Expert Systems with Applications 42 (2015) 7979–7990

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Modelling rules for automating the Evented WEb by semantic technologies

Miguel Coronado [a], Carlos A. Iglesias [a], Emilio Serrano [b,*]

[a] Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación, Departamento de Ingeniería de Sistemas Telemáticos, Ciudad Universitaria, 28040 Madrid, Spain

[b] Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Informáticos, Departamento de Inteligencia Artificial, Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain

ARTICLE INFO

Article history:
Available online 25 June 2015

Keywords:
Event-stream
Linked data
Semantic Web
Web rule
Knowledge management
Social sensor

ABSTRACT

The Live Web is characterised by a new way of interacting with the Web through dynamic streams of relevant real-time contextual information to users. These sources of massive data usually overwhelm them, because they are not able to consume that amount of data. *Task Automation Services* (TASs) are platforms or apps that allow their users to author automation rules to combine events from streams while reducing the effort for handling incoming information. While these platforms are a reality, they suffer from two major drawbacks: (i) the only incoming data streams available are those the TASs developers decided to include in the system, and (ii) they lack of a mechanism to reason over large scale data outside their platform. To face these challenges, this paper contributes in (i) reviewing the existing state of the art including research and commercial work given their relevance. Based on the lessons learnt from this review, (ii) we propose the *Evented WEb ontology* (EWE), that models the Evented WEb domain, and in particular those concepts around TASs. EWE enables scalability, interoperability and definition of rules with reasoning over *Linked Open Data* (LOD) cloud. To illustrate these contributions, (iii) a semantic TAS has been implemented that benefits from the advantages EWE offers, and solves a realistic problem using semantic technologies. Finally, (iv) to validate the ontology covers the domain it models, a thorough ontology evaluation is presented.

1. Introduction

The term *Live Web* (Windley, 2011), also called the *real-time Web* (De Francisci Morales, Gionis, & Lucchese, 2012) or the *event Web* (Singh & Jain, 2010), describes a new stage in the evolution of the Web that extends the Web 2.0 or interactive Web. Instead of simply browsing static webpages or even interacting with a website, the Live Web is characterised by a brand new style of interaction through dynamic streams of information, called *personal streams*, to present contextual and relevant experiences to users (Windley, 2011). There are several sources of personal streams such as: social networks, sensor networks, and mobile phones. These sources provide the necessary location-aware, relationship-aware, preference-aware and sensory context to achieve a new generation of context-aware applications (Beach et al., 2010).

Nevertheless, given that each user is a potential source of events, a typical user often gets flooded with a large number of notifications; e.g. tweets, CRM notifications, chat messages, etc. As a result, they only read a small fraction of those they receive (De Francisci Morales et al., 2012), wasting tons of potentially useful data. Thus, several automation tools have emerged in order to simplify personal-stream management. Some of these tools are social media management tools (Kietzmann, Hermkens, McCarthy, & Silvestre, 2011) such as: TweetDeck,[1] with advanced filtering and scheduling facilities; Rapportive,[2] that combines Linkedin profiles with Gmail contacts; and, data-driven mashup tools (Yu, Benatallah, Casati, & Daniel, 2008), which provide users the ability to create new applications based on the available services.

What makes these tools really useful for consumers is that they combine incoming data available from different personal streams, presenting the information to the user in a manner that it is more useful than it was by separate. Thus they create new personalised

* Corresponding author.
E-mail addresses: miguelcb@dit.upm.es (M. Coronado), cif@dit.upm.es (C.A. Iglesias), emilioserra@fi.upm.es (E. Serrano).

[1] http://tweetdeck.twitter.com/.
[2] http://rapportive.com/.

http://dx.doi.org/10.1016/j.eswa.2015.06.031
0957-4174/

streams and services that behave in a particular way for each different user.

In this regard, a number of now prominent websites, mobile applications, and desktop applications feature *rule-based task automation*. Typically, these services provide users the ability to define which action should be executed when some event is triggered, i.e. each user defines its own automations. Some examples of this simple task automation could be 'When I am mentioned in Twitter, send me an email', 'When I come within 500 meters of this place, check-in in Foursquare', or 'Turn Bluetooth on when I leave work'. We call these services *Task Automation Services* (TASs). Ifttt (IFTTT put the internet to work for you, 2015), Zapier (Zapier the best apps. Better together, 2015) or Tasker (Tasker total automation for android, 2015) are a few enlightening examples.

The great number of users these platforms accumulate, speak for itself in terms of usefulness. However, they suffer from two major drawbacks: (i) the only incoming data streams available are those the platform is prepared for; and, (ii) they lack of a mechanism to use and reason over large scale data outside their platform such as the *Linked Open Data* (LOD) cloud or context data. These shortcomings decrease TASs flexibility, narrowing rule capabilities.

Semantic Web technologies are suitable for being incorporated to TASs and *expert systems* implemented by these services in order to face the two challenges explained above. These technologies have enhanced experts systems in a large number of application domains such as: information retrieval (Thangaraj & Sujatha, 2014; Song, Liang, Cao, & Park, 2014), traffic information services (Zapater, Escrivá, García, & Durá, 2015), emergency management (Onorati, Malizia, Diaz, & Aedo, 2014; Serrano, Poveda, & Garijo, 2014), recommendation systems (Martín-Vicente et al., 2014; Ying et al., 2014), reputation systems (Hermoso, Centeno, & Fasli, 2014; Serrano, Rovatsos, & Botía, 2012), ambient intelligence (de Alba, Campillo, Fuentes-Fernández, & Pavón, 2014; Serrano, Moncada, Garijo, & Iglesias, 2014), and cloud computing (Rezaei, Chiew, Lee, & Aliee, 2014; Jula, Sundararajan, & Othman, 2014).

In this scenario, we present a fourfold contribution. The first contribution of this paper is surveying the state of the art in Task Automation Services to gain insight into this novel field and to support the hypothesis that these systems have the shortcomings explained above. The second contribution is the development of an ontology to model the Live Web called the *Evented WEb ontology* (EWE). With EWE, personal streams and the events they produce are modelled semantically so they may be automatically discovered (König-Ries, Opasjumruskit, Nauerz, & Welsch, 2012). Once these personal streams and events are semantically described, composition rules based on them can reason over the LOD. The third contribution is an implementation of a semantic TAS using EWE which addresses the shortcomings observed in the revised TASs. Finally, the fourth contribution is an ontology evaluation that validates that EWE covers the domain it models by comparing its coverage and accuracy to popular alternative approaches.

The rest of the paper is organised as follows. First Section 2 describes the current state of TASs, including the drawbacks which motivate this paper. Next, Section 3 presents the EWE ontology: its design methodology, the main classes, properties, mapping to existing external ontologies, and examples of EWE use. An implementation of a semantic TAS featuring EWE and a use case are presented in Section 4. Then, in Section 5, EWE is formally evaluated. Finally, Section 6 concludes and gives future works.

2. Related works in Task Automation Services (TASs)

Task automation systems of personal streams are a novel research field whose relevance has significantly increased in the last few years. Pushed by the great amount of personal streams available nowadays, these systems are also gradually being adopted by the users as one of their bedside apps and webpages. For that reason, we consider surveying the most relevant related works an important contribution of this paper.

We already named Ifttt (IFTTT put the internet to work for you, 2015), Zapier (Zapier the best apps. Better together, 2015) and Tasker (Tasker total automation for android, 2015) but there are plenty of commercial TASs. Some of these are: Elastic.io (Elastic.io integrate once. Connect many, 2015) and Cloudwork (Cloudwork connect your business apps, 2015), that work on the Web as Ifttt and Zapier do; and, AutomateIt (Automateit turn your smartphone into a genius-phone, 2015), Atooma (Atooma a touch of magic, 2015) and on{x} (On{x} automate your life, 2015), which are smartphone apps as Tasker. This trend is closely related to the *Web of Things* (WoT) where several devices and middlewares allow the users to author their personal automation too, e.g. SmartThings (SmartThings life like never before, 2015), Webee (Webee Experience, Connected, 2015) or Wigwag (WigWag smart starts with a brain, 2015).

Apart from the foundation the different TASs work on (either the Web or a device), there are other characteristics that may differentiate them from each other. The target audience, i.e. the market fit, is one of the most relevant. Ifttt, AutomateIt and Atooma focus on the general audience. Zapier and Cloudwork target business users. Finally, Tasker and onx aim at more technical users. Several other minor technical differences may be considered, some of them are summarised in Table 1, while the complete list of features used in our study is available online (Coronado, Iglesias, & Serrano, 2015b). It is of particular interest, the information gathered in Table 2, that shows the different terms used to define the same elements. This table illustrates the lack of standardisation about this topic that we aim to address with EWE ontology, being one of the contributions of our work.

Apart from commercial systems, there are several approaches to TASs in the specialized literature. Related to the WoT, Paraimpu (Pintus, Carboni, & Piras, 2012) allows user to interconnect Http-enabled smartobjects and Web Services. In the terms we use, it is a TAS that supports configuring sensors and devices as data streams. Among the current TASs described before, several of them support using physical sensors and actuators from different approaches. Ifttt, includes channels to handle sensors such as WeMo[3] or PhillipsHue.[4] Its approach bypasses communication to them using third party APIs. SmartThings or Webee directly support managing sensors and but only those they provide. In this regard, Parimpu is more flexible than any of them since it allows configuring your own smartobjects as channels. However, final users need programming skills to configure them. As Karger (Karger, 2014) points out, one of the main disadvantages of these systems is that it is impossible to integrate new data suppliers and consumers unless the TAS company chooses to do so. To overcome this problem, Opasjumruskit et al. designed Mercury (Opasjumruskit, Expósito, & König-Ries, 2012), a powerful TAS that features service discovery. This service is able to find appropriate sensors, services, actuators, etc.; to perform certain functionality. Mercury relies on semantic annotation of Web of device data sources so it can reason about them. Regarding better user interfaces, Atomate it (Van Kleek, Moore, Karger, André, & Schraefel, 2010) focuses on providing a constrained-input natural language interface for composing automations in natural language. Although it is not an entirely free-text approach, it is very intuitive and the authors claim that users

[3] http://www.wemothat.com/.
[4] http://www2.meethue.com/.

Table 1
Comparison of Task Automation Services' advanced features.

Feature	Ifttt	Zapier	Kinetics	Elastic.io
Rules with more than one condition	No	No	Yes	No
Rules with more than one action	No	No	Yes	Yes
Event filtering	No	Yes	Yes	No
Rule sharing	Yes	Yes	No	No
Channel categorization	No	Yes	-	No
Support for devices	Yes	No	Yes (KRL)	No
Built-in context	No	No	Yes	No
Visual rule editor	Yes	Yes	No (KRL)	Yes
Temporal reasoning	No	No	Yes	No
Feature	Atooma	Tasker	Automateit	On{x}
Rules with more than one condition	No	Yes	No	Yes
Rules with more than one action	No	Yes	Yes	Yes
Event filtering	No	Yes	Yes (few)	Yes
Rule sharing	Yes	No	Yes	Yes
Channel categorization	No	Yes	No	Yes
Support for devices	Yes	Yes	Yes	Yes
Built-in context	No	Yes	Yes	Yes
Visual rule editor	Yes	Yes	Yes	No (js)
Temporal reasoning	No	No	No	Yes (js)

increase their satisfaction compared to the standard interfaces of commercial TASs.

When addressing the TASs limitations with Semantic Web technologies, there are plenty of options for modelling and implementing rule-based knowledge. Some of the best known efforts with this regard are RuleML (Boley, Paschke, & Shafiq, 2010; RuleML Overview & Motivation, 2010), *Semantic Web Rule Language* (SWRL) (O'Connor et al., 2005; SWRL: A Semantic Web Rule Language Combining OWL & RuleML, 2005), and *Rule Interchange Format* (RIF) (Kifer, 2008; RIF Production Rule Dialect, 2008). This paper proposes the use of *SPARQL Inferencing Notation* (SPIN) (SPIN Overview & Motivation, 2011) in the TASs modelling for three main reasons. Firstly, SPIN is based on *SPARQL Protocol and RDF Query Language* (SPARQL) (SPARQL Query Language for RDF, 2008). SPARQL is well established and supported by numerous engines and databases. Therefore, SPIN rules can be directly executed on the databases without requiring intermediate engines (SPIN, SPARQL Inferencing Notation, 2011). These engines would introduce communication overhead, reducing the TAS efficiency, and an extra component in the architecture, complicating the TAS maintenance. Secondly and as a consequence of the first reason, no extra language is needed when implementing a semantic TAS with SPIN (SPIN Frequently Asked Questions, 2015), just the *Resource Description Framework* (RDF) and SPARQL. Thirdly, unlike most rule-based approaches over *Web Ontology Language* (OWL) such as SWRL, SPIN offers closed world semantic and the unique-name assumption, which we find more intuitive for the TAS application domain (SWRL Frequently Asked Questions, 2015). However, despite the explained arguments, all the aforementioned technologies are suitable for building a semantic TAS by using rules which are based on the EWE ontology presented in this paper.

Some researchers consider that TASs borrow inspiration from Web mashups; i.e. applications generated by combining content, presentation, or application functionality from disparate Web sources (Vladimir, Budiselic, & Srbljic, 2015). This is because TASs also compose services and combine data from different Web data sources. After analysing the existing TASs, we identify five dimensions in which they may be compared to Web mashups as shown in Fig. 1. The first dimension is the *number channels or widgets*, defined as the number of sources or elements that are available to be used in the compositions. It depends on the number of data sources that are integrated in the mashup engine or the TAS. In both cases, it depends on the developers that support the system, so their performance in this dimension is similar. The second dimension is the power or the automations, or the resulting mashup; i.e. the *capabilities* offered to users. Mashups lead this field since they offer data visualisation, filtering, and processing to name a few. Next, the *ease of use* is also considered; i.e. how easy is to configure the automations or build the mashups for the user. In this dimension, mashups usually involve complex data pipelining when not programming skills. On the other hand, TASs are characterised by presenting their users intuitive interfaces to build their automation. The ease of use in one the upsides that made TASs so popular these days, since almost all Internet user is a potential TAS user. Customisability or *personalization* is another advantage of TASs. Each user is able to compose their own task automations, as opposed to mashups, that are packed serviced delivered to the user. More importantly, TASs allow users to utilize their own personal streams. However, since TASs are focused on being as easy to use as possible, some customisation capabilities are skipped. Finally, *portability* is the fifth dimension. This is the capability of exporting and importing automations/mashups to different engines or systems. Both, TASs and mashups perform very low in this dimension because typically there is not any support for this.

As explained, all these dimensions are compiled in the chart shown in Fig. 1. According to the considerations given, the reason of the higher penetration of TASs compared to mashups is their personalization and ease of use. There is a third kind of system considered in the chart: semantic TASs, i.e. TASs which employ semantic technologies. These provide several advantages over classic TASs in most of the dimensions shown in the chart. The use of the EWE ontology, presented in Section 3, supports these enhanced TASs, as illustrated in the use case detailed in Section 4.

3. Evented WEb ontology (EWE) model

In this section, we present the EWE Ontology, which models the most important aspects of TASs from a descriptive approach, that enables service discovery and semantic rule definition featuring reasoning over LOD. It stands as a reference model to define and describe TASs. EWE is available online (Coronado, Iglesias, & Serrano, 2015a).

After analysing the TASs described in Section 2, we derive a model that contains the most relevant concepts (and relations between them) presented by those services, using the most widespread terminology. The EWE Ontology comprises two major

Table 2
Nomenclature in Task Automation Services.

Concept/name	Ifttt	Zapier	Kinetic	Triggers	Tasker	On{x}
Rule	Recipe	Zap	Rule	Trigger	Profile	Rule
Channel	Channel	App	Endpoint	–	– (Plugin)	Object
Event	Trigger	Trigger	Event	Input	Context	Trigger
Action	Action	Action	Action	Output	Act./Task	Action

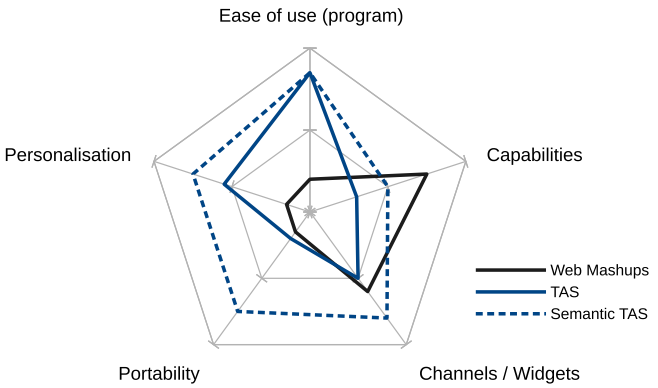

Fig. 1. Comparision between mashups and rule based TASs.

parts: the main objects and properties defined in the EWE ontology, and the mapping that relates EWE to existing ontologies such as *Tag Ontology* (TAGS) or *Friend of a Friend ontology* (FOAF) (Amini, Ibrahim, Othman, & Selamat, 2014). The ontology design methodology and examples of rules defined with EWE are also given in this section.

3.1. EWE design methodology

The final model we propose is the result of an iterative development process consisting of three steps: (i) we analyse each of the TASs considered, identifying features and functionalities, then we extract the concepts and properties they address; (ii) we define a model that formally describes those elements; and finally, (iii) we evaluate the model against the different use cases considered, i.e. we check how suitable it is for describing rules from sites such as Ifttt or Zapier. After each iteration, we repeat the process, including some new elements that the results have shown to be relevant, and remodelling others to best describe the domain. Each iteration refines the model, i.e. classes and properties are included, modified, or even removed from the ontology to make it not only broader but also more accurate, thus more useful.

Therefore, the ontology design has been undertaken by an iterative incremental development as in most agile software development methodologies. Fig. 2 presents a diagram of a excerpt of the resulting model showing the major classes and properties.

3.2. EWE elements: Main classes and properties

This section addresses the main objects and properties defined in the EWE ontology.

3.2.1. Main classes

The core of the ontology comprises four major classes: Channel, Event, Action and Rule. The description of particular TASs or use case scenarios may inherit from them, creating new classes that are specific to the domain. We detail the usage of the main classes below.

The class *Channel* defines individuals that either generate Events, provide Actions, or both. In the context we refer to, Channel usually defines Web services. For instance, according to this definition *Dropbox* is channel because it generates Events every time a new file is uploaded or changed, and it provides the capability (Action) to rename files or move them to another folder. Of course, Dropbox channel would offer much more Events and Actions. Furthermore, sensors and actuators are also modelled as Channels since they produce Events and/or provide Actions; e.g. a wearable device with GPS programmed to generate alerts when it is near certain locations.

The class *Event* defines a particular occurrence of a process, which may trigger rules in the TAS. As opposed to the definition given in other ontologies (Raimond & Abdallah, 2007), in EWE events are instantaneous, i.e. they have no duration over time. The Event class may be subclassed to define particular types of Events. For instance, the class *NewChatMessage* is subclass of Event and defines the type of event that is generated when a new chat messaged is typed. Instances of Event class offer information of the particular event; e.g. instances of *NewChatMessage* have information of the chat message and the date when it was sent. Event individuals are generated by a certain Channel, when triggered by the occurrence of the process that defines them, e.g. typing and sending a message in *GoogleHangouts* triggers the generation of an event instance of type *NewChatMessage*. We say that the channel that generates the event is the event generator; *GoogleHangouts* is the event generator. Definitions of Event subclasses are not bound to a Channel since different channels may generate the same events; e.g. both, *GoogleHangouts* and *Whatsapp* channels may generate instances of *NewChatMessage*.

The class *Action* defines an operation or process provided by a Channel. Actions produce effects whose nature depends on the action nature. These include: producing a log message, modifying a public or private state on a server, a physical action such as switching on a light, etc. The effect can even trigger an Event generation, either by the same Channel or a different one. In a similar manner to Events, the Action class may be subclassed to specific actions. Again, Action definitions are not bound to a Channel, because different channels can support the same actions; e.g. *Linkedin* and *Facebook* channels provide the *ChangeProfilePicture* action.

The class *Rule* defines an *Event–Condition–Action* (ECA) rule, triggered by an Event that produces the execution of an Action. Rules define particular interconnections between instances of the Event and Action classes transferring information from the event to the action. EWE employs the SPIN framework (Vambenepe, 2009) to model the execution logic of the rules. ECA rules can be fully described as SPARQL (Kim, Moon, & Kim, 2014) construct queries, i.e. as SPIN rules. The advantage of using SPARQL for describing the rules is that it facilitates the inclusion of reasoning over LOD since the SPARQL endpoints that are available in the cloud may be directly queried.

3.2.2. Main properties

In addition to the description of the main classes, we offer a list of the important properties and the purpose of including them within the overall EWE ontology. These are *hasParameter*, *hasCategory*, *activeChannel*, *hasCreator* and *spin:rule*.

The property *hasParameter* presents the parameters of an Action or an Event. Instances of Event and Action can rarely be defined without some configuration parameters. They differentiate each individual, e.g. the body and the sender of an *EmailReceivedEvent*. Each parameter should be provided by the appropriate *subPropertyOf* of *hasParameter*. Given the *hasParameter* example, the body is given by a *hasBody* property, and the sender by a *hasSender* property both sub properties of hasParameter.

The property *hasCategory* indicates that a Channel, Event or Action belongs to a certain category. An element may have more than one category. The EWE Ontology does not provide a taxonomy of channels, events, or actions; but it facilitates building that classification. The Channel categorization is important for channel discovery and recommendation. This happens not only with discovery and recommendation methods based on profiling, but also with methods based on semantic similarity. Besides, expert systems may use the channel categorization and help a user to find

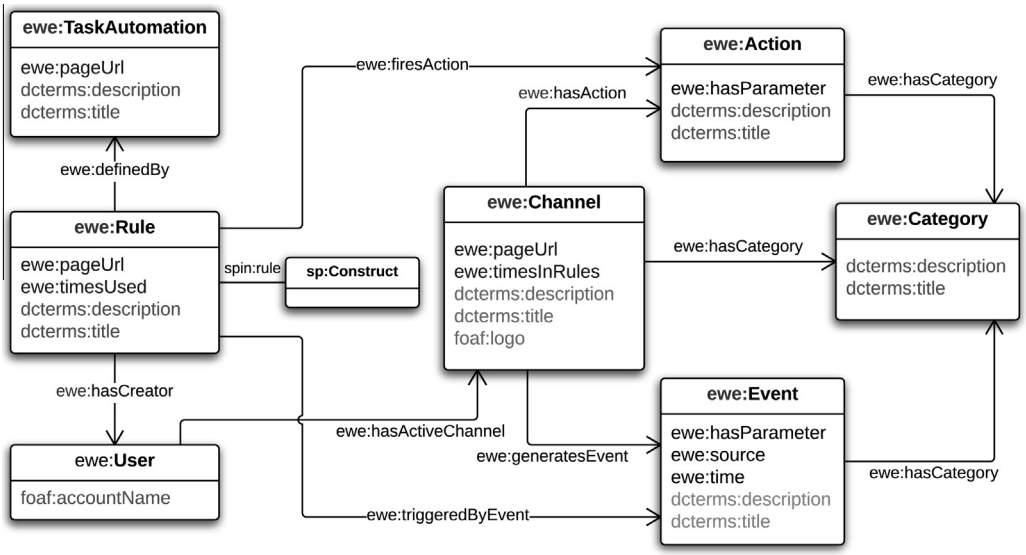

Fig. 2. Detail of the EWE ontology model.

alternatives to other channels; e.g. when a channel is not available due to geographical restrictions. The range of the hasCategory property is the class Category, a subclass of the *Concept* class from the *Simple Knowledge Organization System* (SKOS) (Amini, Ibrahim, Othman, & Nematbakhsh, 2015) ontology.

The *hasActiveChannel* property links users to Channel on which they have an account, i.e. a channel they can include in the Rule definition. Combined with the category of the channels, this information may be used to reason over the alternatives that a user may have to a particular choice of channel.

The property *hasCreator* links instances of Rule to its creator, an *onlineAccount* from the FOAF ontology. The authors of the rules are significant information for data analysis purposes and recommendation systems.

The *spin:rule* property links rule instances with the SPARQL execution logic described using the spin vocabulary. In addition to that property, *ewe:triggeredByEvent* and *ewe:firesAction* properties are defined to link the rule with the event that triggered it and the consequent action that was executed. These two properties simplify the query required to select the events and actions that are related to a rule instance, removing the extra triple patterns required to navigate from the Rule instance to the Event and Action instances through the *spin:rule*.

3.3. Mappings from external ontologies in EWE

EWE has been developed based on a number of existing classes and properties from external ontologies as possible in the spirit of the linked data philosophy. Hence, EWE enhances search, interoperability and linking data because several tools and search engines are capable of using those vocabularies. The connections between EWE and external ontologies are summarised in Fig. 3 and detailed below.

- **SPIN**: the contribution of SPIN to EWE is essential, since rule grounding is done by means of *sp:Construct* instances. The implementation of *SPIN* rules as members of EWE rules connects Events to Actions in the same way they are connected in TASs.
- **FOAF**: EWE *User* class inherits from FOAF *onlineAccount* and is connected by an *account* to a FOAF *Agent*. This assures better interoperability and search operation, as well as enhances profiling by allowing using external context.

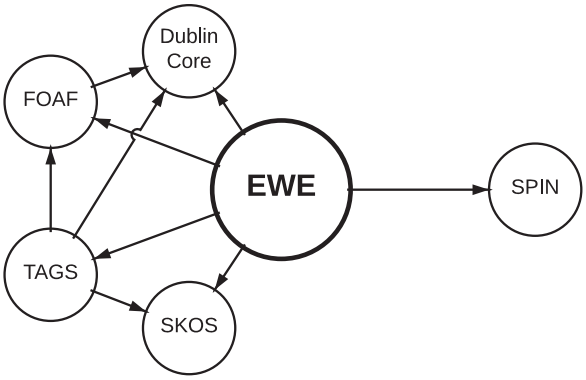

Fig. 3. External ontologies mapping.

- **TAGS**: the TAGS ontology provides a common definition of tags. This ontology can extract information from third party sources and also can provide smart connection and recommendations depending on the tags set. The *taggedWithTag* property, from the TAGS ontology, links rules to each of their associated tags. This is thus another method for classifying Rules.
- **SKOS**: SKOS is proposed for representing structured vocabularies for the Semantic Web. Several tools search and reason using the relations they define. The EWE concept *Category* inherits from the SKOS Concept.
- **dcterms**: most of the administrative properties of every class of EWE are defined according to *DCMI Metadata Terms* (dcterms) metadata elements. This improves the searches of elements of the ontology.

3.4. Examples of EWE use

In order to give a better idea of how specific Channels, Events and Actions are defined using EWE; we show a short excerpt of a Gmail channel definition in Listing 1. The channel represents the Gmail channel implemented in a particular TAS (e.g. Gmail channel from Ifttt[5]). We use the punning mechanism of OWL2 to attach

[5] Ifttt Gmail channel website: http://ifttt.com/gmail.

```
ewe-mail:Gmail a owl:Class ;
    rdfs:subClassOf     ewe:Channel ;
    rdfs:type           ewe:Channel ;
    dcterms:title       "Gmail"^^xsd:string ;
    dcterms:description "Webmail␣by␣Google"^^xsd:string ;
    ewe:generatesEvent  ewe-mail:NewEmail ;
    ewe:hasAction       ewe-mail:SendEmail .

ewe-mail:NewEmail a owl:Class ;
    rdfs:subClassOf     ewe:Event ;
    dcterms:title       "Any␣new␣email␣in␣inbox"^^xsd:string ;
    dcterms:description "New␣email␣arrives␣in␣Gmail"^^xsd:string .

ewe-mail:SendEmail a owl:Class ;
    rdfs:subClassOf     ewe:Action ;
    dcterms:title       "Send␣an␣email"^^xsd:string ;
    dcterms:description "Send␣an␣email␣from␣Gmail"^^xsd:string .

ewe-props:hasBody       rdfs:subPropertyOf :hasParameter.
ewe-props:hasSender     rdfs:subPropertyOf :hasParameter.
ewe-props:hasToAddres   rdfs:subPropertyOf :hasParameter.
ewe-props:hasSubject    rdfs:subPropertyOf :hasParameter.
```

Listing 1. Channel definition excerpt.

properties to that channel. As a result, the description explicitly states that the channel may generate events of a particular class (*ewe-mail:NewEmail*) and provides a particular action for them (*ewe-mail:SendEmail*). The example also includes a few related *hasParameter* subproperties.

An example of event and action instances with grounded parameters, which are based on the concepts defined in the listing given above, is presented in Listing 2.

As explained, rules are conveniently defined as SPARQL construct queries, and then stored as SPIN objects in RDF format. Since SPIN syntax is more complex than SPARQL syntax, for the sake of readability, we present an example of EWE rule in SPARQL format in Listing 3. The example shows a rule that is triggered every time an event of type *NewEmail* happens. Events are filtered so only those with the label 'important' are considered. Then, a *NewChatMessage* action is generated. The message sent contains the email address captured from the incoming event. As seen, the event–condition part of the ECA rule is given by the WHERE clause and the FILTER function. The action part is given by the CONSTRUCT clause, using the variables bound in the WHERE clause.

The rule of the example does not restrict the incoming event to be of a particular Channel. It will be triggered by events of the appropriate type regardless of what channels generates them: the Gmail channel, the Yahoo channel, or any other. Therefore, EWE allows defining rules where the Channels, Events, and Actions involved are not bound. This example illustrates how EWE enables a new kind of rule, looser than the classical rules defined by the TASs explored in the related works section.

4. A prototype of a semantic TAS with EWE

This section illustrates the use of EWE to implement a semantic TAS which facilitates service discovery and enables reasoning over LOD. For that purpose, we have developed a prototype of a TAS with several channels integrated within the platform. These channels generate events that are described following the EWE model and processed by the engines using SPIN rules. This section also contributes to give the reader an explanation of (i) how events are distributed in a semantic TAS based on EWE, (ii) how Actuators and Web Services handle actions, and (iii) how the Rule Engine processes incoming events and fires rules. Nevertheless, the prototype described in this section is shown as a use case, and it is not intended to be a reference implementation, especially in terms of performance, scalability, etc.

Fig. 4 presents the general architecture of the prototype whose components are distributed in three layers. The generation layer shows that events may come from either Web Services or Sensors, thus there are channels of different nature. Since several processing modules take part in the execution flow, and in order to support other modules to be included in the future, a transportation layer is needed to distribute these events among them. Engines in the processing layer are in charge of executing the rules when their triggers appear. In particular, the Semantic Rule Engine is able to connect to SPARQL endpoint of LOD cloud and to use that information to evaluate the rule conditions.

When Web Services and Sensors generate events, the Service API Adapters or the Sensor Network, respectively, generate a message that is pushed to the *Message Oriented Middleware* (MOM) in the transportation layer. A payload with the RDF representation of the event has to be included in these messages. The MOM is in charge of distributing the messages to the subscribers according the publish-subscribe pattern (Eugster, Felber, Guerraoui, & Kermarrec, 2003). Thus, the Semantic Rule Engine in the processing layer subscribes to the event messages from the MOM. In the same manner, the Sensor Network and the Service API Adapters subscribe to the action messages. When these two modules receive the action messages from the MOM, they forward these messages

```
:new-email#1 a ewe-mail:NewEmail ;
    ewe-props:hasSubject    "the␣email␣subject"^^xsd:string ;
    ewe-props:hasBody       "the␣email␣body"^^xsd:string ;
    ewe-props:hasSender     "from-this@email.com"^^xsd:string .

:send-email#1 a ewe-mail:SendEmail ;
    ewe-props:hasSubject    "the␣email␣subject"^^xsd:string ;
    ewe-props:hasBody       "the␣email␣body"^^xsd:string ;
    ewe-props:hasToAddres   "to-this@email.com"^^xsd:string .
```

Listing 2. Event and action instances.

```
CONSTRUCT {
  ?action a ewe:NewChatMessage .
  ?action ewe-props:message ?msg .
}
WHERE {
  ?event a ewe-mail:NewEmail .
  ?event ewe-mail:FromAddress ?fromAd .
  BIND (fn:concat("Important␣message␣from:␣", ?fromAd) AS ?msg) .
  FILTER {
    ?event ewe-props:haslabel "important"
  }
}
```

Listing 3. Rule in SPARQL form.

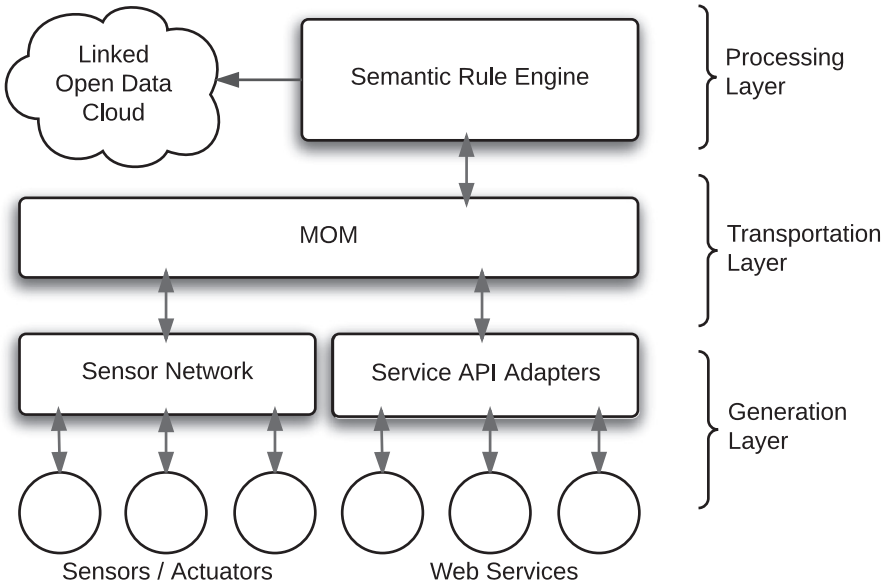

Fig. 4. General architecture of TAS implementation with EWE support.

to the Actuators or the Web Services, respectively, which are responsible for interpreting and executing the actions. This communication based on messages gives loose coupling to the architecture and fits the Semantic Web vision (Berners-Lee, Hendler, & Lassila, 2001).

The Semantic Rule Engine is responsible for: processing the incoming event messages; executing the rules; and discarding the messages once they have been processed to avoid executing the same rule multiple times for the same event. In our prototype, which employs the SPIN notation and a SPARQL engine, the manner that rules are processed relies on the fact that these rules can be expressed as SPARQL queries (see example in Listing 3). These SPARQL queries are run every time an event is received in the SPARQL engine. After executing the rules in their SPARQL query

form, the new actions constructed by these queries are pushed to the MOM. Then the MOM can deliver these actions to the subscribers, e.g. Sensor Networks and Service API Adapters.

In order to provide a complete overview of how an event is generated, processed by the rule engine, and the constructed action is executed; we illustrate all the steps for processing the rule "Whenever I receive an email labelled as important, text me a notice" shown in Listing 3. First of all, a user (e.g. Ann) should have created this rule. Then, the rule is loaded in the Semantic Rule Engine. When Ann receives an email in her Gmail account, the Service API Adapter, which is connected to the Gmail service, generates an *ewe-mail:NewEmail* event message that is pushed to the MOM. Then, the MOM distributes the new message to the subscribers. In our case, the Semantic Rule Engine receives the message, extracts the RDF payload, and executes the rule. In case the email has the label 'important', a new action message will be constructed (*ewe:NewChatMessage* action) and pushed to the MOM. Consequently, the MOM distributes the new action message to the subscribers. In the example, the Service API Adapter receives the message, extracts the service parameters, and sends a request to the Hangout Service API, which texts Ann informing about the important email just received.

Based on this general architecture for a prototype, we have modelled a scenario about a meeting arrangement. Fig. 5 shows an instance of a meeting event similar to those the semantic engine work with. As seen in the figure, some of the classes and instances are labelled as *Enhanced* (according to the legend). We refer to types, properties and instances that are not directly provided by the event generator (the Web Services or Sensors), but obtained from semantic endpoints (from the LODcloud). The information retrieved is sometimes several steps away from that information initially given. For instance, with the email address of the attendees, we can derive: their social ids described with the FOAF ontology; then their Linkedin account; and, then their public profile. In a similar manner, the location of the meeting, its agenda, or the project to which the meeting is related; may be retrieved from the semantic endpoints.

There are a large number of vocabularies available that we can use to describe different types of events as well as related information that may be derived from the data LOD cloud. In Table 3, we present several channels that are available in Ifttt. For each of them, we suggest how the information provided by EWE may enhance the TAS as well as which mappings could be used.

This enables the definition of rules that include clauses in the condition part that are not directly related to the information given within the event. We could define a rule that reads "When I confirm the attendance to a meeting, add all other attendees to Linkedin as contacts". In this case, the Semantic Engine would have to browse the LOD to get the foaf:Agent of the attendees, and in case they have a foaf:OnlineAccount that describes a Linkedin account, obtain the information needed to add them as contacts. This is an enlightening example of reasoning over large scale data outside the platform which is possible thanks to the contribution presented in this paper, the *Evented WEb ontology* (EWE) model. The implementation of the semantic TAS used in this section is available online (Crespo, Coronado, & Iglesias, 2014).

5. EWE evaluation

In this section, we describe the evaluation process that was carried out to show how EWE success at modelling some of the TASs revised in the related works section while the shortcomings of these approaches are addressed.

According to Brank et al. (Brank, Grobelnik, & Mladenić, 2005), when evaluating an ontology is more practical to focus the evaluation of different levels of it separately. They identify several levels —lexical, hierarchical, semantic, application, etc. — and determine which evaluation approaches are more suitable for evaluating each level. In general, selecting which levels should be evaluated depends on the nature of the ontology itself. In our case, the EWE ontology was designed to describe all relevant features from existing TASs. Altogether, those TASs form a wide domain to be modelled, which possesses important differences between their feature sets. Hence, the evaluation described will focus on the *lexical* level. Besides, in Section 4 we show an architecture based on EWE to improve the capabilities of a typical task automation scenario. Thus, that use case addresses the *application* level evaluation.

Since there is no golden standard to compare with, a data-driven evaluation was selected. To do so, we first analysed a set of selected TASs and extracted their main features to define a corpus; then, we define and apply metrics to get the results from the corpus.

5.1. Feature extraction

The feature extraction process consists of the formalization of a conceptual model as a list of features. Then, we will use those extracted features to compile the corpus to use. In the process, we thoroughly analysed Ifttt, Zapier, on{x}, and Tasker, to obtain four theoretical models that represent them. Ifttt and Zapier were included because they have many features in common with most of the TASs we analysed in Section 2, but Tasker and on{x} were also considered because they have many unique features.

Algorithm 1. Feature extraction process

```
Data: Conceptual model of the Task Automation Service
Result: Formal list of features related to the Task Automation
  Service
foreach Concept in Model do
    addFeature: ∃ Concept
    foreach Property of Concept do
        addFeature: hasProperty(Concept, Property)
    end
    foreach Relationship of Concept do
        addFeature: hasRelation(Concept, Relationship)
        foreach Restriction of Relationship do
            addFeature: restriction(Concept, Relationship, Restriction)
        end
    end
end
```

We capture the feature list of each TAS by applying Algorithm 1 to the corresponding conceptual model. As it shows, we considered features of four types: *concepts*, *properties*, *relationships* and their *restrictions*. A sample of the feature extraction process outcome is shown in Table 4. The summary of the results of the process carried out with the four TASs selected is also presented in Table 5.

With the whole list of features extracted, we compile the corpus to use in the evaluation. The list of features extracted from each TAS is modelled as a logical vector where each position represents a feature from the corpus: '1' or '0' on a position means that the feature is in the TAS or is not included, respectively.

$$\mathcal{T}_j = (f_1, f_2, \ldots, f_n), \quad f_n \in \{0, 1\} \tag{1}$$

The EWE ontology is also represented as a logical vector in order to apply the metrics as explained below.

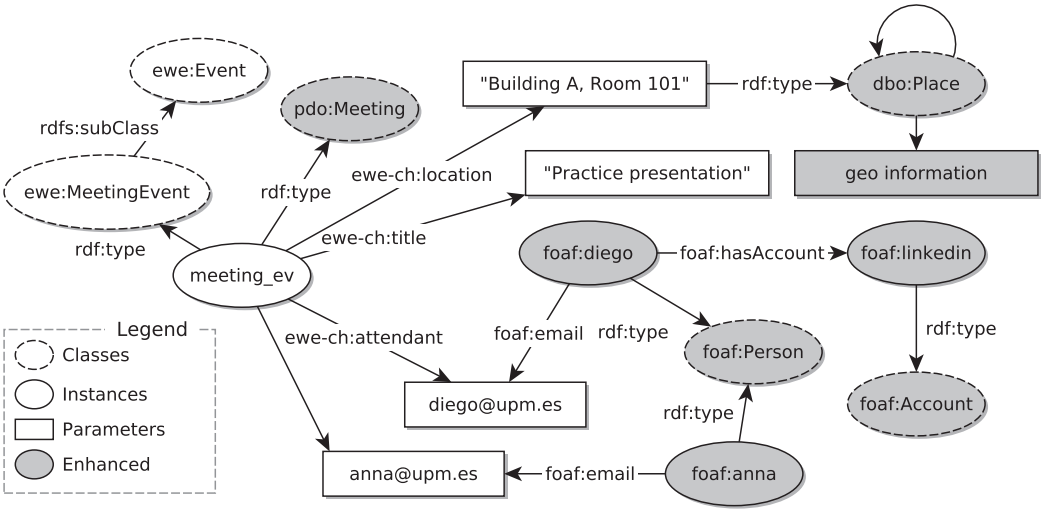

Fig. 5. Use case example event model with linked-data.

Table 3
Examples of vocabularies that may be used to enhance events.

Event generator (channel)	Enhancements	Vocabularies
Blogger, ESPN, Evernote	Post NLU. Emotion analysis	sioc, onyx, sport
Delicious, Evernote	Tagging and categorization	tags, skos
Facebook, Linkedin, G+	Unified account information	foaf, vcard, curric
Foursquarem, Life360	Geolocation information	geo
Last.fm, SoundCloud	Track recognition. Music emotions	music, af, onyx
Wemo, SmartThings	Sensor information, deployment loc.	ssn

5.2. Running the evaluation

We use the *coverage* and *accuracy* functions to the measure of the quality of the ontology. Coverage measures the spread of the ontology: the more features included, the greater the coverage. We also use the accuracy function, to keep the ontology as sharp and concise as possible. Both of these are in a trade-off between embracing many features, and perfectly fitting the corpus.

Coverage describes the range of relevant features from the domain in use that are defined in the ontology. Currently, the state of the art has different ways for evaluating the coverage of an ontology over a knowledge domain. The straightforward approach is measuring the size of the ontology, i.e. counting classes. This can be sufficient for a quantitative view (Ruan & Yang, 2010), even when it lacks the qualitative aspect (Ouyang, Zou, Qu, & Zhang, 2011).

Therefore, coverage is defined as the proportion of features from the TAS that are defined in the ontology. With $\mathcal{T}$ being the TAS and $\mathcal{O}$ the ontology:

$$Coverage(\mathcal{O},\mathcal{T}) = \frac{Dim(\mathcal{O}\cap\mathcal{T})}{Dim(\mathcal{T})} \tag{2}$$

Accuracy is defined as the Jaccard similarity (Jaccard, 1901).

$$Accuracy(\mathcal{O},\mathcal{T}) = \frac{Dim(\mathcal{O}\cap\mathcal{T})}{Dim(\mathcal{O}\cup\mathcal{T})} \tag{3}$$

We introduced the accuracy metric to keep the ontology as tightly suited to the corpus as possible, i.e. as simple and useful as possible. If the ontology embraces too many features, it becomes too big in size, being too complex, and thus hard to use.

The results of the process are generated using formula (2) and formula (3), and are summarised in Table 6. As anticipated, the results are favourable to Ifttt and Zapier, which shows our modeling efforts to focus on the common TASs features. Including most of the features from Tasker or on{x} in EWE would have increased their coverage measure at a cost of lowering the accuracy metric of Ifttt and Zapier.

Then, we undertook a scrapping process to extract data of channels, events, actions and rules from the websites of Ifttt, Zapier, on{x} and Tasker. In this case, the data is extracted from the application itself. The results showed 55,914 rules connecting 718 events to 910 actions that were provided by 222 different channels. To extract the data from the websites, we used *Scrappy* (Fernández-Villamor, Iglesias, & Garijo, 2012), since it returns data in RDF and can be instructed to format it according to EWE.

The crawling, made during the first quarter of 2014, obtained the results that are summarised in Table 7. Although the four TASs studied are similar, the results show great differences in the

Table 4
List of features extracted from Ifttt rules model.

Feature	Type
∃ Rule	Concept
hasProperty(Rule, title)	Property
hasProperty(Rule, dateCreation)	Property
hasProperty(Rule, timesUsed)	Property
hasRelation(Rule, tagged)	Relation
restriction(Rule, tagged, range(Tag))	Restriction
hasRelation(Rule, hasCreator)	Relation
restriction(Rule, hasCreator, range(User))	Restriction
hasRelation(Rule, triggeredBy)	Relation
restriction(Rule, triggeredBy, range(Event))	Restriction
restriction(Rule, triggeredBy, maxCardinality(1))	Restriction

Table 5
Feature extraction process outcome.

Feature type	Ifttt	Zapier	Tasker	On{x}
Concepts	11	14	16	21
Relations	12	24	18	26
Properties	22	13	30	36
Restrictions	31	37	32	40
Total	76	88	96	123

Table 6
Coverage and accuracy results for the EWE ontology.

Target	Coverage	Accuracy
Ifttt	0.96	0.91
Zapier	0.92	0.84
on{x}	0.42	0.38
Tasker	0.51	0.48

Table 7
Scraping process results.

Class/Individuals	Ifttt	Zapier	on{x}	Tasker
Rule	67904	–	39	–
Channel	65	141	–	22
Event	113	463	70	72
Action	77	263	352	218

ratios between channels, events, and actions. Ifttt and Zapier, both Web platforms, present around two actions per channel on average and a higher ratio of events. Tasker and on{x}, both smartphone apps, show opposite results. They present a greater number of actions than events: three times the number of events for Tasker, and five times for on{x}. This is because their control over the device allows them to provide many actions.

Finally, the scraped data were loaded into a SPARQL endpoint in order to make them available as a directory of channels, events and actions available in several TASs. We defined a set of meaningful queries that a common user would make in order to retrieve information about the channels available, the events of a particular type, etc. We coded those queries using SPARQL and executed them. Table 8 gathers some of them together with their execution outcomes.

These experiments show that the EWE ontology this paper contributes with effectively models the TAS domain; i.e. the classes and properties defined by EWE are suitable for describing TAS channels, actions, events and rules. This conclusion is not only supported by the metric results shown in Table 6, but also by the process of automatically extracting data from TASs websites and applications. Finally, the scenario described around the architecture and prototype proposed in Section 4 supports the hypothesis that semantic TASs address the identified TASs drawbacks.

6. Conclusion and future work

This paper offers an overview of *Task Automation Services* (TASs); their features, to what they owe their popularity; and, their major drawbacks. The paper surveys the most important commercial platforms and mobile apps that feature task automation, and

Table 8
SPARQL example queries using the EWE ontology.

Query	SPARQL query	Results
How many channels are supported by each TAS?	`SELECT ?tas (COUNT(?serv) AS ?channels) WHERE { ?serv rdfs:subClassOf ewe:Channel . ?serv ewe:supportedBy ?tas } GROUP BY ?tas`	Zapier (141), Tasker (22), Ifttt (65)
Which channel categories are defined?	`SELECT DISTINCT ?category WHERE { ?serv rdfs:subClassOf ewe:Channel . ?serv ewe:hasCategory ?category }`	Event Management, CRM, Phone, Developer Tools, Email Marketing, Social and 9 more
Which channels are categorised as social?	`SELECT ?servName WHERE { ?serv rdfs:subClassOf ewe:Channel . ?serv ewe:hasCategory ewe:Social . ?serv dcterms:title ?servName }`	Wordpress, buffer, chatter, facebook, Twitter, and 7 more
How many actions can be executed by means of the Tasker TAS?	`SELECT (COUNT(?act) AS ?actsProvided) WHERE { ?chan ewe:hasAction ?act . ?chan ewe:supportedBy ewe:Tasker }`	204 actions
How many rules are triggered by Gmail channel events?	`SELECT (COUNT(?rule) AS ?ruleCount) WHERE { ?srule rdf:type ewe:Rule . ?srule ewe:triggeredByEvent ?Event . ewe:Gmail ewe:generatesEvent ?Event . } GROUP BY ?rule`	7240 rules

compares them to approaches in the specialized literature. This survey is summarised in Table 1. Furthermore, an extended report is available online (Coronado et al., 2015b) as additional material of this paper.

This survey arises several conclusions. Firstly, TASs that focus on different user audience implement different feature sets; e.g. ifft (IFTTT put the internet to work for you, 2015) or Automateit (Automateit turn your smartphone into a genius-phone, 2015), that target the general audience, offer neat and easy to use interfaces with connectivity to smartphone channels. On the other hand, Zapier (Zapier the best apps. Better together, 2015) or Cloudwork (Cloudwork connect your business apps, 2015), that target the business users, pay more attention to the power and capabilities of the rules. Secondly, there is a lack of agreement in the nomenclature used to refer to the same concepts, Table 2 gathers these variations. Thirdly, the TASs studied suffer from two major shortcomings, they lack of: support for channel discovery; and, reasoning over contextual data and *Linked Open Data*(LOD).

Since several authors consider TASs an evolution of Web-mashups, this paper also describes a framework embracing five dimensions to compare them. The conclusion of this study is that TASs perform better in the personalization and easy of use dimensions; while Web-mashups feature better automation capabilities. In the comparison, we also include TASs with semantic capabilities, as the prototype introduced in Section 4 which improves TASs performance in all five dimensions.

Beyond the TAS systems and the mashups survey, this paper central contribution is the definition of the *Evented WEb ontology* (EWE). EWE models TASs from a semantic approach, enabling reasoning over LOD and facilitating channel discovery. The paper introduces a realistic use case scenario where the progress made beyond the state of the art is illustrated. Besides, it presents an implementation of a semantic TAS with support to EWE rules. This prototype employs *SPARQL Inferencing Notation* (SPIN) to describe *Event–Condition–Action* (ECA) rules. Finally, the paper exhaustively evaluates the EWE ontology proposed by using a data-driven approach. The evaluation conclusion is that EWE effectively models four popular commercial TASs while it addresses the shortcomings observed in them.

For sake of reproducibility, the material developed within this work, including code and detailed results, is available online for the interested reader: the EWE ontology (Coronado et al., 2015a), the implementation of the semantic TAS (Crespo et al., 2014), and the detailed results of the TASs survey (Coronado et al., 2015b).

Our main future work is to develop an expert system to assist users to configure new rules in the semantic TAS proposed. To feed the expert system knowledge and thus to enhance the assistance provided, a clustering method can be proposed to group channels, events, actions and rules. This expert system may recommend related rules and variations given a certain rule. Given a channel, it will bring up those similar channels according to different criteria: frequently connected in rules, same category, similar event and action sets, etc. This clustering approach can be used to infer Channel groups, as given by some TASs. We aim to create a disambiguation method that combines channels which are equal but described in different TASs. This method may determine which events and actions correspond to the same purpose to merge them. The EWE ontology has to be adapted to support this issue. Besides, some TASs providers have shown their interest in building a tagging system for rules, as already covered by EWE. In this regard, we intend to extract and analyse the tags when available in order to incorporate them into the expert system knowledge.

Another important future work is the evaluation of alternative manners for passing messages and their performance in the proposed TAS architecture. In this vein, to further explore SPARQL Streaming technologies (Bolles, Grawunder, & Jacobi, 2008; Barbieri, Braga, Ceri, Della Valle, & Grossniklaus, 2009) and the feasibility of processing RDF events in real time, we will extend the prototype presented in this paper with the best alternative assessed. This is not required in a low rate event scenario, as the one described in Section 4. Nonetheless, real time events are desirable in case of having sensors or other high rate streams.

Finally, we intend to develop a visual interface for the semantic TAS to allow users to compare the system features to other TASs.

Acknowledgments

This research work is supported by the Spanish Ministry of Economy and Competitiveness under the R&D project CALISTA (TEC2012-32457); by the Spanish Ministry of Industry, Energy and Tourism under the R&D project BigMarket (TSI-100102-2013-80); and, by the Autonomous Region of Madrid through the program MOSI-AGIL-CM (grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER).

References

Amini, B., Ibrahim, R., Othman, M. S., & Nematbakhsh, M. A. (2015). A reference ontology for profiling scholar's background knowledge in recommender systems. *Expert Systems with Applications, 42*(2), 913–928. http://dx.doi.org/10.1016/j.eswa.2014.08.031. <http://www.sciencedirect.com/science/article/pii/S0957417414005132>.

Amini, B., Ibrahim, R., Othman, M. S., & Selamat, A. (2014). Capturing scholar's knowledge from heterogeneous resources for profiling in recommender systems. *Expert Systems with Applications, 41*(17), 7945–7957. http://dx.doi.org/10.1016/j.eswa.2014.06.039. <http://www.sciencedirect.com/science/article/pii/S0957417414003807>.

Atooma a touch of magic (2015). <http://www.atooma.com/> (Accessed: 2015-03-31).

Automateit turn your smartphone into a genius-phone (2015). <http://automateitapp.com/> (Accessed: 2015-03-31).

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E., & Grossniklaus, M. (2009). C-SPARQL: SPARQL for continuous querying. In: *Proceedings of the 18th international conference on world wide web* (pp. 1061–1062).

Beach, A., Gartrell, M., Xing, X., Han, R., Lv, Q., Mishra, S., et al. (2010). Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the eleventh workshop on mobile computing systems & applications* (pp. 60). New York: ACM Press.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 28–37.

Boley, H., Paschke, A., & Shafiq, M. O. (2010). RuleML 1.0: The overarching specification of web rules. In M. Dean, J. Hall, A. Rotolo, & S. Tabet (Eds.). *Semantic web rules – International symposium, ruleml 2010, washington, dc, usa, october 21–23, 2010. proceedings* (Vol. 6403, pp. 162–178). Springer. http://dx.doi.org/10.1007/978-3-642-16289-3_15.

Bolles, A., Grawunder, M., & Jacobi, J. (2008). *Streaming SPARQL-extending SPARQL to process data streams*. Springer.

Brank, J., Grobelnik, M., & Mladenić, D. (2005). A survey of ontology evaluation techniques. In: *Proceedings of 7th international multi-conference on information society*. Ljubljana.

Cloudwork connect your business apps (2015). <http://cloudwork.com/> (Accessed: 2015-03-31).

Coronado, M., & Iglesias, C. A., & Serrano, E. (2015a). EWE ontology specification. <http://www.gsi.dit.upm.es/ontologies/ewe/> (Accessed: 2015-03-31).

Coronado, M., Iglesias, C. A., & Serrano, E. (2015b). Task automation services study. <http://www.gsi.dit.upm.es/ontologies/ewe/study/full-results.html> (Accessed: 2015-03-31).

Crespo, C., Coronado, M., & Iglesias, C. A. (2014). DrEWE an intelligent platform for task automation. <https://github.com/gsi-upm/DrEWE> (Accessed: 2015-03-31).

de Alba, J. M. F., Campillo, P., Fuentes-Fernández, R., & Pavón, J. (2014). Opportunistic control mechanisms for ambience intelligence worlds. *Expert Systems with Applications, 41*(4, Part 2), 1875–1884. http://dx.doi.org/10.1016/j.eswa.2013.08.084. <http://www.sciencedirect.com/science/article/pii/S0957417413007057>.

De Francisci Morales, G., Gionis, A., & Lucchese, C. (2012). From chatter to headlines. In *Proceedings of the fifth acm international conference on web search and data mining* (pp. 153). New York: ACM Press.

Elastic.io integrate once. connect many (2015). <http://www.elastic.io/> (Accessed: 2015-03-31).

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys, 35*(2), 114–131. http://dx.doi.org/10.1145/857076.857078.

Fernández-Villamor, J. I., Iglesias, C. A., & Garijo, M. (2012). First-order logic rule induction for information extraction in web resources. *International Journal of Artificial Intelligence Tools, 21*, 1250032-1–1250032-2.

Hermoso, R., Centeno, R., & Fasli, M. (2014). From blurry numbers to clear preferences: A mechanism to extract reputation in social networks. *Expert Systems with Applications, 41*(5), 2269–2285. http://dx.doi.org/10.1016/j.eswa.2013.09.025. <http://www.sciencedirect.com/science/article/pii/S0957417413007598>.

IFTTT put the internet to work for you (2015). <http://ifttt.com/> (Accessed: 2015-03-31).

Jaccard, P. (1901). Etude comparative de la distribution florale dans une portion des Alpes et du Jura. *Impr Corbaz*.

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications, 41*(8), 3809–3824. http://dx.doi.org/10.1016/j.eswa.2013.12.017. <http://www.sciencedirect.com/science/article/pii/S0957417413009925>.

Karger, D. (2014). The semantic web and end users: What's wrong and how to fix it. In *Internet Computing*. IEEE.

Kietzmann, J. H., Hermkens, K., McCarthy, I. P., & Silvestre, B. S. (2011). Social media? Get serious! Understanding the functional building blocks of social media. *Business Horizons, 54*(3), 241–251.

Kifer, M. (2008). Rule interchange format: The framework. In D. Calvanese & G. Lausen (Eds.). *Web reasoning and rule systems* (vol. 5341, pp. 1–11). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-88737-9_1.

Kim, K., Moon, B., & Kim, H. J. (2014). RG-index: An RDF graph index for efficient SPARQL query processing. *Expert Systems with Applications, 41*(10), 4596–4607. http://dx.doi.org/10.1016/j.eswa.2014.01.027. <http://www.sciencedirect.com/science/article/pii/S0957417414000487>.

König-Ries, B., Opasjumruskit, K., Nauerz, A., & Welsch, M. (2012). Mercury: User centric device and service processing. In: *Mms* (pp. 112–116).

Martín-Vicente, M. I., Gil-Solla, A., Ramos-Cabrer, M., Pazos-Arias, J. J., Blanco-Fernández, Y., & López-Nores, M. (2014). A semantic approach to improve neighborhood formation in collaborative recommender systems. *Expert Systems with Applications, 41*(17), 7776–7788. http://dx.doi.org/10.1016/j.eswa.2014.06.038. <http://www.sciencedirect.com/science/article/pii/S0957417414003790>.

O'Connor, M., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W., et al. (2005). Supporting rule system interoperability on the semantic web with SWRL. In Y. Gil, E. Motta, V. Benjamins, & M. Musen (Eds.). *The semantic web – ISWC 2005* (Vol. 3729, pp. 974–986). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/11574620_69.

Onorati, T., Malizia, A., Diaz, P., & Aedo, I. (2014). Modeling an ontology on accessible evacuation routes for emergencies. *Expert Systems with Applications, 41*(16), 7124–7134. http://dx.doi.org/10.1016/j.eswa.2014.05.039. <http://www.sciencedirect.com/science/article/pii/S0957417414003194>.

On{x} automate your life (2015). <https://www.onx.ms/> (Accessed: 2015-03-31).

Opasjumruskit, K., Expósito, J., & König-Ries, B. (2012). MERCURY: User centric device and service processing-demo paper. ABIS 2012 (pp. 2–5).

Ouyang, L., Zou, B., Qu, M., & Zhang, C. (2011). A method of ontology evaluation based on coverage, cohesion and coupling. In *2011 Eighth international conference on fuzzy systems and knowledge discovery* (pp. 2451–2455). IEEE. http://dx.doi.org/10.1109/FSKD.2011.6020046.

Pintus, A., Carboni, D., & Piras, A. (2012). Paraimpu: A platform for a social web of things. In: *Proceedings of the 21st international conference companion on world wide web* (pp. 401–404).

Raimond, Y., & Abdallah, S., (2007). The event ontology. (Tech. Rep.). Technical report, 2007. http://motools. sourceforge. <http://motools.sourceforge.net/event/event.html> (Accessed: 2015-03-31).

Rezaei, R., Chiew, T. K., Lee, S. P., & Aliee, Z. S. (2014). A semantic interoperability framework for software as a service systems in cloud computing environments. *Expert Systems with Applications, 41*(13), 5751–5770. http://dx.doi.org/10.1016/j.eswa.2014.03.020. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0957417414001493>.

RIF Production Rule Dialect (2008). <http://www.w3.org/2005/rules/wiki/PRD> (Accessed: 2015-06-10).

Ruan, J., & Yang, Y. (2010). Assess content comprehensiveness of ontologies. In: *2010 Second international conference on computer modeling and simulation* (pp. 536–539). doi:http://dx.doi.org/10.1109/ICCMS.2010.275.

RuleML Overview and Motivation (2010). <http://wiki.ruleml.org/index.php/RuleML_Home> (Accessed: 2015-06-10).

Serrano, E., Moncada, P., Garijo, M., & Iglesias, C. A. (2014). Evaluating social choice techniques into intelligent environments by agent based social simulation. *Information Sciences, 286*, 102–124. http://dx.doi.org/10.1016/j.ins.2014.07.021.

Serrano, E., Poveda, G., & Garijo, M. (2014). Towards a holistic framework for the evaluation of emergency plans in indoor environments. *Sensors, 14*(3), 4513–4535. Retrieved from <http://www.mdpi.com/1424-8220/14/3/4513>.

Serrano, E., Rovatsos, M., & Botía, J. A. (2012). A qualitative reputation system for multiagent systems with protocol-based communication. In: van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M. (Eds.), *International conference on autonomous agents and multiagent systems, AAMAS 2012, Valencia, Spain, June 4–8, 2012 (Vol. 3)* (pp. 307–314). IFAAMAS. Retrieved from <http://dl.acm.org/citation.cfm?id=2343620>.

Singh, V. K., & Jain, R. (2010). Structural analysis of the emerging event-web. In *Proceedings of the 19th international conference on world wide web* (pp. 11–83). New York: ACM Press.

SmartThings life like never before (2015). <http://www.smartthings.com/> (Accessed: 2015-03-31).

Song, W., Liang, J. Z., Cao, X. L., & Park, S. C. (2014). An effective query recommendation approach using semantic strategies for intelligent information retrieval. *Expert Systems with Applications, 41*(2), 366–372. http://dx.doi.org/10.1016/j.eswa.2013.07.052. <http://www.sciencedirect.com/science/article/pii/S0957417413005393>.

SPARQL Query Language for RDF (2008). <http://www.w3.org/TR/rdf-sparql-query/> (Accessed: 2015-06-10).

SPIN Frequently Asked Questions (2015). <http://spinrdf.org/faq.html> (Accessed: 2015-06-10).

SPIN Overview and Motivation (2011). <http://www.w3.org/Submission/spin-overview/> (Accessed: 2015-06-10).

SPIN, SPARQL Inferencing Notation (2011). <http://spinrdf.org/> (Accessed: 2015-06-10).

SWRL: A Semantic Web Rule Language Combining OWL and RuleML (2005). <http://www.w3.org/Submission/SWRL/> (Accessed: 2015-06-10).

SWRL Frequently Asked Questions (2015). <http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ> (Accessed: 2015-06-10).

Tasker total automation for android (2015). <http://tasker.dinglisch.net/> (Accessed: 2015-03-31).

Thangaraj, M., & Sujatha, G. (2014). An architectural design for effective information retrieval in semantic web. *Expert Systems with Applications, 41*(18), 8225–8233. http://dx.doi.org/10.1016/j.eswa.2014.07.017. <http://www.sciencedirect.com/science/article/pii/S0957417414004151>.

Vambenepe, W. (2009). A new SPIN on enriching a model with domain knowledge (constraints and inferences). <http://stage.vambenepe.com/archives/496> (Accessed: 2015-03-31).

Van Kleek, M., Moore, B., Karger, D. R., André, P., & Schraefel, M. (2010). Atomate it! End-user context-sensitive automation using heterogeneous information sources on the web. In *Proceedings of the 19th international conference on world wide web – WWW'10* (pp. 951). New York, New York, USA: ACM Press. http://dx.doi.org/10.1145/1772690.1772787.

Vladimir, K., Budiselic, I., & Srbljic, S. (2015). Consumerized and peer-tutored service composition. *Expert Systems with Applications, 42*(3), 1028–1038. http://dx.doi.org/10.1016/j.eswa.2014.09.033. <http://www.sciencedirect.com/science/article/pii/S0957417414005788>.

Webee Experience, Connected (2015). <http://www.webeeuniverse.com/> (Accessed: 2015-03-31).

WigWag smart starts with a brain (2015). <http://www.wigwag.com/> (Accessed: 2015-03-31).

Windley, P. J. (2011). *The live web: Building event-based connections in the cloud*. Course Technology.

Ying, J. C., Chen, H. S., Lin, K. W., Lu, E. H. C., Tseng, V. S., Tsai, H. W., et al. (2014). Semantic trajectory-based high utility item recommendation system. *Expert Systems with Applications, 41*(10), 4762–4776. http://dx.doi.org/10.1016/j.eswa.2014.01.042. <http://www.sciencedirect.com/science/article/pii/S0957417414000669>.

Yu, J., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding mashup development. *IEEE Internet Computing, 12*(5), 44–52.

Zapater, J. J. S., Escrivá, D. M. L., García, F. R. S., & Durá, J. J. M. (2015). Semantic web service discovery system for road traffic information services. *Expert Systems with Applications*. http://dx.doi.org/10.1016/j.eswa.2015.01.005. <http://www.sciencedirect.com/science/article/pii/S0957417415000202>.

Zapier the best apps. better together (2015). <http://zapier.com/> (Accessed: 2015-03-31).