Sensor Fusion-Based Middleware for Smart Homes

Lorcan Coyle, Steve Neely, Graeme Stevenson, Mark Sullivan, Simon Dobson, and Paddy Nixon

Systems Research Group
School of Computer Science & Informatics
UCD Dublin, Ireland
(Tel : +353 1 716 5351 ; Fax : +353 1 269 7262 ; E-mail : lorcan.coyle@ucd.ie)

Abstract— Smart homes are sensor-rich environments that contain dynamic sets of interacting components. These components often use competing and closed standards and form a message-based architecture. This complicates the development of applications that require information from disparate sources. It becomes difficult to add new components or to allow components from different applications to interact with each another. In this paper we describe Construct, a pervasive computing middleware that is ideally suited for deployment in the smart home. Construct acts as a sensor fusion layer that takes output from each smart home component and makes it available to all applications. This makes it easy to develop applications that require access to heterogeneous sources of sensor data, and to add sensors to existing systems to improve their performance. This paper demonstrates two Construct-enabled smart home applications and shows how access to new sensors leads to improvements in their performance.

1. INTRODUCTION

Home automation and assistive living are perhaps the most well-known examples of context-aware pervasive systems. People expect their living spaces to be comfortable, and place strong requirements on the usability, stability and longevity of any technologies deployed there. For a home to be considered truly "smart" it must provide support for its inhabitants in a way that fits seamlessly into their daily lives. It must respond correctly to the events happening in the home, and adapt as these events (and the tasks they relate to) change and evolve over time. It must allow the various stakeholders – inhabitants, carers and others – to exert appropriate degrees of control over its behaviour, and reconcile any differences. It must allow new devices and services to be deployed (and removed) piecemeal without requiring significant physical, administrative or cognitive effort, ideally without forcing the inhabitants into unwanted vendor or technology lock-in.

Whilst a number of domestic automation systems are available commercially, their use of competing (and often closed) standards and protocols can impede the creation of smart homes in which large numbers of heterogeneous devices can inter-communicate easily. It is vital that developers can abstract away from the detailed topography, protocols, data formats and control of sensors, actuators and other information devices, and instead focus on the processing of information from disparate sources. This provides great architectural flexibility and has the potential to improve responses to the noisy and uncertain information typically encountered in sensor-rich environments. A common approach to addressing such issues is through the use of middleware.

In this paper we introduce a sensor-fusion-based middleware for smart homes. The system – Construct – treats all devices as either sensors or actuators, allowing arbitrary home automation equipment to be controlled. All data are represented uniformly, and their level of abstraction is raised through the use of knowledge-based data-fusion techniques [6]. This aids developers to build smart home applications, providing them with information that is semantically closer to their needs than the raw data provided by individual sensors. Furthermore, Construct makes it easy to integrate sensors that gather data from other sources of interest, such as the web. This provides developers with a rich

body of information with which to better automate the home. The system is fully decentralised and decoupled from individual information sources, allowing robust use of an evolving device population.

Construct collects the inputs from a diverse set of virtual and physical sensors and fuses them into a distributed data store. Applications query this data store directly rather than maintaining point-to-point connections with sensors. This decoupling of consumers of data from producers enables application developers to work with a single data format rather than requiring them to master a number of proprietary formats.

The remainder of this paper is organised as follows. Section 2 describes the current state of the art in middleware for smart homes, Section 3 describes the Construct middleware infrastructure, and Section 4 describes how sensors interact with Construct. Section 5 describes two Construct-enabled smart home applications, and identifies how they can be improved with access to new sensors. Finally, in Section 6 we conclude the paper, outline other smart home applications under development, and describe how the Construct middleware and further documentation can be freely obtained.

2. Middleware in Smart Homes

As smart homes are populated with increasing numbers of interacting components, the complexity of communications and data management rapidly becomes unmanageable without support from a mediating infrastructure. In traditional distributed systems, complexity is managed by building abstraction layers with common services made available to developers. Such abstractions are often termed *middleware*. Middleware can broadly be defined as a layer between application and system software, supporting integration between various products and platforms, whilst maintaining the integrity of the overall solution in terms of robustness and reliability.

From a developer's perspective, middleware can be used to provide services for configuring new sensors or devices. When a new device enters a smart home it needs to negotiate its integration with the environment. Fundamental questions need to be answered: where does it sit in the network: is it a peer, a master, or a slave node? Does it need to be told what to do by some other sensor? What languages does it speak? How fast can it emit and/or absorb data? What functions does it perform?

Decoupling of application and system through middleware provides a number of distinct advantages: indirection affords change of the underlying system, languages and protocols; components can reuse services; introduction of new devices, modules and other systems can be transparent; and a uniform view of the world simplifies the development process. Dealing with common tasks in the middleware layer removes the burden from the programmer.

Development of middleware for general distributed systems has evolved around a number of distinct paradigms including: object-based, tuple-space, message- and event-oriented, and peer-to-peer (P2P). Object-based technologies such as CORBA, COM+ and Java EJB provide a platform on which to build loosely-coupled distributed object systems, complete with operations for registering objects, discovering new services, transaction handling, security and facilitating object message passing. Message- oriented middleware (MOM) decouples client-server communications by encapsulating the interactions between entities in small asynchronous message passing. These forms of event-style systems are popular in financial services and other domains where reliability is paramount.

More recently, P2P systems have emerged as a substraight for building distributed systems upon. P2P systems reduce the amount of configuration required by implementing algorithms that support the dynamic creation of self-organising overlay mesh structures in the network. Look-up algorithms have been shown to scale to a very large collection of nodes [11] and P2P systems can potentially support wireless (and other) ad-hoc networks extremely well, whilst distributing the load and cost of service provision over the node population. This comes at the cost of a more complex and computationally expensive resource location with no guarantees that particular services be, or remain, available. P2P systems remove the need for a controlling node that can become a central point of failure, which improves fault-tolerance.

Technologies and standards focussed specifically on building automation tend to take a more device-centric and protocol-based view of the domain. Protocols such as X10 can be employed to control

devices in active environments. X10-enabled devices generally operate at a very simple level, accepting commands such as on/off or intensity control. LonWorks and BACnet are examples of systems designed to enable the building of control networks. BACnet was specifically designed in an attempt to create a standardised model for representing devices and interactions between them. More recently the oBIX (Open Building Information Xchange) standard has emerged, defining standard XML and Web Services to facilitate exchange of information and services between intelligent buildings and applications.

ProSyst have developed a platform called mBedded Server, compliant with the OSGi dynamic services platform that is fully capable of interconnecting and controlling smart devices and enabling remote maintenance. A common abstraction introduces a unified interface to manage different types of controllable modules e.g., UNPnP, LonWorks and X10. Using this, it is possible to automate and execute operations across a variety of controllable modules.

Despite these efforts, there are an increasing number of sensor systems and modules becoming available that adhere to different (or indeed no) standards but which provide essential information for many applications. Some sensor vendors have focused on IP-enabled platforms using WiFi or Bluetooth for communications. Other systems provide proprietary, often research-based interfaces: Smart-ITs [2], Wavenis [1] and i-Bean [10].

In summary, an analysis of traditional middleware approaches shows that they make assumptions about processing and communications capabilities that cannot be made in smart home environments. The average home user does not want to know or care about how to install their new telephone – it should just work. The dynamic nature of these environments must be handled automatically. Sensor systems do not provide an attractive programming platform for complex interactions. It is difficult to build decentralised and robust applications in a way that does not require significant customisation.

As the complexity and size of smart environments increases user interaction and management requests must be minimised. The requirement for self-management, self-description, self-configuration, self-optimisation and other so called self-* properties points towards the need for more autonomic approaches to middleware targeting smart homes. In the following section we introduce our work on Construct, a knowledge-based middleware, which addresses these requirements.

3. CONSTRUCT

We have designed the Construct infrastructure, to support the development of distributed knowledge-driven pervasive systems. Such systems interact through manipulation of a common data model rather than through the piecing together of services. Sensor fusion techniques are used to integrate noisy data sources in a dynamic and semantically well-founded manner to provide applications with a uniform view of information regardless of how it is derived. Based on the principal of decentralisation, a Construct network is founded upon a set of federated peers, called nodes. Each node makes available services to the providers and consumers of data (collectively known as *entities*) within the environment. Construct provides five core services: *Discovery, Management*, *Sensing, Actuation*, and *Distribution*.

The Sensing service accepts new data from entities and passes it to the Management service. The Management service is then responsible for validating this data before access to it is available though the Actuation service. Data are communicated between nodes using the Distribution service. The result of these interactions is that entities are always working with a local view of the global data set. This is illustrated in Figure 1.

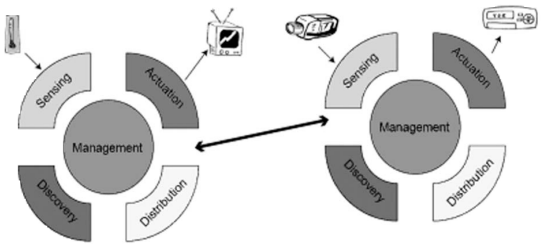

Fig. 1 A representation of our knowledge-driven architecture. Nodes automatically discover and connect to exchange data. Entities join the network and access data through services offered.

In order to contribute data, sensors must first discover a proximate Construct node. Once a sensor has established contact, it opens a connection to that node and sends it data in Resource Descritpion Framework (RDF) form. The node ensures that these data are well formed, and checks to see if they refer to recognised ontologies.

As nodes come online, they automatically discover other Construct nodes using an implementation of the zero-conf protocol [8]. Zero-conf is also used by

entities to locate instances of Construct. The result of this is a reference to the Discovery service, which supplies manifests for each of the local services that support entity interaction.

One of the core features of a knowledge-driven system is a shared model for representing and understanding data. We borrow techniques from the *semantic web* community by requiring that all data be marked up using ontologies. Reasoning over the global data model is supported through mappings between ontological descriptors. If application developers use a shared library of smart home ontologies, interoperability at the data-level between sensors and applications is transparent. New ontologies may be used if they exercise mappings to these ontologies [6].

All data in Construct are modelled using the RDF. RDF is an open-standard that meets our requirement of providing a common language with which to represent information generated within a smart home. Construct's Manager service uses the Jena framework [9] – a well established tool from the semantic web community that provides a rich interface for manipulating RDF data. The Manager service maintains two models that contain entity supplied data and its associated metadata. The Sensor service accepts RDF documents from data providers, while the Actuator service provides support for querying data.

Distributed communication requires trade-offs to be made between reliability, scalability, timeliness, resource usage and complexity. Within the Distribution service we manage the movement of data between devices using a probabilistic routing protocol called *gossiping* [7]. Data within pervasive systems are often frequently repeated (e.g., when the location of a person is updated or the reading from a temperature sensor is refreshed). The implication of this is that we may *not* always need to guarantee complete reliability of message delivery to all nodes: state information is likely to be reaffirmed periodically. These relaxed reliability constraints allow us to use gossiping as a mechanism for scalable, resilient communication. We prevent data saturation by associating each datum with expiration metadata. This value does not indicate that the data are no longer valid, merely that they should no longer be actively gossiped. Providing efficient access to historical data is one area we have identified for future work.

The core gossiping protocol is stateless and is initiated periodically at each node by contacting a randomly selected node and exchanging messages that are missing from each other's message buffer. Although smart homes may have a relatively fixed infrastructure in place that is capable of supporting centralised data storage, a single point of failure is undesirable. We are working towards developing intelligent gossiping algorithms to minimise data distribution delay whilst retaining the benefits of decentralised control.

The loose coupling between all components of the system affords robustness through evolution. Automatic discovery supports the addition of entities and nodes. The removal of a data consumer has no impact on the system in general. The removal of a data provider will impact anything that uses its data unless an ontologically equivalent source is present. The effect on an entity of removing a node may be mitigated by the automatic discovery and reconnection to another node.

All pervasive systems face issues with trust, which are perhaps particularly acute for smart home scenarios. These issues do not revolve around the use of cryptography or connection security, for which adequate solutions exist; rather, they concern the degree to which potentially sensitive information can be shared within a dynamic population of devices of unknown provenance. In some senses, trust and security are the Achilles' heel of pervasive systems: if a device can be "installed" simply by bringing it into the home, little can prevent a malicious visitor snooping – although it would at least require physical presence to accomplish.

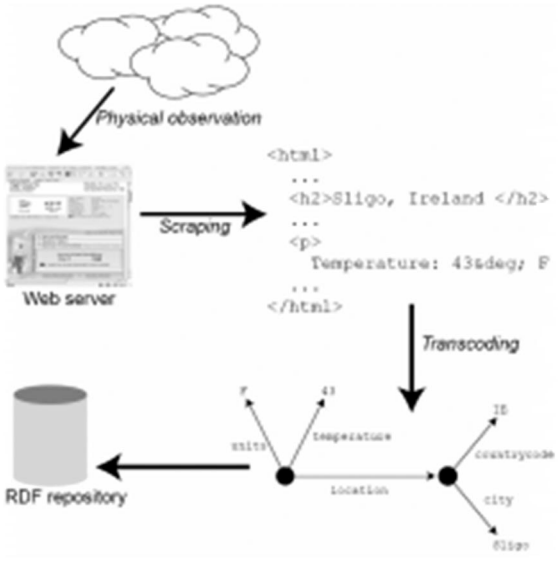

Fig. 2 Web scraping using a weather sensor

4. Construct Sensors

Construct is designed to support an extensible repository of sensors and their associated ontologies. We have categorised our sensors into two types: *physical* and *virtual*. Physical sensors directly detect characteristics of the environment e.g., sensors for location data obtained from Ubisense, Place Lab, Bluetooth spotters, RFID, and GPS.

Virtual sensors obtain information from the Internet, local network or local computer. The use of virtual sensors broadens the scale of context that is available to Construct applications. We have developed a number of sensors for obtaining music listening preferences (by parsing iTunes music databases), news (from online RSS feeds), concert timetables, concert tickets, TV listings, and stock quote information (from web pages). We have also developed a virtual weather sensor to illustrate the ease with which Internet data can be incorporated into a smart home application. This sensor uses web-scraping techniques [4] to read weather data that is published online (shown in Figure 2). The advantage of using virtual sensors rather than physical sensors in this case is that the task of detecting weather and predicting forecasts is left to the experts and there is no need to introduce actual physical sensors into the home. One disadvantage of this technique is that sensors are dependant on a connection to the external source and web-scraping itself is dependant on the source retaining its publishing format. The web-based virtual sensors described here use simple web-scraping but it would equally be possible to take advantage of other techniques such as web-services where these are available.

A number of entities have been written in different languages including Python, Ruby, Java, and C#. Applications that operate over the data from these sources employ a variety of output devices such as web pages, wall-mounted displays and Nabaztags.

5. Smart Home Scenarios

Construct applications can be improved by providing them with access to a wider range of data through the addition of new sensors. To demonstrate the benefits of fusing sensor data in this manner, this section outlines two application scenarios - an activity alarm for in-home assistive living, and a smart heating application that estimates the savings that could be accrued through use of automated storage heaters.

5.1. Activity Alarms for Assistive Living

With the demographic shift towards aging populations in many societies, increasing amounts of research in computer science and engineering is being focused on developing assistive living applications [3]. In many cases, elderly or infirm people are well enough to live in their own homes, but are worried about their ability to contact their family, doctor, or carers in case of an emergency. Panic buttons are one common solution to this problem. However, the assumption is that the householder is always able to use the button in an emergency situation, which may not be the case. It is possible to augment this system by installing activity sensors in devices that the householder commonly uses, such as interior doors, kitchen appliances, the television remote control, or toilet flush. If the householder does not use any of these devices within a certain time period, a call is made to their phone to confirm that they are safe and well. If they do not respond, a further course of action is taken, such as asking their neighbours to check in on them. When the householder is not at home, we can expect the sensors of the house to be quiet.

While this application is not fool-proof, it has the advantage that its sensed inputs are well correlated with healthy behaviour. Another advantage is that alternative sensors can be added depending on the householder's habits. Interactions with other household devices may be more predictive of healthy activity for different users, and it might be useful to put additional activity sensors in those devices, e.g., in radios, house lights, doors, or floors.

Consider the addition to this scenario of a second television set in another room. This new device will influence the behaviour of the householder. It should automatically become a new physical activity sensor providing data for the alarm system. As Construct decouples sources of data from its consumers, it is trivial to incorporate new sensors in this manner. The discovery, integration and sensing all occur seamlessly once the devices are within communication range of each other (e.g., when the TV is plugged into the electrical system). Applications automatically take advantage of the new data without modification.

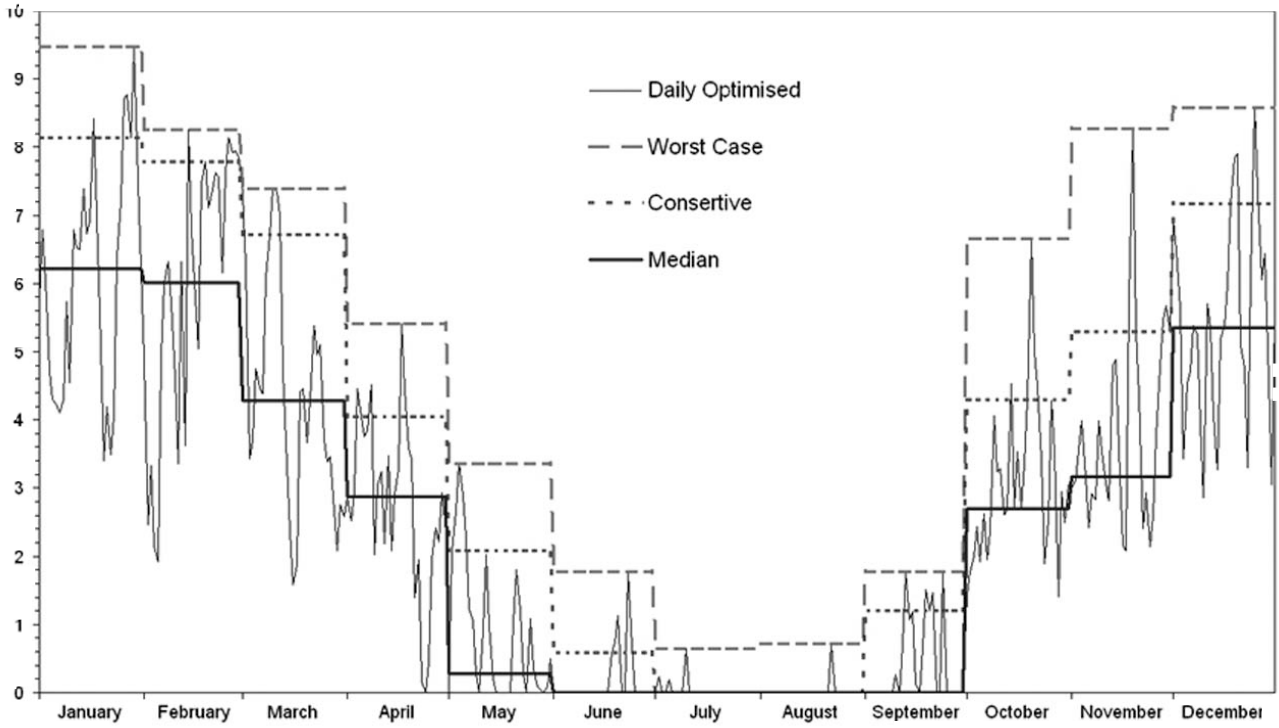

Fig. 3 Number of Hours Spent Heat (Overnight) During a Year.

5.2 Smart Heating Application

The second demonstrator application is an intelligent home heating system that uses storage heaters. Storage heaters use cheap electricity (usually at night-time) to store heat in ceramic bricks and release it during the day when electricity is expensive. This approach is most economical if enough heat is stored each night to heat the house during the following day. If too little heat is stored overnight, a thermostat ensures that the correct temperature is maintained by turning secondary heaters on. However, this necessitates the consumption of more expensive electricity. Typically, the user controls the heating system manually by increasing the amount of heat stored in cold weather and decreasing it in warmer weather.

We attempt to improve this heating application using web-published weather forecasts (using the weather sensor described in Section 4) to automatically regulate the amount of heat stored each night. The application uses the forecast of the following day's temperature to predict how much heat must be stored to keep the home above a desired temperature throughout the following day. These predictions are then used to adjust the thermostat to reflect the expected temperature, resulting in the storage of an appropriate amount of heat. Apart from being more economical for the home owner, the application has the benefit of reducing the cognitive load on the user, in that they no longer have to manually set the heating system.

An evaluation was performed to investigate the potential savings available to a home owner. A model of a home heating system was constructed using a simplified version of storage heating models and controls developed by Wright [12]. This model was used to determine the quantity of heat that must be stored to keep the house above a certain temperature during the following day. A record of daily observed minimum and maximum temperatures in Armagh, UK for 2004 was used to generate a realistic temperature dataset for the evaluation [5].

To simplify the experiment we made the following two assumptions:

- advance knowledge of the temperature for each day is available,
- users will only manually set the amount of heat to be stored each night at the beginning of the month.

To test the economic value of our system, we

developed three strategies that aim to approximate potential user behaviour:

Worstcase : assumes that the user programs the heater at the start of the month to store enough heat every night for the coldest day of that month. In this way, expensive day-time electricity would never be required. This strategy is used as a *straw-man* for the purposes of the evaluation because it is quite inefficient.

Conservative : ensures that the heater stores enough heat each night to protect against the coldest 10% of the days of the following month.

Median : stores enough heat for the median coldest day of each month.

Our application uses a strategy (called *daily-optimised*) that instructs the heater to store just enough heat to remaining above the minimum temperature for the following day. Figure 3 compares the amount of heat that is stored each night of the year using these four strategies. The x axis shows the months of the year, and the y axis shows the number of hours that the heater is turned on for each night. As expected, the daily-optimised strategy varies in the amount of heat stored each night, while the other strategies plateau for each month. It is this variability that leads to cost savings. Comparison with the worst-cast strategy suggests that the conservative strategy stores 24% less heat, the median strategy stores 51% less heat and the daily optimised strategy stores 47% less heat. It should be noted that although the median strategy stores less heat at night, the house requires additional daytime heating half the days of every month. This analysis suggests that autonomous daily correction leads to a more economic heating strategy while eliminating the need for constant calibration by the home owner.

While an approach based on our daily-optimised strategy should ensure that the house temperature does not drop below comfort levels, further improvements could be made with access to other types of data. By taking into account occupancy it should be possible to provide a more efficient solution. If the home is expected to be unoccupied, the owner may be solely concerned about keeping the temperature above freezing (to prevent water pipes from bursting). Using a virtual sensor for calendar information, it would be possible to incorporate occupancy data and alter the strategy to take into account periods where the house will be unoccupied.

6. Conclusions and Ongoing Work

Device management protocols satisfy their role in the control network space yet do not provide the high level abstractions or services that ease development. We have described the case for middleware in smart homes to provide interoperability, and ease of integration of new components, sensors, and applications. Existing middleware solutions are often heavyweight and provide the wrong abstractions for architecting dynamic environments.

To address these issues we have developed Construct – a middleware framework for smart homes. Construct supports application development through manipulation of a common data model, using sensor fusion techniques to limit the effect of noisy data. The infrastructure is fully decentralised to provide scalability, and zero-conf techniques are used to support automatic configuration of devices in a smart home.

We described two scenarios that demonstrate the ease of integrating new components into existing applications, and demonstrated, with a simple evaluation, the potential benefits of fusing a simple virtual sensor into a home heating application. These benefits are two-fold: the user no longer has to set their thermostats manually, thus reducing cognitive load; and the improved system is more economical, as access to contextual information leads to more efficient heating strategies.

As well as the application scenarios described in this paper, we are working on a number of other applications, including a music recommender that makes recommendations collectively to the current occupants of a room; a memory aid for Alzheimer's sufferers that attempts to assist them in the completion of complex but routine tasks; and a smart meeting space that recognises the occurrence of a meeting and attempts to retrieve the minutes and other relevant information from the last related meeting.

The first beta-version of Construct was released under the Limited GNU Public License (LGPL) in late 2006. There is a website[i], a public wiki for developers[ii], and a public mailing list[iii]. Several sensors, demonstrator applications and ontologies describing context data were made available as part of this release.

ACKNOWLEDGEMENTS

This work is partially supported by Science Foundation Ireland under grant numbers 04/RPI/1544, "Secure and Predictable Pervasive Computing" and 03/CE2/I303-1, "LERO: the Irish Software Engineering Research Centre," and by Enterprise Ireland under grant number CFTD 2005 INF 217a, "Platform for User-Centred Design and Evaluation of Context-Aware Services."

REFERENCES

[1] Coronis Systems, Wavenis technology http://www.coronis-systems.com/descriptif.php?id = descr_tech

[2] The Smart-Its project. http://www.smart-its.org/

[3] J.C. Augusto and C.D. Nugent, editors. Designing Smart Homes, The Role of Artificial Intelligence, volume 4008 of LNCS. Springer, 2006.

[4] C. Ball. Screen-scraping with www::mechanize, 2003. Available online at http://www.perl.com/pub/a/2003/01/22/mechanize.html

[5] C. Butler, A. García-Suárez, A. Coughlin, and D. Cardwell. Meteorological Data Recorded at Armagh Observatory : Vol 2 - Daily, Mean Monthly, Seasonal and Annual, Maximum and Minimum Temperatures, 1844–2004. Armagh Observatory Climate Series, 2004.

[6] A.K. Clear, S. Knox, J. Ye, L. Coyle, S. Dobson, and P. Nixon. Integrating Multiple Contexts and Ontologies in a Pervasive Computing Framework. Proc Of Contexts and Ontologies: Theory, Practice and Applications, pages 20–25, Riva Del Garda, Italy, 2006.

[7] P.T. Eugster, R. Guerraoui, A.M. Kermarrec, and L. Massoulieacute. Epidemic Information Dissemination in Distributed Systems. Computer, 37(5):60–67, 2004.

[8] E. Guttman. Autoconfiguration for IP Networking: Enabling Local Communication. IEEE Internet Computing, 5(3):81–86, 2001.

[9] B. McBride. Jena : Implementing the RDF Model and Syntax Specification. Proc Of the 2nd International Workshop on the Semantic Web, Hong Kong, 2001.

[10] S. Rhee, D. Seetharam, S. Liu, N. Wang, and J. Xiao. i-Beans: An Ultra-low Power Wireless Sensor Network. In Interactive Poster in the 5th International Conference on Ubiquitous Computing, 2003.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. LNCS, 2218:329–350, 2001.

[12] A. J. Wright. Electric Storage Heaters in Buildings Simulation. In IBPSA, Building Simulation, pages 38–40, August 1997.

---

[i] The Construct home page can be found at http://construct-infrastructure.org/

[ii] The Construct public Wiki can be found at http://construct-infrastructure.org/wiki/

[iii] The Construct announcements mailing list can be found at http://construct-infrastructure.org/construct-announcements