Intelligent Decision-Making in the Physical Environment

David Lillis, Sean Russell, Dominic Carr,
Rem W. Collier, and Gregory M.P. O'Hare

CLARITY: Centre for Sensor Web Technologies,
School of Computer Science and Informatics,
University College Dublin, Ireland
{david.lillis,sean.russell,dominic.carr,rem.collier,gregory.ohare}@ucd.ie

Abstract. The issue of situating intelligent agents within an environment, either virtual or physical, is an important research question in the area of Multi Agent Systems. In addition, the deployment of agents within Wireless Sensor Networks has received some focus also.

This paper proposes an architecture to augment the reasoning capabilities of agents with an abstraction of a physical sensing environment over which it has control. This architecture combines the SIXTH sensor middleware platform with the ASTRA agent programming language, using CArtAgO as the intermediary abstraction.

1 Introduction

The environment in which an agent is situated is considered to be an essential component of Multi Agent Systems (MASs) by many researchers. An environment provides the surrounding conditions for agents to exist, in addition to providing an exploitable design abstraction upon which a MAS can be developed [1]. As this notion has gained traction, a number of solutions have been proposed to allow environment abstractions be provided for agents.

This paper considers how agents can be provided with an abstraction of a physical environment by leveraging physical sensors. This includes access to sensor data and information about the structure of the sensor network itself. This allows agents to make intelligent decisions about which sensors must be enabled or disabled according to application demands, which sensor data is required and at what frequency, and other decisions relevant to the application domain.

Previous examples of incorporating agents into sensor networks typically concentrates on in-network intelligence, where lightweight agents are deployed on computationally constrained sensor motes in order to perform such tasks as power management [2], adaptive application development [3] and optimisation [4]. This work rather focuses on agents as means not only to make intelligent decisions based on sensor data but also to manage the sensor network itself.

Section 2 outlines some motivations for undertaking the integration discussed in this paper. Sections 3 and 4 respectively discuss the technologies chosen in implementing a solution and the proposed architecture for realising this. Finally, Section 5 presents the conclusions and future work.

J.C. Augusto et al. (Eds.): AmI 2013, LNCS 8309, pp. 235–240, 2013.

2 Motivation

The integration of component frameworks and intelligent reasoning, realised through the use of MASs, has been investigated both as a means of controlling the framework itself and as a method of incorporating intelligent behaviours within an application itself [5]. The use of intelligent agents to maintain these aspects of an application built upon such a component system is desirable as it provides an accessible means for quickly altering the behaviour of the system.

Of particular interest is the integration of generic reasoning systems within Sensor Web middleware. This would provide a coupling between an environment, typified by heterogeneous sensor networks, and intelligent reasoning. One application within which this combination would prove useful is Waste Augmentation and Integrated Shipment Tracking (WAIST) [6], which aims to use a myriad of sensing devices to monitor and validate the legal transportation of waste materials by licensed hauliers. The SIXTH sensor middleware (discussed in Section 3) provides the means of collecting and routing the data from heterogeneous sensors such as GPS, acceleration, light and contact sensors, but does not provide an easy mechanism for the addition or modification of intelligent behaviours.

A typical usage of such a system would be to analyse incoming GPS data. A simple agent could generate an event whenever the speed of a truck falls below a certain threshold, thereby recording all stopping locations. This event could be visualised for the end-user, further analysed by another agent to isolate and remove short stoppages such as traffic lights etc. Other additions could include activating a number of high intensity sensors within a truck when it stops (allowing the conservation of energy when the truck is in motion), or generating events when a truck stops in particular locations such as known dumping sites.

3 Technologies

3.1 Sensor Middleware

SIXTH is a Java-based Sensor Web Middleware incorporating sensed data from diverse data sources both physical and cyber. Through the adaptor layer abstraction, the middleware can be connected with any data source that is accessible programmatically. Examples of physical data samples include those from Wireless Sensor Networks composed of SunSpots, Waspmotes, Shimmers and smartphones. In the case of cyber data, this has been collected from sources such as Twitter, Xively, Foursquare and Facebook.

In terms of physical sensors, an *adaptor* typically wraps a particular sensor network. For example, in the case of SunSPOT motes, an adaptor would run on a machine connected to the SunSPOT base station. This would be responsible for providing access to multiple *sensor nodes* (individual SunSPOT motes), each of which may contain numerous *sensors* (e.g. light sensor, accelerometer). SIXTH exposes its resources to applications through lightweight interfaces such as those responsible for the receipt of possibly filtered sensor data streams, the interface

for the interception of sensor and sensor node status alerts and one allowing the recipient to see (re)tasking conducted by other entities upon resources.

The (re)tasking of sensor nodes is conducted via the Tasking Service, which routes the request to the pertinent adaptor for execution. A request to change the behaviour of a sensor or the entire sensor node is encapsulated within a Tasking Message, for instance the application might wish to decrease the rate at which temperature is being sampled in response to a steady reading. A Tasking Message, if accepted as valid, is translated by an adaptor's Message Wrapper implementation, which performs transformation into a native messaging format for that platform and is then passed to the sensor node.

The architecture of SIXTH is discussed in greater detail in [7]. As SIXTH is an OSGi-based system implemented by means of various component bundles, the current research is partially motivated by previous work done in the area of the management of component-based systems using intelligent agents [5].

3.2 Environment Abstraction

In terms of work in the area of Agent Programming Languages (APLs), the two most well-known environment layer technologies are the Environment Interface Standard (EIS) [8] and the Common Artifacts for Agents Open framework (CArtAgO) [9]. Of these two standards, EIS follows the more traditional view of environments as it implements an interface that models the environment layer as a set of entities. The state of these entities is modelled as a set of beliefs (percepts) and they may be manipulated through a pre-defined set of actions. EIS promotes an opaque view of an environment where the implementation of the actual entities is hidden. Additionally, the set of percepts and actions generated by those entities is specified only in supporting documentation. CArtAgO operates in a similar way, by modelling the environment as a set of artifacts (equivalent to an entity in EIS) that can be manipulated by performing operations (equivalent to an action). Artifact state is modelled as a set of named observable properties that store Java objects. Changes in the observable properties result in the generation of custom events that are passed to the agent layer (e.g. property added, property updated, etc.). In contrast with EIS, CArtAgO is not opaque, and the developer is able to see the implementation of the artifacts and in fact contribute additional artifacts where appropriate.

From a deployment perspective, there are significant differences between the two approaches: in its current incarnation, EIS environments are loaded from the local file space using a dedicated (and inaccessible) Java ClassLoader. This makes EIS difficult to integrate into other technologies, such as OSGi that are also designed to manage ClassLoaders and which advocate that all resources for a deployment should be enclosed within one or more bundles. While it is possible to modify the EIS interface to be OSGi-sensitive, it would require the creation of a new version of EIS that may not be compatible with current EIS-enabled agent toolkits. Instead, the standard EIS approach in such scenarios is either to make the EIS environment into an ad-hoc remote client (e.g. [10]) or to completely embed the system within the EIS environment implementation.

CArtAgO does not suffer from this issue and can be easily integrated into OSGi as a dedicated bundle. A further benefit of CArtAgO is its built-in support for a distributed runtime. This means that any agent toolkit that supports CArtAgO can interact with artifacts deployed using OSGi even if the agents themselves cannot be deployed using OGSi.

3.3 Agent Programming Language

The agent-layer technology used in this paper is ASTRA: an implementation of AgentSpeak(L) [11] APL that is based on Jason operational semantics [12]. ASTRA was chosen because it represents a new breed of APL that provides minimal runtime mechanisms and as such can be easily integrated with other technologies, such as OSGi. ASTRA also has a number of other features, including static typing, support for multiple inheritance, and language level integration with CArtAgO and EIS.

4 Integration

Figure 1 shows a proposed architecture for the integration of ASTRA and SIXTH. For the purposes of clarity, only those elements of SIXTH that are relevant to this integration are shown. Agents interface directly with the CArtAgO layer, which provides access to the components and services provided by SIXTH.

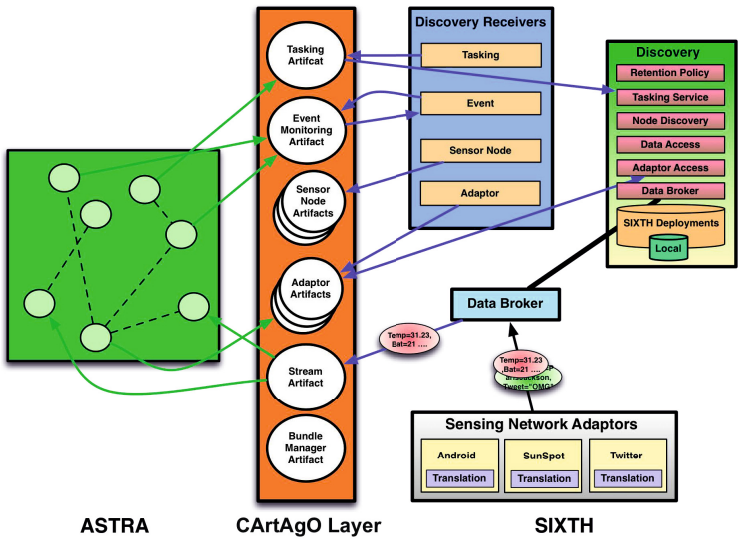

Fig. 1. Proposed integration architecture

The principal artifact types provided by the CArtAgO layer are as follows:

Tasking Artifact. Allows agents to reconfigure sensors by means of tasking messages (e.g. to adjust the sampling frequency). However, it is also possible that sensors can be tasked through other non-agent applications. Thus the Tasking Artifact will also notify interested agents whenever a sensor responds to any tasking message, so that the agents can maintain a correct model of the behaviour of the underlying sensor network.

Event Monitoring Artifact. Agents also require notification about changes in the middleware. For example, this may occur whenever a sensor node is added, removed, fails or becomes uncontactable.

Sensor Node Artifacts. Numerous sensor nodes will typically be attached to a running SIXTH instance. When a notification is received to inform the CArtAgO layer about the presence of a new sensor node, a Sensor Node Artifact is created to model that sensor node and the sensors it contains. The collection of Sensor Node Artifacts that is created serves as an abstraction of the sensor network itself that the agents can use in their process of reasoning about the network.

Adaptor Artifacts. In a similar way to the Sensor Node Artifacts, each Adaptor that is loaded into the SIXTH deployment is also modelled as an individual artifact.

Stream Artifact. Responsible from receiving sensor data from the SIXTH Data Broker and making it available to interested agents. This data is forwarded by the Data Broker whenever sensor data received from the sensors attached to the SIXTH deployment matches queries that the Data Broker has received.

Bundle Manager Artifact. As SIXTH is based on OSGi, its various services and subsystems are implemented by way of individual bundles that can be loaded, unloaded, started and suspended during runtime. This allows for dynamic configuration of the SIXTH system as a whole, to activate necessary services and deactivate those that are no longer required. The Bundle Manager Artifact allows the ASTRA agents to interact with the OSGi runtime to perform this bundle management.

Although this paper focuses primarily on physical sensors, it should be noted that the abstraction discussed above can also accommodate cyber sensors, which are treated in the same way as physical sensors under the SIXTH philosophy. These capture information from virtual sources such as web services.

5 Conclusions and Future Work

The architecture described here allows agents to firstly build an accurate model to represent the structure, capabilities and functioning of the underlying sensor middleware. In addition, it provides them with the ability to dynamically alter the behaviour of the sensors by sending tasking messages, change the structure of the middleware by loading and unloading OSGi bundles as appropriate to the requirements of the agent application. The nature of CArtAgO is such that this implementation is not restricted to the use of intelligent agents written in the

ASTRA APL. The abstraction provided by CArtAgO can be leveraged by any APL or agent framework into which CArtAgO support has been integrated.

A prototype of the architecture shown in Figure 1 has been developed. The next stage of this research will involve a full evaluation to ascertain the usability and effectiveness of this integration.

Acknowledgements. This work is supported by Science Foundation Ireland under grant 07/CE/I1147 and by the Irish Environmental Protection Agency (EPA) (Grant No. 2008-WrM-Ms-1-s).

References

1. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-agent Systems 14, 5–30 (2007)
2. Tynan, R., Muldoon, C., O'Hare, G.M.P., O'Grady, M.J.: Coordinated intelligent power management and the heterogeneous sensing coverage problem. The Computer Journal 54(3), 490–502 (2011)
3. Fok, C.L., Roman, G.C., Lu, C.: Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Transactions on Autonomous and Adaptove Systems (TAAS) 4(3), Article 16 (2009)
4. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 639–646 (2008)
5. Lillis, D., Collier, R.W., Dragone, M., O'Hare, G.M.P.: An Agent-Based Approach to Component Management. In: Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2009), Budapest, Hungary (May 2009)
6. Russell, S., O'Grady, M.J., O'Hare, G.M.P., Diamond, D.: Monitoring and Validating the Transport of Waste. IEEE Pervasive Computing 12(1), 42–43 (2013)
7. O'Hare, G.M.P., Muldoon, C., O'Grady, M.J., Collier, R.W., Murdoch, O., Carr, D.: Sensor Web Interaction. International Journal on Artificial Intelligence Tools 21(02), 1240006 (April 2012)
8. Behrens, T., Hindriks, K.V., Bordini, R.H., Braubach, L., Dastani, M., Dix, J., Hübner, J.F., Pokahr, A.: An Interface for Agent-Environment Interaction. In: Collier, R., Dix, J., Novák, P. (eds.) ProMAS 2010. LNCS, vol. 6599, pp. 139–158. Springer, Heidelberg (2012)
9. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: A Framework for Prototyping Artifact-Based Environments in MAS. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2006. LNCS (LNAI), vol. 4389, pp. 67–86. Springer, Heidelberg (2007)
10. Behrens, T., Dastani, M., Dix, J., Hübner, J.F., Köster, M., Novák, P., Schlesinger, F.: The Multi-Agent Programming Contest. AI Magazine 33(4), 111 (2012)
11. Rao, A.S.: AgentSpeak (L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
12. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Wiley-Interscience (2007)