

# Extending Semantic Web tools for improving Smart Spaces interoperability and usability

Natalia Díaz Rodríguez and Johan Lilius and M.P. Cuéllar and Miguel Delgado Calvo-Flores

**Abstract** This paper explores the main challenges to be tackled for more accessible, and easy to use, Smart Spaces. We propose to use Semantic Web principles of interoperability and flexibility to build an end-user graphical model for rapid prototyping of Smart Spaces applications. This approach is implemented as a visual rule-based system that can be mapped into SPARQL queries. In addition, we add support to represent imprecise and fuzzy knowledge. Our approach is exemplified in the experimental section using a context-aware test-bed scenario.

**Key words:** Smart Space, Fuzzy Ontology, Interoperability, End-user Application Development

## 1 Introduction

Smart Spaces were built in relation with the vision in which computers work on behalf of users, they have more autonomy, and they are able to handle unanticipated situations. This implies the use of Artificial Intelligence (AI), agents and machine learning. One of the main goals of Smart Spaces is to achieve device interoperability through standard machine readable information to allow easy construction of mash-ups of heterogeneous applications. A *Smart Space* (SS) is an abstraction of physical ubiquitous space that allows devices to join and leave the space as well as sharing information. New devices are constantly being introduced, often involving different information formats or coming from diverse sources. Lack of interoperability be-

---

Natalia Díaz Rodríguez and Johan Lilius  
Turku Centre for Computer Science (TUUS), Department of Information Technologies, Åbo Akademi University, Turku, Finland. e-mail: {ndiaz, jolilius}@abo.fi and M.P. Cuéllar and Miguel Delgado Calvo-Flores  
Department of Computer Science and Artificial Intelligence, University of Granada, Spain. e-mail: {manupc, mdelgado}@decsai.ugr.es

tween systems and devices easily becomes a problem, resulting in them not being used efficiently. In order to have devices that interoperate seamlessly, their respective data and functionality must be easily integrated and accessible.

SSs are considered to be context-aware systems; therefore, a key requirement for realizing them, is to give computers the ability to understand their situational conditions [4]. To achieve this, contextual information should be represented in adequate ways for machine processing and reasoning. Semantic technologies suite well this purpose because ontologies allow, independently, to share knowledge, minimizing redundancy. Furthermore, semantic languages can act as metalanguages to define other special purpose languages (e.g., policy languages), which is a key advantage for better interoperability than that one of tools that share no roots of constructs.

We focus on providing the end-user the possibility of rapidly prototype the behaviour of a SS, abstracting away technical details. When presenting a semantic SS to the user, the interface model, even if simplified, must adhere to the Semantic Web (SW) formal models. Domain Specific Languages (DSL) demonstrate support in this abstraction for integration of metamodels using ontologies (e.g. in [16]). Concerning end-user frameworks and GUIs for developing SS applications, we can find works that simplify the tasks to the user when creating their own services, through simple rules.

The survey of programming environments for novice programmers [9] shows how to lower the barriers to programming, which is one of our main aims; let non expert users to take part in the configuration of a Smart Space. Some good examples of end-user visual editors that simplify the tasks to the user when creating their own services or applications, through simple rules, are *If This Then That*<sup>1</sup> for online social services, *Twine*<sup>2</sup> for applications based on sensor interaction or Valpas [12] intelligent environment for assisted living.

A graphical tool to prototype context-aware applications is *iCap* [14], where coding is not required, but rather using window controls and IF-THEN rules. Their public is developers that want to rapidly test and iterate ubicomp applications. Other work in this line is a reconfiguration framework focused on tackling system variability and policy definition in runtime [6]. They use *PervML* as DSL, Feature Modelling techniques and Model Driven Engineering (MDE) principles such as code generation. End-user interaction evaluation is pending.

In [17] a rule-based framework with puzzle-like pieces allows to specify domain-specific situations with service invocations and state changes. Its use in a PDA showed quicker progress in users with less provided help. In *SiteView* [1], a tangible interface allows to control simple conjunctive rules of a "world-in-miniature" to help users create and view the effects of the rules.

A more complex system, aimed at end-users, is [15]. It is based on a pervasive navigation environment which uses spatial and resource annotations from the users' pictures. Pervasive Maps allow to model, explore and interact with complex pervasive environments.

---

<sup>1</sup> <http://ifttt.com>

<sup>2</sup> <http://supermechanical.com/twine/>

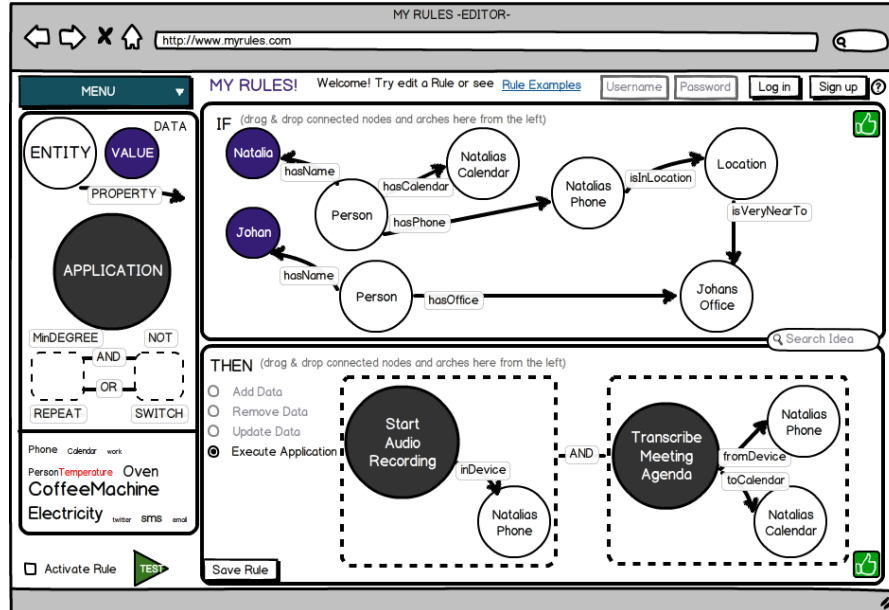




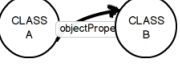
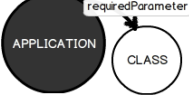


Fig. 1 User interface mock-up and example of semantic rule construction.

In [5], an ECA (Event-Condition-Action) rule system allows end-users to control and program their SS in a drag & drop environment with wildcards filters and textual expressions, allowing in this way complex rules to be formulated. Finally, the dataflow rule language in [3] shows increased expressiveness in young non-programmers. However, previous frameworks lack underlying semantic capabilities and support for fuzzy rule expressions through linguistic labels. There does not exist a GUI for visualizing both fuzzy ontologies and fuzzy rules. Thus, our proposal takes the interaction with context-aware Smart Spaces to a higher level, allowing easier prototyping of the SS's behaviour by a) providing semantics to enhance the context-awareness, and b) permitting imprecise every-day life expressions. Next section elaborates more on our proposal and Section 3 discusses the approach and suggests future directions.

## 2 A framework for rapid application development of context-aware Smart Spaces

Our contribution consist of a graphical interface for representing, visualizing and interacting with SSs information. It allows any end-user to model his own applications without knowledge of programming. Data gathering is possible by aggregation of different ontologies and datasets. The interface is based on simple IF-THEN rules

OWL2	Appearance	OWL2	Appearance	OWL2	Appearance
Class		Data type		Individual	
Data Property		Object Property		Application/Service	

**Table 1** Graph visual model representation mapping to OWL2

applied to graph-based data. A node can be of two types, representing an OWL class (*Entity*, large and white) or a data property value (small and purple). An arc can represent a data property or object property, depending on the type of the destination node ( $destNode=\{Class\ or\ Value\}$ ). The THEN clause of the rule serves to 1) add, remove or update information in form of arcs and nodes (representing RDF triples) from the knowledge base, or 2) execute a registered browser-based application (with associated service grounding), possibly using concrete and well defined individuals or properties described in the IF clause or Linked Data. Registered web or Linked Data services are represented in large grey nodes. Subgraphs in IF and THEN clauses can be connected with logical operators and included into loops expressed in the rule's consequent. A minimum degree of satisfiability can be expressed for a determined subgraph, since a rule can be mapped to a Mamdani rule in a fuzzy reasoner (e.g. *fuzzyDL* [2]). Fuzzy modifiers are considered in the same way as crisp properties (e.g. *isVeryNearTo* in Fig. 1).

The graph-based and "puzzle"-like pieces to edit rules with take inspiration from the successful *Scratch* framework [11]. Variable bindings are correct, by construction of the user interface, through letting the user allocate pieces only in the positions in which corresponding data ranges and domains are allowed.

This intuitive way of expressing a rule's condition, by dragging and joining compatible (data type-wise) nodes and arcs, can be easily translated into SPARQL query patterns (e.g., conditions in the *WHERE* field) and allow fast formulation of mash-up applications. Table 1 summarizes the mapping applied to transform end-user visual model representations into OWL2 entities.

A great power of visual languages is their ability of categorizations of certain primitives, and the graphical properties, to carry semantic information. Furthermore, elements of their syntax can intrinsically carry semantic information. To develop an effective visual language, i.e., a language that can be easily and readily interpreted and manipulated by the human reader, we followed guidelines for visual language design [7, 10]. These can be summarized as *morphology as types, properties of graphical elements, matching semantics to syntax, extrinsic imposition of structure and pragmatics for diagrams*. In our UI, we attach meaning to the components of the

language both naturally (by exploiting intrinsic graphical properties such as keeping the underlying RDF graph structure) and intuitively [7] (taking consideration of human cognition, e.g. using colour to distinguish literals from classes). E.g., we considered the primary properties of graphical objects [8] to design our language's construct symbols.

## 2.1 End-user graphical model mapping to SPARQL

Through structuring the edition of applications as simple IF-THEN rule statements, and by using an underlying graph-based graphical structure, an end-user can model semantic behaviour, by means of classes, individuals and relationships. The RDF store, which reflects its content on the left side of the UI, shows only legal relationships and properties associated to each entity. Simple SPARQL queries can extract the required data to be presented in each view, each moment the user hovers a specific entity or menu. E.g., given a class, show its object properties associated. For example, to get the object properties of the class *GenericUser*, the following query would return *hasCalendar*, *worksForProject*, *performsActivity*, etc.

```

1 SELECT DISTINCT ?pred
2 WHERE { ?pred rdfs:domain ha:GenericUser.
3   ?pred rdfs:range ?object.
4   ?object a owl:Class.}

```

The mapping that transforms a graphical rule into a SPARQL query is below:

```

1 Initialize counter for ClassNode variables, n to 0.
2 Initialize processedNodes dictionary to empty.
3 <-IF CLAUSE MAPPING->
4 For each ClassNode in IFClause of the Rule:
5   For each Arc leaving from ClassNode:
6     If destNode is a Datatype: // Data Property Triple
7       Add patterns (?indiv_n a ClassName) and
8         (?indiv_n dataProp destNodeDataValue) to WHERE
9       Add originNode and its index n to processedNodes
10      Increment variable index n
11   Else: // The Triple represents an Object Property
12     If originNode is processed, obtain its index x
13     If destNode is processed, get its index z
14     Add pattern (?indiv_x objectProp ?indiv_z) to WHERE
15     Else:
16       Add pattern (?indiv_x objectProp ?indiv_n) to WHERE
17       Add destNode and its index n to processedNodes
18       Increment variable index n
19   Else:
20     If destNode is processed, get its index y
21     Add pattern(?indiv_n objectProp ?indiv_y) to WHERE
22     Add originNode and its index n to processedNodes
23     Increment variable index n

```

```

24     Else:
25         Add pattern (?indiv_n objectProp ?indiv_n+1) to WHERE
26         Add originNode and destNode to processedNodes
27         Increment variable index n by 2
28 <-THEN CLAUSE MAPPING->
29 If THENClause.type is APP: // Execute external App
30   For each ClassNode in THENClause:
31     If ClassNode is processed, obtain its index w &
32       add '?indiv_w' to SELECT
33     Else: "ERROR: Class Nodes in APP parameters need to be
34           defined in IFClause". Exit
34   QueryResult = Run SPARQL Query with {SELECT, WHERE}
35   Execute set of AppNodes with QueryResult as parameters
36 Else:
37   If THENClause.type is ADD: // Add triples
38     For each Arc marked toAdd, add pattern to INSERT
39   Else:
40     If THENClause.type is REMOVE: // Remove triples
41       For each Arc marked toDelete, add pattern to DELETE
42   Run SPARQL query including {SELECT, WHERE, INSERT, DELETE}

```

The algorithm "parses" first the IF, followed by the THEN clause in the graphical model, to finally run a SPARQL query with the parameters collected in some of the array structures for SELECT, INSERT, DELETE and WHERE. Each (*origNode*, *Arc*, *destNode*) structure in the visual model corresponds to a triple pattern (subject, predicate, object). A counter *n* keeps track of node indexes to keep unique naming for each variable associated in the SPARQL query. Every arc is processed and, depending on the type of its destination node (line 6 & 11), the pattern is modelled as a) an individual's data property or b) an object property pattern. Generated patterns are added to the WHERE field of the query. For arcs and nodes in the THEN clause, the visual model's equivalent triple patterns are added to the INSERT or DELETE fields of the SPARQL query, respectively, since these are triples that must be marked as *toAdd*, *toRemove* or *toUpdate*. Finally, in THEN clause, some application (grey) nodes may require input parameters, that can reuse information from entities declared in the IF clause.

**Rule Example Scenario:** To study the viability of the ubiquitous model, we propose a location and context-aware scenario where positioning sensors are available through, e.g., each person's phone. We developed a Human Activity ontology that models different kind of users, their interactions and the activities they perform on the environment. Let us suppose the end-user wants to create a rule which allows her to start recording audio of the weekly meeting with his supervisor, automatically when she gets into his room: "If *Natalia* enters the room of his supervisor *Johan*, start audio-recording the meeting agenda in her phone's calendar". The aim would be keeping track, for future reference, of the agenda points and brainstorming ideas discussed, on the user's calendar. First of all, the user would select, from the GUI left menu the needed entities, the datatype values for identifying the individuals *Johan* and *Natalia*, and the relations which connect these with each other. The query produced by our algorithm is below. Although 4 lines longer, it is equivalent to a straightforward query written by somebody with knowledge of SPARQL:

```
1 SELECT ?calendar1 ?phone2
2 WHERE{ ?user0 a ha:User.
3     ?user0 ha:hasName "Natalia"^^xsd:string.
4     ?user0 ha:hasCalendar ?calendar1.
5     ?user0 ha:hasPhone ?phone2.
6     ?user0 ha:isInLocation ?location3.
7     ?phone2 ha:isInLocation ?location3.
8     ?location3 ha:isNear ?office4.
9     ?user5 a ha:User.
10    ?user5 ha:hasName "Johan"^^xsd:string.
11    ?user5 ha:hasOffice ?office4.}
```

### 3 Discussion and Future work

This paper focuses on providing ordinary end-users with an accessible and functional SS vision through a tool that allows to exploit the potential of SW technologies, without requiring technical knowledge and supporting everyday life tasks. Our visual model mock-up proposal, can as well serve as an educative interface for teaching basic SW technologies and logic programming ideas intuitively. However, our main contribution is a general purpose visual language based on a semantic metamodel, that supports query federation and (imprecise) rule composition for rapid development of mash-up applications. Our end-user model pushes the devised evolution of the SW from a data modelling to a computational medium [13] by bringing the advantages of the SW closer to any non expert user. Our contribution follows visual language design guidelines [7] for an intuitive, *well matched*, visual language, i.e., its representation clearly captures the key features of the represented artefact (in our case RDF triples), in addition to simplify various desired reasoning tasks (i.e., hiding namespaces and query languages). The applications of use range from assisted living and health care to home automation or industry processes.

Future work will complement the prototype framework to support complete fuzzy reasoning, develop the graphical model (and its usability) and propose an activity model to represent a higher level human behaviour, in which the end-user can control daily activities, save and exchange rules. The proposed architecture, its support for imprecise rules and fuzzy reasoning, show the path for dealing with current issues on SSs' usability. Since it is clear that having a *Scratch* for real life problem modelling would be of great use, future works, aiming at tackling the mentioned issues, will overcome and better model a context-aware SW accessible not only by machines but also by any human.

**Acknowledgements** The research work presented in this paper is funded by TUCS (Turku Centre for Computer Science).

## References

1. C. Beckmann. Siteview: Tangibly programming active environments with predictive visualization. In *In: Intel Research Tech Report*, pages 167–168, 2003.
2. F. Bobillo and U. Straccia. *fuzzyDL: An expressive fuzzy description logic reasoner*. In *2008 International Conference on Fuzzy Systems (FUZZ-08)*, pages 923–930. IEEE Computer Society, 2008.
3. A. C. Bolós, P. P. Tomás, J. J. Martínez, and J. A. M. Agües. Evaluating user comprehension of dataflows in reactive rules for event-driven AmI environments. In *Proceedings of the V International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI)*, 2011.
4. H. Chen, T. Finin, and A. Joshi. Semantic web in a pervasive context-aware architecture. In *Artificial Intelligence in Mobile System (AIMS 2003), In conjunction with UBICOMP*, pages 33–40, 2003.
5. M. García-Herranz, P. Haya, and X. Alamán. Towards a ubiquitous end-user programming system for smart spaces. 16(12):1633–1649, jun 2010.
6. P. Giner, C. Cetina, J. Fons, and V. Pelechano. A framework for the reconfiguration of ubi-comp systems. In J. Corchado, D. Tapia, and J. Bravo, editors, *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*, volume 51 of *Advances in Soft Computing*, pages 1–10. Springer Berlin Heidelberg, 2009.
7. C. Gurr. Computational diagrammatics: diagrams and structure. In D. Besnard, C. Gacek, and C. B. Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Springer London, 2006.
8. C. Gurr. Visualizing a logic of dependability arguments. In P. Cox, A. Fish, and J. Howse, editors, *Visual Languages and Logic Workshop (VLL 2007)*, volume 274, pages 97–109, 2007. Workshop within IEEE Symposium on Visual Languages and Human Centric Computing VL/HCC 07.
9. C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, 37(2):83–137, June 2005.
10. D. Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Softw. Eng.*, 35(6):756–779, Nov. 2009.
11. M. Resnick, J. Maloney, A. Monroy-Hernandez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. B. Kafai. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
12. A. Rex. Design of a caregiver programmable assistive intelligent environment. Aalto University, 2011.
13. M. A. Rodriguez and J. Bollen. Modeling computations in a semantic network. *Computing Research Repository (CoRR)*, ACM, abs/0706.0022, 2007.
14. T. Y. Sohn and A. K. Dey. iCAP: An informal tool for interactive prototyping of context-aware applications. In *Extended Abstracts of CHI*, pages 974–975, 2003.
15. G. Vanderhulst, K. Luyten, and K. Coninx. Pervasive maps: Explore and interact with pervasive environments. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 227–234, 29 2010-april 2 2010.
16. T. Walter and J. Ebert. Combining DSLs and ontologies using metamodel integration. In *Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages, DSL '09*, pages 148–169, Berlin, Heidelberg, 2009. Springer-Verlag.
17. T. Zhang and B. Brügge. Empowering the user to build smart home applications. In *Proceedings of 2nd International Conference on Smart Homes and Health Telematic (ICOST2004), Singapore, 2004. Palviainen, Marko Series*. Marko Palviainen, 2004.