Extended Context Patterns – A Visual Language for Context-Aware Applications[⋆] [⋆⋆]

Andrei Olaru[1] and Adina Magda Florea[1]

Computer Science and Engineering Department
University Politehnica of Bucharest
060042 Bucharest, Romania
cs@andreiolaru.ro, adina.florea@cs.pub.ro

Abstract. The paper presents a visual language that can help users of a context-aware application represent the current situation, or situations they wish detected, in a language that is both formally defined, and readable and understandable by humans and machines alike. Inspired from Regular Expressions, the concept of Extended Concept Pattern provides both conciseness and expressive power, allowing for specifying negation, and for indicating repeating or alternative structures.

Keywords: Artificial intelligence, Ambient intelligence, Context-awareness, Software agents, Graph theory

1 Introduction

Context-aware applications [11] are gaining a great deal of traction today, as the use of context data enables an application to appear to the user as smart and useful, its actions making sense in the current situation [5,1]. Currently, however, context-awareness is (a) *programmed* and, more than that, (b) *pre-programmed* in the applications. Let us explain.

First, context-aware actions or rules are usually embedded in the code of the application, and when they are not, they are represented in a language that is machine-oriented and understandable only by programmers, rather than readable by the user of the application. Second, the user is unable to change the behavior of the context-aware application, such that in some situations the application reacts differently than pre-programmed, but closer to the desires of the user. In part, this is because the user is generally unable to easily understand and modify the context-aware behavior of the application.

This paper introduces a visual language for the representation of context-based behavior. It allows the user to work with the representation of context

⋆ This work has been supported by the grant *CAMI: Artificially intelligent ecosystem for self-management and sustainable quality of life in AAL*, AAL Programme, 2015-2018.

⋆⋆ The final version of this paper is available on Springer Link at https://link.springer.com/chapter/10.1007/978-3-319-48829-5_6.

and to express situations that should be detected and actions that should be taken. The constructs offered by the language offer features of increasing complexity, which correspond to an increasing formal preparation that is required to understand how they work. However, usage at lower level should be available to the great majority of users.

The language that we present in this paper is that of *Extended Context Patterns*. It uses as background the formal representations of *Context Graphs* and *Context Patterns*, which have been introduced in previous work to represent the current situation of the user and situations that are desired to be detected [10]. An Extended Context Pattern allows a developer or a user to better understand, use and modify the representation of a situation.

The formalism of Extended Context Patterns is related to the textual linear graph representation that we have previously developed, and improves Context Patterns by giving them more expressive power and making them easier to use. Taking inspiration from Regular Expressions, Extended Context Patterns include *operators* such as transitive closure, alternation, and negation, increasing their power of representation.

Throughout the paper, we will use the following running example: Joe has an elderly mother, Emily, who lives alone. Emily is part of an Ambient Assisted Living (AAL) program. She wears a bracelet that can detect falls and report Emily's location. Emily is also assisted by Nurse Jane, a professional carer who cares for several other people. Joe is somewhat familiar with computers, so he is able to set up some action patterns which are specific to Emily's case. These are used by the AAL system to help Jane in her activity. One of the main issues in this scenario is to know who is the emergency contact in the case in which a fall is detected.

The next section presents some research related to ours. Existing definitions and previously developed concepts are presented in Section 3, helping the definition of Extended Context Patterns in Section 4. The paper ends with the conclusion and future perspectives.

2 Related Work

There are currently several proposals that intend to deal with the representation of context and situations in a formal manner. Generally they rely on graph theory, and enhance the representation with various types of tags and special kinds of relations.

Bearing a high degree of relation with the formalism of Context Graphs and Patterns, semantic networks [15], concept maps [8] and Sowa's conceptual graphs [14] are directed graphs representing concepts and relations between concepts. One of their main advantages is their property of being graphically displayed, helping understanding of what they express. Conceptual Graphs, in particular, allow expressing any logical formula as a hypergraph. While semantic networks lack some power of expression, which conceptual graphs have, both lack an intuitive mechanism of expressing partially-defined situations. Moreover, these formalisms are not particularly focused on, or appropriate for, the representation of a user's focus and context.

Triples and RDF graphs [7] are easier to use for machines, especially from the point of view of internal representation and disambiguation. However, RDF is difficult to read and write for a user directly, due to the need to use URLs and to repeat the reference to concepts for every relation. Situations recognition can be done by means of SPARQL rules, which however would be difficult to work with by humans. [13]

Some context-focused representations, such as CML (see [12]), use a graphical representation together with a machine-readable XML file which is not adequate for use by humans without assistance from an advanced editor or IDE.

Related to this work are also several methods and tools that have been developed by the authors in previous research. The representation of Context Graphs and Patterns benefits from a visual representation, and also from a basic text representation [9]. More importantly, a matching algorithm has been devised that allows matching context patterns against graphs with very good results for the problem at hand. A platform has also been designed that improves the performance of matching, in the case where the graph evolves over time, and a library of patterns is matched against the same graph.

3 Prerequisites

While there are many definitions of context, we look at two of them in particular. The original definition given by Dey et al was that the *context* contains any element that is relevant to the interaction between the user and the application [4]. Practically, it may extend to the entire current situation of the user. Context elements could in theory be categorized into several categories, such as elements of spatial, temporal, social, or computational context [3], without ignoring activity context [6]. Another definition that is relevant to our approach is the one given by Brézillion et al, comparing context to the dressing of a focus – the context is everything that is related to the current focus of the user [2]. A context-aware application is one that has access to all these elements and uses them in order to provide an improved and more intelligent response to the user.

All the elements that are part of the context need a *representation*. Context-aware behavior is defined by reactions to particular features in the context of the user. Therefore a representation is also needed for the situations that need action to be taken. Such an action may be a notification, a rational inference, or provision of certain information to the user (as does, for instance, the Google Now[1] service). Such representations could be offered by Context Graphs and Context Patterns, respectively.

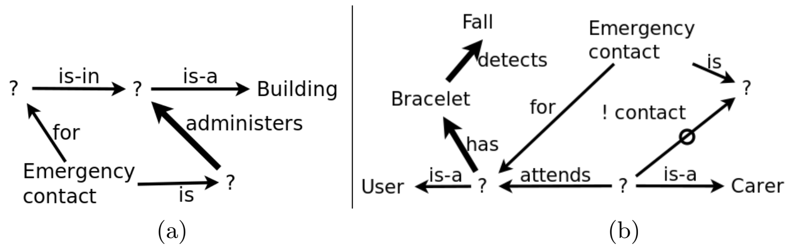

Fig. 1. (a) A Context Pattern showing who should the carer contact in case of an emergency: the administrator of the building where the user is located. The *administers* edge is required for a match to be considered (the edge is *characteristic*). (b) A more complex pattern, containing both *characteristic* (in bold) and *actionable* (marked with a circle) edges. The pattern states that when the bracelet is worn by the user and it detects a fall, the carer who attends the user must be prompted to contact the emergency contact for the user.

3.1 Context Graphs and Patterns

We have introduced Context Graphs and a basic version of Context Patterns in previous work [9]. A *Context Graph* is a representation for the current context of the user. It contains a large number of elements, some of which can be categorized as part of the spatial, temporal, computational, social, or activity context. We mainly consider context as a set of elements that are in some *relation* with the user and the user's current situation, therefore any association can be integrated, not only the categories mentioned before.

Formally, the Context Graph of an agent A is a graph CG_A. Considering a global set of *Concepts* (strings or URIs) and a global set of *Relations* (strings, URIs or the empty string λ, for unnamed relations), the graph is defined as:

$CG_A = (V, E)$, where $V \subseteq Concepts$ and

$E = \{\ (from, to, value, persistence) \mid from, to \in V, value \in Relations\}$

The *persistence* feature of edges allows for them to expire after a certain time, set when they are created.

In order to detect relevant information, or to find potential problems, an agent has a set of *Context Patterns* that it matches against graph CG_A. These patterns describe situations that are relevant to its activity. A pattern with the identifier s is defined by a graph[2] G_s^P:

$G_s^P = (V_s^P, E_s^P)$, where $V_s^P \subseteq Concepts \cup \{?\}$ and

$E_s^P = \{\ (from, to, value, c, a) \mid from, to \in V_s^P, value \in Relations \cup \{\lambda\}\}$

We call nodes labeled with a question mark *generic nodes*. An example of a context pattern is shown in Figure 1(a). It shows that the emergency contact for a user that is in a building is the administrator of that building. Each edge has

[1] `https://www.google.com/landing/now/#whatisit`

[2] We will use the " ^{P} " superscript to mark structures that support generic elements, such as generic nodes.

two features – *characteristic* and *actionable* – showing if a particular edge is absolutely necessary for considering a partial match; and if an edge can be inferred (or actioned upon) in case of a partial match[3]. For instance, the *administers* relation is required to exist for the pattern in Figure 1(a) to be considered.

A more complex example of a Context Pattern is shown in Figure 1(b). It contains 2 characteristic edges, that could not be inferred even if the rest of the pattern matches. It also contains an actionable edge: if the rest of the pattern matches with the Context Graph, the Carer will be prompted to contact the person or organization that is resolved to be the emergency contact. How this can be done is shown in the subsequent examples.

By matching a pattern from the agent's set of patterns against the agent's context graph, an agent is able to detect interesting information and is able to decide on appropriate action to take. We have previously developed an efficient algorithm for such matching [9].

The pattern G^P_s *matches* the subgraph $G'_A = (V', E')$, *iff* there exists an injective function $f_v : V^P_s \rightarrow V'$, so that the following conditions are met simultaneously:

(1) $\forall v^P \in V^P_s, v^P = ?$ or $v^P = f(v^P)$ (same value)

(2a) $\forall (v^P_i, v^P_j, rel) \in E^P_s, (f(v^P_i), f(v^P_j), value) \in E', value \in \{rel, \lambda\}$

(2b) $\forall (v^P_i, v^P_j, \lambda) \in E^P_s, \exists value \in Relations, (f(v^P_i), (v^P_j), value) \in E'$

That is, every non-generic vertex in the pattern has the same label as a different vertex from G'_A (f_v is injective), and every edge in the pattern matches (same label for the edge and vertices) an edge from G'_A. The subgraph G' should be minimal (no edges that are not matched by edges in the pattern). One pattern may match various subgraphs of the context graph. A pattern G^P_s *partially matches with k missing edges* (*k-matches*) a subgraph G' of G, if conditions (2) above are fulfilled for $m_s - k$ edges in E^P_s, $k \in \{1..m_s - 1\}$, $m_s = ||E^P_s||$ and G' remains connected and minimal. Partial matches are useful because, depending on a set threshold for k, they indicate cases where action may be taken automatically, to create the missing edges, or notifications may be sent [10].

4 Extended Context Patterns

In order to improve the power of expression held by Context Patterns, the original concept needed to be amended, in order to accommodate the specification of subgraphs with negative character, as well as the operations of alternation and repetition. The result is the *Extended Context Pattern*.

An *Extended Context Pattern* with the name s is defined as

$P^E_s = (H^{P_E}_s, E^{\neg}_s, E^{(*)}_s, E^{|}_s)$,

[3] Both the *characteristic* and the *actionable* features can be numeric instead of boolean, in the interval $[0, 1]$. For simplicity, we will consider them boolean in this work.

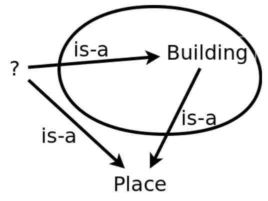

Fig. 2. Example of a hyperedge, represented as an enclosure of a part of the graph. The inbound arity and the outbound arity of the hyperedge are both equal to 1.

where $H_s^{P_E} = (V_s^P, E_s^{P_E})$ is the hypergraph underpinning the extended pattern, and $E_s^{\neg}$, $E_s^{(*)}$ and $E_s^{|}$ are three sets[4] containing information on negative, repetition and alternation hyperedges, respectively. We will use the notation "^{P_E}" for structures that include hypergraph elements used for extended pattern.

In fact, we only allow $H_s^{P_E}$ some limited structural differences from a normal graph pattern G_s^P. Any edge in the hypergraph is either part of one of the three hyperedge sets, or is a binary, directed edge in the graph pattern.

All three types of hyperedges contained in $H_s^{P_E}$ indicate, in fact, subgraphs of $H_s^{P_E}$ with particular properties. A hyperedge $e_i^{P_E} \in E_s^{P_E}$ *covers* an induced subgraph $H_{si}^{P_E} = (V_{si}^P, E_{si}^{P_E})$, with

$V_{si}^P = e_i^{P_E} \subseteq V_s^P$ and

$E_{si}^P = \{e^{P_E} \mid e^{P_E} \subseteq e_i^{P_E}\}$

It is mandatory that any hyperedge $e_i^{P_E}$ that intersects another hyperedge $e_j^{P_E}$ is either completely included in $e_j^{P_E}$ or completely includes $e_j^{P_E}$, and that no two hyperedges cover the same subgraph. That is,

$\forall\, e_i^{P_E}, e_j^{P_E} \in E_s^{P_E}.\ e_i^{P_E} \cap e_j^{P_E} \neq \emptyset \Rightarrow e_i^{P_E} \subset e_j^{P_E} \vee e_j^{P_E} \subset e_i^{P_E}.$

There may exist, however, some binary, directed edges that have one end inside the graph covered by $e_i^{P_E}$ and one end outside it. We call these edges *arity* edges, and they can be *inbound* or *outbound*. For a hyperedge $e_i^{P_E}$ of the extended graph pattern, the set of arity edges is defined as $\bar{H_{si}}$, with:

$\bar{H_{si}} = \overline{H\text{-}in_{si}} \cup \overline{H\text{-}out_{si}}$

$\overline{H\text{-}in_{si}} = \{e \mid e = (v_k^P, v_l^P) \in E_s^{P_E}, v_k^P \notin e_i^{P_E}, v_l^P \in e_i^{P_E}\}$

$\overline{H\text{-}out_{si}} = \{e \mid e = (v_k^P, v_l^P) \in E_s^{P_E}, v_k^P \in e_i^{P_E}, v_l^P \notin e_i^{P_E}\}$

The *pattern-arity* of a hyperedge $e_i^{P_E}$ is the number of arity edges that it has, that is $||\bar{H_{si}}||$. We can define the *inbound pattern-arity* of the hyperedge and its *outbound pattern-arity*.

For instance, Figure 2 shows a graph pattern that contains a hyperedge. The hyperedge covers a graph formed of a single node (*Building*) and no edges, and having one inbound arity edge and one outbound arity edge, amounting to a pattern-arity of 2.

[4] the names of the tree sets are read as "E-neg", "E-star", and "E-or".

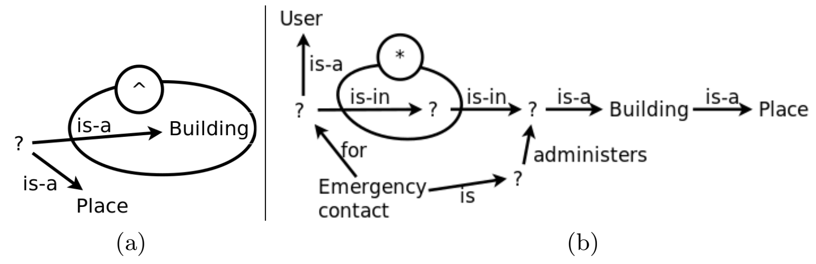

Fig. 3. (a) An example of a negative hyperedge in a graph pattern specifying a place which is not a building. (b) An example of a pattern containing a repetition hyperedge which may match a longer path of spatial inclusion.

Hyperedges in an extended context pattern do not directly take part in the matching process (i.e. no matching hyperedges are searched for in the context graph), but rather influence how the matching is done.

A **negation hyperedge** $e^{\neg} \in E_s^{\neg} \subset E_s^{P_E}, e^{\neg} \subseteq V_s^P$ covers a subgraph of the context pattern that should not be matched in the context graph, in order to obtain a match of the pattern. Opposite from other edges in the pattern, any edges that are contained in the graph covered by the negation hyperedge and that are matched with edges in the context graph, increase the k number of the match.

For example, Figure 3(a) shows an example of a negative hyperedge in a graph pattern. The pattern matches a node which is a place but is not a building.

A **repetition hyperedge** $e_i^{(*)} \in E_s^{P_E}$ is part of a tuple
$(e_i^{(*)}, v_{in}^P, v_{out}^P, e_{in}, e_{out}) \in E_s^{(*)}$,
with $v_{in}^P, v_{out}^P \in e_i^{(*)}$, with e_{in} an arity edge of $e_i^{(*)}$ that is incident to v_{in}^P and e_{out} an arity edge of $e_i^{(*)}$ that is outgoing from v_{out}^P.

The last element of the tuple – e_{out} – is optional. If an *out* edge is specified, the repetition is *binary*, otherwise it is *unary*. A unary repetition hyperedge must have an inbound pattern-arity of at least 1; a binary repetition hyperedge must also have an outbound pattern-arity of at least 1.

In the matching process, the repetition hyperedge acts as a Kleene-star operation on its subgraph. Consider that $e_{in} = (v_a^P, v_{in}^P)$ and, if any, $e_{out} = (v_{out}^P, v_b^P)$. The subgraph covered by a unary repetition hyperedge will match the context graph if:

(1) the context graph contains no subgraph matching H_{si}, or

(2) the context graph (V, E) contains a sequence of n matches of H_{si}, $n \geq 1$, in which v_a^P is matched to $v_a \in V$, v_{in}^P is matched to vertices $v_{in}^{(k)} \in V$ and v_{out}^P is matched to vertices $v_{out}^{(k)} \in V$, with $k = \overline{0, n-1}$. Then, there must exist an edge $(v_a, v_{in}^{(0)})$ matching e_{in}, and a series of $n-1$ edges $(v_{out}^{(k)}, v_{in}^{(k+1)})$, $k = \overline{0, n-2}$, also matching e_{in}.

The subgraph covered by a binary repetition hyperedge will match the context graph if:

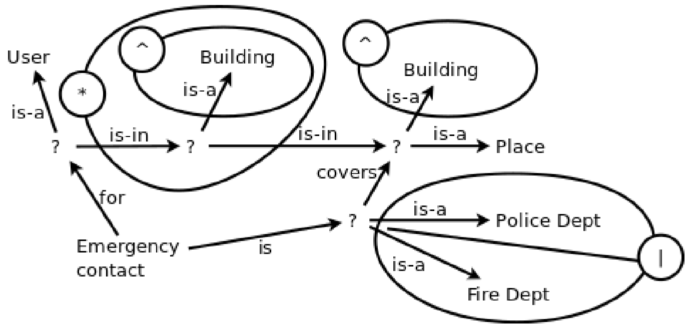

Fig. 4. An example of a pattern containing, among others, an alternation hyperedge.

(1) the context graph contains no subgraph matching H_{si}, but contains an edge (v_a, v_b) matching e_{out}, with v_a matching v_a^P and v_b matching v_b^P; or,

(2) the context graph contains a sequence of n matches of H_{si}, $n \geq 1$, in which v_a^P is matched to $v_a \in V$, v_b^P is matched to $v_b \in V$, v_{in}^P is matched to vertices $v_{in}^{(k)} \in V$ and v_{out}^P is matched to vertices $v_{out}^{(k)} \in V$, with $k = \overline{0, n-1}$. Then, there must exist an edge $(v_a, v_{in}^{(0)})$ matching e_{in}, a series of $n-1$ edges $(v_{out}^{(k)}, v_{in}^{(k+1)})$, $k = \overline{0, n-2}$, also matching e_{in}, and an edge $(v_{out}^{(n-1)}, v_b)$ matching e_{out}.

For example, Figure 3(b) shows a pattern that serves to determine the emergency contact in the case in which the assisted user is inside a building. The building may have a hierarchy of places (floors, areas, rooms, etc), but only the top node of the hierarchy is a building and has an administrator. This pattern matches any such case.

The formalism may be extended to support the case in which edges between the matches may be different (have a different label) from the edge entering the first match.

An **alternation** operation is characterized by a set of hyperedges with sets of arity edges that are identical from the point of view of direction, label, and adjacent vertex outside of the hyperedge:

$alternation \in E_s^{|}$ with $alternation \subseteq E_s^{P_E}$, each alternation characterized by two sets:

– $in\text{-}set = \{(v_a^P, label) \mid v_a^P \in V_s^P \setminus \bigcup_{e_i \in alternation} V_{si}^P\}$, the set of sources and labels for arity edges going towards the hyperedge, such that $\forall e_i^{|} \in alternation \,.\, \forall (v, u, label) \in \overline{H\text{-}in_{si}} \,.\, (v, label) \in in\text{-}set$; and

– $out\text{-}set = \{(label, v_b^P) \mid v_b^P \in V_s^P \setminus \bigcup_{e_i \in alternation} V_{si}^P\}$, the set of destinations and labels for arity edges going towards the hyperedge, such that $\forall e_i^{|} \in alternation \,.\, \forall (v, u, label) \in \overline{H\text{-}out_{si}} \,.\, (u, label) \in out\text{-}set$.

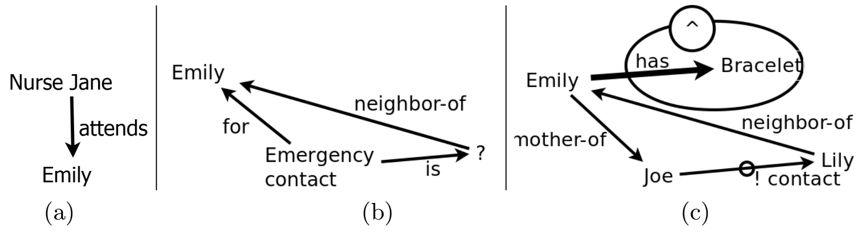

Fig. 5. (a) A simple statement. (b) A simple user-configured pattern. (c) A pattern specific to Emily's case: if Emily does not wear the bracelet, Joe should contact her neighbor Lily.

The alternation set matches a subgraph of the context graph if the subgraph covered by one of the hyperedges in the alternation correctly matches, as part of the pattern.

The example in Figure 4 builds upon previous examples to show a pattern that helps determine the emergency contact in the case when the assisted user is in an exterior area (not in a building). In this case, the emergency contact is the police or fire department that covers an area which includes the area where the user is located.

5 Discussion

Let us look again at the example of Joe and his mother Emily, who is part of an AAL program. Some of the patterns that could be used by a context-aware AAL application have been presented above. They are patterns that show what should happen if a fall is detected, and who is the emergency contact, depending on the context of the user. Such patterns are complex and we cannot expect them to be implemented by normal users. But they are general enough to be already created by developers. Joe can, however, understand what the patterns mean, as the graphical representation is much easier to cope with than other types of representations. Being able to understand why the system chooses to perform an action makes it more acceptable and more dependable.

While a normal user would be unable to create complex patterns, one would surely be able to create simpler ones. For instance, Joe may want to write down that it is Nurse Jane that attends Emily. This can be done by means of a simple graph edge such as the one in Figure 5(a). When Joe understands the system better, he may event insert some patterns, such as the one in Figure 5(b), stating that any of Emily's neighbors can be considered an emergency contact. A more complex user-configured pattern is presented in Figure 5(c). Joe may wish to be prompted to contact Emily's neighbor, Lily, in the case when Emily is not wearing her bracelet (and therefore the AAL system would not have information about Emily's state or whereabouts).

6 Conclusion and Perspectives

This paper presents the visual language of Extended Context Patterns, which builds on the previously developed formalisms for Context Graphs and Context Patterns. Extended Context Patterns allow for the specification of negation, on the one hand, and of variable structures such as repetition and alternatives.

As future work, the matching algorithm must be extended in order to account for the new features of extended context patterns, however the nature of the matching algorithm makes it easy to be adapted to these changes.

Further, the language will be integrated with the tATAmI-2 multi-agent systems for ambient intelligent applications.

References

1. Augusto, J., Nakashima, H., Aghajan, H.: Ambient intelligence and smart environments: A state of the art. Handbook of Ambient Intelligence and Smart Environments pp. 3–31 (2010)
2. Brézillon, J., Brézillon, P.: Context modeling: Context as a dressing of a focus. In: Kokinov, B., Richardson, D., Roth-Berghofer, T., Vieu, L. (eds.) Modeling and Using Context, Lecture Notes in Computer Science, vol. 4635, pp. 136–149. Springer Berlin Heidelberg (2007), `http://dx.doi.org/10.1007/978-3-540-74255-5_11`
3. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College (November 2000)
4. Dey, A., Abowd, G., Salber, D.: A context-based infrastructure for smart environments. Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99) pp. 114–128 (1999)
5. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.: Scenarios for ambient intelligence in 2010. Tech. rep., Office for Official Publications of the European Communities (February 2001)
6. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. Lecture notes in computer science pp. 167–180 (2002), `http://www.springerlink.com/content/jbxd2fd5ga045p8w/`
7. Lassila, O., Swick, R.: Resource description framework (RDF) model and syntax specification. Tech. rep., The World Wide Web Consortium (1998)
8. Novak, J.D., Cañas, A.J.: The origins of the concept mapping tool and the continuing evolution of the tool. Information Visualization 5(3), 175–184 (2006)
9. Olaru, A.: Context matching for ambient intelligence applications. In: Björner, N., Negru, V., Ida, T., Jebelean, T., Petcu, D., Watt, S., Zaharie, D. (eds.) Proceedings of SYNASC 2013, 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, September 23-26, Timisoara, Romania. pp. 265–272. IEEE CPS (2013)
10. Olaru, A.: Context-awareness in multi-agent systems for ambient intelligence. In: Brézillon, P., Gonzalez, A.J. (eds.) Context in Computing - A Cross-Disciplinary Approach for Modeling the Real World, pp. 541–556. Springer New York (2014), `http://link.springer.com/chapter/10.1007/978-1-4939-1887-4_33`
11. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Context aware computing for the internet of things: A survey. IEEE Communications Surveys and Tutorials 16(1), 414–454 (2013)

12. Robinson, R., Henricksen, K., Indulska, J.: XCML: A runtime representation for the context modelling language. Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on pp. 20–26 (2007)
13. Sorici, A., Picard, G., Boissier, O., Florea, A.: Multi-agent based flexible deployment of context management in ambient intelligence applications. In: Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection, pp. 225–239. Springer (2015)
14. Sowa, J.: Conceptual graphs. Foundations of Artificial Intelligence 3, 213–237 (2008)
15. Sowa, J.F.: Semantic networks. Encyclopedia of Cognitive Science (2006)