Ontology-based Rules for Recommender Systems

Jeremy Debattista, Simon Scerri, Ismael Rivera, and Siegfried Handschuh

Digital Enterprise Research Institute, National University of Ireland, Galway
firstname.lastname@deri.org

Abstract. Nowadays, smart devices perceive a large amount of information from device sensors, usage, and other sources which contribute to defining the user's context and situations. The main problem is that although the data is available, it is not processed to help the user deal with this information easily. Our approach is based on the assumption that, given that this information can be unified in a single personal data space, it can be used to discover and learn rules to provide the user with personal recommendations. In this paper we introduce a Rule Management Ontology to support the representation of event-based rules that trigger specific actions. We also discuss how a context listener component can provide recommendations based on the perceived context-data, or in the future, semi-automatically learnt rules.

1 Introduction

With the increasing popularity of smart devices, recommender systems can be useful in providing solutions based on real-time context information perceived by such devices in a ubiquitous environment. As McDonald pointed out, recommender systems "mediate the user experience in the digital world, and they will be increasingly helpful in performing the same role in the physical world, thereby filling an important gap in ubiquitous computing" [11]. Through sensors and device usage information, smart devices can be enabled to continuously accrue context information about repetitive daily tasks performed by the user; such as changing the mobile mode to silent when arriving at work, changing online status during a meeting, and opening certain applications and documents when in presence of other peers.

Data collected by smart devices is usually domain-specific to the application handling it. Having useful information in different domains and in different formats means that the heterogenous context data collected is not unified under one model. *Semantic heterogeneity* is thus another problem limiting the power of smart devices. Although different devices and their embedded sensors could be used for a common objective, their use of non-standard data formats means that the context information that they gather cannot be unified in an interoperable representation. This limitation, associated with the lack of a common domain model, is tackled in the di.me[1] project. The project targets the unification of the user's personal information across various heterogeneous sources, such as social networks and device sensors, into one personal data cloud. This is done with the aim of assisting the user in daily tasks. In the context of di.me, we

[1] http://www.dime-project.eu

restrict our working knowledge base to cover a personal closed-world environment; by extending and introducing models that enable the representation of the user's entire Personal Information Model (PIM). Unlike in Social Semantic Desktops [14], the PIM in di.me does not only cover conventional structured data (such as files, emails, status messages), but also unstructured and abstract data such as user context, situations, and online presence. This type of data is what will drive our intelligent recommender system to perform the desired functions through the di.me userware and intelligent user interface[2].

In this paper we will discuss our plans and progress in relation to the following three project objectives:

1. The definition of rules for recommendation and automation of tasks, based on the user's personal data cloud;
2. Automatic rule learning;
3. The processing mechanism to trigger learnt rules based on the perceived events.

For the first objective we provide a model that allows us to represent the learnt context-driven rules in a declarative manner. This contribution consists of the di.me Rule Management Ontology (DRMO)[3], an activity rule vocabulary integrated in the di.me knowledge representation models that allows PIM knowledge to be exploited for activity rule management. For the second objective, we are currently investigating techniques, such as case-based reasoning (CBR), which allow the system to automatically learn rules based on the user context-aware history. Currently we allow users to define context-aware rules using the intelligent user interface available in the di.me userware. The third objective is that of providing mechanisms to recommend actions to the user based on the real-time events perceived through the userware. We investigate various techniques in order to provide a scalable context listener which can provide recommendations or actions based on the learnt rules and the user's integrated PIM data.

After discussing the related work (Section 2), we discuss the integration of personal data in di.me and how open data sources such as LinkedGeoData[4] and Sindice[5] can be utilised for recommendation (Section 3.1). The Rule Management Ontology and the Context Listener are discussed in Section 3.2 and 3.3 respectively. We then discuss techniques for the automatic learning of context-driven rules (Section 3.4), before providing a real-world scenario (Section 4) and some concluding remarks and discussing future work (Section 5).

2 Related Work

In this section we look at how earlier efforts have tackled rule modelling, and how they were applied in various context-aware and event processing systems respectively. We also discuss techniques on how rules can be learnt automatically.

[2] The UI in question is currently being improved to factor in the results of a number of usability studies. A separate submission detailing its design is currently under review.

[3] http://www.semanticdesktop.org/ontologies/drmo/

[4] http://linkedgeodata.org/

[5] http://sindice.com/

In [4] and [2], rules are defined by an English-like rule language. The SECE system provided five different types of hard-coded events [4], and was later extended towards an ontology-based system [2] in order to enable the automatic discovery of services. This allowed the users to define rules that give recommendations based on open linked data services. The SECE language is specific to their system; thus, unlike our proposed model which is based on Resource Description Framework (RDF), the SECE rule language cannot be easily reused on other frameworks. The authors do not give any indication regarding the use of event and logical operators defined in the rule language. Li et al. [9] presents an event ontology with the aim of describing perceived situations. This model does not represent rules, but users can define rules by creating patterns (called processes) of composed event instances using event and logic operators. The resulting actions are inferred semantic knowledge, from the system knowledge base. On the other hand, in di.me, the actions give the user recommendations and automation based on the user's context-environment. In [10], May et al. present an ontology-based framework based on the Event-Condition-Action (ECA) pattern in order to integrate heterogenous semantic web services via rule definition. This framework allows the definition of sub-languages in order to process events from different web services, unlike the DRMO which is geared towards one domain: the personal information management. Sub-languages include detection processors for events defined in rules and for defining composite events. In contrast, in our model we create properties for composite conditions. May et al. defined the concepts Event-Condition-Action as three separate components since in a rule these might be defined with three different languages. In our case, the Event concept is composed by a number of condition blocks, and if these are satisfied they trigger one or more actions.

To apply our proposed rule model, we need a rule engine to execute the rules. In [9] event processes (rules) take perceived events as input and return an inferred *semantic event* as output. Their objective is to collect contextual data from the surroundings and present the user with an intelligent deduction of the current situation. An inference graph is used to keep track of the detected event concepts. In our context-listener we apply an *event-set* to store perceived events. We are also investigating time-window techniques with the aim of keeping the listener scalable. Rules are then executed on the *event-set* in order to provide recommendations. The framework proposed in [9] is composed of two inference engines; one which identifies rule patterns from perceived events and the other one which translates low-level event data into higher meaningful activities. In our case, the latter is being done by another part of the di.me framework, whilst for the former we do pattern matching using SPARQL[6] queries, since the event data we perceive in the userware is in RDF format. Using SPARQL for pattern matching allow instances of the DRMO ontology to execute on any triple store. Such technique was used by Teymourian and Paschke [16], where by using SPARQL to represent event patterns, they show how semantic events can be used in event detection. When an event is perceived, the system infers new triples using the system's event knowledge base. When the new inferred triples are created, they are sent to a rule-based inference engine and the engine decides if these triples should be stored in the system or discarded. In

[6] http://www.w3.org/TR/rdf-sparql-query/

di.me, SPARQL queries would not infer new knowledge, but indicate that rules should be triggered.

The proposed ontology (DRMO) can be used for automatic rule discovery in a context-aware environment. Similar approaches use ontologies as context models in Case-Base Reasoning (CBR) techniques [1, 8]. Bai et al. [1] proposed an ontology-based CBR (OntoCBR) to compare similarity between contexts, in order to reason and learn rules from a user's environment rather than rely on pre-defined or user-defined rules. In OntoCBR, user context is transformed into a situation case and is compared with other situations which are in the case-base using a semantic similarity measure. Thus recommendations are given if a situation has a match, in order to propose an action which has been carried out in similar situations. Knox et al. [8] make use of sensor data and CBR for activity recognition in a home environment. Cases are represented by an ontology which acts as a relationship between sensor data and resulting activities. The authors show that using CBR, various user activities can be learnt incrementally without having the need of training data. Similar to these two works, we intend to create a case-base from context data using the technique in [8], whilst learning new rules using similarity measures techniques as outlined in [1]. In order to ensure system scalability, we consider the work presented by Bergmann and Vollrath in [3], in which they discuss the idea of having a generalised CBR, where similar cases are generalised into one case in order to reduce the size of the case base.

3 Approach

In this section we first discuss how raw data from sensors and smart devices are integrated within the personal data cloud. We also discuss how linked open datasets such as LinkedGeoData and data source providers such as Sindice can be utilised to provide recommendations for the user, based on their current situation. We then introduce the di.me Rule Management ontology, before exploring how DRMO instances are mapped into SPARQL queries to simulate production rules. These rules are processed, and if their conditions are satisfied, the instructed actions are triggered to provide recommendations and automation in the di.me userware. Finally we discuss how we can learn and define DRMO rules automatically.

3.1 Exploiting Personal and Open Data Cloud for Intelligent Recommendation

The basic goal of the di.me userware is to learn about, gather and integrate the user's information and activities through their personal devices and online accounts, in order to provide a single-entry point to their personal data management and to provide context-aware recommendation and automation. di.me extends an interoperable knowledge representation format based on ontologies for the representation of personal information presented in [14], which also covers context-related personal knowledge. The use of this format allows interoperability with other applications that also utilise the RDF standard. Strang et al. recommends ontology-based modelling as the most appropriate way to engineer the core concepts in a context-aware environment [15]. After

having such personal information crawled and extracted from devices, it is semantically lifted onto the ontology-based PIM representation. The latter can then be used as a knowledge base for defining rules to provide recommendation and automation in the di.me userware.

Even though our primary focus is that of creating a big personal data cloud, external linked open data sources can be used to provide richer recommendations based on the user's current situation. LinkedGeoData could be utilised to, for example recommend a nearby restaurant for a user who is in a situation "Out of Office" during "lunch time". By gathering user context data surrounding the user, the system can invoke the LinkedGeoData SPARQL Endpoint to find restaurants located nearby to the current location of the user. When the data is returned, the userware might make use of a data source crawler such as Sindice to enrich the retrieved data by adding other information which might not be provided by the LinkedGeoData, such as restaurant ratings. Sindice is a web crawler which indexes statements in linked-data resources. This service helps users find certain resources and allows developers to integrate data from various datasources. LinkedGeoData and Sindice data sources can be combined with data in the personal cloud to provide better recommendation. The use of other datasets is also being investigated.

3.2 di.me Rule Management Ontology (DRMO)

The Rule Management Ontology (Figure 1) is inspired by approaches such as [10]. Rules are modelled on the Event-Condition-Action (ECA) pattern concepts. The ECA pattern is a structure used in event driven architectures, where the event part specifies on what event this rule might be triggered, the condition specifies under which conditions the actions should be triggered and the action part contains what is executed to lead the system to a new state, causing data to be changed [6]. Unlike the traditional architectures, DRMO rules are not specific to events (such as "on update" or "on delete"), but all rules can trigger actions if all of their conditions are satisfied. Thus, our pattern is defined as:

$$\textit{if } E[c_1, .., c_m] \implies [a_1, .., a_n] \tag{1}$$

where *event* E represents a rule that consists of a combination of *conditions* c, triggering one or more resulting *actions* a.

A rule is represented as a *drmo:Event*, which is composed (*drmo:isComposedOf*) of a number of *drmo:Condition* 'blocks' and triggers (*drmo:triggers*) one or more *drmo:Action* instances (Figure 1), similar to the work presented in [2, 4]. Thus, a DRMO event corresponds to an antecedent, whereas an action corresponds to the consequent part of a production rule.

In the ontology we define a number of different condition categories as subclasses of *drmo:Condition*. Since our focus is on the personal data cloud, the latter categories (*drmo:ResourceCreated, drmo:ResourceModified, drmo:ResourceDeleted*) represent the different changes affected in the Personal Information Model (PIM), such as receiving a new email (Resource Created), adding access control to peers on files (Resource Modified), and deleting a file (Resource Deleted). In the di.me userware we

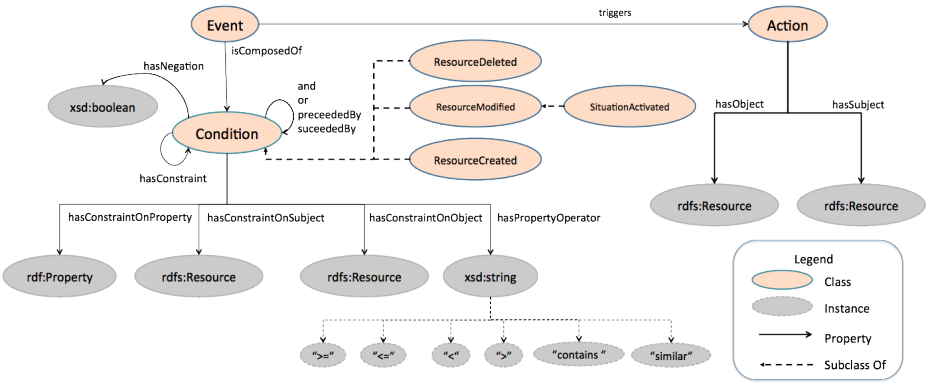

Fig. 1. The Rule Management Ontology

also allow for users to save personal activity context. A change in known user situations can be represented as an instance of *drmo:SituationActivated*, e.g. a change from situation "AtWork" to situation "AtHome". The different subclasses help the rule filtering mechanism according to the perceived system and user events.

Condition blocks can be composed of any information element (resource or class) on the Semantic Web. In di.me, condition blocks are restricted to elements from the OSCAF[7] ontologies and their attributes, and resources from the user's PIM. Multiple conditions are combined together using a number of logical (*drmo:and*, *drmo:or*) and event operators (*drmo:succeededBy*, *drmo:precededBy*). The set of operators are defined as properties in the DRMO (Figure 1) and are a subset of the operators used in [9] and defined in [5]. To explain these operators better, suppose conditions [C_1,C_2] occur at an instance of time t:

- Condition C_1 **AND** Condition C_2. Both conditions have to occur independent of **time** and **order** for the actions to trigger.

$$(C_1 \wedge C_2)(t) = (\exists t_1)(((C_1(t_1) \wedge C_2(t)) \vee (C_2(t_1) \wedge C_1(t))) \wedge t_1 \leq t) \quad (2)$$

- Condition C_1 **OR** Condition C_2. For the rule to trigger its actions, **only one** of the conditions have to be true at any given time.

$$(C_1 \vee C_2)(t) = (C_1(t) \vee C_2(t)) \quad (3)$$

- **SUCCEEDEDBY**(Condition C_1, Condition C_2). The actions are **only** triggered if Condition C_2 happens **at any time after** Condition C_1.

$$SUCCEEDEDBY(C_1, C_2)(t) = (\exists t_1)(C_1(t_1) \wedge C_2(t) \wedge (t_1 < t)) \quad (4)$$

- **PRECEDEDBY**(Condition C_2, Condition C_1). The actions are **only** triggered if Condition C_1 has happened **at any time before** Condition C_2.

$$PRECEDEDBY(C_2, C_1)(t) = (\exists t_1)(C_1(t_1) \wedge C_2(t) \wedge (t_1 < t)) \quad (5)$$

[7] http://www.oscaf.net

Each condition might also be negated using the *drmo:hasNegation* property. This property has a value range of either *true* or *false*. A condition may have one or more constraints (*drmo:hasConstraint*), for example *'if I receive an email from Anna'* the constraint here is that the rule will trigger only when an email from Anna is received. On the other hand, if a rule condition has a PIM resource such as a saved situation, then there might be no constraints, for example *'if I am AtWork'*. Generic conditions can also be defined without constraints, for example the rule *'if I receive an email'* would trigger always whenever a new email is received, irrelevant of the sender.

A condition can have three types of constraints: *drmo:hasConstraintOnProperty, drmo:hasConstraintOnSubject, drmo:hasConstraintOnObject*. These are used to define constraints on the event properties, for example the rule *'If I receive an email from anna@email.com'*, the value 'anna@email.com' is a constraint on the property *'nmo:messageFrom'*. To understand the DRMO constraint properties better we demonstrate the mentioned example as an instance in Listing 1.1:

```
... #prefix definitions

juan:event1 a drmo:Event ;
    drmo:isComposedOf juan:condition1 ; drmo:triggers juan:action1 .

juan:condition1 a drmo:Condition , nmo:Email ;
    drmo:hasConstraint juan:constraint1 .

juan:constraint1 a drmo:Condition ;
    drmo:hasConstraintOnProperty nmo:messageFrom;
    drmo:hasConstraintOnObject juan:constraint1:email1 .

juan:constraint1:email1 a   nco:EmailAddress , drmo:Condition ;
    drmo:hasConstraintOnProperty nco:emailAddress;
    drmo:hasConstraintOnObject  "anna@email.com" .

... #action definitions
```

Listing 1.1. An example of a user-defined rule

In Listing 1.1 the rule (*juan:event1*) is composed of *juan:condition1*, which is of type *drmo:Condition* and *nmo:Email*. *juan:contraint1* is added to the condition to represent the constraint given by the rule *'if I receive an email **from anna@email.com**'*. This is described in the rule instance as a *drmo:hasConstraintOnProperty* 'nmo:messageFrom' and as a *drmo:hasConstraintOnObject* 'anna@email.com'. The URI 'juan:constraint1:email1' is not a resource from the PIM, but it is a generated resource of type *drmo:Condition* since nmo:messageFrom would be expecting a URI of a resource (e.g. 'mailto:anna@email.com') rather than the value itself. These properties help to transform rules from a DRMO instance into SPARQL queries, as we discuss in Section 3.3. The *drmo:hasConstraintOnSubject* property allows for implicit values such as resources from the PIM, for example 'PIM:Anna' to be used in rule constraints. This allow the rule to be triggered every time an email from Anna is received, if her email address is in the user's PIM.

A constraint might also have relational operators in order to compare the perceived event values with the value of the rule's condition (e.g. *"if I receive an email and the subject contains di.me"*). In di.me, "contains" and "similar" can be used as string operators,whereas we define operator instances for numeric datatypes such as $\leq$, $\geq$, $<$, and $>$.

The *drmo:Action* class specifies an action instance (e.g. *Recommend*), whose semantics are understood by the system and result in specific actions. The *drmo:hasSubject* specifies the receiver of the action whilst the *drmo:hasObject* specifies what parameters need to be passed to the executed actions. The parametrisation of action subjects and objects is similar to that defined in [13], where action instances are used to classify emails. Examples of action instances could be recommending a nearby restaurant, where the action instance 'Recommend' would have "Restaurant" as a subject and *pimo:Location* as an object.

3.3 A Context Listener for Rule Activation

A context listener is required to register defined rules and to activate them when their conditions are satisfied by the perceived events. Rule instances are transformed into SPARQL queries and registered to a rule pool in the context-listener. In this section we will first explain how rules are transformed from DRMO instances into SPARQL. Then we show how event processing in the context-listener is done.

Transforming rules from DRMO instances to SPARQL - Rules are defined as instances of the DRMO. As part of the event processing mechanism, the condition part of the rule is transformed into SPARQL queries which might be executed at a specific time window, or each time an event is registered in the PIM. Since we also have logic operators, we discuss how these will be parsed using an infix to postfix technique. For each rule instance in the user's PIM, the transformer maps the *drmo:Condition* blocks to SPARQL queries. The properties *drmo:hasConstraintOnSubject, drmo:hasConstraintOnProperty* and *drmo:hasConstraintOnObject* are mapped to a SPARQL triple pattern {?subject ?predicate ?object . } respectively. A negated condition block (*drmo:hasNegation*) is mapped into the SPARQL query combining[8] 'OPTIONAL', 'FILTER' and 'BOUND'. The negation function is still a feature recommendation for SPARQL 1.1[9]. The transformer parse logical operators *drmo:and* and *drmo:or* in precedence ordering. We decided to implement a postfix technique since it allows us to define the order of execution from an infix notation. Multiple conditions composed with the *drmo:or* property have the condition triples separated with the 'UNION' keyword, whilst those with the *drmo:and* have their condition triples joined in one query. SPARQL 'FILTER's are used when a constraint has the *drmo:hasPropertyOperator* defined. Filters are also used when multiple conditions are composed together using *succeededBy* and *precededBy* (Section 3.2), by restricting patterns using timestamps. The action instances are stored in a separate object together with the rule instance URI, and not part of the SPARQL query. Listing 1.2 shows how the rule instance in Listing 1.1 is transformed into a SPARQL query.

```
SELECT * WHERE {
    ?_cn61 a nmo:Email .
    ?_cn61 nmo:messageFrom ?_varFE208 .
    ?_varFE208 a nco:EmailAddress .
```

[8] http://www.w3.org/TR/rdf-sparql-query/#func-bound
[9] http://www.w3.org/TR/sparql-features/

```
    ?_varFE208 nco:emailAddress 'anna@email.com' .
}
```

Listing 1.2. SPARQL Transformation for rule in Listing 1.1

Event Processing and Pattern Matching - When a rule is transformed to a SPARQL pattern, it is then registered to a rule pool in the context listener (Figure 2). The transformation of DRMO instances into SPARQL queries help us in the pattern matching process. Triple patterns can be considered as the *antecedents* of a production rule, since instance conditions are transformed into triple patterns that have to be matched in the *working memory*. Our working memory consists of an *event-set* which holds events perceived by the userware and the PIM. Each time an event is perceived, the event data is timestamped and stored in the *event-set*. Using the new perceived information,

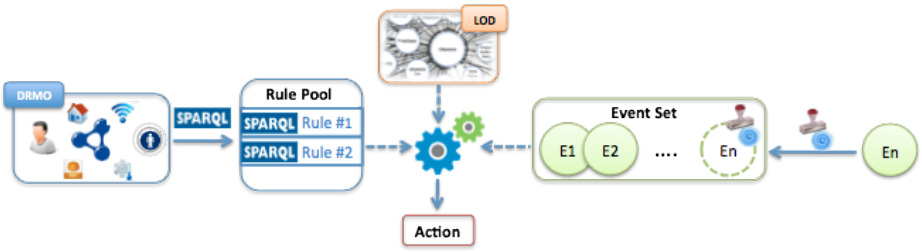

Fig. 2. Context Listener

the context listener will then filter rules by category (see Section 3.2) and type (for example 'nmo:Email') to keep only the ones that might be triggered from the perceived events stored in the set. From these filtered rules, SPARQL queries are executed and results are returned, triggering satisfied rules and thus performing the specified actions. Since this is a real-time listener and events are continuously perceived, we need to cater for events which happened in the past and are no longer required in the Event-Set. The use of a *time window* helps us keep the listener scalable by removing past perceived events, for example, keeping in memory only events that happened in the last hour. An evaluation will be carried out to find the most appropriate time-window limit for our usecases. To improve the scalability and efficiency of our context-listener, we are currently investigating how algorithms such as Rete [7], which speeds up the matching process, can be integrated in this module for optimisation purposes.

3.4 Learning Rules from Context Data

As discussed in previous sections, the semantic framework in the di.me userware gathers context-related information about the user. Perceived context data is semantically lifted onto a unique 'live context' instance of the Context Ontology (DCON)[10] [12]. In

[10] http://www.semanticdesktop.org/ontologies/dcon/

addition, past context snapshots can be timestamped and made persistent as instances of the User History Ontology (DUHO)[11]. Thus it is possible to construct a timeline of events and corresponding actions, as part of a user's history. Using CBR techniques mentioned in the related work section, these instances will be analysed to find similar situation-action patterns, in order to learn and define rule instances automatically. As a simple example, user John frequently forwards emails with a subject containing "ISWC" to his student Mary. By analysing the context logs, the system could identify this pattern and learn this rule. From then onwards, whenever John receives an email with the subject containing "ISWC", the system will recommend John to automatically forward it to Mary.

4 A Rule Scenario

In this section we provide a practical scenario of how a rule instance is defined in the PIM. We show how our ontologies can represent rules for context-aware recommendation. Listing 1.3 provides an example of a rule, represented as an instance of DRMO. The rule represents the scenario *If I find myself in situation "Out of Office", and it is "Lunch Time", recommend me a nearby restaurant*. In this rule we see that juan:event34, which is an existing resource in the PIM, is composed of two conditions, juan:condition8 and juan:condition9. The former refers to the previously saved situation "Out of Office" (<urn:juan:graph:situation14>), and the latter refers to a more complex condition. Here, the *LiveContext* is queried to get the current time. Both condition items are existing resources in the PIM. If the current time instance points at Lunch Time (defined in the PIM), then the action is triggered. "Recommend" is a system defined action that is used by Juan. The action has two properties, juan:subject1 and juan:object1. The former defines the type of recommendation needed and the latter passes any parameters to satisfy the recommendation. The system will transform the instance and parameters into a SPARQL query to an open data endpoint such as the LinkedGeoData which returns a list of possible restaurants near the user's current location.

```
@prefix drmo: <http://www.semanticdesktop.org/ontologies/2012/03/06/drmo#> .
@prefix pimo: <http://www.semanticdesktop.org/ontologies/2007/11/01/pimo#> .

juan:event34 a drmo:Event ;
  drmo:isComposedOf juan:condition8, juan:condition9 ; drmo:triggers juan:action4
       .

juan:condition8 a drmo:Condition ;
   drmo:hasConstraint <urn:juan:graph:situation14> ;
   drmo:and juan:condition9 .

juan:condition9 a drmo:Condition , dcon:LiveContext ;
  drmo:hasConstraint juan:constraint12 ;
  drmo:and juan:condition8 .

juan:constraint12 a drmo:Condition ;
  drmo:hasConstraintOnProperty dcon:currentTime ; drmo:hasConstraintOnObject <urn:
       juan:PIM:LunchHours> .

juan:recommend a drmo:Action ;
    drmo:hasSubject juan:subject1 ;
```

[11] http://www.semanticdesktop.org/ontologies/duho/

```
    drmo:hasObject juan:object1 .

juan:subject1 a rdfs:Literal ;
    rdf:value "Restaurants"^^xsd:string .

juan:object1 a pimo:Location ;
    pimo:hasLocation <urn:juan:PIM:CurrentLocation1> .
```

Listing 1.3. An example of a user-defined rule

5 Future Work and Concluding Remarks

In this paper we described our approach for an ontology-driven recommender system that suggests actions to the user, driven by context and knowledge from their aggregated personal data cloud. We described how personal data from various sources can be combined with the user's sensed situational context in order to detect recurring situations and a vast range of associated actions that can be fully or partly automated. We also explained how open link data services like Sindice, and data sources such as LinkedGeoData, can also be exploited by the system to recommend items that are not part of the user's personal data cloud, i.e. suggest new, possibly unknown items, to the user. A rule management ontology, integrated with existing standard ontologies for the comprehensive modelling of distributed personal information, is proposed. The ontology supports the definition of context-driven recommendation rules that are learnt by an intelligent system, or defined semi-automatically by the user. The rules are converted from RDF to SPARQL queries at runtime by a context listener, which continously matches them against perceived user activities and system events. A matched rule results in a recommendation being provided to the user.

Currently the implemented protoype only caters for user-defined rules, that can be intuitively constructed through an intelligent UI. In the future, techniques to enable the automatic discovery of rules will be further investigated. Further enhancements under consideration are techniques such as the Rete algorithm for a more efficient context listener (performance-wise), and the investigation and evaluation of techniques for managing rule conflicts. To determine the effectiveness and usability of the proposed system, we will perform evaluation to determine i) how many different kinds of use-cases the DRMO ontology is able to cover, ii) the ideal time-window to ensure the context-listener's scalability and efficiency, iii) the user's own assessment of the system's usability and the adequacy of the resulting in-context recommendations.

Acknowledgments. This work is supported in part by the European Commission under the Seventh Framework Program FP7/2007-2013 (*digital.me* – ICT-257787) and in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (*Líon-2*).

References

1. Y. Bai, J. Yang, and Y. Qiu. Ontocbr: Ontology-based cbr in context-aware applications. In *Proceedings of the 2008 International Conference on Multimedia and Ubiquitous Engineering*, MUE '08, pages 164–169, Washington, DC, USA, 2008. IEEE Computer Society.

2. V. Beltran, K. Arabshian, and H. Schulzrinne. Ontology-based user-defined rules and context-aware service composition system. In *Proceedings of the 8th international conference on The Semantic Web*, ESWC'11, pages 139–155, Berlin, Heidelberg, 2012. Springer-Verlag.
3. R. Bergmann and I. Vollrath. Generalized cases: Representation and steps towards efficient similarity assessment. In *Proceedings of the 23rd Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence*, KI '99, pages 195–206, London, UK, UK, 1999. Springer-Verlag.
4. O. Boyaci, V. Beltran, and H. Schulzrinne. Bridging communications and the physical world: sense everything, control everything. In *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, IPTcomm '11, pages 14:1–14:6, New York, NY, USA, 2011. ACM.
5. S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 606–617, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
6. K. R. Dittrich, S. Gatziu, and A. Geppert. The active database management system manifesto: A rulebase of adbms features. In *Proceedings of the Second International Workshop on Rules in Database Systems*, RIDS '95, pages 3–20, London, UK, UK, 1995. Springer-Verlag.
7. C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17 – 37, 1982.
8. S. Knox, L. Coyle, and S. Dobson. Using ontologies in case-based activity recognition. In *FLAIRS Conference*, 2010.
9. Z. Li, C.-H. Chu, W. Yao, and R. A. Behr. Ontology-driven event detection and indexing in smart spaces. In *Proceedings of the 2010 IEEE Fourth International Conference on Semantic Computing*, ICSC '10, pages 285–292, Washington, DC, USA, 2010. IEEE Computer Society.
10. W. May, J. J. Alferes, and R. Amador. An ontology- and resources-based approach to evolution and reactivity in the semantic web. In *Proceedings of the 2005 OTM Confederated international conference on On the Move to Meaningful Internet Systems: CoopIS, COA, and ODBASE - Volume Part II*, OTM'05, pages 1553–1570, Berlin, Heidelberg, 2005. Springer-Verlag.
11. D. McDonald. Ubiquitous recommendation systems. *Computer*, 36(10):111 – 112, oct. 2003.
12. S. Scerri, J. Attard, I. Rivera, M. Valla, and S. Handschuh. Dcon: Interoperable context representation for pervasive environments. In *In Proceedings of the Activity Context Representation Workshop at AAAI 2012*, 2012.
13. S. Scerri, G. Gossen, B. Davis, and S. Handschuh. Classifying action items for semantic email. In *LREC*, 2010.
14. M. Sintek, S. Handschuh, S. Scerri, and L. van Elst. Technologies for the social semantic desktop. In *Reasoning Web. Semantic Technologies for Information Systems*, volume 5689 of *Lecture Notes in Computer Science*, pages 222–254. Springer Berlin / Heidelberg, 2009.
15. T. Strang and C. L. Popien. A context modeling survey. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham, September 2004.
16. K. Teymourian and A. Paschke. Semantic rule-based complex event processing. In *Proceedings of the 2009 International Symposium on Rule Interchange and Applications*, RuleML '09, pages 82–92, Berlin, Heidelberg, 2009. Springer-Verlag.