The Computer Journal Advance Access published September 11, 2014

doi:10.1093/comjnl/bxu088

Using Formal Concept Analysis for Organizing and Discovering Sensor Capabilities

Wassim Derguech[1]*, Sami Bhiri[2], Souleiman Hasan[1] and Edward Curry[1]

[1]*Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland*
[2]*Institut MINE-TELECOM, TELECOM & Management SudParis, Evry, France*
**Corresponding author: wassim.derguech@insight-centre.org*

Smart environments rely on sensor data to provide necessary business intelligence in order to support decision-making. An efficient decision support model in such a context requires that sensor data are provided correctly and timely. Given the dynamicity of sensor environment, the diversity of their features and of user requirements, finding appropriate sensors having the required capabilities or replacing faulty ones constitutes a challenging task. Efficiently describing and organizing sensors in smart environments is essential to deliver a rapid adaptation to errors and availability of data. In this paper, we present an approach for organizing and indexing sensor services based on their capabilities. We introduce a feature-oriented capability model that puts forward the functional aspects of carried actions and model them as resource description framework (RDF) properties rather than focusing on the change in the state of the world. Using this model for describing sensor capabilities, we apply Formal Concept Analysis for organizing and indexing sensor services. We have experimented and evaluated our approach in the Digital Enterprise Research Institute, which has been retrofitted with various sensors to monitor temperature, motion, light and consumption of power within a building.

Keywords: formal concept analysis; capability modelling; sensor capability

Received 30 September 2013; revised 4 August 2014
Handling editor: Mohamed Jmaiel

1. INTRODUCTION

With their decreasing costs and increasing capabilities, sensors are playing a key role in emerging cyber-physical information systems especially smart environments. Indeed, such environments rely on sensor data to provide necessary business intelligence to support decision-making. A possible use case can be a smart building within an energy management application to control the supply and demand of energy.

The main source of information used for running a decision support model relies mainly on sensors deployed within a smart building. An efficient decision support model in such a context requires that sensor data be provided correctly and timely. However, accidents may occur at any time. For example, a sensor may become not responsive or a source of errors. In these cases, the decision support model should provide suggestions to observe another source of data. This can be made easy if sensors are properly described and organized. Creating explicit links between sensors helps to discover similar ones and, consequently, facilitate balancing observations from one sensor to the other.

Given the dynamicity of a sensor environment, the diversity of their features and of user requirements, finding appropriate sensors having the required capabilities or replacing faulty ones constitutes a challenging task especially in medium- and large-scale areas. Efficiently describing and organizing sensors in smart environments is essential to deliver a rapid adaptation to errors and availability of data.

In this paper, we present an approach for organizing and indexing sensor services based on their capabilities. We describe sensor capabilities based on our capability model [1] which we adapt for the focus of this paper.

Current approaches for modelling capabilities (either as part of semantic services or semantic business processes), focus on the change made by the action to the state of the world. Our capability model focuses rather on the aspects of interest to the users and targeted applications, which characterize the carried action. These aspects are formally defined as resource description framework (RDF) properties in domain-specific ontologies. They relate to domain-specific functional dimensions of the carried action. User and targeted application requirements are often expressed based on these features of interest.

Based on our feature-based capability model, we apply Formal Concept Analysis [2] (FCA for short) for organizing and indexing sensor services based on their capabilities. FCA is a well-known mathematical classification tool used in various domains that allows organizing objects described via a set of attributes into a Concept Lattice.

The proposed approach has been experimented on in the Digital Enterprise Research Institute (DERI). The building has been retrofitted with energy sensors to monitor the consumption of power within the building. A building-specific aspect of the dataspace has been presented in [3] with a sensor network-based situation awareness scenario presented in [4]. An event processing technique is applied to process this real-time sensor information to support the energy management applications [5]. However, it has been observed that when a particular sensor is not working properly, there is a limited support to decision-making, data mining and knowledge discovery. Indeed, the identification of critical and meaningful patterns highly depends on the reliability of the data provided.

To build an efficient energy management system in such a context, the main challenge we are dealing with in this paper is how to efficiently organize and index sensors based on their capabilities.

The remainder of the paper is organized as follows. Section 2 presents related work to capability modelling and discovery. Section 3 introduces our capability model and shows how we use it to define sensor service capabilities. Section 4 revisits the theoretical foundations of FCA and shows how we apply it to organize and index sensors based on their capabilities into tree like structure called "sensor capability lattice". It also shows how we can exploit this sensor capability lattice to discover sensors and implications between sensor attributes. Section 5 reports on the evaluation of our work. First, we introduce, in Section 5.1, Linked Energy Intelligence (LEI) dataspace which constitutes our use case for organizing sensor capabilities using FCA. Then we detail, in Section 5.2, two experimentations for verifying the applicability of our approach. Finally, Section 6 draws a conclusion and outlines future work.

2. RELATED WORK

The work proposed in this paper is related to two main research areas: capability modelling and capability discovery. On the one hand, we reviewed related work to web services capability modelling as it constitutes a first pillar to our contribution; on the other hand, we reviewed related work to service capability discovery as it is highly related to service organization/indexing.

2.1. Capability modelling

A capability denotes what an action does either in terms of real world effects or returned information [6]. In the literature, we can distinguish three families of approaches that tackled the problem of capability modelling either directly or indirectly.

The first family includes Semantic Web Services models (WSMOs [7] and OWL-S [8]) which model capabilities as Input, Output, Preconditions and Effects (IOPE paradigm). A capability is expressed by the state of the world before the Web service is executed and the state of the world after successful Web service provision [9]. Such state of the world changes are expressed in terms of axioms. Modelling capabilities as such do not feature in an explicit and easily accessible way domain features. Extracting and managing domain attributes requires some reasoning which can be time consuming and difficult to manage by end-users.

The second family of related efforts concerns semantic annotations of invocation interfaces (SA-WSDL [10] and SA-REST [11]). While these approaches do not directly target capability modelling, they attempt to provide alternative solutions to top-down semantic approaches (WSMO [7] and OWL-S [8]) by starting from existing descriptions such as WSDL [12] and annotate them with semantic information. For example, the SAWSDL specification indicates a possible use of the interface annotation for categorization [13] that might help in introducing a natural language indication of the action being done by the proposed service. However, these approaches define a semantic description of syntactic interaction interfaces rather than concrete capabilities.

The third family includes frame-based approaches for modelling capabilities. Oaks *et al.* [14] give a comprehensive overview of related approaches and propose a model for describing service capabilities as such. The proposed model distinguishes in particular the corresponding action verb and informational attributes (called roles in the paper [14]) in addition to the classical IOPE. While this model makes a step beyond the classical IOPE paradigm, the semantics of capabilities remain defined via the IOPE paradigm and therefore has the same issues as the first family of approaches described above.

All of the previously discussed approaches describe capabilities without featuring functional domain properties. A capability is highly coupled with its implementation (i.e. invocation interface) or related to the description of another concept (i.e. services). We strongly support the idea of considering the capability as an independent concept that describes what a programme, a business process, a service, etc. does from a functional perspective. A capability should not be

limited only to a single label or an action verb but also should consider a proper description of domain-dependent functional properties forming a standalone *entity*.

2.2. Indexing and discovery of service capabilities

The primary purpose of explicit modelling of service capabilities is discovery and selection. Several approaches have been proposed for service discovery and similarity-based selection. In this section, we reviewed relevant contributions towards this issue.

Similarity-based service discovery has been studied extensively for various kinds of services. In [15], multiple evidences are considered to evaluate the similarity of service operations and inputs/outputs based on *clustered concepts* extracted from WSDL documents. To make this approach more user-friendly, another abstraction layer that uses natural language processing is needed to translate user requests into formally specified search requests.

In [16], a hybrid matchmaking approach is proposed for discovering OWL-S services. Among the hybrid filters used in [16], the *subsumed-by* and *nearest-neighbour* filter leverage different Information Retrieval (IR) similarity metrics. In [17, 18], a *replacement degree* is computed for two *service protocols* based on how their sub-protocols can replace each other in the context of mediated service interactions. Compared with the above approaches, we are more focused on providing an easy way for the users to specify their requirements to find a replacement for an unsatisfiable sensor capability by specifying the set of required attributes (i.e. sensor properties).

We have previously provided approaches that describe and discover service capabilities using attribute-featured service descriptions [19, 20]. In these approaches, we construct an indexing structure based on *extension* and *specification* relations. Then we apply heuristic search algorithms to retrieve the closest services to user requests. In the current work, we propose to use FCA as we find it more appropriate for avoiding heuristic search as the indexing structure (i.e. FCA concept lattice) is uniformly created. A search algorithm requires only to navigate the FCA lattice.

3. MODELLING SENSOR CAPABILITIES

3.1. Capability model

Different from existing approaches, which focus on the change of the state of the world made by an action, we propose a semantic frame-based model which rather focuses on and features aspects of interest (to the users and targeted applications) of the carried action. In the following, we present our capability model [1] adapted to the focus of this paper. We first briefly introduce our model and then show how we use it to describe sensor capabilities.

We model a capability as a category (of functionalities) enriched by (zero or many) functional or non-functional features (see the concept of property entry below). These features refine the given category (which corresponds to an abstract capability) by giving more details about aspects of interest of the corresponding action.

More formally, in our model both search requests and service capabilities are defined as a set of `property entries`. A `property entry` is a couple *(property, value)* where property is a domain-specific functional feature or a domain-independent non-functional property. Both *property* and *value* refer to ontological terms shared by service descriptions and search requests.

A `property entry` (P, v) is specified with respect to a `property declaration` defined in a shared ontology. A `property declaration`, $d = (P, V, R)$, defines (i) a property P as a relevant functional or non-functional feature of the capabilities of a given domain, (ii) V the most general value (super class) that property entries defined according to d can have and (iii) a relation R that tells when a value v_1 is more specific than a value v_2 with respect to the meaning of the property P.

A specific property that is present in all capability descriptions is `hasActionCategory` whose value denotes the category of the capability defined in a domain-specific ontology.

We ground our capability model on RDF-S/RDF. We define the class `PropertyDeclaration` as a subproperty of (*RDF-S:subPropertyOf*) the RDF class `RDF:Property`. We model property declarations as RDF properties, instances of the class `PropertyDeclaration`. Property entries are defined as RDF predicates referring to the corresponding declarations.

3.2. Sensor capability models

The sensor capability model extends our capability model as follows. Listing 2 gives a snippet in N3 format of our RDF Sensor Capability Ontology (SCO) which is also illustrated graphically in Fig. 1. First, we define the superclass of all sensor capabilities `SensingCapability` as a subclass of the concept `Capability` (Listing 2, line 10). As all sensors are supposed to provide data about a phenomenon they are observing, we define the action category *sensing* and assign it as the value of the property `hasActionCategory` (Listing 2, line 12–14).

The second step consists of defining features of interest of sensor capabilities. As presented above, these features are specified as property declarations. We distinguish between valued and non-valued features. Non-valued features are features that are either present/fulfilled or not such as `hasStorageOption`. The range of such properties is `boolean`. We define the following property declarations (see Fig. 1).

(i) *sco:isActive* (line 18) is defined as a property that has a boolean value. It reports if the sensor is active.

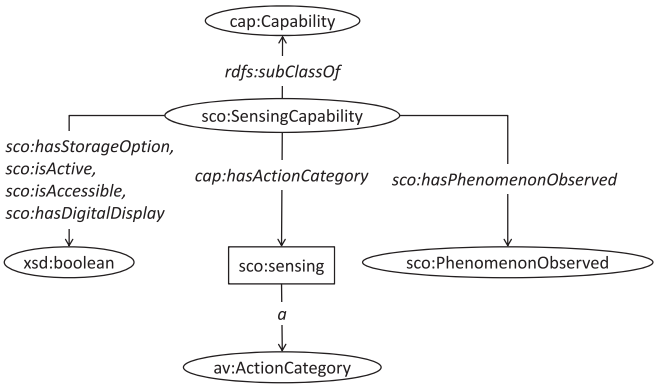

FIGURE 1. Sensor capability ontology.

(ii) *sco:hasStorageOption* (line 22) is defined as a property that has a boolean value. It reports if the sensor has any storage option. This property can be modified to report on the size of the storage capacity that the sensor has.

(iii) *sco:isAccessible* (line 26) is defined as a property that has a boolean value. It reports if the sensor is accessible. That helps to take decisions to physically move and check the status of the sensor or read directly from its digital display if it has one.

(iv) *sco:hasDigitalDisplay* (line 30) is defined as a property that has a boolean value. It reports if the sensor has a digital display that a user can read from.

(v) *sco:hasPhenomenonObserved* (line 34) is defined as a property that has a string value. It reports on the phenomenon that the sensor is observing. A listing of possible values is defined as a Datatype (line 39).

A sensor capability is created as an instance of (*rdf:type*) *sco:SensingCapability* with concrete values of its predefined properties. Listing 1 presents an example of a temperature sensor capability *:TemperatureSensorCapability123* reporting that it is an active and accessible temperature sensor with a digital display and it does not have a storage option.

```
1  @prefix sco: <http://.../sensor_capability_ontology#>.
2
3  :TemperatureSensorCapability123 a sco:SensingCapability;
4              sco:isActive "true"^^xsd:boolean ;
5              sco:hasStorageOption "false"^^xsd:boolean ;
6              sco:isAccessible "true"^^xsd:boolean ;
7              sco:hasDigitalDisplay "true"^^xsd:boolean ;
8              sco:hasPhenomenonObserved "Temperature"^^sco:
                   PhenomenonObserved.
```

Listing 1. Snippet of a temperature sensor capability.

Compared with existing approaches, our capability model presents several advantages. Mainly, it explicitly captures domain-specific functional properties that describe and characterize the carried action according to the aspects of interest to end-users and targeted applications. Indeed, these properties are defined in domain-specific ontologies with respect to specific engineering tasks. Moreover, our domain-specific capability models are easily extensible. If a new property is required for describing a particular sensor aspect/characteristic, we simply need to define it as a *cap:PropertyDeclaration* and define its domain and range. Finally, our feature-based model enables new techniques for indexing and discovering services as detailed in the following.

The main research problem we are dealing with in this paper is the organization of sensor capabilities in order to facilitate their discovery. We have previously proposed other approaches that use a heuristic search for discovering capabilities from an indexing structure created based on the analysis of the property values [19, 20]. In this paper, we propose another approach which is also build on top of our feature-based model and uses FCA to construct an indexing structure that organizes sensor capabilities. We use this structure for the purpose of discovery.

```
1  @prefix sco: <http://vocab.deri.ie/sco#>.
2  @prefix cap: <http://vocab.deri.ie/cap#>.
3  @prefix av: <http://vocab.deri.ie/av#> .
4  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
6  @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
7  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
8  @prefix owl: <http://www.w3.org/2002/07/owl#>.
9
10 sco:SensingCapability rdfs:subClassOf cap:Capability .
11
12 sco:sensing a av:ActionCategory ;
13 rdfs:comment "Measuring a physical quantity and converts it into a signal
       which can be read by an observer."^^xsd:string ;
14 skos:prefLabel "Sensing"^^xsd:string .
15
16 sco:SensingCapability cap:hasActionCategory sco:sensing
17
18 sco:isActive a cap:PropertyDeclaration;
19     rdfs:domain sco:SensingCapability;
20     rdfs:range xsd:boolean.
21
22 sco:hasStorageOption a cap:PropertyDeclaration;
23     rdfs:domain sco:SensingCapability;
24     rdfs:range xsd:boolean.
25
26 sco:isAccessible a cap:PropertyDeclaration;
27     rdfs:domain sco:SensingCapability;
28     rdfs:range xsd:boolean.
29
30 sco:hasDigitalDisplay a cap:PropertyDeclaration;
31     rdfs:domain sco:SensingCapability;
32     rdfs:range xsd:boolean.
33
34 sco:hasPhenomenonObserved a cap:PropertyDeclaration;
35     rdfs:label "hasPhenomenonObserved";
36     rdfs:domain sco:SensingCapability;
37     rdfs:range sco:PhenomenonObserved.
38
39 sco:PhenomenonObserved a rdfs:Datatype;
40             rdfs:comment "An observed phenomenon can be light,
                   motion temperature, etc." ;
41             owl:onDatatype xsd:string;
42             owl:withRestrictions ("Light"^^xsd:string "Motion"^^xsd:
                   string "Temperature"^^xsd:string "Energy"^^xsd:
                   string).
```

Listing 2. Snippet of SCO: sensor capability ontology.

4. FCA FOR ORGANIZING SENSOR CAPABILITIES

The approach that we adopt in this work consists of using FCA [2] for better organizing a repository of capabilities in order to

make their discovery more efficient. We define in this paper the theoretical foundations of FCA while applying it on sensor capabilities. We use FCA in Section 4.1 for creating a concept lattice, a structure that allows for indexing sensor capabilities. Then in Section 4.2, we discuss how a discovery mechanism can be implemented using this concept lattice.

4.1. Creating the concept lattice

FCA is a technique that has evolved from mathematical lattice theory and has been used for data analysis across several domains. Examples of domains include organizing web search results into concepts based on common topics, gene expression data analysis, IR, understanding and analysis of source codes, etc. [21]. It represents a powerful tool for identifying meaningful relationships within a set of objects that share common attributes. It provides as well a theoretical model to build from a *formal context* (see Definition 1) a partially ordered structure called a *concept lattice*.

DEFINITION 1 (Formal Context). *A formal context $\mathcal{FC}$ is a triplet $\langle X, Y, R\rangle$, where X and Y are non-empty sets and $R \subseteq X * Y$ is a binary relation between X and Y.*

For a formal context $\mathcal{FC}$, elements $x \in X$ are referred to as objects and elements $y \in Y$ are called attributes. $\langle x, y\rangle \in R$ denotes that the object x has the attribute y.

In our work, the formal context is actually defined via the set of sensors that we have as well as their respective descriptions. We will use Table 1 (called cross-table) as a running example which describes the relationship between the objects (i.e. Sensors 1–5 represented by the table rows: X = {Sensor 1, Sensor 2, Sensor 3, Sensor 4, Sensor 5}) and their descriptions (i.e. attributes represented by the table columns: Y = {Active, Storage Option, Digital Display, Accessible}, and Table 1). We consider in this example four attributes:

(i) *Active* that indicates if the sensor is in operation;
(ii) *Storage Option* that indicates if the sensor has the possibility to store data on it;
(iii) *Digital Display* that indicates if the sensor is equipped by a digital display for displaying the data; and
(iv) *Accessible* that indicates if the sensor is located in an accessible area.

Another fundamental concept in FCA is the *Formal Concept*. This concept is defined in Definition 2.

DEFINITION 2 (Formal Concept). *A formal concept in $\langle X, Y, R\rangle$ is a pair $\langle E, I\rangle$ of $E \subseteq X$ (called extent) and $I \subseteq Y$ (called intent) such that* $\mathrm{Att}(E) = I$ *and* $\mathrm{Obj}(I) = E$.

$\mathrm{Att}(E)$ *is an operator that assigns subsets of X to subsets of Y, such that* $\mathrm{Att}(E)$ *is the set of all attributes shared by all objects from E.* $\mathrm{Obj}(I)$ *is an operator that assigns subsets of Y to subsets of X, such that* $\mathrm{Obj}(I)$ *is the set of all objects sharing all the attributes from I.*

TABLE 1. Data table with binary attributes.

	Active	Storage option	Digital display	Accessible
Sensor 1	X	X	X	X
Sensor 2	X		X	X
Sensor 3		X	X	X
Sensor 4		X	X	X
Sensor 5	X			

From Definition 2, we can conclude that a concept $C = \langle E, I\rangle$ is created by getting objects from E sharing the same attributes from I. For example, the shaded rectangle in Table 1 represents a formal concept $\langle E_1, I_1\rangle$ = ⟨{Sensor 1, Sensor 2, Sensor 3, Sensor 4}, {Digital Display, Accessible}⟩ because $\mathrm{Att}(E_1) = \{DigitalDisplay, Accessible\}$ and $\mathrm{Obj}(I_1)$ = {Sensor 1, Sensor 2, Sensor 3, Sensor 4}.

From a *formal context* $\mathcal{FC} = \langle X, Y, I\rangle$, we can deduce a set of formal concepts that can be ordered with respect to a subconcept ordering. Definition 3 formally introduces the subconcept ordering.

DEFINITION 3 (Subconcept Ordering). *Having two formal concepts $\langle E_1, I_1\rangle$ and $\langle E_2, I_2\rangle$ from $\mathcal{FC} = \langle X, Y, R\rangle$, $\langle E_1, I_1\rangle \leq \langle E_2, I_2\rangle \iff E_1 \subseteq E_2$ ($\iff I_2 \subseteq I_1$).*

Let us consider the following formal concepts from the example of Table 1:

$\langle E_1, I_1\rangle$ = ⟨{Sensor 1, Sensor 2, Sensor 3, Sensor 4}, {Digital Display, Accessible}⟩
$\langle E_2, I_2\rangle$ = ⟨{Sensor 1, Sensor 2, Sensor 4}, {Digital Display, Accessible}⟩,
$\langle E_3, I_3\rangle$ = ⟨{Sensor 1, Sensor 2}, {Active, Digital Display, Accessible}⟩,
$\langle E_4, I_4\rangle$ = ⟨{Sensor 1, Sensor 2, Sensor 5}, {Active}⟩.

Then
$\langle E_3, I_3\rangle \leq \langle E_1, I_1\rangle$, $\langle E_3, I_3\rangle \leq \langle E_2, I_2\rangle$,
$\langle E_3, I_3\rangle \leq \langle E_4, I_4\rangle$ and $\langle E_2, I_2\rangle \leq \langle E_1, I_1\rangle$.

The set of ordered formal concepts derived from a formal context is called a *concept lattice*, which is another important notion in FCA. A concept lattice can be represented into a graph such as the one depicted in Fig. 2.[1] In this figure, the concept extent near the bottom of the lattice contains only *Sensor 1* since the corresponding intent is related to the biggest number of attributes. The top concept contains all the sensors and its intent corresponds to no attribute. This makes the concept less interesting as it allows for all possible combinations of attributes.

So far, we considered binary attributes (i.e. either the object has or not that attribute). However, in real settings when

[1] All concept lattices in this paper are created using Conexp. [http://conexp.sourceforge.net/].

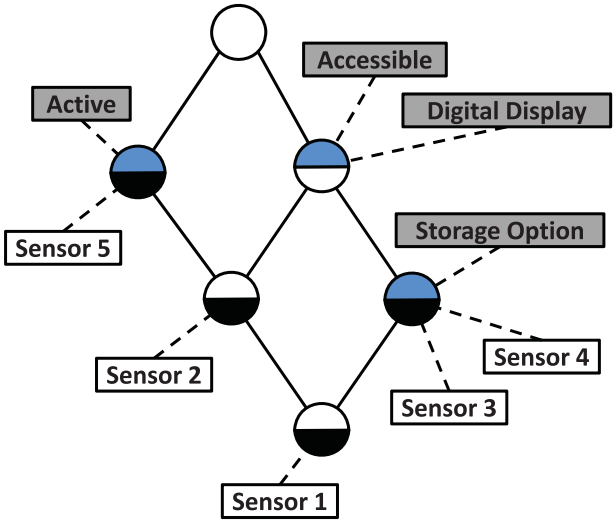

FIGURE 2. Concept lattice of the example of Table 1.

TABLE 2. Data table with a multi-valued attribute.

	Active	Storage option	Digital display	Accessible	Phenomenon observed
Sensor 1	X	X	X	X	Energy
Sensor 2	X		X	X	Energy
Sensor 3		X	X	X	Light
Sensor 4		X	X	X	Temperature
Sensor 5	X				Motion

describing capabilities, there are also multi-valued attributes. Consider Table 2, this table contains an additional attribute *Observed Phenomenon*. This attribute reports whether the sensor is an *Energy* consumption sensor, *Light* detection sensor, *Temperature* sensor or a *Motion* sensor. In this case, we need to transform this multi-valued attribute into a binary attribute.

For the usage of FCA, transforming and preprocessing the data displayed in Table 2 is needed. One of the possible ways consists in using scaling method. Scaling is a transformation method that converts a multi-valued attribute into a context. Table 3[2] represents the transformation of the multi-valued attribute *Phenomenon Observed* into a context.

After the application of FCA on converted tables, the resulted lattice is depicted in Fig. 3.

This concept lattice is our indexing structure; it allows organizing sensor capabilities in a tree. This structure can serve for the discovery of sensors as described in the following subsection.

[2]Please note that PO stands for Phenomenon Observed.

TABLE 3. Data table with a scaled multi-valued attribute.

	PO: Energy	PO: Light	PO: Temperature	PO: Motion
Sensor 1	X			
Sensor 2	X			
Sensor 3		X		
Sensor 4			X	
Sensor 5				X

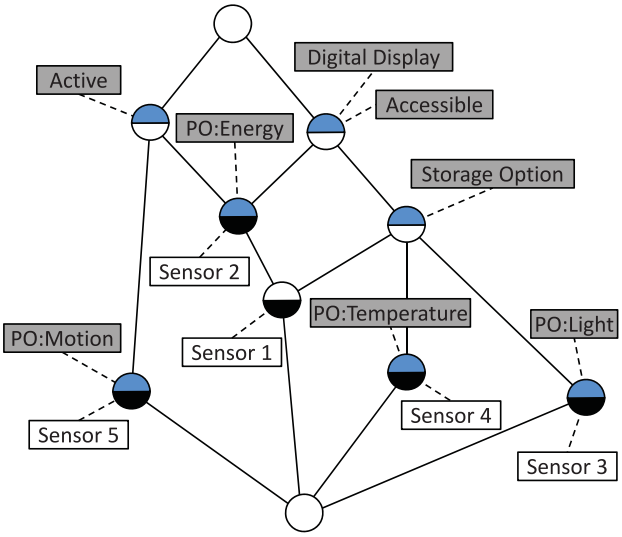

FIGURE 3. Concept lattice of the example of Table 2.

4.2. Concept lattice for sensor and knowledge discovery

In the following, we show the usefulness of using FCA for indexing sensor descriptions via two scenarios. The first is the discovery of sensors and the second is the discovery of implications between sensor attributes.

We propose Algorithm 1 for the discovery of sensors satisfying a set of attributes. It takes as input the concept *Lattice* and a set of *attributes* representing the query. Suppose that our input Lattice is the one depicted in Fig. 4 and the input attributes are 'PO:Energy', 'Digital Display' and 'Storage Option'. In the following, we explain how Algorithm 1 operates:

(1) Lines 2–6: finding the set of formal concepts with an intent that contains the input attributes. The result of this step, as shown in Fig. 4, is the set of formal concepts **FC1**, **FC2** and **FC3**.
(2) Line 7: finding the Highest Common Subconcept of the concepts identified in the first step. In Fig. 4, this can be determined by following the lines down from FC1, FC2 and FC3 and stop where they meet. The result is **FC4**.
(3) Lines 8–16: collecting the set of potential candidates of the query. Every object in the formal concept identified

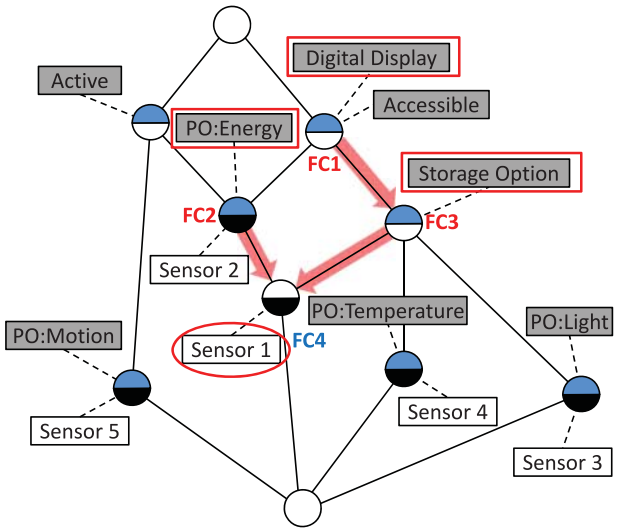

FIGURE 4. Discovering sensors with same attributes.

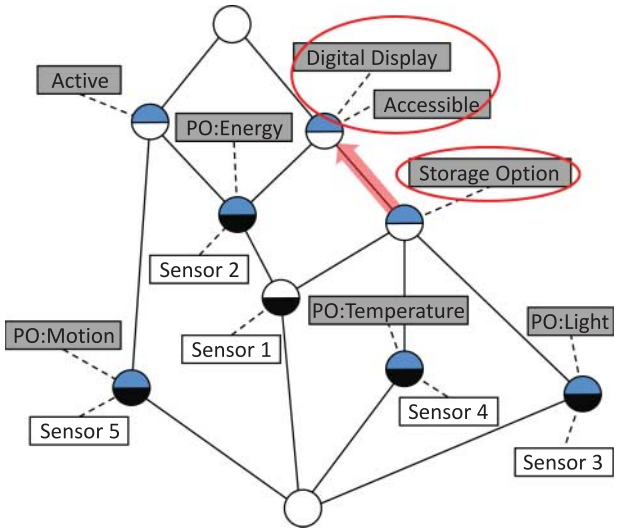

FIGURE 5. Implication: every sensor that has a 'Storage Option' is 'Accessible' and has a 'Digital Display'.

in Step 2 as well as all its subconcepts down to the bottom of the lattice are potential candidates for the input query. In Fig. 4, starting from FC4, 'Sensor 1' is the only result for our input query as there are no subconcepts of FC4 with a non-empty extent.

Algorithm 1 Sensor discovery algorithm.

```
1:  function DISCOVER(Lattice, Attributes)
2:      Concepts ← null
3:      for (Attribute ∈ Attributes) do
4:          Concepts.add(Lattice.
5:              findConceptWithAttribute(Attribute) )
6:      end for

7:      Concept ← FindHCSubC(Lattice,Concepts)

8:      SubConcepts ← Concept.getSubConcepts()
9:      Sensors ← null
10:     while SubConcepts.size() ≠ 0 do
11:         OneConcept ← SubConcepts.getConcept()
12:         Sensors.addAll(OneConcept.getObjects())
13:         SubConcepts.addAll(
14:             OneConcept.getSubConcepts())
15:         SubConcepts.remove(Concept)
16:     end while
17:     return Sensors
18: end function
```

The proposed algorithm relies mainly on the explicit relations between formal concepts. This is useful to discover sensors that share similar attributes. For example, if one of the motion sensors M is not active anymore, it is possible to use one of the other motion sensors in its equivalence class or discovering sensors that share its attributes (i.e. attributes of the sensor M) by using them as input for Algorithm 1.

In the context of replacement of a sensor, this method reduces the change time[3] considerably to simply parsing the lattice until reaching the required equivalence class and select one of its sensors rather than performing a full search over the set of all the available sensors.

This also helps avoid having empty results. In fact, during the navigation of the concept lattice, if the user cannot find the equivalence class that satisfies his request, he can adapt it according to the visited nodes of the lattice. This allows the user to relax his query by reducing the attributes he initially identified in his request.

The other advantage of using FCA is the presence of the explicit subconcept relationship between equivalence classes. This allows to discover additional knowledge among the objects' attributes that are analysed (i.e. sensor attributes). Indeed, as depicted in Fig. 5, we can discover implications such as: every sensor that has a 'Storage Option' is also 'Accessible' and has a 'Digital Display'. In other words: 'Storage Option' implies 'Accessible' and 'Digital Display'.

To conclude, it is important to note that the use of FCA permits to create a concept lattice uniformly. In other words, we always create the same structure with the same input objects. This has the advantage of creating a deterministic discovery algorithm, as there is no need to use any heuristic for parsing this indexing structure. In this paper, we are focusing mainly on the creation of the concept lattice and we study its applicability

[3]Change time: the required time for selecting a replacement sensor for the disabled one.

in our domain: i.e. indexing a set of sensor capabilities. The development and implementation of a complete discovery algorithm is part of our future work. We have used FCA in real settings for organizing sensor capabilities and our experimental settings are described in detail in Section 5.

5. EXPERIMENTATION

In our experimentation, we developed multiple modules for creating a concept lattice starting from an RDF description of sensor services. The workflow as well as the data exchanged between the various modules is shown in Fig. 6.

The first developed tool is the RDF parser. It is a Java application that uses Apache Jena Framework, a Java RDF and Semantic Web library. This module takes as input an RDF file and produces a text file containing a formal context as a table with multi-valued attributes. This module is still under development as we intend to support getting data from an RDF store and not simply an RDF file. It is custom-made for this application; it can manipulate only sensor descriptions that follow the vocabularies introduced previously. The second module of our prototype is another Java application that performs the scaling operation described in Section 4. It starts by checking all the attributes that are not boolean (see Listing 2 for an example) and then it considers each of its values as a separate attribute and assigns a boolean value (i.e. true) to the corresponding objects. The output of this module is a textual file compatible with the Conexp tool format [22] that we use in the following step.

The third step in our work consists of creating the concept lattice with one of the following options: (i) Using Conexp [22] for the creation and visualization of the concept lattice. (ii) Using Colibri-Java [23] for the creation and the analysis of the resulting concept lattices. We use the first option for applying

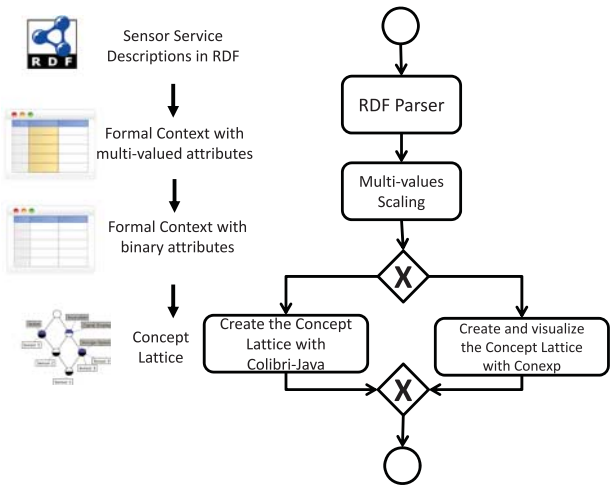

FIGURE 6. Creating a concept lattice from RDF descriptions of sensor services.

our approach in a real-world scenario that will be described in detail in Section 5.1 and the second option for carrying out further statistical analysis on the use of FCA in our domain application in Section 5.2.

5.1. Use case application

We illustrate in this section a use case scenario using a set of real-world sensors deployed within the LEI dataspace. LEI is an ecosystem where energy-related data are made available and interlinked to support decision-making and ultimately energy consumption friendly behaviour [5]. Such data are provided by real-time data sources such as sensors as well as relatively static background knowledge such as building plan and occupancy. The LEI dataspace has been realized in the DERI.

DERI is a premier research institute with ∼130 research students and staff with a worldwide reputation in its area. It is based in a dedicated building with 2190 m^2 of space, comprising 22 unit offices, 160 open plan workspaces, 1 large 80-seat conference room with audio-visual and video conferencing facilities, 4 meeting rooms, 3 kitchens, 1 air-conditioned data centre with a backup generator, 1 sensor network laboratory, a 30-person café, and Ireland's National Museum of Computing History.

There are various sources of power consumption in DERI such as Heating, Ventilation and Air Conditioning systems, lights and electronic devices. The building provides first-class technical infrastructure to its researchers.

The DERI building has been retrofitted with energy sensors to monitor the consumption of power within the building. In total, there are over 50 fixed energy-consumption sensors covering office space, café, data centre, kitchens, conference and meeting rooms, computing museum along with over 20 mobile sensors for devices, light and heater energy consumption as well as light, temperature and motion detection sensors. A building-specific aspect of the dataspace has been presented in [3] with a sensor network-based situation awareness scenario presented in [4]. For this work, we ended up with a total number of 78 sensors.

These sensors are described via a set of attributes:

(i) Active: This attribute reports whether the sensor is in operation.
(ii) Observed Phenomenon: we have four observed phenomena which are 'energy and power consumption', 'motion', 'light' and 'temperature'. This attribute is a multi-valued attribute that needs to be scaled using the transformation previously given in Table 3.
(iii) Protocol: This attribute indicates the protocol used by the sensor. We have in our selection of sensors two possible protocols: UDP used by electricity and power consumption sensors and CoAP used by other sensors. This is a multi-valued attribute that has to be scaled.

(iv) Electricity Phases: This attribute reports on the electricity phases used by the sensor; we have in our use case two options: 3-phase and 1-phase sensors. Again, this is a multi-valued attribute that has to be scaled.

(v) Location: even though this attribute is not an intrinsic property of the sensor, we have used it because it is important information that is required for processing the data provided by the sensor. This is also a multi-valued attribute that enumerates the locations of the sensors, e.g. first floor: west wing, ground floor: canteen, etc. that need to be scaled.

As previously mentioned, the advantage of our capability model is that we can easily extend the domain ontology to add more attributes. The current use case requires additional attributes: Protocol, Elasticity Phases and Location that are added to the domain ontology shown previously in Listing 2. Current changes to this domain ontology are shown in Listing 3. In this listing, we use geo as a namespace for referring to an existing RDF vocabulary for representing information about spatially located things, using WGS84 as a reference datum [24]. It is also possible to customize further this attribute, for example, to the rooms vocabulary [25] if locations of sensors are limited to predefined rooms.

```
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

sco:hasProtocol a cap:PropertyDeclaration;
    rdfs:domain sco:SensingCapability;
    rdfs:range sco:Protocol.

sco:Protocol a rdfs:Datatype;
    rdfs:comment "A protocol can be either COAP or UDP" ;
    owl:onDatatype xsd:string;
    owl:withRestrictions ("UDP"^^xsd:string "COAP"^^xsd:string).

sco:hasElasticityPhases a cap:PropertyDeclaration;
    rdfs:domain sco:SensingCapability;
    rdfs:range sco:ElasticityPhases.

sco:ElasticityPhases a rdfs:Datatype;
    rdfs:comment "Elasticity Phases considered are 3−phase or 1−phase" ;
    owl:onDatatype xsd:string;
    owl:withRestrictions ("3−phases"^^xsd:string "1−phase"^^xsd:string).

sco:hasLocation cap:PropertyDeclaration;
    rdfs:domain sco:SensingCapability;
    rdfs:range geo:SpatialThing.
```

Listing 3. Snippet of SCO with the required extensions for our Use Case.

All the sensor capabilities were automatically generated from an Excel file containing the original descriptions that were manually checked. Manually checking RDF descriptions was possible as the number of sensors used was not huge. We have not carried out any evaluation of our RDF parser, because it is custom-made for our dataset and conceptual model. The correctness of the algorithm we applied for the RDF parser is out of the scope of this paper; however, the data has been manually verified after parsing and scaling.

The resulting concept lattice from Conexp [22] is depicted in Fig. 7. The top concept in this lattice represents the set of all active sensors $\langle\{\text{Sensor 1}, \text{Sensor 2}, \ldots, \text{Sensor 78}\}, \{\text{Active}\}\rangle$. This formal concept contains in its extent all the sensors of our dataset because they are all active. We can see in this concept lattice several formal concepts that represent the set of motion sensors $\langle\{\text{Sensor 61}, \ldots, \text{Sensor 66}\}, \{\text{OP: Motion}\}\rangle$, the formal concept for temperature sensors $\langle\{\text{Sensor 67}, \ldots, \text{Sensor 72}\}, \{\text{OP:Temperature}\}\rangle$ and the light sensors $\langle\{\text{Sensor 73}, \ldots, \text{Sensor 78}\}, \{\text{OP: Light}\}\rangle$. These three formal concepts are all sub-concepts of the concept $\langle\{\text{Sensor 61}, \ldots, \text{Sensor 78}\}, \{\text{1st Floor: East Wing}\}\rangle$. This helps to deduce that all motion, temperature and light sensors are in the same location, i.e. First Floor:East Wing.

5.2. Evaluation

To evaluate the applicability of our approach in highly dynamic environments, we carried out two experiments highlighting mainly the efficiency of using FCA in terms of the number of concepts created in a concept lattice given a formal context and the time required to build it. Throughout this evaluation, we reused an existing implementation of FCA in Java, namely Colibri-Java [23]. Colibri-Java is a library that offers the required tools from the preparation of the context to the creation of a concept lattice that has been experienced in [26].

Experiment 1: Context size vs. Lattice size The object of this first experiment is to analyse the size of the generated concept lattice with respect to its original formal context and find out the limits of using FCA in our domain application.

During this experiment, we wanted to verify the correlation between the context size and the corresponding lattice size. We randomly created multiple sets of sensor capabilities. For each set we generated its corresponding concept lattice. In terms of attributes, we considered a total number of 16 attributes for describing each sensor capability with three different coverage ratios ranging from 0 to 60, 50 or 30% (i.e. each sensor capability has between [0,10], [0,8] or [0,5] attributes, respectively).

Figure 8 shows the results of our evaluation. On the horizontal axis, we present the size of the original formal context that goes from 0 to 970 objects and on the vertical axis, we present the size of the corresponding concept lattice. From this figure, we can clearly note that concept lattices grow considerably in size with respect to their context. This leads us to the conclusion that if we want to consider applying our approach in an environment with a big number of sensors, it would not be very easy for an end-user to visualize the different concept classes generated. On the same figure, we can also note the impact of the coverage ratio. We can see that the concept lattice gets larger when large contexts have bigger

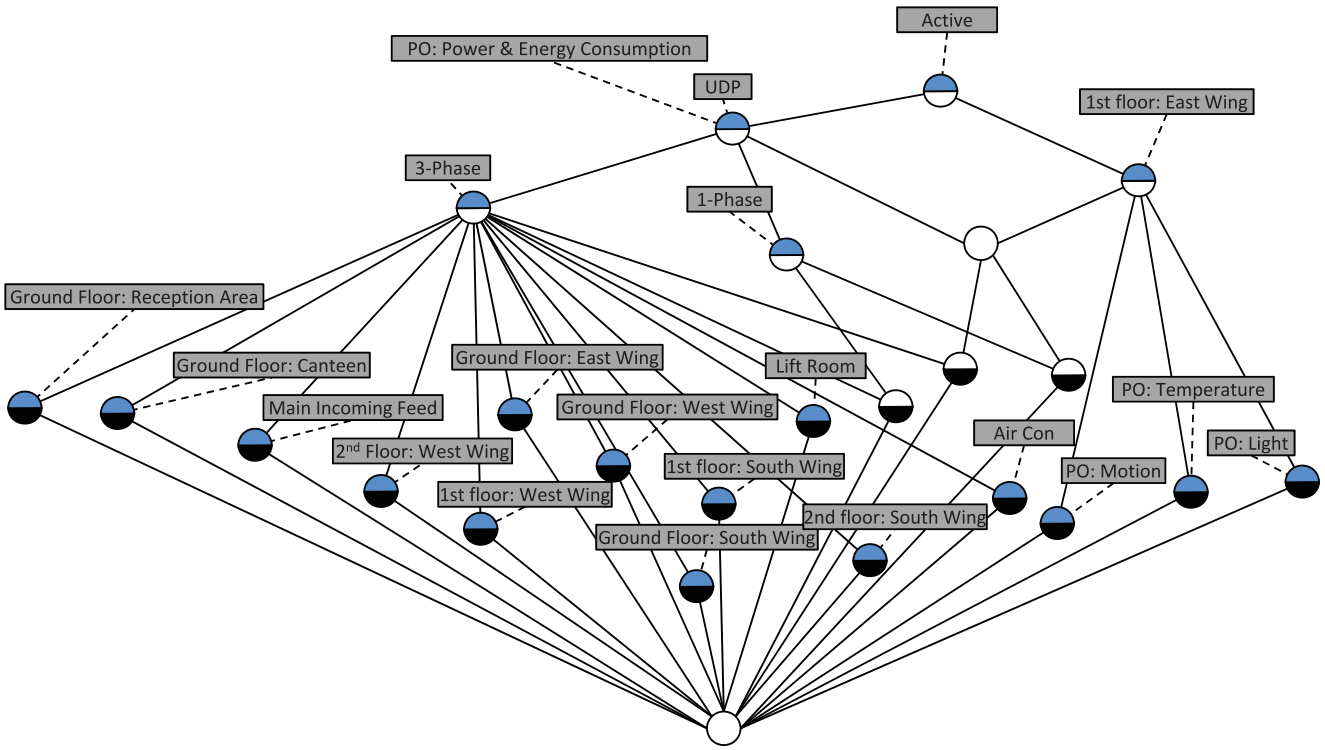

FIGURE 7. Concept lattice of the LEI-DERI use case.

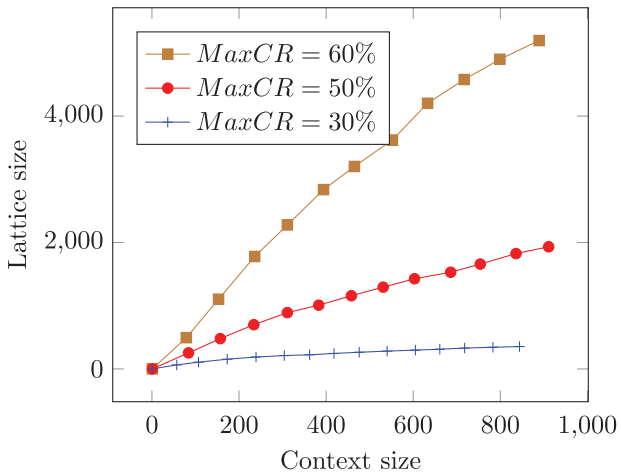

FIGURE 8. Context size vs. lattice size vs. maximum coverage ratio (max CR).

coverage ratio. This means that when describing sensors, we have to avoid over-describing them and carefully choose the most discriminating attributes.

Experiment 2: Lattice size vs. Construction Time The object of this second experiment is to measure the required time for creating a concept lattice with respect to its size in order to verify the applicability of our approach at a large-scale (big number of sensor capabilities) and dynamic environments.

Please note that, for these experiments, we ignored the required time for creating a context starting from the RDF descriptions of sensor capabilities. It focuses only on the computation time required for the creation and parsing of a concept lattice. In other words, in these experiments, we focus on the third step of the diagram depicted in Fig. 6.

We randomly created multiple sets of sensor capabilities with a fixed coverage ratio of the attributes between 0 and 50% (each object has between 0 and 8 out of a total of 16 attributes). For each set, we generated its corresponding concept lattice and measured the required construction time.

Figure 9 shows the results of our evaluation. On the horizontal axis, we present the size of the concept lattice and on the vertical axis, we present the required time for its construction. From this figure, we can clearly note that the required construction time grows exponentially depending on the size of the concept lattice. However, for a concept lattice with over 5000 concepts, the construction time is still <200 ms. This time can be considered acceptable in small- or medium-sized buildings where decision-making can be postponed until the data have been updated within a few seconds. Nevertheless, in highly sensitive environments, even a few milliseconds can have a huge impact.

The focus of our work is on applying our approach in environments similar to DERI where the number of concepts does not exceed 1000. In such settings, as we can see from

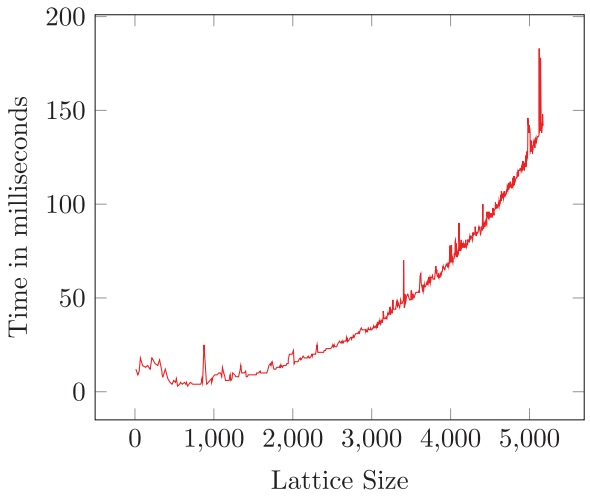

FIGURE 9. Lattice size [0–5000] vs. Construction time in milliseconds.

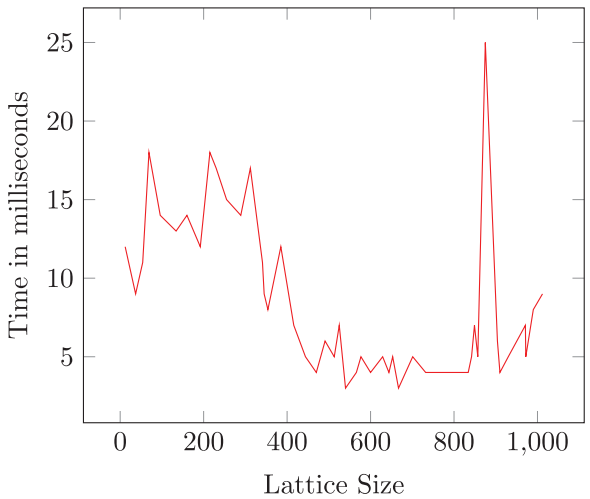

FIGURE 10. Lattice size [0–1000] vs. Construction time in milliseconds.

Fig. 10, the maximum construction time can reach only 25 ms. Even though the main criticism towards our approach is the fact of reconstructing the concept lattice for any change in the environment (e.g. a new sensor, change in a sensor attribute, etc.), we can consider that this remains acceptable with such low construction time.

6. CONCLUSION

We propose in our work to model capabilities as attribute-featured entities where each attribute reports on a particular characteristic of the described action. Our conceptual model is flexible enough to consider even non-functional attributes to include, for instance, quality of service attributes. We applied this approach for modelling sensor capabilities, from a real-world dataspace, featuring both functional and non-functional attributes. On top of these sensor descriptions, we used FCA for indexing these sensors. The resulting indexing structure is called concept lattice that can serve for several use cases, for example, the discovery of a replacement sensor. Using FCA in such a use case is recommended only if the number of objects (i.e. sensors) is not very big. Actually, this constitutes the major disadvantage of our approach. However, as seen in Section 5.2, this approach remains efficient while applied on a set of sensor services in a medium-sized building such as DERI.

Our experiments show that reconstruction of the entire concept lattice while managing 5000 sensors does not exceed a few milliseconds. This allows us to conclude that when dealing with dynamic environments, our system should reconstruct the entire concept lattice either regularly or when a change in the environment has been detected. This method can be efficient but it is very costly and could not be very useful in highly dynamic environments when several sensors could be added and removed frequently or where the values of attributes might change after a short period. Actually, as part of our future work, we plan to provide the required algorithms for updating this indexing structure (i.e. removing or adding a sensor description) in order to deal with dynamic changes in the environment.

The current version of our work supports only literal attribute values (e.g. boolean, integer, float and string). However, our conceptual model supports other complex attribute types [1] such as conditional values, enumeration values, dynamic values, etc. To resolve this issue, we can simply transform any attribute type into boolean without considering its actual value. This method can be useful for a rapid construction of the concept lattice and spends less effort in the scaling operation, but we can lose in terms of expressiveness of the concept lattice. Our future work includes also investigating other scaling operations for covering complex attribute types.

FUNDING

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

REFERENCES

[1] Bhiri, S., Derguech, W. and Zaremba, M. (2012) Modelling Capabilities as Attribute-Featured Entities. *Web Information Systems and Technologies—8th International Conference, WEBIST 2012—Revised Selected Papers*, Porto, Portugal, April 18–21, Lecture Notes in Business Information Processing, Vol. 140, pp. 70–85. Springer, Berlin, Heidelberg.

[2] Ganter, B. and Wille, R. (1999) *Formal Concept Analysis—Mathematical Foundations*. Springer, Berlin.

[3] Curry, E., O'Donnell, J., Corry, E., Hasan, S., Keane, M. and O'Riain, S. (2013) Linking building data in the cloud:

Integrating cross-domain building data using linked data. *Adv. Eng. Inf.*, **27**, 206–219.

[4] Hasan, S. and Curry, E. (2014) Approximate semantic matching of events for the internet of things. *ACM Trans. Internet Technol.*, **14**, 2:1–2:23.

[5] Curry, E., Hasan, S. and O'Riain, S. (2012) Enterprise Energy Management using a Linked Dataspace for Energy Intelligence. *Sustainable Internet and ICT for Sustainability (SustainIT)*, Pisa, Italy, October 4–5, pp. 1–6. IEEE.

[6] OASIS reference model for service oriented architecture 1.0. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf (accessed May 25, 2014).

[7] WSMO: Web service modelling ontology. http://www.wsmo.org/ (accessed May 25, 2014).

[8] OWL-S: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/ (accessed May 25, 2014).

[9] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C. and Fensel, D. (2005) Web service modeling ontology. *Appl. Ontology*, **1**, 77–106.

[10] SA-WSDL: Semantic Annotations for WSDL. http://www.w3.org/2002/ws/sawsdl/ (accessed May 25, 2014).

[11] SA-REST: Semantic annotation of web resources. http://www.w3.org/Submission/SA-REST/ (accessed May 25, 2014).

[12] WSDL: Web Services Description Language. http://www.w3.org/TR/wsdl/ (accessed May 25, 2014).

[13] Martin, D., Paolucci, M. and Wagner, M. (2007) Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective. *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, Busan, Korea, November 11–15, Lecture Notes in Computer Science 4825, pp. 340–352. Springer, Berlin, Heidelberg.

[14] Oaks, P., ter Hofstede, A.H.M. and Edmond, D. (2003) Capabilities: Describing What Services Can Do. *Service-Oriented Computing—ICSOC 2003, First International Conference*, Trento, Italy, December 15–18, Lecture Notes in Computer Science 2910, pp. 1–16. Springer, Berlin Heidelberg.

[15] Dong, X., Halevy, A.Y., Madhavan, J., Nemes, E. and Zhang, J. (2004) Similarity Search for Web Services. *(e)Proc. 30th Int. Conf. on Very Large Data Bases*, Toronto, Canada, August 31–September 3, pp. 372–383. Morgan Kaufmann.

[16] Klusch, M., Fries, B. and Sycara, K.P. (2006) Automated Semantic Web Service Discovery with OWLS-MX. *5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, Hakodate, Japan, May 8–12, pp. 915–922. ACM, New York, NY, USA.

[17] Zhou, Z., Gao, F. and Shu, L. (2011) Service Protocol Replaceability Assessment in Mediated Service Interactions. *Proc. IEEE Int. Conf. on Communications, ICC 2011*, Kyoto, Japan, June 5–9, pp. 1–5. IEEE.

[18] Zhou, Z., Gaaloul, W., Shu, L., Tata, S. and Bhiri, S. (2013) Assessing the replaceability of service protocols in mediated service interactions. *Future Gener. Comput. Syst.*, **29**, 287–299.

[19] Gao, F. and Derguech, W. (2012) Ubiquitous Service Capability Modeling and Similarity-Based Searching. *Web Information Systems Engineering—WISE 2011 and 2012 Workshops—Combined WISE 2011 and WISE 2012 Workshops, Revised Selected Papers*, Sydney, Australia, November 28–30, Lecture Notes in Computer Science 7652, pp. 173–184. Springer, Berlin, Heidelberg.

[20] Derguech, W. and Bhiri, S. (2013) Modelling, Interlinking and Discovering Capabilities. *ACS Int. Conf. on Computer Systems and Applications, AICCSA 2013*, Ifrane, Morocco, May 27–30, pp. 1–8. IEEE.

[21] Belohlávek, R. and Vychodil, V. (2010) Background Knowledge in Formal Concept Analysis: Constraints Via Closure Operators. *Proc. 2010 ACM Symposium on Applied Computing (SAC)*, Sierre, Switzerland, March 22–26, pp. 1113–1114. ACM.

[22] Conexp. http://sourceforge.net/projects/conexp/ (accessed: May 25, 2014).

[23] Colibri-java. (accessed May 25, 2014).

[24] Basic geo (wgs84 lat/long) vocabulary. http://www.w3.org/2003/01/geo/ (accessed May 25, 2014).

[25] Cyganiak, R. Buildings and rooms vocabulary. http://vocab.deri.ie/rooms (accessed May 25, 2014).

[26] Lindig, C. and Gbr, G.D. (2000) Fast Concept Analysis. *Working with Conceptual Structures—Contributions to ICCS 2000*, Aachen, Germany, August, pp. 152–161. Shaker.