless sensor and actuator networks. In *Ambient intelligence*. Springer Verlag, Berlin, 2005.

[24] K. Sohraby, D. Minoli, and T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley-Interscience, 2007.

[25] J. Sung, T. S. Lopez, and D. Kim. The epc sensor network for rfid and wsn integration infrastructure. In *Proc. of the 5. IEEE Intern. Conf. on Perv. Comp. and Comm. Workshops*, pages 618–621. IEEE, 2007.

[26] F. Thiesse. Architektur und integration von rfid-systemen. In E. Fleisch and F. Mattern, editors, *Das Internet der Dinge*, pages 101–117. Springer, '05.

[27] P. Vrba, F. Macrek, and V. Mařík. Using radio frequency identification in agent-based control systems for industrial applications. *Eng. Appl. Artif. Intell.*, 21(3):331–342, 2008.

[28] W. Wang, J. Sung, and D. Kim. Complex event processing in epc sensor network middleware for both rfid and wsn. In *Proc. of the 11th IEEE Symp. on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 165–169. IEEE, 2008.

[29] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *Network, IEEE*, 18(1):15–21, 2004.

[30] L. Zhang and Z. Wang. Integration of rfid into wireless sensor networks: Architectures, opportunities and challenging problems. *Grid and Cooperative Computing Workshop*, pages 463–469, 2006.

Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm

Ruben Stranders, Alex Rogers and Nicholas R. Jennings
University of Southampton
School of Electronics and Computer Science
{rs06r,acr,nrj}@ecs.soton.ac.uk

ABSTRACT

In this paper, we introduce an on-line, decentralised coordination algorithm for monitoring and predicting the state of spatial phenomena by a team of mobile sensors. These sensors have their application domain in disaster response, where strict time constraints prohibit path planning in advance. The algorithm enables sensors to coordinate their movements with their direct neighbours to maximise the collective information gain, while predicting measurements at unobserved locations using a Gaussian process. It builds upon the max-sum message passing algorithm for decentralised coordination, for which we present two new generic pruning techniques that result in speed-up of up to 92% for 5 sensors. We empirically evaluate our algorithm against several on-line adaptive coordination mechanisms, and report a reduction in root mean squared error up to 50% compared to a greedy strategy.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Theory, Experimentation

Keywords

Mobile Sensors, Coordination, Gaussian process, Spatial Field Monitoring

1. INTRODUCTION

In disaster response, and many other applications besides, the availability of timely and accurate information is of vital importance. Thus, the use of multiple mobile sensors for information gathering in crisis situations has generated considerable interest.[1] These mobile sensors could be autonomous ground robots or unmanned aerial vehicles. In either case, while patrolling through the disaster area, these sensors need to keep track of the continuously changing state of spatial phenomena, such as temperature or the concentration of potentially toxic chemicals. The key challenges in so doing are twofold. First, the sensors cannot cover the entire environment at all times, so the spatial and temporal dynamics of the monitored phenomena need to be identified in order to predict environmental conditions in parts of the environment that can not be sensed directly. Second, the sensors need to coordinate their movements to collect the most informative measurements needed to predict these environmental conditions as accurately as possible.

Recent work has addressed similar challenges by modelling the spatial and temporal dynamics of the phenomena using Gaussian processes (GPs) [12]. GPs are a powerful Bayesian approach for inference about functions, and have been shown to be an effective tool for capturing the dynamics of spatial phenomena [3]. This principled approach to modelling the environment has been used to compute informative deployments of fixed sensors [7], and informative paths for single [9] and multiple mobile sensors [13].

However, the algorithms used to compute these informative deployments and paths are not suitable in our domain, since they are geared towards solving a one-shot optimisation problem in an off-line phase. Moreover, these algorithms are centralised. In hostile environments, this is undesirable, because it creates a single point of failure, thereby increasing the vulnerability of the information stream. Other work has employed on-line decentralised path planning using artificial potential fields to keep sensors in specific favourable formations [5], or through multi-agent negotiation techniques to partition the environment and allocate the sensors to these partitions [1]. However, in general, this work has used representations of the environment, that are less sophisticated than Gaussian processes, and are thus, less applicable for modelling complex spatial and temporal correlations.

To address this shortcoming, Low *et al.* [8] combine these approaches and use Gaussian processes to represent the environment, and use Markov decision processes to compute non-myopic paths for multiple mobile sensors in an on-line fashion. Whilst such a non-myopic approach avoids the problem of local minima, it incurs significant computational cost (it is only empirically evaluated for systems containing just two sensors), and is again a centralised solution.

Thus, against this background, in this paper, we present a new on-line, decentralised coordination algorithm for teams of mobile sensors. This algorithm computes coordinated paths with an adjustable look-ahead, thus allowing the trade off between computation and

[1]For example, one of the missions of both the Aladdin project (`http://www.aladdinproject.org`) and the Centre for Robot Assisted Search & Rescue (`http://crasar.csee.usf.edu`) is to use autonomous robots for information gathering in disaster response scenarios.

solution quality, and uses GPs to represent generic temporal and spatial correlations of the phenomena. To this end, we represent each sensor as an autonomous agent. These agents are capable of taking measurements, coordinating their actions with their immediate neighbours, and predicting the state of the spatial phenomenon at unobserved locations. We then use the max-sum algorithm for decentralised coordination [4] such that the agents can negotiate a joint plan by exchanging messages with their immediate neighbours. By applying max-sum in this manner, every agent controls its own movements using information it possesses locally, and the coordination mechanism is decentralised. We choose max-sum because it has been shown to generate good solutions to decentralised coordination problems, while limiting computation and communication. However, a standard application of max-sum is still too computationally costly within our particular domain. Thus, we introduce two novel and generic pruning techniques that speed up the max-sum algorithm, and hence, make decentralised coordination tractable in our application.

In more detail, in this paper we contribute to the state of the art in the following ways:

- We cast the multi-sensor monitoring problem as a decentralised constraint optimisation problem (DCOP), and present a new decentralised on-line coordination mechanism based on the max-sum algorithm to solve it.
- We present two novel, generic pruning techniques specifically geared towards reducing the number of function evaluations that is performed by max-sum. Thus alleviating a major bottleneck of this algorithm.
- We empirically show that a specific instantiation of our approach prunes 92% of joint moves for 5 sensors, and outperforms a greedy single step look-ahead algorithm by up to 50% in terms of root mean squared error.

The remainder of this paper is structured as follows. In section 2 we give a formal problem description. Section 3 describes how spatial phenomena are modelled. In section 4, we present our distributed algorithm, which we empirically evaluate in section 5, before concluding in section 6.

2. PROBLEM DESCRIPTION

The problem formulation described in this section was inspired by [9], and has been extended to deal with multiple sensors and limited local knowledge. Consider an environment in which M sensors monitor spatial phenomena that are modeled by a scalar field $\mathcal{F} : \mathbb{R}^3 \rightarrow \mathbb{R}$, defined on one temporal and two spatial dimensions, at a finite set of locations $V = \{v_1, v_2 \dots\} \subset \mathbb{R}^2$, and an indeterminate[2] number of of discrete time steps $T = \{t_1, t_2, ...\}$. To the measurement at location $v \in V$, and time t we associate a continuous random variable, $\mathcal{X}_{v,t}$. The set of all random variables is denoted by $\mathcal{X}$. The layout of the physical environment is given by a graph $G = (V, E)$, where E encodes the possible movements between locations V. The locations accessible from v are denoted by $adj_G(v)$. Since it is generally not possible to visit all locations V during a single time step, each sensor selects an adjacent location at which to take a measurement at time step $t + 1$. Values at locations V are subsequently predicted with a statistical model using all measurements that the sensors have gathered so far. In order to do this, we model the scalar field $\mathcal{F}$ with a GP (see next section) that encodes both its spatial and temporal correlations.

Now, in order to select their movements, sensors need to be able to predict the informativeness of the samples that are collected along their paths with respect to the missing ones. Here, the informativeness of a set of samples O is quantified by a function $f(O)$, that, depending on the context, can take on different forms [9]. Our choice for this function f will be derived in the next section.

Given this formalisation, we define the multi-sensor monitoring problem as follows. For every time step t, maximise $f(O_t)$, where $O_t = \cup_{i=1}^{M} O_t^i$ is the set of samples collected by all M sensors up to time step t at which the prediction is made. Moreover, while doing so, sensors can only communicate with and be aware of their immediate neighbours, such that no single point of control exists.

This problem is very challenging even for a single sensor. We therefore propose a distributed algorithm that computes paths with an adjustable look-ahead in Section 4, but first we discuss the way in which the spatial phenomena are modelled and derive function f.

3. MODELING THE SPATIAL PHENOMENA

In order to predict measurements at unobserved locations, we model the scalar field $\mathcal{F}$ with a GP. Using a GP, $\mathcal{F}$ can be estimated at any location and at any point in time based on a set of samples collected by the sensors [12]. In more detail, a single sample o of the scalar field $\mathcal{F}$ is a tuple $\langle \mathbf{x}, y \rangle$, where $\mathbf{x} = (v, t)$ denotes the location and time at which the sample was taken, and y the measured value. Now, if we collect the training inputs $\mathbf{x}$ in a matrix $\mathbf{X}$, and the outputs y in a vector $\mathbf{y}$, the predictive distribution of the measurement at spatio-temporal coordinates $\mathbf{x}_*$, conditioned on previously collected samples $O_t = \langle \mathbf{X}, \mathbf{y} \rangle$ is Gaussian with mean μ and variance σ^2 given by:

$$\mu = K(\mathbf{x}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y} \tag{1}$$

$$\sigma^2 = K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{x}_*) \tag{2}$$

where $K(\mathbf{X}, \mathbf{X}')$ denotes the matrix of covariances for all pairs of rows in $\mathbf{X}$ and $\mathbf{X}'$. These covariances are obtained by evaluating a function $k(\mathbf{x}, \mathbf{x}')$, called a covariance function, which encodes the spatial and temporal correlations of the pair $(\mathbf{x}, \mathbf{x}')$. Generally, covariance is a non-increasing function of the distance in space and time. For example, a prototypical choice is the squared exponential function where the covariance decreases exponentially with this distance:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\tfrac{1}{2}|\mathbf{x} - \mathbf{x}'|^2 / l^2\right) \tag{3}$$

where σ_f and l are called *hyperparameters* that model the signal variance and the length-scale of the phenomenon respectively. The latter determines how quickly the phenomenon varies over time and space[3]. If these hyperparameters are unknown before deployment of the sensors, they can be efficiently learnt on-line from collected samples using Bayesian Monte Carlo [10].

One of the features of the GP is that the posterior variance in Equation 2 is independent of actual measurements $\mathbf{y}$. This allows the

[2] In uncertain and dynamic scenarios, the mission time is often not known beforehand.

[3] A slightly modified version of Equation 3 allows for different length-scales for the spatial and temporal dimensions of the process.

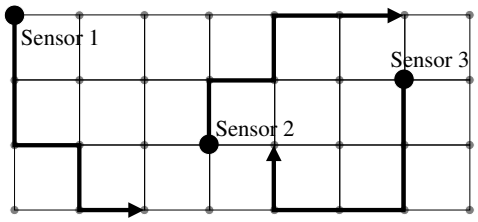


Figure 1: Joint plan of length 5 for sensors on a lattice graph.

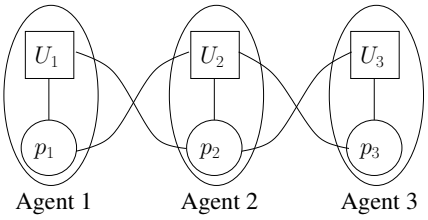


Figure 2: Factor graph with three agents.

sensors to determine the variance reduction that results from collecting samples along a certain path without the need of actually collecting them. Using this feature, we define the value $f(O)$ to be the *reduction in entropy* that results at the coordinates of O after taking these samples[4]. This function exhibits the property of *locality* [7], that is exploited by our algorithm. This means that the correlation between two samples decreases rapidly (exponentially in the case of Equation 3) with increasing distance, such that samples that are far apart can be considered uncorrelated, and thus, mobile sensors that are far apart need not explicitly coordinate.

4. DECENTRALISED COORDINATION

In this Section, we present an algorithm for solving the problem described in Section 2 that computes joint moves with an finite, adjustable look-ahead. An example of such a joint move for three sensors, consisting of a path of length 5 for each of them, is shown in Figure 1. The sensors run the algorithm every n time steps[5] to plan joint paths of length $l \geq n$. The algorithm is flexible, in that it allows paths of various lengths, and additional constraints, to be considered. For instance, the sensors can coordinate over all possible paths of length l, or only those moves that keep them within their own partition of the environment.

Given any particular potential joint move, our algorithm must calculate the reduction in entropy, $f(\cup_{i=1}^{M} O_t^i)$, that will result. In order to do so, we approximate $f(\cup_{i=1}^{M} O_t^i)$ by $\sum_{i=1}^{M} f(O_t^i)$ using the chain-rule of entropies[6]. This modified problem can then be cast as a decentralised constraint optimisation problem (DCOP), which lends itself to an agent-based solution paradigm. In this paradigm, sensors are modelled as agents that jointly plan their paths in order to maximise the value of the collected samples.

More formally, we denote the decision variable of agent i as p_i, which takes values in the set $A_i = \{a_i^1, \ldots, a_i^{q_i}\}$, representing all q_i moves that agent i is currently considering. In Figure 1, for example, this set contains all paths of length 5 from the agent's current position. Now, the incremental value of adding the samples $\tilde{O}^i$ collected by agent i along a considered path $p_i \in A_i$ to the set of samples $O_{t+1} \setminus \tilde{O}^i$ that are collected by the other sensors is equal to $f(O_{t+1}) - f(O_{t+1} \setminus \tilde{O}^i)$. This quantity is the contribution of agent i to the total value of the samples, given the movements of the other agents. In what follows, we will refer to this quantity as the agent's *utility*, which is denoted by $U_i(\mathbf{p}_i)$, where $\mathbf{p}_i$ is the vector of decision variables on which agent i's utility depends (thus, $p_i \in \mathbf{p}_i$ will always hold). By the property of *locality* of f, this dependency relation is usually limited to a small set of observations, and thererfore, to a small set of nearby agents. Now, the agents will collectively attempt to find joint move $\mathbf{p}^* = [p_1^*, \ldots, p_M^*]$, such that:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \sum_{i=1}^{M} U_i(\mathbf{p}_i) \tag{4}$$

or, in other words, the joint move that maximises the total value obtained by the agents.

4.1 The Max-Sum Message-Passing Algorithm

The coordination problem encoded by Equation 4 is a DCOP, which can be solved by a wide range of algorithms. Unfortunately, many of these algorithms either compute the optimal solution at exponential cost, either in terms of the number or size of messages that are exchanged between agents (e.g. DPOP [11]), or require little local computation and communication, but produce approximate solutions (e.g. the Distributed Stochastic Algorithm [6]). However, there exists a class of algorithms usually referred to under the framework of the Generalised Distributive Law [2], that can be used to obtain good approximate solutions. The max-sum message passing algorithm is one member of this class that is of particular interest here. This algorithm has been shown to compute better quality solutions than the approximate class with acceptable computation compared to representative complete algorithms [4].

In more detail, the max-sum algorithm operates on a *factor graph*: an undirected bipartite graph in which vertices represent variables p_i and functions U_j. In such factor graphs, an edge exists between a variable p_i and a function U_j iff $p_i \in \mathbf{p}_j$, (i.e., p_i is a parameter of U_j). Using the max-sum algorithm we exploit the fact that an agent's utility depends only on a subset of other agents' decision variables (locality), and that the global utility function is a sum of each agent's utility. Figure 2 shows an example factor graph that that encodes Equation 4 for the coordination problem of Figure 1. In this example, the utility of agent 1 depends on its own action, and that of agent 2, so $\mathbf{p}_1 = \{p_1, p_2\}$. Similarly, $\mathbf{p}_2 = \{p_1, p_2, p_3\}$, and $\mathbf{p}_3 = \{p_2, p_3\}$.

In yet more detail, using max-sum, each agent computes:

$$\tilde{U}_i(p_i) = \max_{\mathbf{p}_{-i}} \sum_{i=1}^{M} U_i(\mathbf{p}_i) \tag{5}$$

in a distributed way (i.e. based on local information and communication with direct neighbours). Agent i's optimal move p_i^* is then obtained as follows:

$$p_i^* = \arg\max_{p_i} \tilde{U}_i(p_i) \tag{6}$$

In order to do this, messages are passed between the functions U_i,

[4]Note that Guestrin *et al.* [2005] show that mutual information can also be used to define this value. However, the gain in doing so is small, and it is much more computationally expensive to evaluate.

[5]For simplicity, and w.l.o.g. we assume that the speed of the sensors is 1 unit per time step.

[6]This rule states that $H(\mathcal{X}_{o_1} \ldots \mathcal{X}_{o_n}) = H(\mathcal{X}_{o_1} | \mathcal{X}_{o_2}, \ldots \mathcal{X}_{o_n}) + H(\mathcal{X}_{o_2} | \mathcal{X}_{o_3}, \ldots \mathcal{X}_{o_n}) + \cdots + H(\mathcal{X}_{o_n})$.

Algorithm 1 Algorithm for computing pruning message from function U_m to variable p_n

1: compute $\overline{U_m}(p_n) \leq \min_{\mathbf{p}_{-n}} U_m(p_n, \mathbf{p}_{-n})$
2: compute $\underline{U_m}(p_n) \geq \max_{\mathbf{p}_{-n}} U_m(p_n, \mathbf{p}_{-n})$
3: send $\langle \overline{U_m}(p_n), \underline{U_m}(p_n) \rangle$ to p_n

and the variables in $\mathbf{p}_i$ as described below[7]:

- **From variable to function:**

$$Q_{p_n \to U_m}(p_n) = \alpha_{nm} + \sum_{\substack{U_{m'} \in adj(p_n) \\ U_{m'} \neq U_m}} R_{U_{m'} \to p_n}(p_n) \quad (7)$$

  where α_{nm} is a normalising constant that is chosen such that $\sum_{p_n} Q_{n \to m}(p_n) = 0$, to prevent the messages from growing arbitrarily large.

- **From function to variable:**

$$R_{U_m \to p_n}(p_n) = \max_{\mathbf{p}_{-n}} \left[U_m(\mathbf{p}_m) + \sum_{\substack{p_{n'} \in adj(U_m) \\ p_{n'} \neq p_n}} Q_{p_{n'} \to U_m}(p_{n'}) \right] \quad (8)$$

Finally, $\tilde{U}_i(p_i)$ is obtained by summing the most recent messages $R_{U_{m'} \to p_i}(p_i)$ received by p_i. This computation is guaranteed to be exact when the factor graph is acyclic. However, since dependencies are usually mutual (i.e. agent i's action influences agent j's utility and vice versa), the factor graph will normally contain cycles. In this case, max-sum computes approximate solutions (i.e $\tilde{U}_i(p_i) \approx \max_{\mathbf{p}_{-i}} \sum_{i=1}^{M} U_i(\mathbf{p}_i)$). Despite this, however, there exists strong empirical evidence that max-sum produces good results even in cyclic factor graphs [4].

4.2 Speeding up Message Computation

The straightforward application of max-sum to solve Equation 4 is not practical, because the computation of the messages from function to variable (Equation 8) is a major bottleneck. A naïve way of computing these messages for a given variable p_n is to enumerate all joint moves (i.e. the domain of $\mathbf{p}_m$), and evaluate U_m for each of these moves. Since the size of this joint action space grows exponentially with both the number of agents, and the number of available moves for each agent, the amount of computation quickly becomes prohibitive. This is especially true when evaluating U_m is costly, as is the case in the mobile sensors domain[8]. Therefore, we introduce two pruning algorithms to reduce the size of the joint action space that needs to be considered. These algorithms are then applied to our mobile sensor domain, but they can just as easily be applied within other settings.

4.2.1 The Action Pruning Algorithm

The first algorithm attempts to reduce the number of moves each agent needs to consider *before* running the max-sum algorithm.

[7]In what follows, we use $adj(U_m)$ to denote adjacent vertices of function U_m in the factor graph, i.e., the set of variables in the domain $\mathbf{p}_m$ of U_m. Similarly, $adj(p_n)$ denotes the set of functions in which p_n occurs in the domain.

[8]Specifically, determining the value of a sample involves the inversion of a potentially very large matrix $K(\mathbf{X}, \mathbf{X})$ (see Equation 2).

Algorithm 2 Algorithm for computing pruning messages from variable p_n to all functions $U_m \in adj(p_n)$

1: **if** a new message has been received from all $U_m \in adj(p_n)$ **then**
2: compute $\perp(p_n) = \sum_{U_m \in adj(p_n)} \underline{U_m}(p_n)$
3: compute $\top(p_n) = \sum_{U_m \in adj(p_n)} \overline{U_m}(p_n)$
4: **while** $\exists a \in A_n : \top(a) < \max \perp(p_n)$ **do**
5: $A_n \leftarrow A_n \setminus \{a\}$
6: **end while**
7: send updated domain A_n to each $U_m \in adj(p_n)$
8: **end if**

This algorithm prunes the dominated states that can never maximise Equation 4, regardless of the actions of other agents. More formally, a move $a' \in A_n$ is dominated if there exists a move a^* such that:

$$\forall \mathbf{a}_{-n} \sum_{U_m \in adj(p_n)} U_m(a', \mathbf{a}_{-n}) \leq \sum_{U_m \in adj(p_n)} U_m(a^*, \mathbf{a}_{-n}) \quad (9)$$

Just as with the max-sum algorithm itself, this algorithm is implemented by message passing, and operates directly on the variable and function nodes of the factor graph, making it fully decentralised:

- **From function to variable**: Function U_m sends a message to p_n, containing the minimum and maximum values of U_m with respect to $p_n = a_n$, for all $a_n \in A_n$. (see Algorithm 1).
- **From variable to function**: Variable p_n sums the minimum and maximum values from each of its adjacent functions, and prunes dominated states. It then informs neighbouring functions of its updated domain (see Algorithm 2).

Using this distributed algorithm, functions continually refine the bounds on the utility for a given state of a variable, which potentially causes more states to be pruned. Therefore, it is possible that action pruning starts with a single move, and subsequently propagates through the entire factor graph.

Now, given the highly non-linear relations expressed in Equation 2, on which the agents' utility functions U_m are based, it is very difficult to calculate these bounds exactly, without exhaustively searching the domain of $\mathbf{x}_m$ for utility function U_m. Needless to say, this would defeat the purpose of this pruning technique. Nonetheless, experimentation showed that by computing these bounds in a greedy fashion, a very good approximation is obtained. Thus, the lower bound $\underline{U_m}(a_n)$ on a move a_n is obtained by selecting the neighbouring agents one at a time, and finding the move that reduces the utility of agent m's move the *most*. In a similar vein, the upper bound $\overline{U_m}(a_n)$ is obtained selecting those moves of other sensors that reduce the utility the *least.*

4.2.2 The Joint Action Pruning Algorithm

Whereas the first algorithm runs as a preprocessing phase to max-sum, the second algorithm is geared towards speeding up the computation of the messages from function to variable (Equation 8), *while* max-sum is running. A naïve way of computing this message to a single variable p_i is to determine the maximum utility for each of agent i's actions by exhaustively enumerating the joint domain of the variables in $\mathbf{p}_m \backslash \{p_i\}$ (i.e. the Cartesian product of the domains of these variables), and evaluating the expression between brackets in Equation 8, which we denote by $\tilde{R}_{U_m \to p_n}(\mathbf{p}_m)$. This

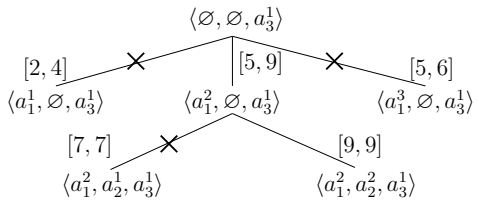


Figure 3: Search-tree for computing $R_{U_m \to x_3}(a_3^1)$ showing lower and upper bounds on the maximum value in the subtree.

expression is the sum of the utility function U_m and the sum of messages Q.

However, instead of just considering joint moves, we now allow some actions to be undetermined, and thus, consider *partial* joint moves, denoted by $\hat{\mathbf{a}}$. By doing so, we can create a search tree on which we can employ branch and bound to significantly reduce the size of the domain that needs to be searched. In more detail, to compute $R_{U_m \to p_n}(a_n^i)$ (a single element of the message from U_m to variable p_n) for a single state $a_n^i \in A_i$ in the domain of p_n, we create a search tree $\mathcal{T}(a_n^i)$ as follows:

- The root of $\mathcal{T}(a_n^i)$ is a partial joint move $\hat{\mathbf{a}}_r = \langle \varnothing, \ldots, \varnothing, a_n^i, \varnothing, \ldots, \varnothing \rangle$, which indicates that a_n^i is assigned to p_n, and the remaining variables are unassigned (denoted by $\varnothing$).
- The children of a vertex $\langle a_1^{(1)}, \ldots, a_k^{(k)}, \varnothing, \ldots \varnothing, a_n^i, \varnothing, \ldots, \varnothing \rangle$ are obtained by setting the first unassigned variable p_{k+1} to each of its $|A_{k+1}|$ moves.
- The leafs of the tree represent a (fully determined) joint move $\mathbf{a}_m$ (i.e. $\forall i : p_i \neq \varnothing$). In the tree, only leafs are assigned a value, which is equal to $\tilde{R}_{U_m \to p_n}(\mathbf{a}_m)$.

The maximum value found in $\mathcal{T}(a_n^i)$ is the desired value. Now, in order to use branch and bound to find this value, we need to put bounds on the maximum value found in a subtree of $\mathcal{T}(a_n^i)$. These bounds depend on U_m and the received messages Q. Now, in many cases we can put bounds on the maximum of the former, that is obtained by further completing a partial joint move $\hat{\mathbf{a}}'$ in a subtree of $\mathcal{T}(a_n^i)$. The bounds on U_m, combined with the minimum and maximum values of Q for $\hat{\mathbf{a}}'$ (again, by further completing the partial joint move), gives us the desired bounds.

Figure 3 shows an example of a partially expanded search tree for computing a single element $R_{U_m \to x_3}(a_3^1)$ of a message from function U_m to variable p_3. Given the lower and upper bounds on the maximum, subtree $\langle a_1^1, \varnothing, a_3^1 \rangle$ can be pruned immediately after expanding the root. Similarly, subtree $\langle a_1^3, \varnothing, a_3^1 \rangle$ is pruned after expanding leaf $\langle a_1^2, a_2^2, a_3^1 \rangle$, which has the desired maximum value.

To compute these bounds on the maximum of $U_i(\hat{\mathbf{a}})$ in the mobile sensor domain, note that partial joint move $\hat{\mathbf{a}}$ represents a situation in which only a subset of the agents have determined their move. Using this interpretation, we can obtain bounds as follows. The upper bound on this value is obtained by disregarding the agents that have not determined their action (i.e. agents i for which $p_i = \varnothing$). Since the act of collecting a sample always reduces the value of other samples, disregarding the samples of these 'undecided' agents will give an upper bound on the maximum. To obtain a lower bound on the maximum, we use the locality property of f, which tells us that the interdependency between values of samples weakens as their distance increases. So, in order to calculate the lower bound, we move the undecided agents away from agent i's destination.

4.3 Ensuring Network Connectivity

In many situations, it is also important that the sensors maintain network connectivity in order to transmit their measurements to a base station. More importantly in the context of our algorithm, agents need to be able to communicate in order to negotiate over their actions. Not surprisingly, we can use the algorithm to accomplish this, by penalising disconnection from the network in the utility function U_i. To this end, we assume that every agent maintains a routing table that specifies which agents can be reached through each immediate neighbour. Thus, a move is only allowed if all agents will still be reachable through the remaining links. Otherwise, the agent risks disconnection from the network, in which case a large penalty is added to its utility function U_i.

5. EMPIRICAL EVALUATION

To empirically evaluate our approach, we simulated five sensors on a lattice graph measuring 26 by 26 vertices. The data was generated by a GP with a squared exponential covariance function (see Equation 3) with a spatial length-scale of 10 and a temporal length-scale of 150. This means that the spatial phenomenon has a strong correlation along the temporal dimension, and therefore changes slowly over time. At every m time steps, the sensors plan their motion for the next l time steps ($l \geq m$). In what follows, this strategy is referred to as MSm-l. Now, instead of considering all possible paths of length l from an agent's current position, which would result in a very high computational overhead, the action space is limited to the locations in G that can be reached in l time steps in 8 different directions, corresponding to the major directions on the compass rose. In the first experiment, we benchmarked MS1-1 and MS1-5 against four strategies:

- **Random:** Randomly moving sensors.
- **Greedy:** Sensors that greedily maximise the value of the sample collected in the next move without coordination.
- **J(umping) Greedy:** The same as Greedy, except that these sensors can instantaneously jump to any location.
- **Fixed:** Fixed sensors that are placed using the algorithm proposed in [7].

The averaged root mean squared error (RMSE) for 100 time steps is plotted in Figure 4(a). From this figure, it is clear that both MS strategies outperform the Greedy, Random, and Fixed strategies. Furthermore, the prediction accuracy of MS1-5 is comparable to that of JGreedy, whose movement is not restricted by graph G. Moreover, it shows that increasing the length of the considered paths from 1 to 5, reduces the RMSE by approximately 30%.

In the second set of experiments, we analysed the speed-up achieved by applying the two pruning techniques described in Section 4.2. Figure 4(b) shows the percentage of joint actions pruned plotted against the number of neighbouring agents. With 5 neighbours, the two pruning techniques combined prune around 92% of the joint moves. With such a number of neighbouring agents, the agents

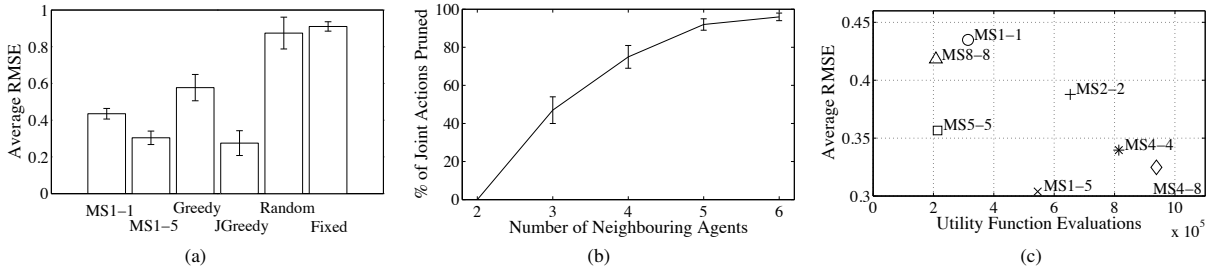


Figure 4: The Experimental Results. Errorbars indicate the standard error in the mean.

are strongly clustered, which occurs rarely in a large environment. However, should this happen, the utility function needs to be evaluated for only 8% of roughly 8^5 joint actions, thus greatly improving the algorithm's efficiency.

In the third experiment, we performed a cost/benefit analysis of various MSm-l strategies. More specifically, we examined the effect of varying m and l on both the number of utility function evaluations, and the resulting RMSE. Figure 4(c) shows the results. The results of MS1-1, MS2-2, MS4-4, MS5-5, and MS8-8 show an interesting pattern. Up to and including $m = l = 4$, both the number of function evaluations and the average RMSE decrease. This is due to the fact that planning longer paths is more expensive, but results in lower RMSE. However, for $m, l > 4$, the action space becomes too coarse (since only 8 directions are considered) to maintain a low RMSE. At the same time, the number of times the agents coordinate reduces significantly, resulting in a lower number of function evaluations. Finally, MS1-5 and MS4-8 provide a compromise; they compute longer paths, but coordinate more frequently. This leads to more computation compared to MS5-5 and MS8-8, but results in significantly lower RMSE, because agents are able to 'reconsider' their paths.

A video demonstrating the mobile sensors with the techniques from Section 4 can be found at `www.youtube.com/ijcai09`.

6. CONCLUSIONS

In this paper, we presented an on-line decentralised coordination algorithms for multiple sensors. We showed how the max-sum message passing algorithm can be applied to this domain in order to coordinate the motion paths of the sensors along which the most informative samples are gathered. We also presented two general pruning to speed up the max-sum algorithm. The first attempts to prune actions of the sensors that are not part of the optimal joint move. The second uses branch and bound to reduce the fraction of the joint action space that needs to be searched in order to compute the messages from functions to variables, which is the main bottleneck of the max-sum algorithm. We empirically showed that for 5 sensors, these techniques prune 92% of joint moves, thus significantly reducing the number of utility function evaluations, which are particularly expensive in the mobile sensor domain. Moreover, we showed that root mean squared error with which the spatial phenomenon is predicted by the MS1-5 strategy is approximately 50% lower compared to a greedy single step look-ahead algorithm. Our future work in this area is to extend the proposed approach by adopting techniques from sequential decision making to do non-myopic path planning, while keeping computational costs in check.

7. REFERENCES

[1] M. Ahmadi and P. Stone. A multi-robot system for continuous area sweeping tasks. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1724–1729, 2006.

[2] S. Aji and R. McEliece. Generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325 – 343, 2000.

[3] N. Cressie. *Statistics for Spatial Data*. Wiley-Interscience, 1993.

[4] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, pages 639–646, May 2008.

[5] E. Fiorelli, N. Leonard, P. Bhatta, D. Paley, R. Bachmayer, and D. Fratantoni. Multi-auv control and adaptive sampling in monterey bay. *IEEE Journal of Oceanic Engineering*, 31(4):935 – 48, 2006.

[6] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz, Jr., and M. Tambe, editors, *Distributed Sensor Networks*, chapter 11, pages 257–295. Kluwer Academic Publishers, 2003.

[7] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 265–272, New York, NY, USA, 2005. ACM Press.

[8] K. Low, J. Dolan, and P. Khosla. Adaptive multi-robot wide-area exploration and mapping. In *7th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS-08)*, Estoril, Portugal, May 2008.

[9] A. Meliou, A. Krause, C. Guestrin, and J. M. Hellerstein. Nonmyopic informative path planning in spatio-temporal models. In *Proc. of AAAI-07*, pages 602–607, 2007.

[10] M. A. Osborne, A. Rogers, S. D. Ramchurn, S. J. Roberts, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In *Proceedings of the seventh International Conference on Information Processing in Sensor Networks*, 2008.

[11] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland, Aug. 2005.

[12] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[13] A. Singh, A. Krause, C. Guestrin, W. J. Kaiser, and M. A. Batalin. Efficient planning of informative paths for multiple robots. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2204–2211, 2007.

Distributed Adaptive Sampling, Forwarding, and Routing Algorithms for Wireless Visual Sensor Networks

Johnsen Kho Long Tran-Thanh Alex Rogers Nicholas R. Jennings

School of Electronics and Computer Science,
University of Southampton,
Southampton, SO17 1BJ, UK.
{jk05r,ltt08r,acr,nrj}@ecs.soton.ac.uk

ABSTRACT

The efficient management of the limited energy resources of a wireless visual sensor network is central to its successful operation. Within this context, this paper focuses on the adaptive sampling, forwarding, and routing actions of each node in order to maximise the information value of the data collected. These actions are inter-related in this setting because each node's energy consumption must be optimally allocated between sampling and transmitting its own data, receiving and forwarding the data of other nodes, and routing any data. Thus, we develop two optimal decentralised algorithms to solve this distributed constraint optimization problem. The first assumes that the route by which data is forwarded to the base station is fixed, and then calculates the optimal sampling, transmitting, and forwarding actions that each node should perform. The second assumes flexible routing, and makes optimal decisions regarding both the integration of actions that each node should choose, and also the route by which the data should be forwarded to the base station. The two algorithms represent a trade-off in optimality, communication cost, and processing time. In an empirical evaluation on sensor networks (whose underlying communication networks exhibit loops), we show that the algorithm with flexible routing is able to deliver approximately twice the quantity of information to the base station compared to the algorithm using fixed routing (where an arbitrary choice of route is made). However, this gain comes at a considerable communication and computational cost (increasing both by a factor of 100 times). Thus, while the algorithm with flexible routing is suitable for networks with a small numbers of nodes, it scales poorly, and as the size of the network increases, the algorithm with fixed routing is favoured.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*

General Terms

Algorithms, Experimentation, Management, Performance

Keywords

Decentralised mechanism, distributed constraint optimization, information metric, inter-related adaptive sampling and routing

1. INTRODUCTION

Due to their flexibility and ease of deployment, wireless sensor networks, composed of battery powered sensor nodes that wirelessly communicate and route information sampled from the environment through the network to a base station, are becoming increasingly prevalent in a wide variety of applications, including environmental monitoring [8], area surveillance [1, 10], and object tracking in battlefields [4]. In particular, the rapidly increasing computational power of the nodes deployed within such networks has allowed them to perform ever more sophisticated tasks, and recently, *wireless visual sensor networks* (WVSN), whose nodes consist of spatially distributed smart camera devices, which are capable of performing basic capturing and processing of visual data, before forwarding it to the base station to be fused and analysed, have received increasing attention within the research community [13].

Such networks are intended for distributed image acquisition over large, and possibly hostile environments, and as such, are required to operate for extended periods of time with minimal human intervention. However, the increased computational power of the nodes within a WVSN (compared to those typically deployed within a conventional wireless sensor network), the large amounts of visual information that they collect, and the high energy cost of wirelessly communicating this information through the network, mean that efficient energy management is a key challenge in these networks.

To date, this challenge has been addressed through two complementary approaches: namely (i) hardware and (ii) software solutions. Within the former, advances in chip manufacture have successfully reduced the power consumption of nodes, helping to improve their longevity, and, in turn, the network's lifetime [3]. From the latter perspective, work has addressed the two main actions that such sensor nodes can vary in order to make their energy management more efficient: (i) their *sampling rate* (i.e. *how much* visual data they acquire) and (ii) their *communication* of data capabilities (those include selecting the most energy efficient path between the node and the base station given that the nodes may have different energy constraints and communication costs).

In particular, recent work has explored decentralised coordination algorithms that enable the nodes to autonomously adapt and adjust their sampling and communication behaviours. This coordination is computationally expensive since the sampling and communication decisions are inter-dependent. This is because each node's energy consumption must be optimally allocated between *sampling* and *transmitting* its own data, *receiving* and *forwarding* the data of other nodes, and *routing* any data. In such a setting, the choices of one node can potentially affect all other nodes in the network. However, much of this work has specifically addressed nodes

that are assumed to be extremely low power, computationally constrained devices. As such, it considers simple heuristic algorithms that allow the nodes to make local decisions to improve the overall performance of the network (see Sect. 5 for more details).

While such approaches have proved valuable in the context in which they were developed, when applied to WVSN they do not fully exploit the computational power available to the nodes. Furthermore, the large amounts of visual data that the nodes within the WVSN collect and communicate means that the communication resources available to the decentralised coordination mechanism are much greater than those of many conventional wireless sensor networks. When taken together, the move to WVSN means that there is now the possibility of deploying algorithms that can optimally maximise the overall effectiveness of the network through distributed computation, rather than local heuristics. It is this challenge that we address in this paper, and to this end, we adopt an agent-based approach in which each node is represented as an autonomous agent (with private information regarding its own state), and the complex, inter-connected, and rapidly changing network, as a multi-agent system. The individual agents must then cooperatively coordinate their activities to achieve system-wide goals.

Against this background, in this paper, we develop a novel optimal decentralised algorithm that varies each node's sampling, transmitting, and forwarding rates to ensure all nodes in a network focus their limited resources on maximising the information content of the data collected at the base station. This algorithm assumes that the route by which data is forwarded to the base station is *fixed* (either because the underlying communication network is a tree, or because an arbitrary choice of route has been made), and uses a distributed dynamic programming approach to extensively truncate the search space of potential allocation decisions. We then extend this approach to deal with *flexible* routing, in which each node not only makes optimal decisions regarding the sampling, transmitting, and forwarding actions, but also determines the optimal route by which this data should be forwarded. To ground and evaluate these algorithms, we empirically evaluate them and show that they represent a trade-off in optimality, communication cost, and processing time. In more detail, we show that when deployed on sensor networks with loopy topology (i.e. where data can follow multiple paths to the base station), the algorithm with flexible routing is able to deliver approximately twice the quantity of information to the base station compared to that of the algorithm using fixed routing. However, this gain comes at considerable communication and computational cost (increasing both by a factor of 100 times).

The remainder of this paper is organized as follows. In Sect. 2 we state the formal model of our distributed constraint optimization problem. In Sect. 3 we detail our two novel algorithms and show how we find the optimal local allocation decisions. Our experimental results are presented in Sect. 4. We present previous work in this area in Sect. 5 and we conclude in Sect. 6.

2. PROBLEM DESCRIPTION

Here, we formalise the generic adaptive sampling, transmitting, forwarding, and routing problem that we face. To this end, let n be the number of nodes within a WVSN system and the set of all nodes (or agents) be $I = \{1, \ldots, n\}$. Each node $i \in I$ can sample at s_i different rates over a period of time. Its set of possible sampling (or frame) rates is denoted by $C_i = \{c_i^1, \ldots, c_i^{s_i}\}$. Each element of this set, c_i^j, is a positive integer that describes the number of samples that the node takes during any specific time interval.

Each node has private information regarding the information content of the samples that it acquires, and this is represented by an array of 2-tuples $\mathfrak{F}_i = \left[(0,0), \left(c_i^1, v_i^{c_i^1}\right), \ldots, \left(c_i^{s_i}, v_i^{c_i^{s_i}}\right)\right]$, where the first value of each tuple is the number of samples that the node may take and $v_i^{c_i^j}$ is the corresponding information content for those c_i^j samples. Given the fact that more samples will generally generate visual data with a higher information content, we define $v_i^{c_i^j} = \alpha_i \cdot c_i^j$, where α_i is a weighting factor (with support $[0,1]$) that models the typical situation that the sensors within the network are heterogeneous, having different capabilities (i.e. the resolution of their cameras, the quality of their optics, or the sophistication of their image processing algorithms) and fields of view, and thus, samples from some sensors will contribute more to the total amount of information collected at the base station than others [11]. However, we note that our algorithms are not restricted to this linear information valuation function and, in some domains, it may be more valid to model the information as a strictly concave function where continuing to increase the sampling rate generates decreasing gains in information content [2]. We assume that should the node choose to acquire no samples, it will yield zero information value. Furthermore, we assume that the visual data captured by a node needs to be processed at the base station with that of other nodes, and therefore the information content of the data will only be accounted for if it arrives successfully at the base station.

We further assume that each node has an energy budget, B_i (also a private value initially known only to the node), such that its total energy consumption can not exceed this budget. We consider three specific kinds of energy consumption for each node in the network; namely the energy required to (i) acquire, e_i^{s}, (ii) transmit, e_i^{Tx}, and (iii) receive, e_i^{Rx}, a single sample. We disregard the energy required for other types of processing since it is negligible in comparison. Now, since the node has to transmit its own data towards the base station, the total energy required for this activity is thus $E_i^{\mathrm{S}} = e_i^{\mathrm{s}} + e_i^{\mathrm{Tx}}$ per sample (we will later on refer to the combination of these processes as *sensing*). Similarly, the node could potentially spend a portion of its energy to help its neighbourhood nodes to *forward* their own samples (and/or data that this node is forwarding for another node). This incoming data forwarding process requires a total energy of $E_i^{\mathrm{F}} = e_i^{\mathrm{Rx}} + e_i^{\mathrm{Tx}}$ per sample.

Each node initially stores its collected samples into its local memory buffer in order to be transmitted at a later stage. The transmission period and interval are predetermined. During each transmission phase, the transmitter module of each node is turned on for the purpose of transmitting data or message packets to the base station in a multi-hop fashion. Battery-powered visual sensor nodes typically offer reasonably small on-board memory and, hence, at the end of the transmission phase, each node's memory buffer is flushed, reinitialized, and ready to store new sampled data [6].

We describe the route through which the samples, c_i^j, will be transmitted to the base station by the vector $R(c_i^j) = (r_i^1, \ldots, r_i^b)$, where $r_i^l \in I$. The first element of this vector is the origin node that actually takes the samples. Each subsequent element of this vector is unique and r_i^l will forward the data to r_i^{l+1}. Thus, for the data value of c_i^j samples to be taken into account, its routing set must contain the base station node as its last node.

Given the formal description of the problem above, we now wish to maximise the value of the collected data that arrives at the base station. That is, we wish to solve:

$$\mathcal{D}_i^* = \arg \max_{\{\mathcal{D}_i, \mathfrak{F}_i\}} \sum_{i=1}^{n} \sum_{c_i^j \in C_i} d_i^{R(c_i^j)} v_i^{c_i^j} \tag{1}$$

In this expression, $d_i^{R(c_i^j)} \in \mathcal{D}_i \in \{0,1\}$ is a decision variable

where "1" represents a state where the node carries out the corresponding c_i^j sampling action and the samples follow the $R(c_i^j)$ route to arrive at the base station, and "0" represents the state where the node does not carry out the corresponding c_i^j sampling action. This objective function is maximised subject to the energy budget constraint on each node $i \in I$, such that:

$$B_i \geq c_i^j E_i^{\mathrm{S}} + f_i E_i^{\mathrm{F}} \tag{2}$$

where f_i represents the total incoming data (or forwarded samples from its set of neighbourhood nodes D_i) and is given by:

$$f_i = \sum_{d \in D_i} c_d^j + f_d \tag{3}$$

where $i \in R(c_d^j)$. Note that the total outgoing number of samples from node i is thus $c_i^j + f_i$. We must also constrain the node to chose one and only one sampling rate, such that:

$$\sum_{c_i^j \in C_i} d_i^{R(c_i^j)} = 1 \tag{4}$$

for all different possible routes in the network.

The problem, as formulated above, is similar to *multiple-choice knapsack* problems[1] (i.e. NP-complete resource allocation or distributed constraint optimization problems) [12], that exhibit the *optimal substructure* property[2]. Given this insight, we propose algorithms based on the sort of computationally efficient dynamic programming technique that are often used on such knapsack problems for solving multi-agent distributed coordination problems.

3. THE ALGORITHMS

We now focus on the algorithms used by the nodes to make optimal use of their energy resources in order to cooperatively sense, forward, and route data to the base station. Our approach places higher priority on those samples that have a higher information content, and this is achieved by exchanging coordination messages between connected nodes. To this end, we distinguish three types of messages being exchanged by the nodes; namely (i) *actual data messages* containing visual data sampled by the nodes, and two types of *coordination* messages composed of (ii) *meta-data messages* describing the information content of the visual data together with the number of samples taken to produce that data (i.e. $v_i^{c_i^j}$ and c_i^j respectively), and (iii) *control messages* containing the allocation decisions. In WVSNs, the size of the actual data messages overwhelms that of the coordinations messages and, hence, exchanging these before sending the actual data can significantly increase the information collected at the base station by making more efficient use of each node's constrained energy.

The goal of the algorithms that we derive is to calculate the optimal sampling and routing actions of each node. This is given by:

$$\begin{aligned} Cmax_I = &\{(i, c_i^j, R(c_i^j)) | d_i^{R(c_i^j)} = 1, \\ &\forall i \in I, \forall c_i^j \in C_i, \forall d_i^{R(c_i^j)} \in \mathcal{D}_i^*\} \end{aligned} \tag{5}$$

[1]There are m items and the set of all items $T = \{1, \ldots, m\}$. Each item $t \in T$ has a value v_t and a weight w_t. The items are subdivided into o categories and exactly one item must be taken from each category. The maximum weight that can be carried in a bag is G. Given these, we need to determine which items to include in the bag such that the total weight does not exceed its given limit, while the total value is maximised.

[2]This property means that the optimal solution can be constructed efficiently from optimal solutions to its subproblems.

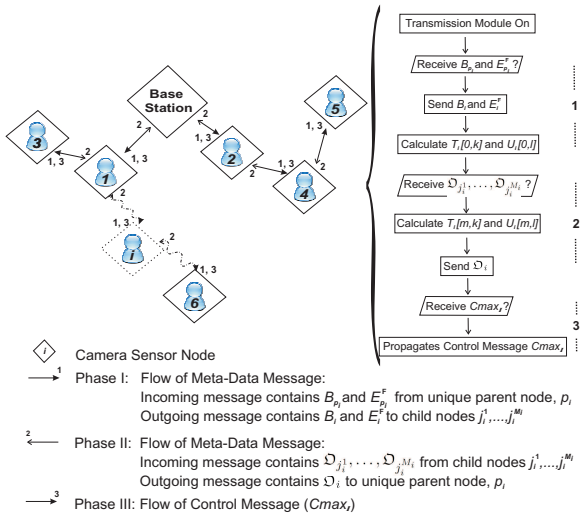


Figure 1: The flow of the algorithm in a connected and undirected tree-structured WVSN. We assume that communication is bi-directional and multiple nodes within range can thus establish a connection. Dotted node i could represent any subtrees in the network.

and represents a set of 3-tuples indicating for each node in the network, the sensing and forwarding rates that it should adopt, and the route that it should use to transmit its own and its forwarded data to the base station, in order to maximise the objective function in (1), subject to the constraints in (2) and (4). We now present our two novel adaptive sensing, forwarding, and routing algorithms. Both of them are efficient as they satisfy the data flow conservation of the network where no energy is wasted by transmitting data that later will not be forwarded to the final destination.

3.1 Algorithm With Fixed Routing

In this case, each node $i \in I$ can only forward its data to exactly one other node (which will later be referred as its *parent*). This may be because the underlying communication network of the WVSN is tree structured, or because it actually exhibits loops but an arbitrary choice of route has been made (effectively turning the loopy communication network into a tree). An example of a WVSN whose underlying network structure is a tree structure is shown in Fig. 1. Note that in such tree-structured networks, there is only one unique route between each node and the base station (e.g. $R(c_4^j) = (4, 2, \text{base station})$ and $R(c_3^j) = (3, 1, \text{base station})$).

In general, the nodes within a network will deplete their energy resources at different rates since they will have different sampling rates, and will be transmitting different quantities of visual data. Each node $i \in I$ thus needs to compute the highest information value it can transmit by using at most $b_i^k \leq B_i$ of its energy. As described earlier, the energy consumption of node i only includes E_i^{S} and E_i^{F} (i.e. the energy to sense and forward a sample respectively). It is therefore sufficient that b_i^k satisfies:

$$\begin{aligned} &b_i^k = c_i^j E_i^{\mathrm{S}} + f_i E_i^{\mathrm{F}} \quad \text{where } c_i^j, f_i \geq 0 \\ &b_i^k \leq B_i \end{aligned} \tag{6}$$

where c_i^j is its own number of samples and f_i is the number of forwarded incoming samples.

Now, let $\mathfrak{O}_i = \left[\left(b_i^1, Vmax_i^1, Cmax_i^1\right), \ldots, \left(b_i^{K_i}, Vmax_i^{K_i},\right.\right.$

Algorithm 1 Optimal adaptive sensing and forwarding with fixed routing.

```
1: loop
2:   if sTime = NOW then                                   ▷ Time to sample.
3:     readings ← PerformSampling(sTime)                   ▷ Sampling action, c_i^j.
4:     SetSTime(sTime + sRate)
5:   end if
6:   if tTime = NOW then                                   ▷ Time to transmit, transmission module is turned on.
7:     [B_{p_i}, E^F_{p_i}] ← WaitMetaData(p_i)            ▷ Receives B_{p_i} and E^F_{p_i} from its unique parent node, p_i.
8:     for each j_i^m ∈ J_i do                             ▷ Iterates each child node in J_i = {j_i^1, ..., j_i^{M_i}}.
9:       SendMetaData(j_i^m, [B_i, E^F_i])                 ▷ Sends B_i and E^F_i to child node j_i^m.
10:    end for
11:    CalcFirstRowTables(readings)                        ▷ Calculates the 1st rows of T_i and U_i using (7) and (10) respectively.
12:    if ¬leafNode then
13:      for each j_i^m ∈ J do
14:        O_{j_i^m} ← WaitMetaData(j_i^m)                 ▷ Receives O_{j_i^m} from child node j_i^m.
15:        CalcTheRestTables(O_{j_i^m})                    ▷ Calculates the other rows of T_i and U_i using (8) and (11) respectively.
16:      end for
17:    end if
18:    O_i ← CalcMetaDataArray()                           ▷ Determines O_i which is basically the last row of U_i.
19:    SendMetaData(p_i, O_i)                              ▷ Sends O_i to unique parent node, p_i.
20:    Cmax_I ← WaitControlMessage(p_i)                    ▷ Receives control message from unique parent node, p_i.
21:    PropagateControlMessage(j_i^m, Cmax_I)              ▷ Sends control message to each child node, j_i^m ∈ J_i.
22:    PerformTransmit(readings, Cmax_I)
23:    SetNodeOptimalBehaviour(Cmax_I)                     ▷ Sets node's optimal sensing and forwarding actions.
24:    SetTTime(tTime + tRate)                             ▷ Node sets its next transmitting time.
25:    readings ←{}
26:  end if
27: end loop
```

$Cmax_i^{K_i}\big)\Big]$ denote an array of 3-tuples sorted incrementally by b_i^k where $k = 1, \ldots, K_i$, and b_i^k is the energy limit that satisfies (6) and will later on be referred to as the labels of $\mathfrak{O}_i$. $Vmax_i^k$ is the maximum information value that node i can transmit to its parent by using at most b_i^k, and $Cmax_i^k$ is the set of sensing and forwarding actions that will produce data with the value of $Vmax_i^k$.

The algorithm installed on each node runs in three phases (see Fig. 1 and Algorithm 1). In the first, meta-data message propagation is initiated by the base station. To this end, messages containing the value of each node's energy budget, B_i, and energy consumption for forwarding, E_i^{F}, are propagated down the tree (i.e. as soon as any node receives this information from its unique parent node, p_i (see state 1 or line 7), it sends its own information to its set of children, $J_i = \left\{j_i^1, \ldots, j_i^{M_i}\right\}$ (line 9)). Having sent this information each node i then enters an idle mode in which it waits for the $\mathfrak{O}$ meta-data arrays from its child nodes.

In the second phase, after all the $\mathfrak{O}$ meta-data arrays have arrived from its children (denoted by $\mathfrak{O}_{j_i^1}, \ldots, \mathfrak{O}_{j_i^{M_i}}$, see state 2 or lines 14-15), node i then calculates its own $\mathfrak{O}_i$ (line 18). To do so, it constructs a table, T_i, which has $M_i + 1$ rows numbered from 0 to M_i, and K_i columns, where K_i is the number of all the b_i^k values that satisfy (6). See Table 1 in which each column k has label b_i^k. Let $T_i[x, y]$ denote the element of the table that is in the x^{th} row and the column with label b_i^y. As every node could choose not to sample (yielding zero value), then $\mathfrak{O}_{j_i^m}[0] = T_i[m, 0] = 0$ for all $0 \leq m \leq M_i$, where $\mathfrak{O}_{j_i^m}[x]$ is the x^{th} element of $\mathfrak{O}_{j_i^m}$. Moreover, we can assume that the set of labels in each $\mathfrak{O}_{j_i^m}$ that node i has received is the same as the set of labels in its table T_i. We will show how we can guarantee this later on. Hence, T_i's other elements are filled as follows:

$$T_i[0, k] = \max\{v_i^{c_i^j}\} \tag{7}$$

$$T_i[m, k] = \max_{0 \leq r \leq k} \left\{T_i[m-1, r] + Vmax_{j_i^m}^{k-r}\right\} \tag{8}$$

for all $1 \leq k \leq K_i$ and $1 \leq m \leq M_i$, where $(c_i^j, v_i^{c_i^j}) \in \mathfrak{F}_i$, and $\mathfrak{F}_i$ is the array of 2-tuples defined in the previous section.

According to (7), we can see that $T_i[0, k]$ stores the maximum information value of data that can be delivered to node i's parent by sensing only (with the energy consumption not exceeding the energy limit b_i^k). Due to the fact that each of the sets of labels in $\mathfrak{O}_{j_i^m}$ is equivalent to the set of labels of table T_i, (8) gives the maximum value of data that node i can deliver to its parent (noting that this data does not only include its own sensed data but also its children's data that will potentially be forwarded through it). Hence, $T_i[1, k]$ is the maximum value that can be sent by taking into account the sensed data and the data from j_i^1, with respect to the b_i^k energy limit. $T_i[2, k]$ stores the maximum value when the data from child node j_i^2 is also included. In general, $T_i[m, k]$ is the maximum information value that node i can transmit to its parent, given the b_i^k energy limit. The data considered is the potential forwarded data from child nodes $j_i^1, \ldots, j_i^m$ and node i's own sensed data.

Note that while it is necessary to construct the entire table, as in conventional dynamic programming solutions to the multiple-choice knapsack problem, it is only the last row that provides useful meta-data regarding the maximum information values of data that can be transmitted given different feasible values of b_i^k. Indeed, it is only the last element of this row that represents the maximal information value that node i can transmit to the parent node.

To illustrate how the elements of the table are calculated in a clearer way, consider Tables 1 and 2 in which the information values of node i's sensed data and the $Vmax_{j_i^m}^k$ values of $\mathfrak{O}_{j_i^m}$ arriving from its child nodes j_i^m respectively are chosen arbitrarily for illustrative purposes. The rows of Table 1 represent the set of nodes whose data has been taken into account. For instance where $row = i$, if node i has b_i^0, b_i^1, b_i^2, b_i^3, or b_i^4 amount of energy limit, in return it will be able to sense it own data with the maximum value of 0, 12.34, 14.56, 28.25, or 50.98 correspondingly. These values are calculated using (7). $\mathfrak{O}_{j_i^1}$ then arrives (see Table 2 where $row = \mathfrak{O}_{j_i^1}$) from its child node j_i^1 containing the maximum values that this node could potentially forward to node i.

The elements of T_i's second row (i.e. $row = \{i \cup j_i^1\}$) can thus be calculated using (8). These elements represent the maximum information that node i could send by taking into account not only its own sensed data, but also the data that could be potentially forwarded from its child node j_i^1. For instance where $column = b_i^1$, node i could allocate all its b_i^1 energy resources to either sense and transmit its own data or to forward data from its child node j_i^1. In this case, the node chooses to sense and transmit its own data since it has a higher value. Where $column = b_i^2$, however, the node again allocate all its b_i^2 energy resources to either sense its own data or to forward its child node j_i^1's data. Alternatively it could as well divide its b_i^2 energy resources by allocating a portion of b_i^1 energy resources to its own and another b_i^1 to its child node. In this case, it turns out that the latter alternative yields the highest information value of 19.32. T_i's other elements are calculated in a similar way.

Now, the next step of the algorithm is to calculate $\mathfrak{O}_i$. To do so, let U_i denote a table similar to T_i. However, its labels b_i^l, now, satisfy the following:

$$\begin{aligned} &b_i^l = (c_i^j + f_i)E_{p_i}^{\mathrm{F}} \quad \text{where } c_i^j, f_i \geq 0 \\ &b_i^l \leq B_{p_i} \end{aligned} \tag{9}$$

Table 1: The T_i table of node i. Its $\mathfrak{O}_i$ meta-data array is represented by the dotted rectangle.

T_i	b_i^0	b_i^1	b_i^2	b_i^3	b_i^4	b_i^k
$\{i\}$	0	12.34	14.56	28.95	50.98	⋱
$\{i \cup j_i^1\}$	0	12.24	19.32	45.89	58.23	
⋮	⋮	⋮	⋮	⋮	⋮	
$\{i \cup j_i^1 \cup \ldots \cup j_i^{M_i}\}$	0	12.34	28.78	45.89	58.23	⋱

Table 2: $\mathfrak{O}_{j_i^m}$ meta-data arrays that have arrived from each child nodes $j_i^1, \ldots, j_i^{M_i}$.

$\mathfrak{O}_{j_i^m}$-s	b_i^0	b_i^1	b_i^2	b_i^3	b_i^4	b_i^k
$\mathfrak{O}_{j_i^1}$	0	6.98	15.67	45.89	48.99	⋱
⋮	⋮	⋮	⋮	⋮	⋮	
$\mathfrak{O}_{j_i^{M_i}}$	0	6.79	28.78	35.89	51.88	⋱

where B_{p_i} is the energy budget of i's unique parent node, p_i, and $E_{p_i}^{\mathrm{F}}$ is the value of energy consumption of the parent for forwarding a sample. Recall that these values were delivered to node i in the first stage. Let L_i denote the number of all b_i^l that satisfy (9). Similarly, we can calculate table U_i's elements in a similar fashion to those of T_i as described earlier, but with the new labels:

$$U_i\left[0,l\right] = \min\left(\max\{v_i^{c_i^j}\}, T_i\left[0,K_i\right]\right) \tag{10}$$

$$U_i\left[m,l\right] = \min\left(\max_{0\le r\le l}\left\{U_i\left[m-1,r\right] + Vmax_{j_i^m}^{l-r}\right\}, T_i\left[m,K_i\right]\right) \tag{11}$$

for all $1 \le l \le L_i$ and $1 \le m \le M_i$, where $(c_i^j, v_i^{c_i^j}) \in \mathfrak{F}_i$.

We can now construct the meta-data array of node i such that $\mathfrak{O}_i = \left[\left(b_i^1, U_i\left[M_i,1\right], Cmax_i^1\right), \ldots, \left(b_i^{L_i}, U_i\left[M_i,L_i\right], Cmax_i^{L_i}\right)\right]$, where $U_i\left[M_i,l\right]$ is the maximum information value that node i can transmit to its parent node (by using at most b_i^k energy) which can subsequently forward the received i's data by using at most b_i^l energy. $Cmax_i^l$ is the set of sensing actions that will produce data with the value of $U_i\left[M_i,l\right]$. Hence, once $\mathfrak{O}_i$ is sent to the parent node, its labels will be the same as those in table T_{p_i} of the parent node. This second phase meta-data message containing $\mathfrak{O}_i$ propagates up the network arriving back at the base station (line 19).

In the third phase of the algorithm, each parent node will have received meta-data arrays from all of its children. The base station will be able to calculate the highest information value it can potentially receive from all the nodes beneath it in the network. Based on the structure of $\mathfrak{O}_i$, each node i can easily determine what amount of data it should receive from each child node and, hence, how many samples it should acquire and transmit itself. A control message containing this set is then propagated down the network (see state 3 or lines 20-21), and this control message informs each node of its optimal decisions (lines 22-23). In this way, there is a guarantee that all of the data transmitted by each node will reach the base station. The control message eventually reaches the leaf nodes which then start to acquire and transmit visual data as planned.

3.2 Algorithm with Flexible Routing

Next, we consider the algorithm which assumes flexible routing, and makes optimal decisions regarding both the sensing and forwarding actions that each node should perform, and also the route by which data should be forwarded to the base station (see Fig. 2 for an illustration of this case). In order to make the routing decision tractable, we place one minor restriction on the routes that our algorithm can consider. Specifically, we assume that the nodes always forward their data toward the base station; that is, they will not forward data to a node that is further from the base station (in terms of hop count) than themselves. We believe this is a reasonable assumption. There may be cases where several nodes are better off taking longer paths. However, in general such paths will deplete the energy resources of a greater number of nodes, and are thus unlikely to be optimal solutions. Furthermore, we assume that the data samples of a node can only be sent as a bundle (i.e. they are indivisible). The data readings of different nodes can, however, be sent through different routes to the base station.

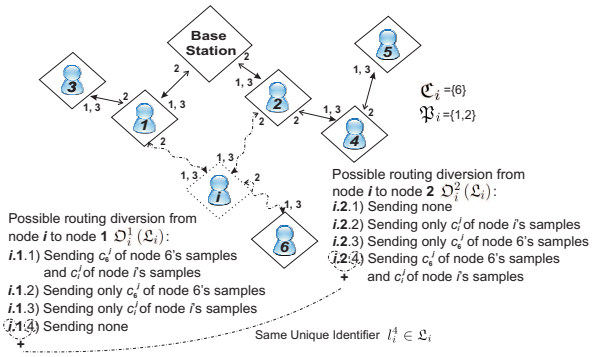


Figure 2: The flow of the algorithm that assumes flexible routing and makes optimal decisions regarding both sensing, forwarding, and next-hop (or routing) decisions. The phases involved in this algorithm are similar to those in the algorithm for fixed routing.

With these assumptions, we now need to organize the nodes into different *levels*. The first consists of all the nodes that have a 1-hop shortest path to the base station (nodes 1 and 2 in Fig. 2). Nodes that belong to the second level have a 2-hop shortest path to the base station (nodes i, 3, and 4). Nodes 5 and 6 have a 3-hop shortest path. Now, according to this hierarchy, each node can only forward its data to higher level nodes within its transmission range. In Fig. 2, for example, node i has two potential shortest routes to choose from; namely (i) node 1 which results in route $R(c_i^j) = (i, 1,$base station) and (ii) node 2 which results in route $R(c_i^j) = (i, 2,$base station). Node i does not consider routing through node 6 since 6 is a greater hop count away from the base station than it is. Furthermore, as we can see from the figure, node i has potentially two bundles of data to consider (its own and data that it is forwarding for node 6). In addition, it has two possible shortest paths to choose between (either through node 1 or 2 for each of the bundled data). Thus, a number of routing options exist for this node. It could send both bundles of data through node 1, such that both $R(c_i^j)$ and $R(c_6^j)$ contain $(i, 1, \ldots)$, or it could send them through node 2. Other alternatives are to send each of them separately through each possible route, such that $R(c_i^j)$ contains $(i, 1, \ldots)$ and $R(c_6^j)$ contains $(i, 2, \ldots)$, or the other way around.

Now, let $\mathfrak{P}_i$ denote the set of parent nodes (which are nodes with a one hop shorter shortest path to the base station) of node i and $\mathfrak{C}_i$ denote the set of its descendants. Thus, at each node $i \in I$, there are at most $|\mathfrak{P}_i|^{|\mathfrak{C}_i|+1}$ possibilities to forward the bundled data, where $|\mathfrak{P}_i|$ and $|\mathfrak{C}_i|$ are the sizes of $\mathfrak{P}_i$ and $\mathfrak{C}_i$ respectively. This is because each node has to forward $|\mathfrak{C}_i| + 1$ bundles through $|\mathfrak{P}_i|$ different shortest paths. Next, let $\mathfrak{L}_i$ denote the set of these possibil-

Algorithm 2 Optimal adaptive sensing and forwarding with flexible routing.

1: **loop**
2: **if** $sTime = NOW$ **then** ▷ Time to sample.
3: $readings \leftarrow$ PERFORMSAMPLING($sTime$) ▷ Sampling action, c_i^j.
4: SETSTIME($sTime + sRate$)
5: **end if**
6: **if** $tTime = NOW$ **then** ▷ Time to transmit, transmission module is turned on.
7: **for** each $p_i^{\mathfrak{n}} \in \mathfrak{P}_i$ **do** ▷ Iterates each parent node, $p_i^{\mathfrak{n}} \in \mathfrak{P}_i$.
8: $[B_{p_i^{\mathfrak{n}}}, E^{\mathrm{F}}_{p_i^{\mathfrak{n}}}] \leftarrow$ WAITMETADATA($p_i^{\mathfrak{n}}$) ▷ Receives $B_{p_i^{\mathfrak{n}}}$ and $E^{\mathrm{F}}_{p_i^{\mathfrak{n}}}$ from parent node $p_i^{\mathfrak{n}}$.
9: **end for**
10: **for** each $j_i^m \in J_i$ **do** ▷ Iterates each child node in $J_i = \{j_i^1, \ldots, j_i^{M_i}\}$.
11: SENDMETADATA($j_i^m, [B_i, E_i^{\mathrm{F}}]$) ▷ Sends B_i and E_i^{F} to child node j_i^m.
12: **end for**
13: CALCFIRSTROWTABLES($readings$) ▷ Calculates the 1^{st} rows of T_i and $U_i^{p_i^{\mathfrak{n}}}$ (for each parent node, $p_i^{\mathfrak{n}}$ in $\mathfrak{P}_i$) using (7) and (10) respectively.
14: $\mathfrak{C}_i \leftarrow \{i\}$ ▷ Adds this node to the set of descendants $\mathfrak{C}_i$.
15: **if** $\neg leafNode$ **then**
16: **for** each $j_i^m \in J_i$ **do**
17: $\mathfrak{O}^i_{j_i^m} \leftarrow$ WAITMETADATA(j_i^m) ▷ Receives $\mathfrak{O}^i_{j_i^m}$ from child node j_i^m.
18: CALCTABLESWITHIDENTIFIER($\mathfrak{O}^i_{j_i^m}$) ▷ Calculates the other rows of T_i using (8) by identifying the same forwarding partition with the same unique identifier.
19: $\mathfrak{C}_i \leftarrow \mathfrak{C}_i \cup j_i^m$ ▷ Adds child node j_i^m to the set of descendants $\mathfrak{C}_i$.
20: **end for**
21: **end if**
22: **for** each $p_i^{\mathfrak{n}} \in \mathfrak{P}_i$ **do**
23: $\mathfrak{L}_i \leftarrow$ PARTITIONPOSSIBLEFORWARDING($\mathfrak{C}_i$) ▷ Partitions the possible forwardings using a mapping function that decides the direction of each bundle, u_i^j, from one of its descendants in $\mathfrak{C}_i$.
24: $\mathfrak{O}_i^{p_i^{\mathfrak{n}}} \leftarrow$ CALCMETADATAARRAY($\mathfrak{L}_i$) ▷ Calculates the other rows of $U_i^{p_i^{\mathfrak{n}}}$ using (11) to forms its own $\mathfrak{O}_i^{p_i^{\mathfrak{n}}}$ meta-data for parent node $p_i^{\mathfrak{n}}$.
25: SENDMETADATA($p_i^{\mathfrak{n}}, \mathfrak{O}_i^{p_i^{\mathfrak{n}}}$) ▷ Sends $\mathfrak{O}_i^{p_i^{\mathfrak{n}}}$ to parent node $p_i^{\mathfrak{n}}$.
26: **end for**
27: $Cmax_I \leftarrow$ WAITCONTROLMESSAGE($p_i^{\mathfrak{n}}$) ▷ Receives control message from parent node $p_i^{\mathfrak{n}}$ in $\mathfrak{P}_i$.
28: PROPAGATECONTROLMESSAGE($j_i^m, Cmax_I$) ▷ Sends control message to each child node, $j_i^m \in J_i$.
29: PERFORMTRANSMITINCROUTING($readings, Cmax_I$)
30: SETNODEOPTIMALBEHAVIOURINCROUTING($Cmax_I$) ▷ Sets node's optimal sensing, forwarding, and next-hop decisions.
31: SETTTIME($tTime + tRate$) ▷ Node sets its next transmitting time.
32: $readings \leftarrow \{\}$
33: **end if**
34: **end loop**

ities (with $|\mathfrak{L}_i| = |\mathfrak{P}_i|^{|\mathfrak{C}_i|+1}$) and each $l_i^t \in \mathfrak{L}_i$, a possible partition of forwarding at node i. That is, $l_i^t = \left[F\left(u_i^1\right), \ldots, F\left(u_i^{|\mathfrak{C}_i|+1}\right)\right]$ where u_i^j is the j^{th} bundle that might arrive at node i from one of its descendants, $F\left(u_i^j\right)$ is a mapping function that decides the forwarding direction (or path) for this particular bundle, and $u_i^{|\mathfrak{C}_i|+1}$ is node i's own bundle of samples.

Given this, our algorithm with flexible routing is similar to that with fixed routing, and as before, it runs in three phases (see Algorithm 2). The first, in which the parent nodes send their information regarding $B_{p_i^{\mathfrak{n}}}$ and $E^{\mathrm{F}}_{p_i^{\mathfrak{n}}}$ to their child nodes (where $p_i^{\mathfrak{n}} \in \mathfrak{P}_i$), is identical (see lines 7-13). There are, however, slight modifications in the next phase. These modifications are needed to keep track of all the possible partitions of forwarding for nodes which have more than one shortest path routes to the base station. In more detail, in the second phase, instead of sending one $\mathfrak{O}_i$ to a unique parent (as in the case of tree-structured networks), here, each node i has to calculate all the $\mathfrak{O}_i^{p_i^{\mathfrak{n}}}\left(l_i^t\right)$ meta-data arrays for each $l_i^t \in \mathfrak{L}_i$ partition of forwarding for each $p_i^{\mathfrak{n}} \in \mathfrak{P}_i$ (see lines 23-25). Specifically, this is achieved by first calculating the T_i table as we did for the first algorithm (line 17). In this case, however, we join each of the arriving $\mathfrak{O}^i_{j_i^m}\left(l^t_{j_i^m}\right)$ from its children $j_i^1, \ldots, j_i^{M_i}$ with those that belong to the same forwarding partition with the same unique identifier (line 18). The unique identifier is formed and attached to a particular partition of forwarding when there are more than one possible routes to forward to (line 23). As in Fig. 2, a feasible unique identifier could be the index of $l^t_{j_i^m}$. Next, we calculate $U_i^{p_i^{\mathfrak{n}}}$ tables for each $p_i^{\mathfrak{n}} \in \mathfrak{P}_i$ as in the first algorithm (line 24). The rest of the second phase and third phase remain the same as that of the algorithm with fixed routing described previously (see lines 27-30).

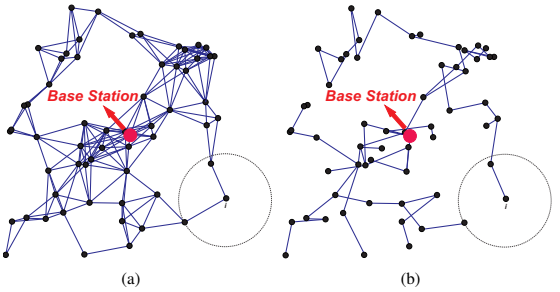


Figure 3: (a) A randomly created and connected WVSN (of 60 nodes) whose underlying communication network exhibits loops. (b) The resulting tree-structured network formed when each node makes an arbitrary choice of the route that its data will take toward the base station. The dotted circle in each graph represents the wireless range of node i. In both these networks, all nodes are set with the same transmission range.

4. EMPIRICAL EVALUATION

We now seek to evaluate their performance and effectiveness when applied to typical WVSN whose communication networks exhibit loops. Our goal in this empirical evaluation is to quantify the performance of the algorithms in terms of the quantity of information that they deliver to the base station, and the communication and computational costs of the coordination. We expect the algorithm with flexible routing to deliver more information, but make greater demands of computation and communication resources (because of the large number of alternative routes for the data that it must evaluate). However, given that the algorithm with fixed routing can always be applied in this setting by ignoring the fact that there exist alternative routing options, and just making an arbitrary choice, we are interested in the trade-off between the loss in information and the saving in resources that results. We first describe the experimental setup and then go onto the actual evaluation.

In our experiments, we create instances of a WVSN by randomly deploying the nodes within a unit square, and connecting them according to a randomly determined radio transmission range (extending this range as necessary to ensure that there are no unconnected nodes). Each resulting WVSN exhibits a loopy communication network such that for each node there are multiple alternative routes to the base station. We consider twenty different sampling actions for each node such that the possible sampling rates, C_i, of each node are initialized to $C_i = \{1, \ldots, 20\}$. The corresponding information content $v_i^{c_i^j}$ for each $c_i^j \in C_i$ sample is generated using the generic information metric (defined in Sect. 2), with the factor, α_i, randomly drawn from a uniform distribution with support $[0, 1]$. The energy budget of each node is randomly generated with a predetermined maximum value that ensures the network as a whole is energy constrained. We scale this predetermined max-

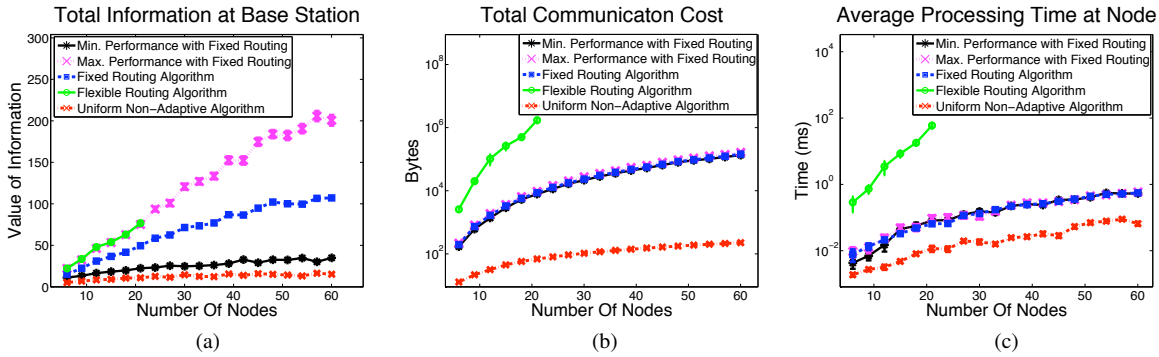


Figure 4: Simulation results showing the performance of the algorithms with flexible, fixed (with maximum and minimum performance), and uniform non-adaptive routing against (a) total information collected at the base station, (b) total communication cost for coordination, and (c) average computation time at each node.

imum value with the number of nodes in the network since larger networks require sensors to forward data for a larger number of nodes. We assume that each real valued number inside a coordination message (e.g. the value of B_i or c_i^j) occupies 4 bytes of communication cost, and the energy consumption for sensing and forwarding a sample is fixed throughout the entire experiment[3].

We apply our algorithm with flexible routing just once, directly on the loopy communication network of the WVSN (see Fig. 3(a) for an exemplar scenario), such that it determines both the optimal sensing and forwarding actions, as well as the routes. Prior to applying our algorithm with fixed routing, we allow each node to make an arbitrary choice of the route that its data (and any data that it forwards for other nodes) will take toward the base station. This effectively turns the loopy communication network into a tree-structured one, with each node effectively selecting their parent in the tree (see Fig. 3(b)). We then apply our algorithm with fixed routing to calculate the optimal sensing and forwarding decisions of each node. For each instance of the WVSN, we repeat this process 100 times, averaging over the unique instances of trees that result. We perform repeated experiments by creating 100 instances of the WVSN with 6, 9, ..., 60 nodes for the algorithm with fixed routing, and only up to 21 nodes for the algorithm with flexible routing (due to its increased computational cost).

We also benchmark our two algorithms against a uniform non-adaptive algorithm with fixed routing. This algorithm dictates that each sensor $i \in I$ in the network should simply choose to allocate its energy budget, B_i, equally to itself and each of its descendants such that it will naïvely sample and transmit the minimum of $\left(\frac{B_i}{|\mathfrak{C}_i| \cdot E_i^{\mathrm{S}}}, \frac{B_{p_i}}{|\mathfrak{C}_{p_i}| \cdot E_{p_i}^{\mathrm{S}}}\right)$ times regardless of whether the samples will eventually be forwarded towards the base station. $|\mathfrak{C}_i|$ and $|\mathfrak{C}_{p_i}|$ are the numbers of descendants of node i and node i's parent, p_i, respectively, and B_{p_i} is the energy budget of node p_i. E_i^{S} and $E_{p_i}^{\mathrm{S}}$ are the energy required by node i and p_i correspondingly in order to sense a sample.

We present the results of the simulation process described above in Fig. 4. The error bars shown represent the standard error in the mean, and we note that in some cases, the error bars are smaller that the plotted symbol size. Considering first Fig. 4(a), we observe that the algorithm with flexible routing delivers close to twice the quantity of information to the base station as does the fixed routing algorithm. This is as expected since in loopy communication networks, there are typically many alternative routes available for routing data, and the flexible algorithm is able to exploit them[4]. The uniform non-adaptive algorithm, however, performs poorly as it has no intelligence of adapting the nodes' actions. In the same figure, we also show the mean maximum and minimum performance of the algorithm with fixed routing (averaged over different trees for the same loopy network). Note that by making an appropriate choice of parent, we can derive performance close to that of the algorithm with flexible routing (without incurring any additional computation or communication cost as will be explained shortly).

However, the increased information delivered by the algorithm with flexible routing comes at considerable communication and computational cost. Figures 4(b) and 4(c) show the total size of coordination messages exchanged by the nodes and the average computation time of each node (both are presented on a logarithmic scale). Specifically, Fig. 4(b) shows that typically only a few tens of kilobytes of coordination message packets are required by the algorithm with fixed routing, while the algorithm with flexible routing exhibits approximately two orders of magnitude more; with a few megabytes of coordination message packets being exchanged. Likewise, Fig. 4(c) shows that the average computation time of a node required by the algorithm with fixed routing is typically less than 1 millisecond, while that of the algorithm with flexible routing approaches 100 milliseconds (a two orders of magnitude increase)[5]. The increase in terms of computation time is due to the additional time which the flexible routing algorithm requires in order to enumerate each possible partitions of forwarding.

More generally, these results indicate that the algorithm with flexible routing is able to deliver significantly more information to the base station, but incurs considerable additional computation and communication costs in doing so. The choice of algorithm thus depends on the application domain. If the network is small, and the size of the actual data messages is large, then the algorithm

[3]Note that we do not consider the failure, addition, or removal of nodes. Also, we do not consider the dropping or corruption of meta-data or control message packets, and hence assume that message packets are always transferred successfully to the destination.

[4]We remark that the quantity of information delivered does not increase monotonically. This is an artifact of the experimental setup since the scaling of the nodes' energy budget does not fully account for the necessary increase in sample forwarding.

[5]Measurements were performed on a 3GHz desktop PC. Typically, the nodes within a WVSN will use much lower powered processors and, thus, while we would expect the ratio between the algorithms to be the same, the overall computation time is likely to be longer.

with flexible routing is most appropriate. However, this algorithm scales poorly as the size or connectivity of the network increases (due to the exponential growth in the number of possible combinations of routing options that it must evaluate). In such cases, the size of the coordination messages may rapidly approach that of the actual data messages and, hence, coordination may not actually yield any energy saving. To address this, the algorithm with fixed routing may be run on the original loopy network by having each node make an arbitrary choice of route. While the quantity of information delivered to the base station will be reduced (by up to 50% in our experiments), this solution will scale well and use minimal communication and computational resources.

5. RELATED WORK

The work that is most closely related to ours is that of Padhy *et al.* who developed a decentralised adaptive sampling and routing protocol named *Utility-based Sensing and Communication Protocol* [8]. Within this mechanism, each node adjusts its sampling rate depending on a valuation function that assigns a value to newly sampled data. This protocol is intended for low power, computationally constrained devices, and as such, relies on a heuristic approach to estimate the opportunity energy cost used by each sensor for sampling, forwarding, and routing data. The protocol is not efficient and the integration of the node's actions is very limited since there is no guarantee that the transmitted data will actually be forwarded to the base station. For instance, there might be cases where nodes with data of a high value are unable to send their data to the base station because intermediate nodes have depleted their energy. The protocol could thus result in no data collection.

In a somewhat similar setting, Mainland *et al.* present a market-based approach for determining efficient node resource allocations [5]. Rather than manually tuning node resource usage, or providing specific algorithms as we do here, their approach defines a virtual market in which nodes sell goods (e.g. data sampling, listening, or forwarding) in response to global price information that is established by the end user. However, this approach involves an external coordinator to set prices in order to induce any particular global behaviour, and it is not clear how this price determination should be performed in order to elicit desirable system-wide properties.

Within the multi-agent systems literature, another useful technique that has emerged for solving distributed coordination problems is that of *distributed constraint optimization* (DCOP). A number of algorithms in the area of DCOP have been developed; including *asynchronous distributed optimization* (ADOPT) [7] and *distributed pseudotree optimization procedure* (DPOP) [9]. Both are guaranteed to converge to the optimal solution while using only localized communication and computation. However, they are not specifically tailored to the specific problem that we address here, and since these algorithms are complete, they require an exponential increase in the total message size being exchanged (unlike the case of our algorithm with fixed routing). This is unrealistic for WVSNs in which the nodes are typically installed with limited computational, storage, and memory resources.

6. CONCLUSIONS

In this paper, we have considered the problem of adaptive sampling, forwarding, and routing within WVSNs in order to manage the limited energy resources of nodes in an effective and efficient way. We have developed two novel optimal decentralised algorithms: one which assumes fixed routing and calculates the optimal sensing and forwarding actions that each node should perform, and one which assumes flexible routing, and makes optimal decisions regarding both the integration of actions that each node should choose, and also the route by which this data should be forwarded to the base station. In an empirical evaluation, we showed that the algorithm with flexible routing delivered approximately twice the quantity of information to the base station, but at considerably higher communication and computational cost. Thus, while the algorithm with flexible routing is suitable for networks with a small numbers of nodes, it scales poorly, and as the size of the network increases, the algorithm with fixed routing is favoured.

Our ongoing work in this area includes relaxing the restriction that the nodes may only forward data to nodes that are closer to the base station (in terms of hop count) than themselves and, in particular, we would like to characterise the circumstances in which this may yield some benefit. More significantly, we would also like to develop a principled algorithm for making the choice of route when applying the algorithm with fixed routing to loopy WVSNs (rather than having the nodes make an arbitrary choice of parent in order to convert the loopy network into a tree-structured network as we have done here). Our empirical results indicate that the performance of the algorithm with fixed routing is very close to that of the algorithm with flexible routing if the appropriate fixed route is selected (see Fig. 4)[6], and thus, there is great potential in doing so.

7. REFERENCES

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach. Distributed embedded smart cameras for surveillance applications. *Journal of IEEE Computer*, 39(2):68–75, 2006.

[2] J. Kho, A. Rogers, and N. R. Jennings. Decentralised control of adaptive sampling in wireless sensor networks. *ACM Transactions on Sensor Networks (In Press)*, 5(3), 2009.

[3] R. Kleihorst, B. Schueler, A. Danilin, and M. Heijligers. Smart camera mote with high performance vision system. In *Proceedings of the ACM SenSys Workshop on Distributed Smart Cameras*, October 2006.

[4] A. Ledeczi, A. Nadas, P. Volgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dora, K. Molnar, M. Maroti, and G. Simon. Countersniper system for urban warfare. *ACM Transactions on Sensor Networks*, 1(2):153–77, 2005.

[5] G. Mainland, D. C. Parkes, and M. Welsh. Decentralised, adaptive resource allocation for sensor networks. In *Proceedings of the 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation*, pages 315–28, 2005.

[6] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, pages 374–381, 2006.

[7] P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–80, January 2005.

[8] P. Padhy, R. K. Dash, K. Martinez, and N. R. Jennings. A utility-based sensing and communication model for a glacial sensor network. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*, pages 1353–1360, 2006.

[9] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 266–271, August 2005.

[10] F. Qureshi and D. Terzopoulos. Distributed coalition formation in visual sensor networks: A virtual vision approach. In *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems*, pages 1–20, 2007.

[11] B. Rinner, T. Winkler, W. Schriebl, M. Quaritscha, and W. Wolf. The evolution from single to pervasive smart cameras. In *Proceedings of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–10, September 2008.

[12] P. Sinha and A. A. Zoltners. The multiple-choice Knapsack problem. *Operations Research*, 27(3):503–15, May-June 1979.

[13] T. Zahariadis and S. Voliotis. Open issues in wireless visual sensor networking. In *Proceedings of the 6th EURASIP Conference Focused on Speech and Image Processing, Multimedia Communications and Services*, pages 335–338, June 2007.

[6]Note the algorithms are not necessarily identical in this case, since the algorithm with flexible routing allows individual nodes to forward data through multiple routes.