JOURNAL OF CIVIL ENGINEERING AND MANAGEMENT
ISSN 1392-3730/eISSN 1822-3605
2015 Volume 21(3): 356–375
doi:10.3846/13923730.2014.890645

AN INTEGRATED FRAMEWORK UTILISING SOFTWARE AGENT REASONING AND ONTOLOGY MODELS FOR SENSOR BASED BUILDING MONITORING

Michael DIBLEY, Haijiang LI, Yacine REZGUI, John MILES

BRE Institute of Sustainable Engineering, Cardiff University, UK

Received 02 May 2012; accepted 21 Dec 2012

Abstract. Smart building monitoring demands a new software infrastructure that can elaborate building domain knowledge in order to provide advanced and intelligent functionalities. Conventional facility management (FM) software tools lack semantically rich components, and that limits the capability of supporting software for automatic information sharing, resource negotiation and to assist in timely decision making. Recent hardware innovation on compact ZigBee sensor devices, software developments on ontology and intelligent software agent paradigms provide a good opportunity to develop tools that can further improve current FM practices. This paper introduces an integrated framework which includes a ZigBee based sensor network and underlying multi-agent software (MAS) components. Several different types of sensors were integrated with the ZigBee host devices to produce compact multi-functional sensor units. The MAS framework incorporates the belief-desire-intention (BDI) abstraction with ontology support (provided via explicit knowledge bases). The different software agent types have been developed to work with sensor hardware to conduct resource negotiation, to optimize battery utilization, to monitor building space in a non-intrusive way and to reason about its usage through real time ontology model queries. The deployed sensor network shows promising intelligent characteristics, and it has been applied in several on-going research projects as an underlying decision making service. More applications and larger deployments have been planned for future work.

Keywords: multi-agent system (MAS), belief-desire-intention (BDI), ontology, building management system (BMS), domain knowledge, ZigBee sensor unit.

Introduction

Contemporary buildings are becoming more complex with the increasing demands for integrating more functionality into buildings. Alongside the normal building services such as water, electricity, safety, security and heating, ventilation, and air conditioning (HVAC), etc. systems, functionality such as building space usage monitoring becomes one of the core concerns (motivated by low carbon footprint and other requirements) for modern building management. Because of the involvement of human behaviours, the non-intrusive sensor based monitoring approach becomes particularly useful and more amenable than the use of video cameras. Problems arise in the process of sensor data collection and transmission, human and building interactivity, relevant factors' sensitivity analysis for decision making, and so on. Smart monitoring of buildings demands a new software infrastructure that can elaborate building domain knowledge in order to provide advanced and intelligent functionalities. The conventional facility management (FM) software tools lack such semantically rich components, and that limits the capability of software for supporting automatic information sharing, resources negotiation and in the timely assistance of decision making.

Ontology provides a way to convert human knowledge to explicit and hence computer understandable domain knowledge. With the appropriate implementation scheme, some of the low level FM tasks can thus be conducted automatically by software components with certain level of intelligence by elaborating those explicit domain knowledge models (Li *et al.* 2010; Rezgui, Miles 2011; Dibley *et al.* 2012). In the context of AEC/FM domain, at present there are a large amount of taxonomy resources, so there is a need for the purpose of practical use to go through the plethora of existing taxonomies and ontological resources and constructs to find a feasible development route that can take full use of the existing resources while remaining simple and straight forward. Software agents constituting a multi-agent system (MAS) relate to that in the sense that they can utilise knowledge bases (KBs) (encompassing domain and some general knowledge in a formal representation, and inference "machinery") in order to render themselves intelligent. Specifically agents can use the services of KBs in order to direct their behaviour for the purpose

Corresponding author: Haijiang Li
E-mail: *lih@Cardiff.ac.uk*

of pursuing their goals. The knowledge requirement of the KBs' content to support specialised behaviour is thus specific to agent types. A multi-agent framework including different types of agents (supported by ontology), targeting specific functionalities, is therefore needed to provide the expected better FM solutions.

This paper introduces an integrated framework (named OntoFM) that includes a ZigBee sensor network and some underlying multi-agent software components. The intended research target is to show that the application of software agency based on the belief-desire-intention (BDI) (Rao, Georgeff 1995) formalism, supported with semantic knowledge bases that are synchronised in near real time to the environment, delivers several benefits in the realisation of a software system to support facility management. The rest of the paper is organized as follows. First, relevant theory and development resources have been reviewed. This is followed by the integrated system design and implementation methodology, where a multi-layered systematic architecture is explained. The detailed implementation process is introduced next, including ZigBee sensor unit assembly, ontology development, multi-agent framework implementation and system integration. The next section outlines the system deployment and evaluation. Due to the lack of the standard evaluation process for such a FM oriented integrated system development, a use case and scenario based system evaluation procedure has been devised. The testing process has been conducted repeatedly, and the preliminary test results were used for next stage improvements to further refine the hardware design and software developments. Finally, the conclusion and discussion are given.

1. Ontological development resources and MAS applications

The relevant ontological development resources and MAS applications in AEC/FM domain are briefly reviewed first in this section, followed by a description of MAS development methodologies that inspired the development work. Finally applications of MAS in the ACE/FM domain are outlined.

1.1. Ontological resources for the FM domain

A wide range of resources containing high implicit or explicit semantic content exists that is applicable to the OntoFM. Resources include descriptions of high level abstract (common sense) concepts, and domain descriptions in: engineering, mathematical, physical contexts, as well as product and simple process models. The resources vary in type, amount of detail and level of abstraction, and by virtue of the language used, vary in expressiveness (and "ontological precision" (Guarino 2006)) and consequent succinctness. The ontological precision increases progressively from knowledge captured in a systematic list, taxonomy, object oriented design to axiomatic theory (Guarino 2006). Commonly, the level of abstraction with respect to dependence on purpose and domain can be aligned with one of three layers from: so called upper level, domain or application. The upper layer captures the most general and reusable terms, including common sense concepts. The lower levels specialise the concepts above. Layering facilitates interoperability by ensuring consistency between domains. There are several well-known upper ontologies and the Suggested Upper Merged Ontology – SUMO (Pease 2008) is one example. SUMO describes fundamental concepts in first order logic, is highly axiomised and includes 1000 terms, 4000 axioms, 750 rules. It is an IEEE initiative, an open standard and is mapped to the WordNet lexicon (Crawley *et al.* 2008). A high level distinction in SUMO entities object and process derived from physical entity can be aligned with the "endurants and occurrents" classifications by Fielding *et al.* (2004). Endurants and occurrents refer to an entity's existence relationship with time and never form part of relationships with each other. Fielding *et al.* (2004) also define top level classes that capture dependency and scope.

At the same upper layer of abstraction as SUMO are the Top Level Ontologies of Universals and Particulars, developed by Guarino and Welty (Gómez-Pérez *et al.* 2004). The Universals Ontology has been derived from the philosophical considerations: rigidity, identity and dependency – meta properties used in Ontoclean (Guarino, Welty 2009). The formulation of the Individuals Ontology is structured on the base concepts of abstract, concrete and relation (Gómez-Pérez *et al.* 2004). The PhysSys (Borst 1997) ontology set defines abstract reusable ontologies for: mereology, topology, systems theory, component, physical and process. It utilises the EngMath ontology (Gruber, Olsen 1994) for mathematical KR, holistically realising "… three conceptual viewpoints: technical components, physical processes and mathematical relations" (Borst 1997). Specifically EngMath facilitates ontological mathematical modelling in engineering using Ontolingua, providing "... conceptual foundations for scalar, vector and tensor quantities as well as functions of quantities, and units of measure" (Gómez-Pérez *et al.* 2004). Based on aspects of SensorML, OntoSensor (Russomanno *et al.* 2005) is an OWL-DL ontology that includes a few concept-to-concept links to SUMO. It was developed for the purpose of data fusion and the modelling effort focuses on the sensor data rather than the associated processes (Preece *et al.* 2007).

1.2. MAS development methodologies

Several conventional Multi-Agent System (MAS) development methodologies have been presented over recent years showing various characteristics including their lifecycle coverage, level of guidance detail, provision of guidelines and heuristics, pattern provision, resemblance to conventional software engineering methodologies, availability of supporting tool and any provision or re-use of existing notation. The nature of development methodologies for formal (in a mathematical sense)

MASs where the system specification captured in a logical representation can often be directly executed, is different to that for conventional systems. In such systems agents are typically theorem provers, where goals and beliefs etc. are derived from the logical representation of the specification. Little or no refinement, as seen in the analysis and design phases in traditional software engineering is therefore needed.

Methodologies for MAS take primarily one of two forms, either adapting objected oriented or knowledge engineering methodologies. Each has its advantages. Using object oriented methodologies as a basis has the advantage of familiarity for programmers and the potential to reuse a range of notations (with modified semantics where necessary) and tools. Although there are major distinctions between agents and objects, some commonalities can be drawn. Agents can be regarded as objects that are loosely coupled and "active", that communicate asynchronously using a high level language. The challenge in adapting existing object methodologies is to conceptually model the autonomous/non-passive nature of agents. Interactions and collaboration should also be addressed (Iglesias *et al.* 1999). Like objects, agents have a stable identity and are cohesive, but their environment and collaboration involvement (even in closed systems) is dynamic, which is not usually the case with object systems. Alternatively, knowledge engineering methodologies have been used as a starting point. As agents are often knowledge consumers, knowledge engineering practices assist with that aspect. However, any basis for modelling the behavioural aspects of agents as autonomous entities (with motivational, means-end solving etc characteristics), or their distribution, is beyond the scope of knowledge engineering methodologies. Several researchers have reported the use of the European standard knowledge engineering methodology Common KADS (Schreiber *et al.* 1994). Some methodologies are presented in Table 1.

The reviewed methodologies resemble conventional software development methodologies in their structuring

Table 1. Selected MAS development methodologies

Name	Notable features	Summary
Multiagent Systems Engineering Methodology (MaSE) (Wood, DeLoach 2000)	Targets closed, static (agent lifecycle and inter-relationships) systems having 10 or less agent types. A goal hierarchy diagram captures the system specification. Wide use of UML diagramming (but sometimes different semantics e.g. class relationships represents high level communication) and automatic code generation with an accompanying tool. BDI supported in the last phase of agent architecture selection.	Analysis consists of goal and use case identification, and generation of sequence diagrams, then role identification and allocation to parallel tasks (tasks detail how goals are reached). In design, agent types are generated from roles with regard to concurrency, interactions are then detailed and agent architecture devised. For agent types and their interactions, deployment diagrams are produced.
Gaia (Wooldridge *et al*. 2000)	Targets "coarse grained computational agents" that have static/predictable inter-relationships and service provision. Suites systems types that aim to improve some collective utility cf. guarantee the best solution. Covers analysis and design. Central is the identification of roles and related "... responsibilities, permissions, activities, and protocols" and their (role) interaction (Wooldridge *et al*. 2000).	Analysis and design phases generate a range of models. For the former: roles and interaction models, and agent (types), services and acquaintance (communication between agents) models for the latter.
"Nikraz" (Nikraz *et al*. 2006)	Design phase specifically supports JADE. Testing not covered. Simple structure diagrams that show goal composition are prepared during analysis, later elaborated in implementation (parameterised for re-use, and structured for appropriate commitment), and again used later in the lifecycle to drive plan implementation.	Primarily analysis, design and implementation/testing phase. Analysis identifies candidate agent types, allocates responsibilities to the types, identifies collaborators, elaborates details and identifies deployment environment for each type. In design the agent types reviewed with a view to deployment (messaging overhead etc) and interactions are elaborated. Next non agent interactions are detailed together with the supporting ontology. JADE infrastructure resources are integrated.
Prometheus (Winikoff, Padgham 2004)	Detailed guidance at each phase, comprehensive coverage from specification to detailed design, and some support from a freely available tool. Supports agents based on "goals and plans" (Winikoff, Padgham 2004).	A three phase methodology: (1) System specification identifies system goals and use cases, (2) architectural design identifies agent types and use case scenarios which are elaborated into agent interactions, and (3) detailed design elaborates the agent types internal architecture (Padgham, Winikoff 2004).
Tropos (Bresciani *et al*. 2004)	Mental attitudes (including BDI) supported from analysis onwards. Development support for requirements to implementation. UML class uses with a meta model definition. Pattern application in (macro) architectural design.	Six phases: early and late requirements analysis, (macro) architectural design, detailed design, implementation.

into analysis and development phases, and to some extent in some of the content of those phases. The agent (micro) architecture and societal architecture development is supported by varying degrees. The BDI model is directly supported by most and while it is supported from the outset by Tropos, any overall advantage remains to be quantified. The use of UML notation is common. Some methodologies are more suited to particular MAS characteristics, while other distinguishing factors are the integration of tools and direct support for existing run-time frameworks at the implementation stage. Aspects of a particular methodology could easily be modified in most cases if more suitable techniques were not identified. No research on comparison metrics for multi-agent system development methodologies has been found.

1.3. MAS applications in AEC/FM domain

The widest application of agency in the AEC/FM domain is for the support of collaborative processes including concurrent engineering, management of supply chains, project scheduling and control, and e-commerce (Ren, Anumba 2004). These processes exist in AEC/FM, and require support, as a result of the distributed and disjointed nature of the AEC/FM sector in terms of organisation, project execution, decentralised control, authority and information and heterogeneous tools, working practices and information representations (Rueppel, Lange 2006). The scope of the support includes the application of standards and legal requirements, information retrieval and accommodation of time differences or preferences for different working hours (as proxy for the "missing" participant).

Examples in the area of concurrent engineering are the realisation of collaborative design frameworks such as, for example, for assisting the activity of fire protection engineering and for facilitating collaborative concurrent structural design processes. To support collaborative working in concurrent structural design Bilek and Hartmann (2006) utilise a multi-agent collaborative framework (constituting a middle "tier") that mediates between the individuals involved in the project effort and the resources on which the project depends. The resources with which the agents interact include product models as well as software tools, databases and other supporting resources. Agents are grouped according to the facility they provide such as workflow and coordination agents, product model agents, expertise agents, software wrapper agents. Workflow agents and coordination agents for example use resources in the layer below such as Petri nets (to model resource sharing, concurrency and time dependant activities) to achieve their goals.

Another application of agent frameworks is modelling the social behaviour of humans in building egress (Pan *et al.* 2005). The authors of that research state their belief that such systems are "particularly suitable for simulating individual cognitive processes and behaviour and for exploring emergent phenomena such as social or collective behaviours". Typifying the agent paradigm, the agents represent humans and are able to perceive their environment (doors, exit signs, other people, obstacles such as furniture), have ability to make choices and exhibit social behaviour, and are able to perform actions (walk, run, turn). In a simulation of the agents exiting from a building in an emergency situation, the authors report: "competitive behaviour, queuing behaviour and herding behaviour (is modelled) through simulating the behaviour of human agents at microscopic level". The results assist in facility design and management and checking conformance to regulations. Further examples in AEC/FM where agency has been exploited include monitoring and planning for construction sites (Zhang *et al.* 2009), and a sensor based security system for intelligent buildings (Luo *et al.* 2003). Research relating to agency in intelligent building in general is discussed in the next subsection.

2. System design and development rationale

In the context of the OntoFM, the following requirements have been concluded:

- An architecture that is practically implementable and deployable in "real" applications, with a good degree of framework support. MAS support including transport, hosting and lifecycle control.
- An internal agent architecture to support the realisation of pro-active rational agents including a motivational aspect, deliberative aspect and a procedural action element. Typically most solutions are close to the intuitive theories of BDI.
- Viable integration with the OWL knowledge sources.
- A publically available framework implementation.

Figure 1 below shows an integrated architecture for the developed OntoFM framework. In short, the infrastructure layer is managed by software agents (serving as an intermediate layer), and the upper layer includes different user applications. Infrastructure layer includes sensor hardware (wired and wireless), and the interface software executables working for data collection and transmission, and different information resources, such as database, ontologies, building information models, etc. Through the middle agent layer, the low level resources will be well matched with the requirements coming from application layer.

2.1. MAS framework selection

The JADE MAS infrastructure framework is widely reported as forming the basis of many published work in the domain, and meets all the requirements of the OntoFM. The framework provides support for agent infrastructure implementation incorporating FIPA messaging, agent hosting, lifecycle control, and other

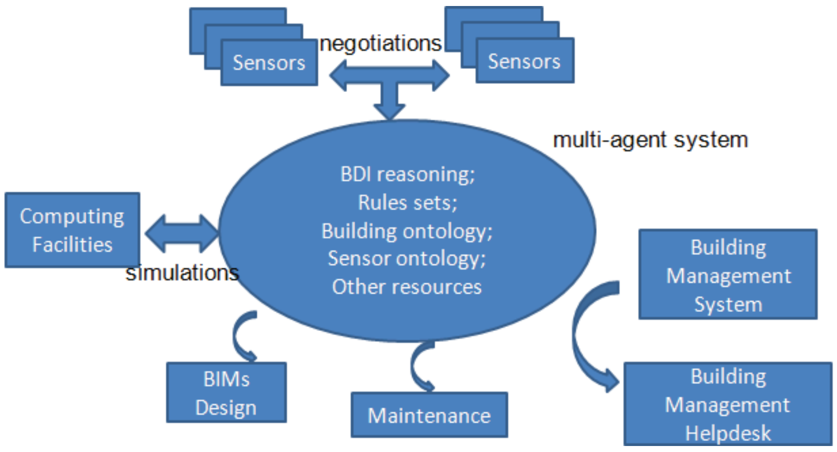


Fig. 1. Multi layered system architecture

infrastructure services such as agent location. Library support for FIPA-agent communication language (ACL) conformant messaging is provided for message construction and transport, but no semantics are forced. Depending on the application the programmer can implement the level of compliance that is appropriate.

The two most favoured internal architecture solutions, both JADE based, were JADEX and PRACTIONIST. Both support deliberation and means-end reasoning roles derived from theories of practical reasoning, both are publically available and well documented. The PRACTIONIST framework though is published as a "release candidate" with the statement that it has not been extensively tested. The PRACTIONIST framework is attractive however due to its goal centric implementation and some support for reasoning about various attitudes, incorporating the modal representation of beliefs. However in the Onto FM, most of the agents' knowledge of the world is captured in OWL ontologies or is closely integrated, and consist of fairly complex representations, and so for a meaningful exploitation of PRACTIONIST's internal mechanisms, a fairly extensive mapping effort would be required. Thus although the rationality of a PRACTIONIST agent is more transparent, in a practical application, JADEX was preferred due to its fairly open plan mechanism that allowed the addition of some customisations that add further transparent rationality. The decision was made to utilise the same internal architecture for all the main agent types in the OntoFM. However future agents that are integrated into the OntoFM may favour a different architecture choice. Potentially finer grained agent types that do not use OWL KBs would be better suited to PRACTIONIST implementations, but a more complete evaluation is left to further work.

In the JADEX framework, agent behaviour is defined with the specification of belief conditions, pre-conditions on sub goals and plans, and post-conditions (the post-condition is the intention in the case of plans). Behaviour can be further defined with other facilities including activation and inhibit conditions formulated as Java statements and belief states, event triggering, retry goal criteria, plan exclusion criteria, and goal and plan failure actions. Agent behaviour can thus potentially be achieved in a number of different ways and hence early in the implementation stage of the methodology. Additionally JADEX provides a number of modularised capabilities which encapsulate agent behaviour (fully configured cohesive goals, plans etc. targeting well known scenarios) such as commonly used functionality e.g. registering an agent with yellow pages facility and protocol implementations such as contract-net.

The implementation of agents in JADEX comprises of definitions in a configuration file having a framework provided universal schema to help maintain static agent configuration integrity, together with supporting plan implementations written in Java. The implementation task in the methodology targets the identification and definition of agent specific goals, sub goals, plans and all associated parameters including trigger conditions such as events and belief states. Contrasting a JADEX implemented agent with that implemented in JADE directly, JADEX combines its rule engine and the aforementioned agent definition to substitute a JADE "behaviour". The (forward chaining) rule engine uses an efficient pattern matching algorithm, to realise both means-end reasoning and (goal) deliberation.

2.2. Agent communication

Regarding agent communication, the JADE Semantics Addin (JSA) framework was not adopted, primarily because as the OntoFM system is closed and dialogue is uniform so the flexibility delivered by that framework implementation is not required. Moreover conformance to protocols that deal with more uniform dialogue does not require much coding overhead in JADEX, and as a result, message content is simplified. While the support of different sub systems in agents is feasible, the use of JSA would add complexity which is not required in a closed system. The architecture of JSA is closely integrated with semantics of speech acts, so no technical integration issues would have been expected. The semantic language SL was selected for use in agents' message content for the following reasons:

- A library is available for construction and parsing of SL statements.
- The schema allows the capture of nested and up to very expressive statements (the expressivity of a statement is the (semantic) "power"/richness captured, dependant on the constructs that it uses e.g. propositional forms are less expressive than first order predicate and modal logic forms). Complex grounded and ungrounded expressions (specifically *Content Element* instances) can be created describing objects and sets of objects (using single first order predicates or identifying referential expressions), and formulas can be combined, modified or quantified in those expressions using the defined connectives and modifiers (*and, equiv, implies, not, exists*). Formulas can also be combined with modal operators capturing attitudes: believes, uncertain, intends and action operators.
- The use of SL s a contents language is a FIPA standard in contrast to for example OWL. Although that consideration is less relevant for internal agent dialog, SL expressions can be readily consumed by external tools.

From the FIPA ACL content reference model, the classes predicate, concept and occasionally agent action were elaborated with Java based ontologies by constructs that typically map corresponding OntoFM ontology constructs (including reification in the case of object or data properties), to capture a (simplified) sub-set of those ontologies that is adequate for dialog, and to capture the required agent actions. Specifically the simplified classes contain a reference to the corresponding ontology class URI. The requirement for the creation of the Java based ontology described could be considered a disadvantage due to the (limited) redundancy rendered, but without the FIPA SL schema a similar model would have been required, probably expressed in OWL ontology. While a restricted subset of the SL vocabulary's semantics for the OntoFM application could have been selected and defined in such ontology, custom implementation would still have been needed to process the statements in messages. The selected semantics supported would necessarily be restricted by the lower expressivity of OWL, as well as just providing support for those semantics required for the immediate application. Moreover it was considered undesirable to require all agents to be OWL based.

The Java ontology defined for the purpose of SL message content (and for some limited belief base components, typically for the purpose of buffering mechanisms) has a narrow scope and limited expressivity. While most OWL constructs can be mapped to the object oriented domain such as Java, the ontology was restricted to a few constructs so that the re-expression of knowledge remains holistically relatively straight forward. The Jastor project libraries (Szekely, Betz 2009) can be used to generate Java ontologies from OWL in the form of Java Beans classes, and while there are fundamental differences between semantics of the OWL KR and its corresponding Java representation, the agents' usage does not impinge on those areas. An example is the difference in the semantics of the definition of necessary and sufficient conditions including a role restriction for example, where in OWL any individual with a definition that matches the former will be inferred to be a member of that class, in contrast to a typical mapped Java implementation (including Jastor's), where the Java implementation just upholds the constraint corresponding to the role restriction (Kalyanpur *et al.* 2004).

2.3. Building information model

Due to the complexity of generating and maintaining a realistic ontological building representation, and moreover for the frameworks' application as a user friendly tool, the use of a manually generated building representation was not considered feasible. Thus a requirement for agents to interpret a building model was added. An increasingly well supported (by modelling tools) open standard is the IFC (Industrial Foundation Classes) schema, and a comprehensive and popular modelling tool supporting that schema via export and import is Autodesk Revit Architecture. Thus good building modelling integration in the OntoFM was rendered by its adoption of the IFC as the primary "import" format for building models.

The library OpenIfc Java Toolbox (Tulke, Tauscher 2009) is a facility that allows programmatic access to IFC models. The library allows the reading and writing of STEP (Standard for the Exchange of Product model data) physical files containing IFC schemas, via an object oriented representation of IFC entities, as well as providing some data management functionality. The Java classes are a close mapping of the basic EXPRESS entities and attributes of the IFC schema, so the developer requires experience of the IFC in order to use the library. The tool kit captures a binding between the EXPRESS schema language (Schenck, Wilson 1994) and Java which has less expressivity and different constructs, so a simple complete mapping is not always possible. For example EXPRESS includes local (to entities) and global rules, and derived attributes. However it was found in practice that no information was missing when working with OpenIfc Java Toolbox, so the difficulties in the mapping do not currently affect the representation, at least in this instance. An alternative to the tool box library is the direct use of the STEP SDAI (Standard Data Access Interface), but the former is significantly simpler for the programmer to use.

The building model is accessed by agents to elaborate semantic knowledge bases and to resolve geometric queries that are beyond the scope of semantic representation.

2.4. Implementation languages

Primarily implementation language selection decisions were derived from any constraints for compatibility with the APIs and implementation languages of selected frameworks, together with a legacy constraint stated by a sponsor at the project outset that the infrastructure should be implemented with Microsoft products. The constraint was so that any implementation would be directly compatible with the sponsor's existing products. Fortunately that was favorable regarding the interfacing to some hardware, especially the National Instruments USB device interfaces, as the interface library provision is only available on the Microsoft.Net platform. The decision to implement the upper agent layer using Java was determined by the very good support for both MAS and OWL ontology interfacing and KB support (specifically reasoners) by that language. Moreover the provisions are only available in that language. To facilitate communication between the two different language based virtual machine types hosted by the layers, an interface provision was thus required. The open source project IIOP.Net is a Remoting channel implementation that is customised to use the IIOP protocol, it hosts a CORBA Object request broker (ORB), and performs translation between the .Net and CORBA type systems. Using Java's RMI/IIOP facility, an object based interface can thus be realized. Additionally the library also supplies an executable to process the meta data contained in .Net assemblies in order to generate Interface Definition Language (IDL) files. The IDL files can then be used with the Java IDL compiler ("idlj") to generate Java language bindings. Regarding the use of Java libraries in the .Net environment, the free software IKVM offers the potential ability to translate Java byte code into the .Net Intermediate Language (IL). Jena was successfully converted but the relatively high number of dependencies of the Jena framework meant that a high overhead in terms of configuration and run time support was required.

3. Ontology design and development

Several ontology development methodologies have been published and this section summaries their salient defining characteristics. Those characteristics are listed in Table 2. The motivation is to identify the most appropriate in the context of the OntoFM, or to inform the development of a custom methodology. The developed OntoFM ontologies are introduced as well in this section (more ontology development details included in Dibley *et al.* (2012).

On comparing the published methodologies, there is a variation in scope and level of specification of the processes described. As expected the main focus of most is authoring, but some also cover, to various extents, lifecycle management and development support activities such as knowledge acquisition, evaluation, integration, merging and alignment, and configuration management. The methodologies also exhibit varying application independence e.g. Cyc (application dependant), SENSUS (intermediate dependence), and On-To-Knowledge Methodology (OTKM) (independent) (Gómez-Pérez *et al.* 2004).

Regarding Neon, a specific feature is its support for the development of contextualised networked ontologies. Various Meta Object Facilities (based Meta models) are defined including those that allow the specification

Table 2. Defining characteristics of selected ontology development methodologies

Name	Defining feature/s
Neon	Complete and detailed support ("step by step" guidance) for reusing existing resources in 9 scenarios e.g. starting with taxonomy, semantic, from "scratch". Familiar alignment with familiar software engineering paradigms. Various granularities of reuse are supported: whole ontology reuse, ontology module reuse, reuse of individual ontology statements, and reuse of ontology design patterns. Supports "contextualised networked" ontology development with the specification of Meta Object Facility based meta models, covering ontologies, rules, mapping and modularisation.
Methontology (Fernandez-Lopez *et al*. 1997)	One of the most comprehensive and is typical in that it has distinct phases aligned with software engineering methodologies. Those phases are: specification, conceptualization and formalisation (conversion of the conceptual model into a formal model [formal up to the formality of the KR, not necessarily in a mathematical sense]), implementation (transforming the formal model into a representation with a KR language).
Cyc (Gómez-Pérez *et al*. 2004)	Customises and extends an existing, extensive high level ontology. New ontologies are specialised from an extensive existing ontology, with tool support.
SENSUS (Swartout *et al*. 1997)	Customises and extends an existing, extensive high level ontology.
On-To-Knowledge Methodology (OTKM) (Sure *et al*. 2004)	After capture of requirements, a semi formal ontology is created which is later formalised into the target ontology. Evaluation of that ontology from different perspectives then follows. A maintenance phase is specified. Refinement, evaluation and maintenance phases can iterate.
Uschold and King methodology (Uschold, King 1995)	Employs process stages: purpose identification, building (capture, coding, integrating), evaluation and documentation.
Grüninger and Fox methodology (Grüninger, Fox 1995)	The Grüninger and Fox methodology introduce formality after the scope of the ontology has been identified. The scope is derived from informal usage scenarios and "competency questions". "The competency questions and their answers are then used to extract the main concepts and their properties, relations and formal axioms".

of ontology mapping and modularisation. Specifically regarding modularisation, the OWL specification only provides limited support for modularisation via its definitions of owl:imports semantics. Making no (Meta level) distinction between "native" and imported entities, the nominated ontology is simply included as a whole. Neon's modularisation facility in contrast permits partial importing. Also facilitated by the modularisation facility is an information hiding provision which, similarly to that in object oriented (OO) engineering, allows the specification of reusable "interfaces". The technique allows, for example, parts of an ontology to be developed (evolve) "behind" that interface without requiring changes in the interface clients, thus leading to easier maintenance of deployed systems. A benefit of partial importing is that its application could be an alternative to ontology pruning for specific applications, for the purpose of attaining performance improvements in ontology classification and realisation for example, again easing maintenance.

Regarding ontology design, most methodologies include strategies to identify concepts and to derive a taxonomy, and here the approach varies between top down, bottom up or middle out (Gómez-Pérez *et al.* 2004). With a top down strategy, where the most abstract entities are identified first, the level of abstraction can be introduced in a consistent way but the structure may suffer from unnecessary abstraction, and commonality may be dispersed if the abstraction of artificial entities is too fine. The converse, bottom up, where the most concrete entities are identified first results in very high detail in the taxonomy. Often many entity layers are not needed and common characteristics can reside in multiple entities which can in turn lead to inconsistency. A "middle out" approach is a compromise; identifying the core entities first and then abstracting and specialising them as needed leads to less redundancy and better structure. A number of authors have presented comparison criteria for ontology methodology comparison. Fernández López (1999) presents nine criteria including the level of specification, level of application dependence, concept identification techniques, comparison with the IEEE standard for software lifecycle processes, link to any KR formalisms, as well as others. Gómez-Pérez *et al.* (2004) elaborate on some of these categories.

3.1. Common ontology design principles

In the area of fundamental formulation, Fielding *et al.* (2004) identify the classifications of "endurants and occurrents", "dependent and independent" and "universals and particulars". Endurants and occurrents refer to the temporal existence of an entity and never form part of relationships with each other (Fielding *et al.* 2004). In SUMO the separate high level entities object and process reflect the temporal distinction for example. The authors' dependency classifications describe the necessity for existence of membership of a whole e.g. the concept of door function relies on a door and other entities. The criteria of universalness make the distinction between type or class and individual or instance. Those classifications and associated properties were taken into account throughout the ontology implementation.

Another general ontology design consideration was that of semantic closure, arising as by default OWL semantics adopts the open world assumption (OWA). The assumption though suits the nature of the domain and the KR used to model it. The relatively high KR expressivity allows rich semantic expression, so a complete model may be unnecessary, or it may be impractical or impossible to capture. The application of the OWA means that incomplete knowledge can still be consistent. However there are areas which in contrast are complete and so explicit closure with appropriate axioms can give additional useful inference e.g. the sensor ontology states that an enabled and fully functional passive infra-red (PIR) sensor signal indicates movement, so closure indicates no movement. However closure is not appropriate in the relation between movement and occupancy i.e. a room can be occupied even if no movement is detected. An alternative approach to implementation without using closure statements is via Pellet's integrity constraints where axioms can be nominated as having closed world assumption (CWA) based semantics and thus interpreted as such by the reasoner, for example using annotation.

A further consideration in modelling OWL ontologies is the lack of the unique names assumption (UNA). While Pellet has an option to assert the UNA via the API, for compatibility with Protégé tools and general reasoner compatibility, design time and run time ontology updating by agents add *owl:different From* properties (or the construct *owl:All Different* for a set of pair wise different) for appropriate individuals. Missing statements relating to UNA and OWA have a significant negative impact on the ontologies, particularly where modelling involves statements with universal role restrictions. However other model statements can lead to the desired intermediate (different from) inferences, e.g. individuals can be inferred as different through their inclusion in roles having functional properties.

3.2. Ontology interaction support libraries

Jena is an ontology application programming interface (API). The API presents Java classes representing the ontology language constructs, together with classes to facilitate model reading and specification, thus allowing object oriented program development supporting OWL ontology manipulation. It was selected due to its support for OWL (and OWL2 with some extensions), its support for the query language SPARQL, and its integration with the Pellet reasoner providing abstract interfaces. Additionally, Jena has a number of built-in reasoners capable of delivering RDFS inference among others, which found useful application. Another popular API, namely the Manchester OWLAPI (Hamscher *et al.* 2000) has a number of advantages including its support of a range of syntaxes, its integration with a number of reasoners, and its interfaces for explanations. However it does not currently support SPARQL queries.

The Pellet reasoner was adopted for its support of OWL2 reasoning, SPARQL query support, comprehensive SWRL rule support (when combined with Jena's ARQ query engine), and its support of explanations. An anticipated application of rules was, for example, as a convenient way to apply temporal constraints. The Pellet reasoner provides coverage of nearly all the SWRL operators ("built-ins") that support manipulation of a range of Extensible Mark-up Language (XML) schema data types in rules, and that coverage is adequate for the expected potential scope of use. Due to the emergence of the semantic web, the support for semantic KBs is good, particularly using the OWL KR (Bergman 2010) Support is provided by editing tools and reasoners. Additionally a range of work on ontology development methodologies has been published (Gómez-Pérez *et al.* 2004).

3.3. OntoFM ontologies design hierarchy

Figure 2 shows the developed ontologies in OntoFM and their relationships, and how agent, sensor hardware and ontologies work together to achieve the research targets. The sensor infrastructure supplies hardware interfaces and access to those resources which is scalable to several hundred units with a throughput of up to 1 Hz.

The application of ontology to the OntoFM offers several benefits to the system:

– The use of ontology allows the reuse of domain dependent and independent knowledge.
– The externalization of knowledge means that the exchange of statements is accurately defined. That delivers benefits internally within the agent layer but is particularly useful for interfacing to intelligent external tools.
– The capture of knowledge of the complex domains can be represented very concisely such that a large proportion is inferred, and thus is easier to maintain.
– The semantics of the OWL knowledge representation (KR) used allows complex knowledge modeling but without necessarily "full" definitions, for example role restrictions define some facts about relationships but lack detail about the types and numbers of the fillers. That modeling suits the nature of the complex domain where such complete knowledge is not known at modeling design time, or it can change. Moreover the statements, appropriately formulated, remain consistent.
– The formal representation allows consistency checking, which in a complex model is very beneficial in the identification of model authoring errors, or at run time during KB updating.

During development the ontology set has undergone several fairly extensive evolutions, but very little software needed to be altered to accommodate these changes, while the programmatic exploitation of the improved model was achieved with the simple addition of Java code. While accurate and philosophically sound modeling was a main concern in the authoring of ontologies, the need for practical simplicity and appropriate reasoner output was recognized. The scope of semantic expression did not include numerically based domains e.g. geometry that would have delivered little or no benefit by capture in an ontology. Instead, alternative mechanisms are used where appropriate so for example in the case of building geometry, some ontology entities cross-reference a semantically compatible representation in contained in the IFC model, and that representation is processed numerically.

4. Agent reasoning implementation

Five primary agent types have been developed: *Zone Agent*, *Sensor Node Agent*, *Yellow Pages Agent*, *Utility Agent* and *Facility Manager Agent*. Through elaborating the domain beliefs, the zone agent is used to generate zone centric knowledge which relies on the intelligent

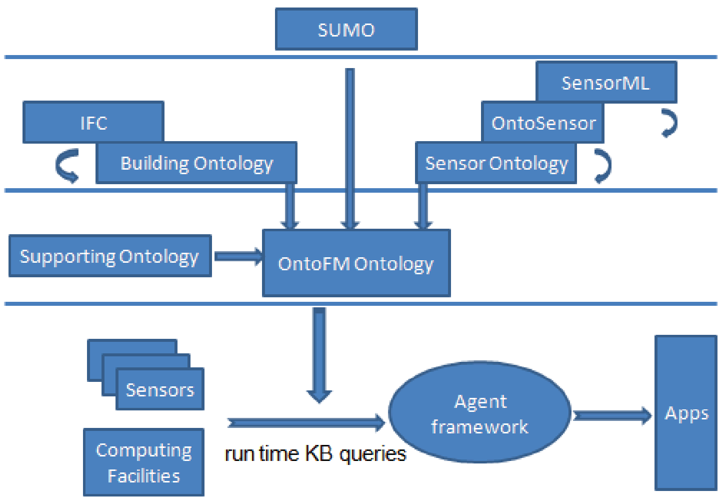


Fig. 2. OntoFM ontology development architecture

selection and utilisation of sensor resources (facilitated by the knowledge base and building model). The sensor node agent is used to work with sensor (wired and wireless) related tasks, including resources provision, sensor configuration, minimizing power consumption, etc. The yellow pages Agent is used to serve as a central registration server, any agent needs to register itself first in order to get the relevant resources/information. Finally the utility agent is mainly used for analysis of performance of goals followed by other agents and for data logging and the facility manager agent is used to issue goals. The implementation of sensor node agent type is explained (as an example) in detail below due to its comprehensiveness.

4.1. Sensor node agent type implementation

The primary goal of the sensor node agent type is to deliver resource provision in terms of monitoring data to other agents that request it, while managing efficiently that provision, especially in the case of finite resources. The battery powered wireless sensor units managed by the sensor node agent have a finite power source and the system (but primarily this agent and benevolent clients) aims to maximise the interval between those battery replacements. The device lease class plays a central role in dialog between agents relating to resource provision. A summary of selected high level goals for Sensor Node Agent type is given in Table 3.

4.1.1. Use of ontologies

The sensor node agent type makes extensive use of the sensor node ontology to support means-end reasoning and some deliberation. Similarly to the zone agent type, the sensor node agent creates and configures a number of different KBs for use in different reasoning applications. The use of RDFS inference offers much shorter inference delivery for event identification, relying on limited expressivity compared to the application of full OWL inference rules. A Pellet inference supported KB is also configured and is widely used for general full expressivity reasoning. Typical applications of inference are to analyse the connection of a given device in order to determine the handling of lease requests, to evaluate the power mode for the host node, and to elaborate sensor clusters to find connected channels and devices. Additionally the connection topology is analysed for other characteristics such as connection to a mains electrical outlet (thus not battery powered) and other queries involving the T box, or to determine if a device is wired or wireless. The use of abstraction describing sensor type and characteristics is not so extensively used as clients typically request specific sensor individuals, i.e. that role is usually completed by client agents.

4.1.2. Service provision

In support of the sensor node agent type's primary goal to deliver requested sensor provision, the agent performs several other high level goals in support of that. Those goals involve finding infrastructure resources, identifying the resources available (in order to "advertise" to other agents), as well as managing those resources efficiently. Although the infrastructure components have default behaviour, the lack of intelligence in that layer means

Table 3. Sensor node high level agent goals summary

Goal	Responsibility	Notes
Initialise	Read configuration file	Configure agent identity, provide message routing info
Manage infrastructure node connections	Discover infrastructure nodes	Periodically poll the known endpoints for new resource availability. Maintain "active nodes" list
	Extract sensor events	Poll active endpoints for (infrastructure) events
Advertise sensors	Register services with YP	
	Retrieve resource list	
Manage clients	Listen for subscriptions, sensor lease requests, general requests	Interpret SL message
	Subscribe client	maintain lease subscriptions lookup table
Collaboration	Register with Yellow Pages (YP), Advertise resources in YP agent	Register agent type, associated zone identifier, hosted devices, with the YP agent. Refresh on any change
	Describe resources/sensors	Elaborate descriptions using the sensor ontology for rapid data response from clients. Triggered on addition of new knowledge of devices
	Service requests	Reply to data requests after verifying lease status. Request data from infrastructure (sensor read)
	Notify subscribers	Formulate SL message and notify lease holder of new data
Negotiation	Manage sensor leases	
Wireless network management	Configure wireless nodes, configure individual sensor channels and manage power settings	Serve sensor lease requests with wireless sensor node availability, minimise power consumption of nodes. Manage networks (configure nodes and sensors)
Manage wired networks	Monitor sensor availability	Grant leases for available devices

that the sensor agent has a central role even when no resources are requested by clients, particularly regarding the wireless hosted resources. The implementation of simple default behaviours in the wireless network was a necessary design decision made in order to reduce redundancy and possible conflict.

Effective sensor node agent behaviour relies on the sensor ontology KBs in order to direct actions in the plans (means-end reasoning), as well as on the algorithmic implementation of plans. The dialog over resources is based on a sensor lease class which facilitates requests and allows verification of status. The lease lifecycle is dependent on the wireless network status and success of the sensor node agent's actions. In general node and device configurations are not executed immediately. The sensor node agent can modify the lease interval requested, and is able to select the device that fulfils the lease from a number of alternative devices the requestor has nominated. Alternative selections are granted depending on availability of the device and its host.

The agent manages some meta data relating to devices and their activation. For example a device activation history is maintained that is used to implement signal conditioning to suppress spurious transient signal generation that are characteristic of some sensor types when they are first powered on, particularly PIR devices. For that purpose "suppress" intervals (derived from datasheets) are mapped to abstract types. Other Meta data is managed for wireless network nodes.

4.1.3. Device leases

The device lease class when used as the content of SL expressions in inter-agent messages plays a central role in realising resource negotiation and verification of status. The lease resolutions and states are:

- Resolutions: None, Initialised, Pending, Granted, Delayed, Denied – determined by device availability;
- States: Active, Inactive – set by the start and end times.

In the case that a given requested device is attached to a node that is available in the network, following successful node and device configuration of wireless devices or without further action for some wired devices, the requested lease will be assigned the granted resolution. If the node is not available or the configuration fails for another reason, the lease is assigned as delayed. The resolution is also assigned as delayed if Meta data is held stating the node is currently unavailable, and in that circumstance the configuration action is not attempted. Those leases with a start time later than the current time are assigned the pending resolution. If the host is not recognised, the lease is set to the denied state.

Regarding the setting of the duration of leases requested, typically very short leases are used to "sample"/read a value, while longer durations are used to "subscribe" to, and thus receive asynchronous notification of events such as motion and switch activation. Regarding the timing of issuing lease requests, client agents that employ scheduled reading can request leases in advance to allow lead time for activation. Another temporal consideration regarding leases on a shorter time scale is that some sensors require an interval for circuit for stabilisation as mentioned above in connection with signal conditioning. As an example a device that has a relatively long stabilisation time is the "Napion" motion sensor range at 30 seconds.

4.1.4. Device management

In order to deliver the best timely responses to new leases, the sensor node agent attempts to action newly requested pending leases immediately. Some leases can be granted without further action as mentioned, such as those for wired devices, or those leases which are requested for sub intervals of those already active. Next, if a compatible active lease exists the agent extends it. If the lease requested is for a wireless hosted device then the agent then initiates wireless network management.

Regarding the management of wireless networked resources, the agent has the role of assigning behaviours defined in the infrastructure implementation to wireless nodes, and configuring the devices attached to the hosts appropriately. Those devices are both actuators and sensors; the actuators control the power to sensors. The node behaviours are mapped to certain sensor KB inferences and so when appropriate, i.e. it is inferred that a request for a new resource requires a different node behaviour to the existing one, that node behaviour set command is issued before the device configurations are issued. Those node behaviours assign configurations for ZigBee node devices such as its radio components, timers, and timer activation of preset actions for example for network management, resulting in characteristics such as power consumed, sensor availability, and sensor availability "lead time". The target host node availability is dependent on its current configuration (behaviour), or there may be other reasons for its unavailability such as an expired power source.

From the range of node behaviours available, the *sleep-and-listen* mode is very desirable for assignment to nodes that have no active leases for hosted devices, but it is not commonly used. One reason for not using that mode extensively is due to lead times in availability, particularly where there is little redundancy in device roles from the client agent perspective, and given that typically clients assign and change roles in a very dynamic fashion. The purpose of the device Meta data though is to track the configuration of nodes, exemplified by the case where a node is not available to retrieve its status. Another factor is that before activation of the *sleep-and-listen* mode, the agent has to ensure that as well as a feasible electrical configuration for waking the device is available and that there is also a feasible physical scenario. An undesirable situation is if a node was put into this power mode and the wake up scenario was rarely encountered e.g. motion detection in a rarely assessed room.

In contrast to "on demand" node management, the agent performs routine network management where devices with associated expired leases are powered off, and host nodes are put into a standby mode when possible. The power modes standby and low power are the most commonly used modes.

4.2. Agent messaging

The implementation pattern for message listening by agents is division by (Java based) ontology, where each ontology has a corresponding plan handler. Thus listening plans are implemented for devices, zones and the most general event based messaging. The implementation is a fairly wide category approach to reduce the overhead of defining multiple finely grained message handling events, while also deriving some structuring from the ability of the JADEX's internal search/match implementation for Agent Communication Language (ACL) message processing. The framework uses search matching on ACL Meta data, and here with suitable plan implementation, the ontology identifier is an adequate discriminator. The relevant plan then further discriminates on the semantic language SL content in most cases, typically using run time class checking after de-serialising the SL statement. Specifically in the case of an Identifying Referential Expressions or action expressions for example, the agents determine the type of the received message act depending on the run time class of the extracted primary predicate (in practice the predicate name is a constant so is checked instead in some cases).

The message handling plan is responsible for message interpretation and propagation, which typically involves the request for an action, belief or intentional attitudes (externalised intentions, in the form of commitments, describe goal entailed resource use and duration for example), or for the updating of beliefs. All action requests are honoured and all notifications are trusted. Most (about 85%) message encoding uses the SL, but where the expressivity of SL is not necessary, some implementations (for ease of implementation) use a custom binary encoding based on standard Java serialisation. Most message content cross-references ontology URIs and typically, new concepts introduced by the (Java based) communications ontologies are reifications of relationships made by the agent, adding further properties about its beliefs. For example a concept called zone characterisation adds a timestamp value and the agent's identifier. The JADEX framework provides implementations of several FIPA defined interaction protocols such as contract net and auction variants. However, the IFMS interaction, while observing FIPA messaging semantics, remains relatively simple, so no such provision is utilised. The setting of timeout values for messaging related activity had to consider the dynamic behaviour of agents. Synchronous queries involving reasoning can take several seconds so relatively large timeouts are required. Further refinements of settings were completed during deployment testing.

4.3. Performance consideration

For performance considerations, in some plans buffering is occasionally used. The buffering is of some infrastructure related knowledge, e.g. location information, and some ontology derived knowledge. However the use of buffering was only used where strictly necessary, due to the synchronisation requirement and overhead in maintenance introduced. Buffering of semi-static and short lived data, such as conclusions from complex ontology queries and infrastructure related knowledge, did significantly improve performance in specific situations, particularly location finding related interactions with the yellow pages agent. Those situations typically involve agent's participation in inter-agent dialog. Implementations were usually the result of unit or integration testing conclusions. In contrast to the development of the infrastructure, the application of patterns in the design of agents was very limited. The implementation of agents using the predefined internal agent architecture, and integration with JADE MAS framework, meant that design and implementation is typically at a higher level of abstraction than infrastructure design and it is at the more fundamental level that those patterns find application.

The development of the infrastructure layer and to some extent the agent layer were developed by following conventional object oriented development, using selected Unified Process workflows. The process is characterized by use case driven, iterative and incremental development as well as "architecture centric" (Jacobson *et al.* 1999). The iterative and incremental nature allowed in particular the dynamic system behavior to be investigated and evaluated, principally from a realistic system deployment, from which observations were feed back into analysis and design. A case tool, namely Visual Paradigm was central to the development, especially the early iterations. The case tool provides code generation and class diagram creation but no C# "round trip" engineering in the version used, which hindered, in the case of infrastructure development, the ease of maintaining the model in the later development stages.

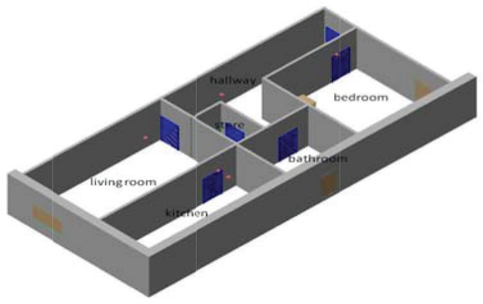
5. ZigBee sensor network deployment and reasoning evaluation

Two deployments were used to test and evaluate the system. The first was a small domestic flat (Fig. 3 (a)), which was primarily used for initial development and early testing, while the second deployment is a large meeting area for students in a university building, together with several adjacent offices (Fig. 3 (b)). The first deployment uses up to 5 wireless nodes and a few wired devices. In the second deployment, there are 10 wireless units and a small set of wired sensors. The type of sensors attached to the wireless devices varies but includes ambient light level sensing, one or two motion sensors and a temperature sensor (Fig. 3 (c)). The selection criteria of sensor hardware were primarily sensing capabilities that match indoor environmental conditions and very low power consumption. In addition some platforms host proximity sensors attached to doors.

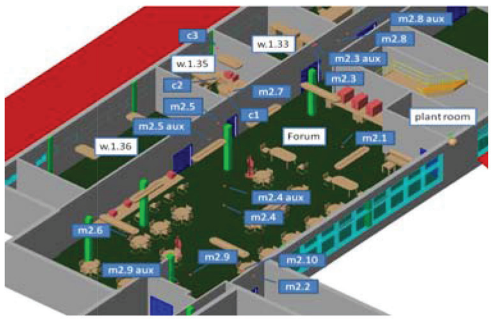
The testing hardware uses several PCs to host the infrastructure modules, agent platform and agent executables. A National Instruments digital input/output unit and a ZigBee network controller are connected via USB. The sensor hardware deployment (for a flat) is outlined in Table 4. An excerpt of a rendered IFC model (for university open area) is shown in Figure 3 (d) embedded with a real deployment photo. The large arrows in (d) point to disks in the ceiling region that represent sensors, sensor clusters or a wireless node with sensors attached. Figure 4 shows a wireless sensor system deployment diagram supporting a range of sensors on an ETRX357x platform.

Table 4. Domestic flat sensor hardware outline

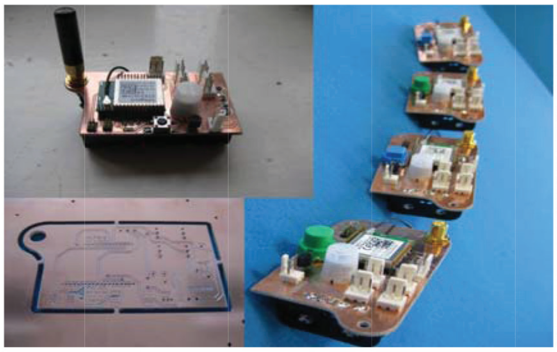
Room	Sensor deployment explanation	"highest" sensing capabilities
Kitchen	Wireless unit providing coverage of the living_room/kitchen doorway and interior, temp and lux monitoring	Opening monitor counting, environment
Living_room	Wireless unit providing coverage of the living_room/hallway entrance and interior, temp and lux monitoring	Opening monitor counting, environment
Hallway	Two wired motion sensors	Continuous motion

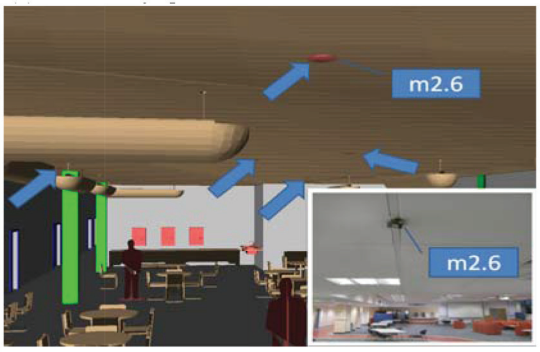


(a) Domestic flat sensors deployment



(b) A university open area for sensor test


(c) ZigBee sensor unit



(d) Virtual environment for a university open area

Fig. 3. ZigBee sensor units and network deployment

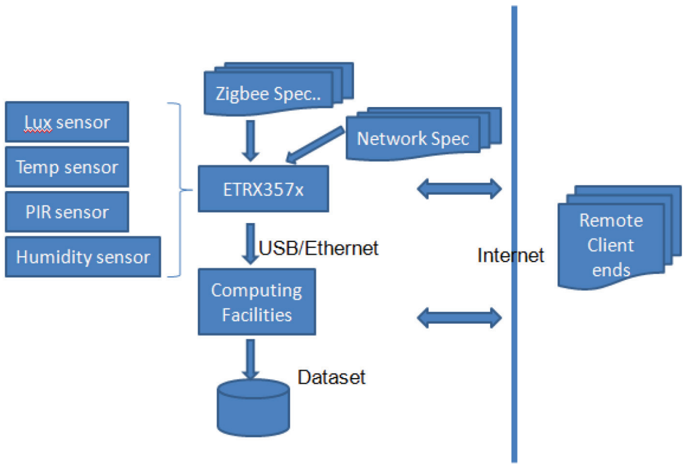


Fig. 4. An example of a wireless sensor network deployment

Table 5. ZigBee Node behaviour characteristics

Behaviour	Usage
Low power	Typical usage as "sleepy" device. 1 sec network (firmware, part of the ZigBee stack implementation) based polling for good performance.
Standby	Reduced network polling, sets attached devices to a disabled state to reduce power consumption, removes listen, etc.
Sleep-and-listen	Deep sleep only woken by external event e.g. PIR activity. Very low power as radio and polling, timers etc. are deactivated. The agent will only use this mode if there is hardware connected, it is feasible that an associated event will occur and it is acceptable to have the node unavailable for an interval. The agent adequately configures any devices used to detect the wake up event. High level goals and historical leases are taken into account as well as the wake up constraints before setting this behaviour. The agents typically check for previous events and linked activity to assert that the node will become available when pursuing such event based goals. By querying the ontology events capable of generating wake up events can be counted.
Empty	A behaviour that does nothing. The other behaviours repeat failed steps until success, such as would occur due to transmission failure (NACK) or timeout (not present), so the empty behaviour should be assigned to those nodes that are not available, to eliminate unnecessary radio traffic.
Power definable	Typically the agent could set "awake" mode so that the node can act as a router. Agents do not currently use this mode directly, but it is used as a super class for other behaviours.
On-board timer power mode control	An on board timer controlled power definable useful for USB connected host that is power critical. Not currently used by agents but used for testing.

A number of behaviours for assignment to sensor nodes are implemented and their characteristics are outlined in Table 5.

5.1. ZigBee network communication test

During development of the ZigBee network interface, some unit testing was carried out by hard coding a few dialogs (replies to some implemented commands) to substitute the serial interface. After integration to the serial library, a terminal program into which responses were manually typed was then used initially before testing with the ZigBee serial hardware interface. However timing constraints, and the level of detail required to formulate meaningful responses, limited the practical usefulness of the terminal program to simple scenarios. The user interfaces for the ZigBee network (shown in Fig. 5) are primarily for status display and a facility to assign "behaviours" to sensor nodes was implemented for testing purposes. In Figure 5, *"sensor node"* screenshot shows data collecting information – reading in data through digital I/O etc.; the *"ZigBee network interface"* shows the communication between different sensors. The arrows represent communication channels and indicate the direction of the flow of data. Those behaviours consist of some configuration commands and the issuing of some write commands that enabled visual diagnostics (the development kit units have LED status indicators on some of the channels). For the next integration stage, a utility agent was developed to, in a controlled and predictable way, request leases, read and log data.

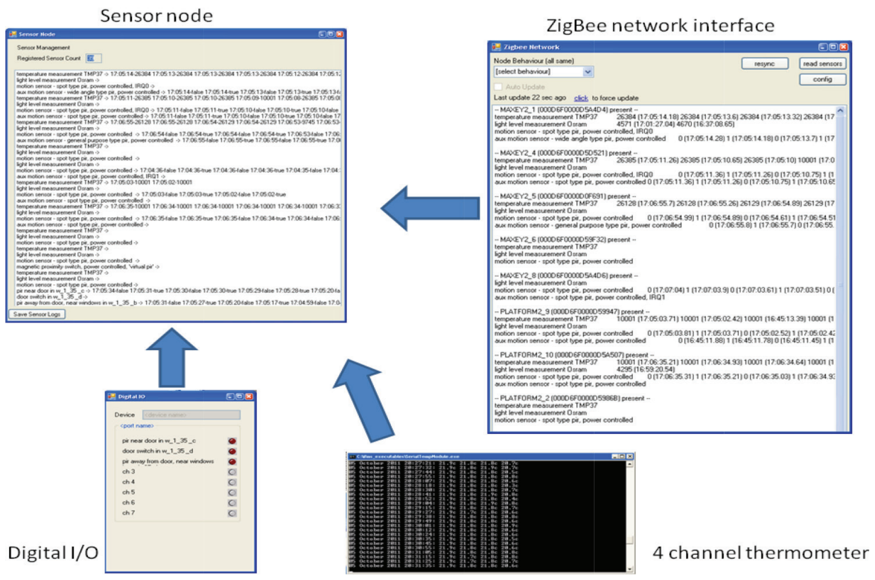


Fig. 5. Some system infrastructure executables

Regarding the ZigBee interface's operation with the rest of the infrastructure, including the registration of its sensors and the updating of data, the same interfaces as those used as by the wired network are employed and such testing of the associated functionally was covered, so no further testing was required in that area, apart from the simple testing of additional façades in some cases. The ZigBee interface's implementation is primarily event driven and includes several multi-threaded mechanisms for processing serial data, issuing commands and synchronising wireless node proxy objects. The mechanisms interact and so during testing, the settings for various triggering mechanisms, timeouts for synchronisation objects and for other behaviours such as the default activation of the timeout invocation for the handling of error states were revised to give the desired overall behaviour under different scenarios.

The sensor manager interface implemented for registration and updating by device interfaces typically realise the application of the façade pattern (Gamma *et al.* 1995), exemplified by a restricted set of high level methods using types supported by the IDL to Java mappings. While the IIOP.Net libraries allow the custom specification of language construct mappings, the primitive built-in types were adequate for use in the façade definition e.g. substitution of simple array for complex collection types used internally. A simple type used in the façade for which custom implementation was required was the date type, which handles daylight saving time and time zone. Regarding the call semantics across the remoting channels, the original implementation was kept as simple as possible by using a combination of pass by value and reference, and the use of uni-directional implementations where possible, avoiding the requirement for the client to register a listener sink for call-back implementations. The sensor manager specifically, apart from the façade interface, hosts other interfaces suitable for use within the infrastructure layer and for an ASP based web monitor. Figure 6 below shows some selected ZigBee network interface class hierarchies, including behaviours, node types etc.

The sensors currently connected include temperature, motion detection (PIR), proximity switches on doors and windows, and ambient light. Most sensors and actuators are hosted by ZigBee wireless platforms. Actuators are supported both in hardware and software but currently are only used to control sensor power. The classes capturing sensor history, which realise persistence, were generated from a case tool and employ the NHibernate (Maulo 2006) object relation mapping framework, so therefore benefit from database performance enhancement delivered by those libraries. As well as the use of the façade

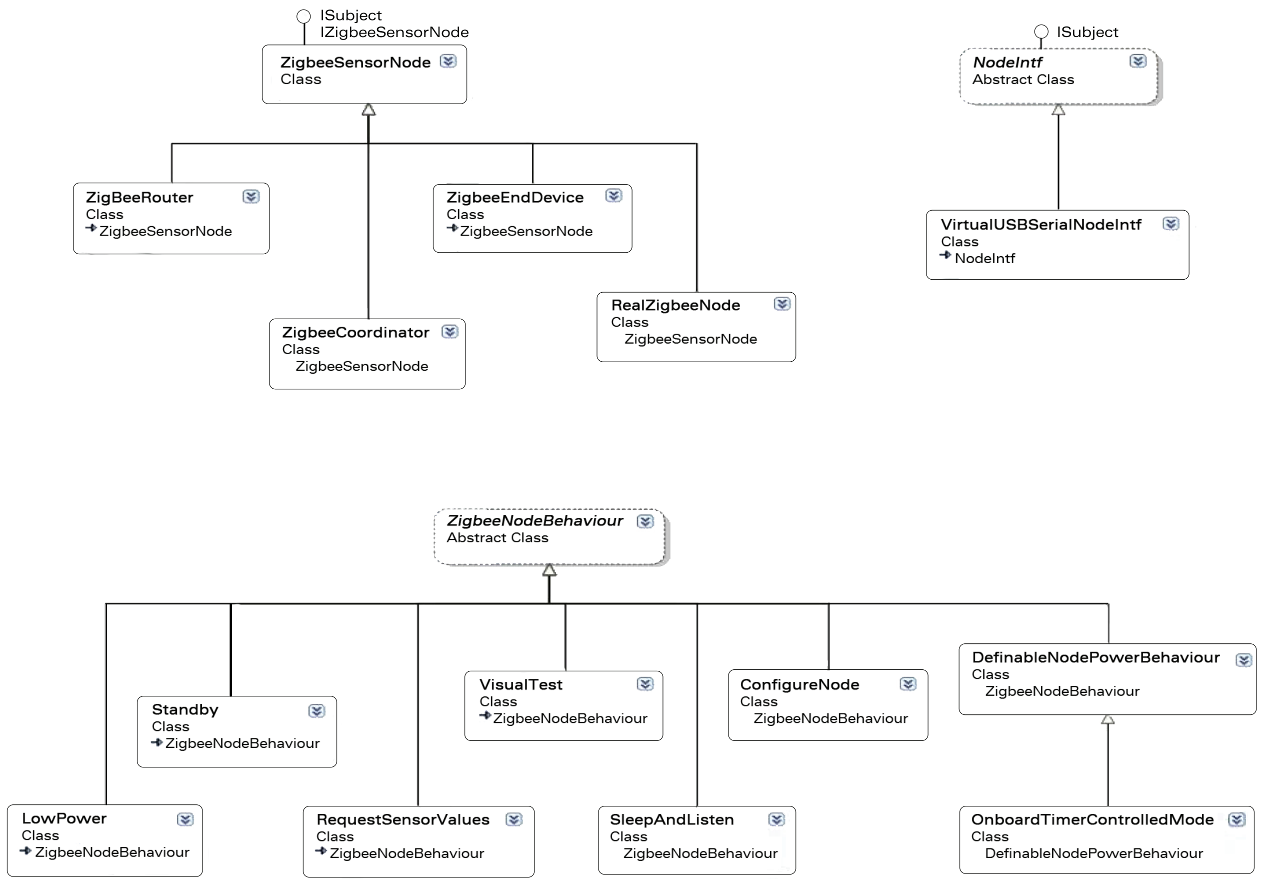


Fig. 6. Selected ZigBee network interface class hierarchies

pattern, the infrastructure layer employs further design patterns (Gamma *et al.* 1995), including: subject/observer, state, singleton, factory, proxy and smart pointer.

Due to the early development of the infrastructure layer, before development of the client (agent) layer and before any ontology development, some sensor and actuator hardware interface implementations use an XML configuration file. The situation allowed some easy immediate testing implemented in local procedures. While the configurations only contain a very minimal description of connected hardware, the information is replicated in the sensors ontology. Currently the sensor node agent is able to read the configuration from the sensor manager interface (hosted by the sensor node executable with which devices register) and partially verify consistency with the sensor ontology from that. The issue arises from the relatively simple XML configuration file content which is adequate to describe connected wired sensors, but is inadequate to fully describe the wireless sensor network, nor would the latter be desirable. Thus currently consistency between the XML files and the ontology has to be manually checked. A readily implemented solution is for any (trusted) client agent to write the configuration subset extracted from the sensors ontology to the infrastructure sensor node which would then update its XML based persistent configuration. For the same reason the infrastructure also contains some (class) modelling of sensor and actuator devices which creates a small degree of redundancy with the sensors ontology. The sensor and actuator classes however remain fairly abstract.

5.2. Tests for sensor node, digital input/output and thermometer modules

The unit and integration testing of the wired network supporting modules together with the sensor node executable, was completed using routine software engineering practices. The testing involved debugging software implementations employing the NHibernate object relational mapping libraries in conjunction with an SQL database, Microsoft .Net Remoting technologies, National Instruments USB driver libraries and a RS232 serial library.

Test cases were derived from the use cases for the system. After the initial debugging, the testing effort focussed on ensuring the delivery of good performance in terms of preserving all detected environment events while still delivering low processor usage. Where asynchronous notification of new data was not available, polling was required, but the overhead is very modest and as data through puts are also modest, no specific difficulties were encountered in that area. The implementation of pulse timing of the (wired) devices connected to the National Instruments interfaces, for example, was easily realised. That implementation includes "light weight" mechanisms to detect changes at a relatively fast polling rate (a 500 millisecond interval), and upon detecting changes, the interfaces are then queried to resolve those devices having new states and their associated values.

The initial testing revealed that the customised settings for the configuration of the .Net Remoting channels were adequate. Primarily those customisations relate to the "lifetime" specification of server side objects, typically activated as singletons that realise the primary interfaces.

Further testing relating to the sensor node executable revealed some degradation in update performance of an early implementation when tables grew to include a relatively large (>5k) number of entries. The sensor node design includes object-relational mapping (ORM) derived classes to implement the data histories and originally those objects were manipulated directly in synchronous client .Net Remoting associated threads. As a solution the sensor histories were buffered and the ORM objects synchronised with the database in a separate thread. A 3 hour buffer for historical data for each device allowed fast update from sensor interfaces and fast query from agents. In practice data is only rarely requested from outside that time interval, but for the servicing of requests where older data is required, some custom SQL statements were added within the NHibernate framework to further improve performance over the default (framework's) implementation.

5.3. Sensor role allocation

In order to improve the operation in terms of the effectiveness of (resource utilising) plans to deliver its designed result, the selection of sensor role allocation was re-evaluated, and extra selection criteria were added where possible. More specifically, where multiple leases are requested, which is typical, the ordering of those requested were reviewed to identify any benefits from early availability of specific device roles. For example, in the determine occupancy plan, the capture of the motion of persons moving away from zone entrances immediately following entry, can deliver early plan sub conclusions. In that example, such detection capability is delivered by motion sensors near boundaries. The order of lease request would not affect the immediacy of sensor availability but other factors can. However, the evaluation of preference can incur additional overhead. For example, wired device leases are always executed by the sensor node agent immediately, due to their typically always active configuration. The overall net benefit of added sensor selection criteria is therefore not clear without further investigation. Another example relates to wireless network devices. Those leases for sensors that are hosted by nodes already in a suitable configuration are advanced to the "granted" state almost immediately. In order to avoid the scenario where an agent may wait for a particular lease/role to become active instead of employing an alternative sensor in that role that would be ready almost immediately, it can use the existing lease query dialog to identify "ready" potential alternatives.

5.4. Tests for agents

The artefacts involved in testing of agents, moving from the narrowest scope to the widest were:

- Methods, typically implemented as common stateless methods manifested as static methods of "utility" classes for use by any agent type. They primarily realised miscellaneous functionality such as the custom object serialisation for use in a few messages (cf. the semantic language SL), sunset/sunrise time related functionality etc. Such functionality was easily tested using test "harnesses" for unit testing.
- Classes. The agents' plan implementations and common classes are implemented following the object oriented paradigm. Typical classes support IFC model interaction, sensor and building ontology manipulation and update, and the motion and entry exit tracker implementations. Again testing at this scope was easily completed with the creation of test harnesses. The testing of plans holistically is covered in the following scopes.
- Simple goal and corresponding single (candidate) plan implementation which can be triggered by the BDI architecture based mechanisms e.g. due to events (user defined and message events), and belief changes. Testing at this scope additionally includes plans that are triggered by a simple trigger match for sub goals dispatched in plan implementations. The motivation for implementation of the latter as goals cf. methods is the lifecycle control support by virtue of its hierarchy, as well as the other BDI manifestation "flags" that allow the specification of goal behaviour. Testing was typically completed by "hard coding" the dispatch of those goals to be tested after the creation of an appropriate context.
- Goal/plan implementations involving BDI manifestations that include (non simple) trigger and preconditions specifications in Java, belief state and belief change triggering, goal retry criteria, context and drop conditions, and the JADEX support for goal deliberation such as cardinality control and inhibit specification. Some of that testing required the hard coding of some of the conditions to create appropriate contexts while other scenarios were created with support from other assemblies. Examples are the sensor node agent type's management of its infrastructure connections as well as its management of sensor leases and ZigBee nodes.
- Goals involving more complex deliberation such as the zone agent type's evaluate occupancy high level goal. The test deployment at the domestic flat was a convenient environment for the purpose of initial testing, involving in some cases the hard coding of contexts and goal dispatch.
- Complete agent types, the primary types being the zone and sensor node agent types. The testing at this scope was completed in the same way as immediately above.

The software units mentioned above could typically be meaningfully tested using a single stepping debugger, unlike the more complex assemblies involving BDI manifested behaviour and asynchronous messaging. The assemblies were tested using scenarios derived from the agent responsibilities. The utility agent was also used to test modules of other agents' functionality before integration into the target agent type/s. One such test involved the evaluation of the zone agent type's lease management facility which was extended in later tests to include the subscription to sensors and the reading of values, incorporating the later integration testing of the infrastructure. The "hard wiring" during testing in order to create controlled contexts included the fixing of any deliberation to "force" the desired scenario (thus removing temporally some aspects of pro-activeness of the agent for the predictable and convenient activation of scenarios). Message exchange scenarios such as the request and reply of some agent attitudes including beliefs, e.g. zone characterisations, were tested in isolation before integration into assemblies. The details of selected tests for the sensor node agent type are shown in Table 6. The number of scenarios for each test (for the sensor node type) was

Table 6. Some sensor node agent type tests

Functionality	High level details	Test case/s – selected illustrative example/s	Result/see also
Manage leases, resolve supplier of resource (device, device cluster etc). efficient re-use of leases, modifying existing where feasible (eliminate unnecessary node reconfiguration)	Requests by client agents. The nature of requested the leases includes requests for new leases, those that can extend existing ones, and requests for unavailable devices	Activity log	Working as expected
Manage ZigBee nodes' power state, evaluate configuration, issues configuration commands, maintain nodes	Target node available Target node unavailable but becomes available (temporarily power off some nodes) Node becomes unavailable then available, hosting resources with active leases	As above Log showing leave state transitions As above	Working as expected Working as expected
Manage power states of sensors	As above	As above	Working as expected

less numerous in comparisons to the zone agent. Testing with the sensor node agent in the university deployment handled higher data throughput so that agent was used in order to derive conclusive results for tests.

The integration testing was performed from several formulations. Initially controlled testing took the form of "staging" scenarios where a person moved between different rooms with different building interactions e.g. unlocking a door, pausing before opening the door, activating a light switch to render a slow exit, perform an uninterrupted exit etc. Controlled behaviour varied from entering an office and taking different routes to desks/ seating causing the activation of different sensors. Additionally scenarios such as initiating internal movement while another person exited the room were tested. Permutations using various openings where they existed and activity were formulated and tested. Test were formulated on a "glass box" basis in order to identify worst case scenarios e.g. activity near an opening while a person entered or exited through that opening. In contrast uncontrolled test cases where the environment was observed and recorded were also carried out. Recording consisted of marking on paper the tracks of persons through the observed zones with approximate timestamps. Most effort to date has been on the former controlled test scenarios. In all cases the agent activity logs were inspected to determine the success.

Conclusions and discussion

This paper explains an integrated framework that demonstrates the use of semantic modelling, together with the application of the BDI model of agency and the implementation of an infrastructure incorporating sensor hardware that has enabled the aims of the system to be met. The upholding of rationality by the intelligent pro-active agents in the upper layer in a way that is transparent and explicit is a key feature. Additionally the solution needed to be practically executable and meet realistic performance constraints. Agents' behaviour is closely integrated with their beliefs and those beliefs include historical records about the outcomes of past behaviour (as well as others about the environment). Those beliefs, realising experience, contribute towards directing future behaviour. Specifically deliberation takes account of past behavioural outcomes so, for example, where options exist, earlier action that failed is not continually repeated. The application of inferences to support BDI agent behaviour is wide (for example, agent deliberation – goal feasibly, goal selection; means/end reasoning – sensor assignment, identification of sets of alternatives sensor roles and preference, configuration of hardware, control of hardware). The requirement to minimise resource utilisation adds significant complexity in terms of algorithmic plan implementation cf. always "on" data mining approach, but the application of intelligent management gives the advantage of more sustainable hardware units that are easily deployed. The system derives significant behaviour from executing reasoning with semantic knowledge but some behaviour remains captured implicitly in algorithmic implementations in plans.

Moreover, the semantics captured in the ontologies in the OntoFM ontology are shared and reused consistently both internally within agents and for well-defined communications between agents. Additionally explicit semantic definitions addresses one of the aims of the system, namely to facilitate well defined communication between agents and external tools. The knowledge can be readily consumed by tools in different disciplines and even at different lifecycle stages, where terminology and semantics could vary. Furthermore the ontological knowledge sources in OntoFM have been typically derived from existing published consensus of knowledge, ensuring high quality. The main resources used are the OntoSensor ontology (in turn is derived from the SensorML schema) which formed the basis of the sensors ontology, and the IFC schema inspired the building ontology. At a domain independent level, theories of mereology and topology have been incorporated into further smaller system ontologies for common usage. The formal KR additionally brings, as mentioned above, the benefit of consistency checking in the models, both at design time and in the dynamic assertion of individuals at run time.

A further area that could add extra flexibility to agents is the use of XQuery and XPath (Herman 2008) facilities applied to the dynamic analysis of ontologies. XQuery is a query language for XML while XPath is the syntax for specifying a path to a set of nodes in an XML tree structure. Therefore the facility could be usefully employed to query OWL ontologies where such functionality is not supported by SPARQL. A specific example for use in the OntoFM ontology could be for examining routes between zones when analysing the movement of people in buildings. Similarly in the analysis of the ZigBee network mesh, for example, counting "hops" between an end device and a controller would be a useful application. Another area of future work is to improve agent learning capability. It is expected that the main benefit would be in the enablement of further inferences by the ontology in contrast to the intrinsic informational value in the learned statements themselves. The creation of temporal relationships between ontologically described events that the agent generates is a starting point, but others may be relevant depending on the context. Another learning scenario is detecting changes in inference due to the addition of new individuals. So it may be the case, for example, that an ontology update triggers a more specific inference for a zone individual. A change listener could be configured, via the Jena API, to listen for all triples added or removed so this is one approach that could be used, with filtering for those related to individuals of interest.

Acknowledgements

The authors would like to thank the EPSRC-CASE PhD sponsorship.

References

Bergman, M. 2010. *Listing of 185 ontology building tools. Adaptive information, adaptive innovation, adaptive infrastructure* [online], [cited 15 May 2012]. Available from Internet:
http://www.mkbergman.com/904/listing-of-185-ontology-building-tools

Bilek, J.; Hartmann, D. 2006. Agent based collaborative framework for concurrent structural design processes, in *Joint International Conference on Computing and Decision Making in Civil and Building Engineering*, 14–16 June 2006, Montréal, Canada, 918–929.

Borst, W. N. 1997. *Construction of engineering ontologies for knowledge sharing and reuse*. Holland: University of Twente. 227 p.

Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J. 2004. Tropos: an agent-oriented software development methodology, *Autonomous Agents and Multi-Agent Sytems* 8(3): 203–236.
http://dx.doi.org/10.1023/B:AGNT.0000018806.20944.ef

Crawley, D.; Hand, J.; Kummert, M.; Griffith, B. 2008. Contrasting the capabilities of building energy performance simulation programs, *Building and Environment* 43(4): 661–673. http://dx.doi.org/10.1016/j.buildenv.2006.10.027

Dibley, M.; Li, H.; Rezgui, Y.; Miles, J. 2012. An ontology framework for intelligent sensor-based building monitoring, *Automation in Construction* 28: 1–14.
http://dx.doi.org/10.1016/j.autcon.2012.05.018

Fernandez-Lopez, M.; Gomez-Perez, A.; Juristo, N. 1997. METHONTOLOGY: from ontological art towards ontological engineering, in *Proceedings of the AAAI97 Spring Symposium*, 24–26 March 1997, Stanford, USA, 33–40.

Fernández López, M. 1999. Overview of methodologies for building ontologies, in *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5)*, 1991, Stockholm, Sweden, 4.1–4.13.

Fielding, J.; Simon, J.; Ceusters, W.; Smith, B. 2004. Ontological theory for ontological engineering: biomedical systems information integration, in *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, 2004, Whistler, BC, USA, 114–120.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. 1995. *Design patterns: Elements of reusable object-oriented software.* Reading, MA: Addison-Wesley. 416 p.

Gómez-Pérez, A.; Fernandez-Lopez, M.; Corcho, O. 2004. *Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic web*. Springer. 420 p.

Gruber, T.; Olsen, G. 1994. An ontology for engineering mathematics, in *Fourth International Conference on Principles of Knowledge Representation and Reasoning*, 1994, Gustav Stresemann Institut, Bonn, Germany. Morgan Kaufmann. 18 p.

Grüninger, M.; Fox, M. 1995. Methodology for the design and evaluation of ontologies, in *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing held in conjunction with IJCAI-95*, 1995, Montreal, Canada. 10 p.

Guarino, N. 2006. Ontology and terminology – how can formal ontology help concept modeling and terminology?, in *EAFT-NordTerm Workshop on Terminology, Concept Modeling and Ontology*, 2006, Vaasa, Italy. 12 p.

Guarino, N.; Welty, C. 2009. An overview of OntoClean, in *Handbook on ontologies. international handbooks on information systems.*Verlag, Springer, 201–220.

Li, H.; Rezgui, Y.; Miles, J.; Wilson, I. 2010. Low carbon ontology development using information retrieval techniques, in *Proceedings of the 8th European Conference on Product & Process Modelling*, 14–16 September 2010, Cork, Ireland, 215–221.

Hamscher, V.; Schwiegelshohn, U.; Streit, A.; Yahyapour, R. 2000. Evaluation of job-scheduling strategies for grid computing, *Grid Computing–GRID*, 191–202.

Herman, I. 2008. *W3C Semantic Web activity*, *World Wide Web Consortium*. World Wide Web Consortium [online], [cited 12 Oct 2012]. Available from Internet:
http://www.w3.org/standards/semanticweb/.

Iglesias, C.; Garijo, M.; Centeno-Gonzalez, J. 1999. A survey of agent-oriented methodologies, in *Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*. London: Springer-Verlag. 14 p.

Jacobson, I.; Booch, G.; Rumbaugh, J. 1999. *The unified software development process*. Addison-Wesley Professional. 512 p.

Kalyanpur, A.; Pastor, D.; Battle, S.; Padget, J. 2004. Automatic mapping of OWL ontologies into Java, in *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, 2004, Banff, Alberta, Canada. 8 p.

Luo, R. C.; Lin, S.; Su, K. 2003. A multiagent multisensor based security system for intelligent building, in *Proceedings of the IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems*, 30 July – 1 August 2003, Tokyo, Japan, 311–316.
http://dx.doi.org/10.1109/MFI-2003.2003.1232676

Maulo, F. 2006. NHibernate for .NET [online], [cited 11 Oct 2012]. Available from Internet:
https://community.jboss.org/wiki/NHibernateForNET?_sscc=t

Nikraz, M.; Caire, G.; Bahri, P. 2006. *A methodology for the analysis and design of multi-agent systems using Java Agent Development Framework (JADE)*. Turin: Telecom Italia Lab. 40 p.

Pan, X.; Han, C.; Law, K. 2005. A multi-agent based simulation framework for the study of human and social behavior in egress analysis, in *Proceedings of the 2005 International Conference – Computing in Civil Engineering*, 2005, Cancun, Mexico, 1–12.

Pease, A. 2008. *The suggested upper merged ontology (SUMO) – Ontology portal* [online], [cited 22 Oct 2012]. Available from Internet: http://www.ontologyportal.org

Preece, A.; Gomez, M.; Mel, G.; Vasconcelos, W.; Sleeman, D.; Colley, S.; Porta, T. 2007. An ontology-based approach to sensor-mission assignment, in *Proceedings of the Annual Conference of Interntional Technology Alliance*, 2007, Maryland, USA. 16 p.

Rao, A. S.; Georgeff, M. 1995. Belief, desire, intention (BDI) agents: from theory to practice, in *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, 1995, San Francisco, USA, 312–319.

Ren, Z.; Anumba, C. 2004. Reveiw: multi-agent systems in construction – state of the art and prospects, *Automation in Construction* 13: 421–434.
http://dx.doi.org/10.1016/j.autcon.2003.12.002

Rueppel, U.; Lange, M. 2006. An integrative process model for cooperation, *Journal of Information Technology in Construction* 11 (Special Issue: Process Modelling, Process Management and Collaboration): 509–528.

Russomanno, D. J.; Kothari, C.; Thomas, O. 2005. Building a sensor ontology: A practical approach leveraging ISO and Open Geospatial Consortium (OGC) models, in *The 2005 International Conference on Artificial Intelligence*, 2005, Las Vegas, Nevada, USA. 7 p.

Schenck, D. A.; Wilson, P. 1994. *Information modeling the EXPRESS way*. Oxford University Press, USA. 416 p.

Schreiber, G.; Wielinga, B.; Hoog, R.; Akkermans, H.; Velde, W. 1994. CommonKADS: a comprehensive methodology for

KBS development, *Intelligent Systems and Their Applications* 9(6): 28–37.

Sure, Y.; Staab, S.; Studer, R. 2004. On-to-knowledge methodology (OTKM), in *Handbook on ontologies, international handbooks on information systems*. Ontoprise Gmbh. 17 p.

Swartout, B.; Ramesh, P.; Knight, K.; Russ, T. 1997. Toward distributed use of large-scale ontologies, in *AAAI Technical Report* SS-97-06, 138–148.

Szekely, B.; Betz, J. 2009. *Jastor – typesafe, ontology driven RDF access from Java*. [online], [cited 11 Oct 2012]. Available from Internet: jastor.sourceforge.net

Tulke, J.; Tauscher, E. 2009. *Open industry foundation classes (IFC) tools* [online], [cited 11 Oct 2012]. Available from Internet: http://www.openifctools.org/Open_IFC_Tools/Home.html

Uschold, M.; King, M. 1995. Towards a methodology for building ontologies, in *Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995, Montreal, Quebec, Canada. 13 p.

Winikoff, M.; Padgham, L. 2004. The Prometheus methodology, in *Methodologies and software engineering for agent system. multiagent systems, artificial societies, and simulated organizations*, Vol. 11. Springer, 217–234.

Wood, M. F.; DeLoach, S. 2000. An overview of the multiagent systems engineering methodology, in *Agent-Oriented Software Engineering, Lecture Notes in Computer Science*, Vol. 1957, 2001. Berlin: Springer Verlag, 207–221.

Wooldridge, M.; Jennings, N.; Kinny, D. 2000. The Gaia methodology for agent-oriented analysis and design, *Autonomous Agents and Multi-Agent Systems* 3(3): 285–312. http://dx.doi.org/10.1023/A:1010071910869

Rezgui, Y.; Miles, J. 2011. *Harvesting and managing knowledge in construction: from theoretical foundations to business applications*. London: Spon Press. 232 p.

Zhang, C.; Hammad, A.; Bahnassi, H. 2009. Collaborative multi-agent systems for construction, *Journal of Information Technology in Construction* 14: 204–228.

Michael DIBLEY. Dr, has recently completed his PhD (An Intelligent System for Facility Management, 2011) in Cardiff University, Engineering School. He now works in BRE Institute of Sustainable Engineering, Engineering School, Cardiff University as a Research Associate. His research interest relates to sensor development, sensor based building monitoring, ontology development, and multi-agent system.

Haijiang LI. Dr, a Lecturer (Engineering Informatics & Structural Applications) working in the BRE Institute of Sustainable Engineering, Engineering School, Cardiff University. His main research interest lies on Advanced and Innovative Computing supported large scale systems integration and decision making. It covers Building Information Modelling (BIM) based AEC systems integration, High Performance Computing & Cloud Computing, ontology development and multi-agent systems, intelligent building monitoring sensor system etc., which have been applied into large scale infrastructure modelling, sustainable design and development, adaptable buildings, robust & resilient building design, life cycle infrastructure integration, knowledge management, intelligent computing for building energy simulation, virtual reality and visualization.

Yacine REZGUI. Prof., the Director of the BRE institute of sustainable engineering. He has successfully completed over 20 national (TSB/EPSRC, WAG) and European projects. He is currently involved in the FP7-2011-NMP-ENV-ENERGY-ICT-EeB KnowholEM project (285229) which delivers optimized ontology-based building energy management solutions. His expertise includes ontology engineering, service-based computing, multi-agent systems, and building energy. He has published extensively in the above areas, including a newly edited book (Rezgui and Miles, 2011 – Spon).

John MILES. Is a Professor in Engineering Informatics. He was the former institute head, and his expertise lies on conceptual engineering design, engineering optimization.