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

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

[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.

ATSN09 at AAMAS09 Organizers:... Date:...

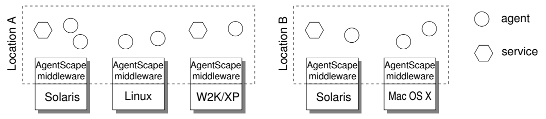

Figure 1: Conceptual model of AgentScape middleware

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

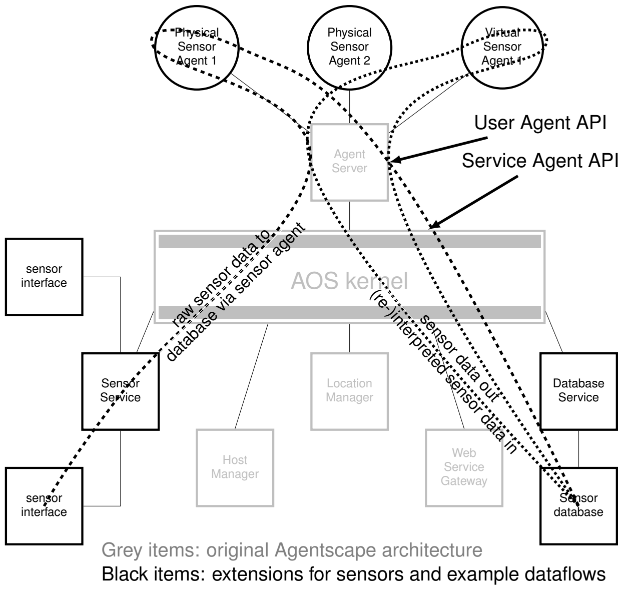

Figure 2: Agentscape sensor architecture

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.