A Semantic Middleware Architecture for Supporting Real *Smartness*

Maria J. Santofimia*, David Villa*, Felix J. Villanueva*, Soledad Escolar†, and Juan Carlos Lopez*
*School of Computing Science, University of Castilla-La Mancha, Ciudad Real, Spain.
†Institute of Technology and Information Systems, Ciudad Real, Spain.
Email: {mariajose.santofimia, david.villa, felixjesus.villanueva, soledad.escolar, juancarlos.lopez}@uclm.es

***Abstract*—Smart environments, enabled by the Internet of Thing (IoT) paradigm, advocate for more intelligent and interconnected systems, electronic devices, tools, and appliances. While most efforts are nowadays addressed to provide connectivity or *smartness* to IoT devices, unfortunately, few have realised the importance of supporting automatic service composition and service reconfiguration capabilities at middleware level. Due to the openness that characterise such environments, the range of possible interactions and available services and devices cannot be totally defined nor prescribed in advanced. This uncertainty demands mechanisms to dynamically adapt existing systems, and their functionality, to address unforeseen needs. In order to do so, a general understanding of contexts, services, and device capabilities is needed. From that understanding, new responses can be automatically devised on run-time. This paper presents a semantic middleware specifically devised to support automatic and autonomous service reconfiguration and composition. The novelty consists in moving the semantics, traditionally held at the programming-interface level, to a common-sense knowledge-base system, with higher expressive power and reasoning capabilities.**

I. Introduction

The middleware technology gained more importance as applications were involving more distributed and heterogeneous devices and services. The middleware layer was originally devised to provide an abstraction layer, with mechanisms and tools, that simplified the way how these elements connect and communicate. The advent of paradigms such as the Internet of Things (IoT) or Internet of Everything (IoE) brought about new challenges that extended those previously faced by traditional middleware technologies [1] demanding appropriate responses.

The IoT faces two important challenges: on the one hand, enabling seamless communication and interconnection of devices, services, or object in general and, on the other hand, to digest and understand the vast amount of unstructured data generated by these networks. Interoperability support is grounded in the IoT protocol stack but, eventually, depends on the rules that determine how devices or services might communicate. Similarly, the data gathered by IoT devices need to be semantically enriched in order to extract structured knowledge from them.

The work in [2] summarises the most relevant approaches that, to the date, have been proposed for information retrieval in IoT, claiming the need to provide *semantic awareness* [3] in the IoT. Sowa proposed in [4] analysing the notion of context from a triple dimension involving its syntax, semantics, and pragmatics. The majority of the work founds in the literature only concerns one of such dimension, leading to partial meaning being captured. In this sense it is common to find semantic middlewares that simply consider an ontology [5] in the sense of a taxonomy. Some other works like [6] extend that taxonomy to consider relationships, as a proper ontology, using to this end description logics. However, their expressive power do not exceed that of first-order logic. The work in [7] provides a complete list of state-of-the-art approaches for semantic normalisation along with a new proposal. These methods succeed in addressing the semantic dimension but overlooked the syntax and pragmatic ones.

Providing semantics to IoT objects and data should therefore be addressed from the same triple dimension identified by Sowa and applied to smart spaces in [8]. In the application of this triple perspective, the use of common-sense knowledge is essential. Minsky [9] considered that the lack of common sense is responsible for the deficiencies of current computational systems. In this sense, he criticised three aspects of current programs. The first issue is the lack of common-sense knowledge and the skills required to use such knowledge. For example, from the fact that a parcel is tied up with a string, there are many *obvious* facts that are straightaway associated with it. The string can be used to pull the parcel but not to pull it, or the fact that if you push it too hard you might break it. These are some of the facts that despite being obvious for humans are unknown by computational systems, unless explicitly told about. The second issue refers to the lack of explicit goals, in the sense that systems are told *what* to do, instead of *why* doing so. This ignorance causes systems to fail in their task when something goes wrong, since no reasons have been provided that help reconfiguring the task. For example, "people like to go indoors when it rains", this fact tell us the *what*, and the *why* is provided by the fact that "people do not like to get wet". Knowing that fact makes it possible to fulfil the ultimate goal of preventing people from getting wet, for example by using an umbrella, if going indoors is not a possibility. The third issue is related to resourcefulness and how people use common sense to make analogies when some required knowledge is missing.

Eventually, a semantic middleware is intended to provide the means to support context understanding and automatic response elaboration. In this sense, a twofold aim is pursued. On the one hand, the middleware framework should

encompass the required knowledge to identify the undergoing situation. On the other hand, it should also be aware of the device and service capabilities that support the acting skills. A semantic middleware should therefore be expected to support the automatic implementation of responses, by reconfiguring and combining existing system capabilities. To this end, the middleware framework should count on the appropriate machinery to translate events into actions effects and to understand available devices and services as middleware capabilities. Based on these requirements, this work describes the foundations supporting Dharma, a semantic middleware for smart spaces. The paper is organised as follows. Section II describes the proposed approach for supporting the description of the three dimensions proposed by Sowa. Section III describes the different modules comprising the proposed architecture. Finally IV presents the most relevant conclusions of this work.

II. A THREE-LEVEL PERSPECTIVE

Approaches such as the Context Modelling Language (CML) [10] claims to provide support for reasoning through a database modelling technique that captures the context syntax and semantics. However, reasoning capabilities are here limited to answering SQL-like queries. In this sense, inferences or deductions are therefore limited to the information explicitly stated by means of rules.

Although aimed at the field of meaning in natural language, the theory of situation semantics, proposed by Barwise and Perry [11] has been extrapolated to context-awareness. However, as stated in [4], situations cannot be completely described by propositionally enumerating all the aspects involved in the situation since aspects such as intuitions about context escape this modeling strategy. Sowa also proposes his own theory [12] for context modeling, based on conceptual graphs of semantic networks. Under this theory, contexts are modeled as propositional containers of additional conceptual graphs.

Based on the three-dimensional view proposed by Sowa and given the limitations of previous approaches in which only one perspective was considered at a time, we propose a semantic middleware for IoT that comprehensively addresses the aforementioned dimensions. The following subsections describe the information modelled at each level and the mechanisms employed to it. Moreover, we will indicate how the information handled at each level is made available to the semantic middleware.

A. The syntax level

This dimension concerns the lexicon shared between the middleware and the context, systems, and services it interacts with. Several domains are therefore considered:

- Actions and events: this domain includes human actions that can be captured by the effects they produce on sensing devices. Additionally, it also considers the actions that can be performed by electronic devices, such as the actuator that turns on a light. WordNet [13] provides a hierarchical structure for actions. This Knowledge-Base will be used as a source for the employed lexicon.
- Internet of Things objects: this domain describes the taxonomy of electronic devices, message types, objects, etc., considered by the IoT paradigm. The work in [14] provides a full description of IoT components (smart devices, persistent nodes, actors, etc.). The identified entities will be incorporated here as part of the lexicon.

The syntax level basically consists in enumerating the vocabulary that will be employed by the semantic middleware when interacting with external elements. Additionally, basic relations such as IS-A or HAS-A will be also considered at this level.

These knowledge will be held in a Knowledge-Base (KB) system with support for efficient search and retrieval operations. We propose the use of Scone[1], an open-source KB system written in Common Lisp. The main difference with respect to other approaches consists in the way search and inference operations are implemented. Scone adopts an efficient marker-passing algorithm [15] that support most of the search and inference works involved in common-sense reasoning: inheritance of properties, roles, and relations in a multiple-inheritance type hierarchy; default reasoning with exceptions; detecting type violations; search based on set intersection; and maintaining multiple, overlapping world-views at one time in the same knowledge base.

The considered vocabulary, along with its primary relationships, is therefore modelled using the Scone language and asserted to its knowledge base.

B. The semantic level

At the semantic level, the taxonomy proposed at the syntax level is going to be enriched with common-sense knowledge. To this end, an expressive-enough language is required that, at the same time, does not fail to be computationally efficient. The Scone system pays special attention at providing an expressive, easy to use, scalable and efficient approach for accomplishing search and inference operations.

The semantic model considered here was proposed and formalised in [16] and can be summarised as follows: devices provide services that perform actions on objects. Additionally, events take place in a context and involve the actions whose effects are perceived. This semantic model is described in the Scone KB and, additionally, provided as the middleware programming interface, as described later on this work.

C. The pragmatic level

Both semantic and pragmatic level concerns about meaning, however, the semantic level assumes that there exists a precise meaning for every concept, while the pragmatic one goes one step further and considers how that meaning may vary depending on the surrounded circumstances [17].

The work in [18] describes how Scone is used to capture the context pragmatics using its *multiple-context* mechanism. Here, this mechanism is adopted to model actions and events in terms of the world states before and after their occurrence. This mechanism plays an essential role in enabling

[1] https://github.com/sfahlman/scone

reconfiguration and composition capabilities in the proposed middleware architecture. Hence, when there is not a basic service for accomplishing a certain action, alternatives are explored to select those that produce the same world state after they take place. Some adjustment works might be required (reconfiguration) or more than one service might be involved in producing the desired world state. The process how the service space is explored is described underneath.

III. The semantic middleware architecture

The motivation behind the middleware architecture proposed here is to enable system capabilities to react to unforeseen situations on the basis of available resources. The everyday world can be seen as an open world in which different sources of change are concurrently taking place. This *openness* makes it difficult to characterise, in advance, the possible changes that might take place. The infinite list of possible states that the system might find itself makes it unfeasible to address it as though it would be a state planning problem. This limitation calls for more intelligent mechanisms that replicate human ability to tackle unforeseen situations. For example, when we devise clever uses of things we have at hand to overcome the shortage of a certain item (e.g., switch on your cell phone during a blackout to illuminate your surroundings).

Leveraging system capabilities to articulate automatic and unsupervised responses is a task that should address several challenges. On the one hand, available means or devices cannot be foreseen in advance. Then, predefined system responses cannot be elaborated in terms of available actuators because they might or might not be available when required.

On the other hand, more than one service might be involved in the devised response. Then, it should be stated how these services are to be bound or simultaneously executed. This task regards the service composition process in contrast to service combination. Combining music, for example, differs from composing it, due to the required understanding of additional aspects such as harmony, rhythm, chords. Similarly, composing services requires that understanding of what the service does and how, whereas combining basically involves service input/out matching. Most common approaches found in the state of the art claim to perform service composition while simply combining services. Nonetheless, composition involves combination, but it is not restricted to it. To date, among all those that claim to perform service composition few are really addressing the key challenges that are involved in composition:

- **Basic services:** the composition process involves basic services as its raw elements. A complete and updated list of available services should be held for the composition task.
- **Automatic binding:** different technologies, protocols, or service features, make the service binding process unfeasible without automatically abstracting those issues from the composition task.

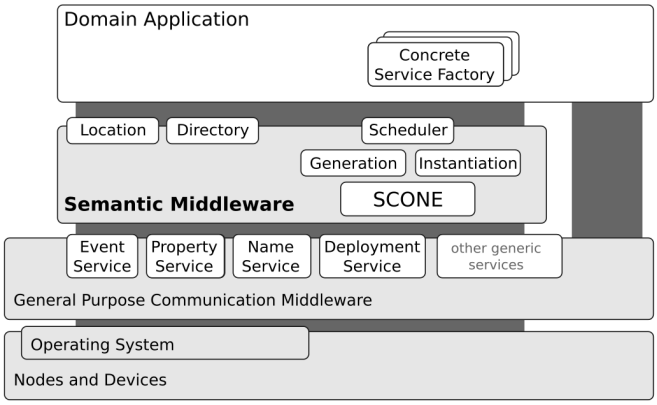

Fig. 1. Overall view of the different modules involved in the proposed middleware architecture

- **Uniform treatment of services:** despite their heterogeneity, all services have to provide the same interface, as if they were being provided by the same service source.
- **External automatic service adaptation:** at some point it can be necessary to carry out an adaptation process during the composition, since external source services may differ in the interaction models and protocols. It is necessary to mask out these differences by means of translation and interface masks.

The middleware architecture proposed here addresses these requirements with a set of modules, as depicted in Figure 1. The proposed architecture is based on a general purpose communication middleware as it is ZeroC Ice[2]. ZeroC ICE is an object-oriented and CORBA-like middleware technology that provides the mechanisms (tools, API, libraries) to easily build object-oriented client-server applications. This layer rests on the operating system or firmware of the devices and nodes. Finally, at the domain application layer, concrete service factories might be available for deploying basic or composite services. Additionally, a well-known API will be available for external developers to use the different services provided at the semantic middleware layer.

The following subsections describe the details of the different modules comprising the proposed architecture. Additionally, Figure 2 depicts these modules from the point of view of their interrelations and the functionality they provide when a `lookup()` request is launched. The Scone KB, at the core of the diagram, holds the common-sense knowledge that describes *how the world works*, including information such as that a typical human being has two arms and two legs. It also holds domain specific knowledge comprising information about concrete physical spaces, such as a specific building, with a complete description of the different rooms, floors, etc. Domain specific knowledge also encompasses the different devices or IoT objects that are available at that specific domain, and the services they provide. This knowledge will support the

[2]https://zeroc.com/

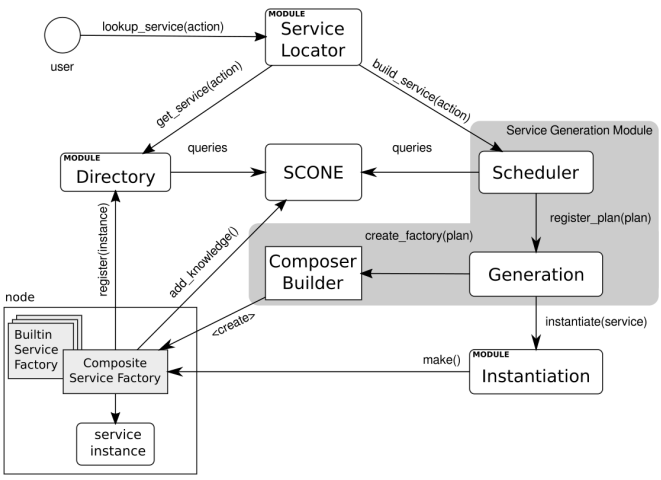

Fig. 2. Roles involved in a service-lookup operation

automatic service composition and reconfiguration as a mean to overcome the need to face unexpected needs.

A. Service deployment module

Several stages are involved in the deployment process:

- If the device has not been previously asserted to the KB, it has to be described in terms of *what it is* and *what it does*. For example, a lighting device:

```
(new-type {light} {thing})
(new-type {emitter} {thing})
(new-type {light-emitter} {dharma-device})
(new-is-a {light-emitter} {emitter})

(new-relation {emits} :a-inst-of {emitter} :b-inst-of {thing})
(new-statement {light-emitter} {emits} {light})
```

  Recall that, at the syntax level, a taxonomy of IoT devices was employed to determine the terms that are to be employed during the service deployment. In this sense, developers will be assisted with this list of device terms. All the devices available at the semantic middleware layer are labelled as `dharma-device`.
- Once the provider device has been described and asserted to the Scone KB, the next step consists in describing the services this device provides. For example, two services can be offered by the lighting device, as they are *light-up* and *light-down*.

```
(new-type {light-service} {dharma-service})

(x-is-the-y-of-z {light-up} {performs-action} {light-service})
(x-is-the-y-of-z {light-down} {performs-action} {light-service})
```

  The naming policy used for the actions performed by services follows the taxonomy of actions and events extracted from WordNet. Once again, meeting the vocabulary agreed during the modelling stage will assure compatibility with applications implemented by different developers, in a *plug-and-play* manner.
- Describe the actions that can be carried out by the offered services in terms of relevant elements and states of the world involved (*before* and *after contexts*). Please refer to [16] for further details on modeling actions and events using Scone). Action roles (e.g.: `action-type`, `object-type`, or `recipient-type`) symbolise those domain elements that characterize the world states.

```
(new-action-type {light-up}
    :agent-type {thing}
    :object-type {light}
    :recipient-type {bulb})

;; Before context
(new-context {light-up BC} {dharma})
(new-is-a {light-up BC} {before context})
(x-is-the-y-of-z {light-up BC} {before context} {light-up})

(in-context {light-up BC})
(new-is-not-a {bulb-instance} {light-emitter})
(new-not-statement {bulb-instance} {emits} {light})

;; After context
(in-context {dharma})
(new-context {light-up AC} {dharma})
(new-is-a {light-up AC} {after context})
(x-is-the-y-of-z {light-up AC} {after context} {light-up})

(in-context {light-up AC})
(new-is-a {bulb-instance} {light-emitter})
(new-statement {bulb-instance} {emits} {light})
```

These steps will have to be carried out each time a new device is to be deployed. However, once that this knowledge has been provided to the Scone KB no further assertions are required apart from providing the service endpoint. The `$proxy` argument corresponds to the service endpoint.

```
(new-indv {light-service-instance} {light-service})
(x-is-the-y-of-z {$proxy} {offered-service} {light-service-instance})
```

B. Service locator module

One of the main strengths of our semantic middleware consists in the orthogonal interface provided for searching and instantiating services. Independently of the action that needs to be executed and the properties that configure it, the service-locator module (SLM) will provide a common and orthogonal mechanism, as it is the `lookup()` method, as listed underneath.

```
interface ServiceLocator {
    Service* lookup(string action, PropertyService::ThingDict params);
};
```

The application invokes the `lookup()` method over the SLM. The `action` argument corresponds to the sought action (e.g.: `light-up`). Please note that the action argument should match one of the terms defined by the taxonomy. The `params` argument consists of a list of properties that should be met by the provided services (e.g., the location where the service should be deployed). These properties will be enumerated using a `key:value` template where the `key` will be elements of the taxonomy and `value` will be the requirements to be met.

```
service = SDM.lookup_service("light-up", {'location':thing.from_str("Room 1")})
```

This `lookup` query described above is searching for a service that implements the `light-up` action in `Room 1`.

C. Directory module

The SLM implements the facade pattern, providing a simple and unique way of looking for services that implement a given action, obviating the details regarding its existence and nature (basic or composite). Firstly, the SLM checks the Directory

Module (DM) for an available service providing the requested action. This would be the case of the `light-up` action as we deployed a `light-emitter` device offering a service implementing that action.

However, this is not always the case, nor the most powerful use of the semantic middleware. On the contrary, if there is not a service offering that action, the SLM will launch a query for building a service from available basic services, which will be the role of the service generation module.

D. Service generation module

The Service Generation Module (SGM) will be employed when a request for a given action cannot be satisfied with available services. Despite this failure, a combination of basic services or a reconfiguration of an existing one might end up solving the problem. To this end, the SGM implements a Hierarchical Task Networks (HTN) approach to guide the KB search over available services [19].

The actions that can be performed by a system are determined by the devices and services available at each moment in time. Those actions that cannot be performed, due to the lack of services that provide a specific functionality, are named here *non-feasible actions*. Whenever the system demands the execution of a non-feasible action, the planner comes into play.

As listed below, the `Planning` algorithm starts with an empty plan, the Π plan, to be completed with the list of actions, which at the same time are provided by services. This course of actions is intended to emulate the required non-feasible action. The course of actions is provided as a set of actions performed on objects A and O respectively, and the results R of accomplishing such actions.

Algorithm 1 Planning(Π, A, O, R)

1: $\pi = (A, O, R)$
2: **if** A is non-feasible **then**
3: get all the actions $A = (a_1, a_2, ..., a_n)$ that have the same result A
4: **while** a_i is non-feasible **do**
5: delete a_i from A
6: **end while**
7: **while** feasible action a_i does not have an equivalent target object **do**
8: list all the objects $Objects = (o_1, o_2, ..., o_n)$ of action a_i
9: check if those o_i are equivalent to or can be O
10: **end while**
11: Recursively call $\pi = Planning(a_i, o_i, resultOf\ a_i)$
12: **end if**
13: Add π to Π
14: Return Π

In line 3 of the planning algorithm those actions whose effects are equivalent to the targeted action are selected. Among the selected actions, lines 4-6 get rid of those that cannot be directly instantiated by any of the available services. Upon each of the *feasible actions*, lines 7-10 are devoted to identify the object on which the given action can be performed which, at the same time, is equivalent to the targeted object. Finally, line 11 recursively instantiates the planning algorithm in trying to fulfil the action requirements, by accommodating the context conditions to those in which the action can take place. Finally, only when all the actions that are part of the plan can be instantiated will the planning algorithm abandon the execution, giving as a result, the course of actions which makes up the plan.

The yielded plan is then registered and, implementing the builder pattern, the composite service is created. From the point of view of the client, there will be no difference in instantiating a composite or a basic service. Additionally, the SGM will request the service instantiation module to eventually instantiate the service.

E. Service instantiation module

Traditional middlewares expect developers implementing clients for a given service to know, beforehand, the member functions of the service interface. Invoking a method that is not provided by the interface would produce an error. On the contrary, the proposed middleware is based on premises such as "*uniform treatment of services*" and "*automatic binding*". These requirements are incompatible with different interfaces that should be known in advanced by the clients instantiating the service.

The strength of the proposed solution is that the basic services that will eventually comprise the composite service do not have to be known in advance. Traditional middlewares have overcome this limitation by constraining the basic services that can be involved in the composition. To support this contribution, services have to be orthogonally instantiated, meaning that every service will implement the `Service` interface. This interface provides the *performAction()* member function that guarantees that all the actions that can be carried out by a service are executed through this facade.

```
#include <PropertyService/PropertyService.ice>

module Semantic {
 interface EventSink {
 void report(string sourceid, string magnitude,
 PropertyService::Thing value, PropertyService::ThingDict metadata);
 };

 interface Device extends PropertyService::PropertySet {};

 interface DeviceFactory {
 Device* make(string role);
 };

 interface Service {
 void performAction(string action, PropertyService::ThingDict params, EventSink* callback);
 };


 interface SconeService {
 string sconeRequest(string request);
 };

 sequence<PropertyService::TypeCode> TypeCodeSeq;
 sequence<string> StringSeq;

 interface TypeMapper{
 TypeCodeSeq getTypes(StringSeq names);
 PropertyService::TypeCode getType(string name);
 };
};
```

Moreover, the service instantiation module (SIM) provides support for third-party services that do not implement the `Service` interface. These services have to go through a

dharmification process, consisting in developing a wrapper that translates dharma invocations to the appropriate method calls.

Finally, the SIM will create an instance of the composite service. This instance will be registered in the directory, along with the services previously registered in the system. This registration process will also imply an update of the KB system, to extend the offered functionality with the composite service.

Finally, the directory will keep track of the elaborated plans so that if the same or similar request is made, the computing time can be reduced by reusing the whole plan or a specific part of it.

IV. Conclusion

This paper presents a comprehensive solution for leveraging real *smartness* in smart environments. Similarly to human ability to overcome the lack of the *typical object* when performing a task, the proposed solution aims at enabling automatic service composition and service reconfigurability as the key capabilities for achiving the smart environment paradigm.

This paper describes a three-dimensional approach for capturing the semantics of IoT objects and contexts. The considered semantics go beyond the use of a taxonomy or ontology, to consider a mechanism that adapt the semantics to the considered contexts. These semantics will support context understanding which will eventually lead to the management of unexpected needs. Based on available services, the semantic middleware proposed here, is intended to devise a composite or reconfigured service that faces that need.

The different modules comprising the middleware architecture proposed here are outlined. Some implementation details are provided for the most important modules, as they are the Service Locator, Service Generation, and Service Instantiation modules.

Future work will extend the planner with projection and evaluation capabilities, so that plan failures can be foreseen before being instantiated. Additionally, validation tests need to be carried out to compare this proposal with state-of-the-art solutions.

Acknowledgment

This work has been funded by the Programme for Research and Innovation of University of Castilla-La Mancha, co-financed by the European Social Fund (Resolution of 25 August 2014) and by the Spanish Ministry of Economy and Competitiveness under project REBECCA (TEC2014-58036-C4-1-R) and by the Regional Government of Castilla-La Mancha under project SAND (PEII_2014_046_P).

References

[1] J. Rodríguez-Molina, J. F. Martínez, P. Castillejo, and R. D. Diego, "SMArc: a proposal for a smart, semantic middleware architecture focused on on Smart City energy management," *International Journal of Distributed Sensor Networks*, vol. Smart City (special issue), pp. 1–17, 2013.

[2] F. Zhao, Z. Sun, and H. Jin, "Topic-centric and semantic-aware retrieval system for internet of things," *Information Fusion*, vol. 23, pp. 33 – 42, 2015.

[3] M. Giannikos, K. Kokoli, N. Fotiou, G. F. Marias, and G. C. Polyzos, "Towards secure and context-aware information lookup for the internet of things," in *International Conference on Computing, Networking and Communications, ICNC 2013, San Diego, CA, USA, January 28-31, 2013*, 2013, pp. 632–636.

[4] J. F. Sowa, "Syntax, Semantics, and Pragmatics of Contexts," in *Proceedings of the Third International Conference on Conceptual Structures: Applications, Implementation and Theory*. London, UK: Springer-Verlag, 1995, pp. 1–15.

[5] V. Charpenay, S. Kbisch, D. Anicic, and H. Kosch, "An ontology design pattern for iot device tagging systems," in *Internet of Things (IOT), 2015 5th International Conference on the*, Oct 2015, pp. 138–145.

[6] R. de Diego, J.-F. Martnez, J. Rodrguez-Molina, and A. Cuerva, "A semantic middleware architecture focused on data and heterogeneity management within the smart grid," *Energies*, vol. 7, no. 9, p. 5953, 2014.

[7] S. Hasan and E. Curry, "Thingsonomy: Tackling variety in internet of things events," *IEEE Internet Computing*, vol. 19, no. 2, pp. 10–18, 2015.

[8] M. J. Santofimia, "Automatic service composition based on common-sense reasoning for ambient intelligence," Ph.D. dissertation, School of Computing Science. University of Castilla-La Mancha, 2011.

[9] M. Minsky, "The emotion machine: from pain to suffering," in *Creativity & Cognition*, 1999, pp. 7–13.

[10] K. Henricksen and J. Indulska, "Developing context-aware pervasive computing applications: Models and approach," *Pervasive Mobile Computing*, vol. 2, pp. 37–64, February 2006.

[11] J. Barwise and J. Perry, "Situations and attitudes," *Journal of Philosophy*, vol. 78, no. 11, pp. 668–691, 1981.

[12] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA.: Addison-Wesley, 1984.

[13] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995.

[14] L. Barker, M. White, M. Curran, Z. Patoli, B. Huggins, T. Pascu, and N. Beloff, "Taxonomy for Internet of Things - Tools for Monitoring Personal Effects," in *PECCS 2014 - Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems, Lisbon, Portugal, 7-9 January, 2014*, 2014, pp. 67–71.

[15] S. E. Fahlman, "Marker-Passing Inference in the Scone Knowledge-Base System," in *First International Conference on Knowledge Science, Engineering and Management (KSEM'06)*. Springer-Verlag (Lecture Notes in AI), 2006.

[16] M. J. Santofimia, S. E. Fahlman, X. del Toro, F. Moya, and J. C. Lopez, "A semantic model for actions and events in Ambient Intelligence," *Engineering Applications of Artificial Intelligence*, vol. In Press, Corrected Proof, pp. –, 2011.

[17] L. R. Horn and G. Ward, *The Handbook of Pragmatics (Blackwell Handbooks in Linguistics)*. Blackwell Publishers, 2004.

[18] M. J. Santofimia, S. E. Fahlman, F. Moya, and J. C. Lopez, "Possible-world and multiple-context semantics for common-sense action planning," in *Space, Time and Ambient Intelligence IJCAI 2011 Workshop Proceedings*, 2011, pp. 100–105.

[19] M. J. Santofimia, S. E. Fahlman, F. Moya, and J. C. Lopez, "A common-sense planning strategy for Ambient Intelligence," in *Proceedings of the 14th international conference on Knowledge-based and intelligent information and engineering systems: Part II*, ser. KES'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 193–202.