A Novel Context Ontology to Facilitate Interoperation of Semantic Services in Environments with Wearable Devices

Gregorio Rubio , Estefanía Serral , Pedro Castillejo , José Fernán Martínez

Abstract. The LifeWear-Mobilized Lifestyle with Wearables (Lifewear) project attempts to create Ambient Intelligence (AmI) ecosystems by composing personalized services based on the user information, environmental conditions and reasoning outputs. Two of the most important benefits over traditional environments are 1) take advantage of wearable devices to get user information in a non-intrusive way and 2) integrate this information with other intelligent services and environmental sensors. This paper proposes a new ontology composed by the integration of users and services information, for semantically representing this information. Using an Enterprise Service Bus, this ontology is integrated in a semantic middleware to provide context-aware personalized and semantically annotated services, with discovery, composition and orchestration tasks. We show how these services support a real scenario proposed in the Lifewear project.

1 Introduction

Wearable devices are becoming more advanced, accurate and capable of sensing. To take full advantage of them is necessary to combine their information with the obtained from the context and the own user. The amount and heterogeneity of the information that comes into play means that architectures and data models must be equipped with the features necessary to develop applications increasingly customized to the user and facilitate interoperation and information sharing.

The main concern has been to developed ontologies, middlewares and architectures to resolve problems related with ubiquitous computing, but, basically thinking in services provided to the web. With the advance of technology, new services based on the user information, environmental conditions and reasoning outputs are emerging in the context of Ambient Intelligence (AmI) ecosystems. Adding to these data a user profile a new ontology is developed including the entire information model. The integration of this ontology is simple: it is integrated as a new service in the Enterprise Service Bus (ESB). In this way, the ontology service can be used by other services or applications executing in the system.

The aim of this paper is to describe an ontology that allows integrating information about user context and semantic services. Using a test scenario, we show how this ontology considerably helps to provide context-aware personalized and semantically annotated services, with discovery, composition and orchestration tasks.

2 Related Work

To understand the scope of this paper in this section briefly present some of the research papers and projects related to sensor and context-aware of user ontologies and service-oriented middleware.

There are ontologies related to sensors used in different environments, like sensorML [1] or O&M Observation and Management [2-3] and even in [4] is presented a semantic sensor network ontology, like an approach to describe sensor assets and other ontologies summarized in [5]. Also, have been described ontology for context-aware environments in pervasive computing [6], modeling the context of a user. In [7] it is provided a means of acquiring, maintaining, and reasoning data about context, and different approach to service-oriented middleware to provide services based in sensor measures to other services and applications or built it [8]. Related to sensors, u-services like proposed in ITEA2 DiYSE project[1] [9], where a new middleware to provide u-services is presented.

This paper presents an integrated ontology, about users, with wearable devices, and services provided by a semantic middleware from the measurements provided by sensors, placed in a wearable device or context-aware, mixed with a dynamic user profile, all integrated in a very flexible architecture, in which the ontology is treated, like a service, being integrated trough an enterprise service bus.

3 Ontology proposal

Different context models have been proposed until now to capture context in Pervasive Computing. Some of the most important examples are: object oriented models [10]; key-value models [11]; graphical models [12-14]; etc. However, several studies [6][15-16] state that the use of ontologies to model context is one of the best choices. They state that this model guarantees a high degree of semantic richness, exhibits prominent advantages for reasoning and reusing context, and facilitates the integration of different systems.

An ontology is a formal and semantic representation of a set of concepts and the relationships between those concepts within a domain. Some relevant examples of ontology-based approaches are SOUPA [17], COMANTO [18], and SOCAM [19]. A

---

[1] http://diyse.org:8080

complete background of most of the ontologies proposed in Pervasive Computing can be found in [16]. None of the studied context ontologies cover adequately all the context information identified in the previous subsection; however, the SOUPA ontology is of special interest for this work. SOUPA is a proposal for a Standard Ontology for Ubiquitous and Pervasive Applications that defines core concepts by adopting the following different consensus ontologies: FOAF, which captures personal information and social connections to other people; DAML-Time the Entry Sub-ontology of Time, which represent Time and facilitate the reasoning about the temporal orders of different events; OpenCyc Spatial Ontologies RCC, which allow space to be specified using geo-spatial coordinates or symbolic representation; Rei Policy Ontology, which specifies high-level rules for granting and revoking the access rights to and from different services.

For facilitating information sharing, we define a context ontology that adequately covers the context information needed for Lifewear systems by adopting as far as possible, suitable concepts of the SOUPA ontology. To build the context ontology of Lifewear, we follow a top-down approach, starting from the most coarse-grained concepts and dividing them up into finer-grained concepts. The coarse-grained concepts that we identify are: Environment, System, Person, Policy, Time, and Event. Dividing them into finer-grained concepts, we obtain the classes of the class diagram shown in Fig. 2.

To describe the environment where the user is, we reuse the OpenCyc Spatial and RCC ontologies that propose classes *SpatialThing* that is related to *LocationCoordinates* class. We have extended these ontologies with the *Location* class to describe the different areas that compose an environment by using a symbolic representation more intuitive for users (i.e., Kitchen, Corridor, etc.). Location has two subclasses; *indoor_location* and *outdor_location*; and can be related by the following relationships:

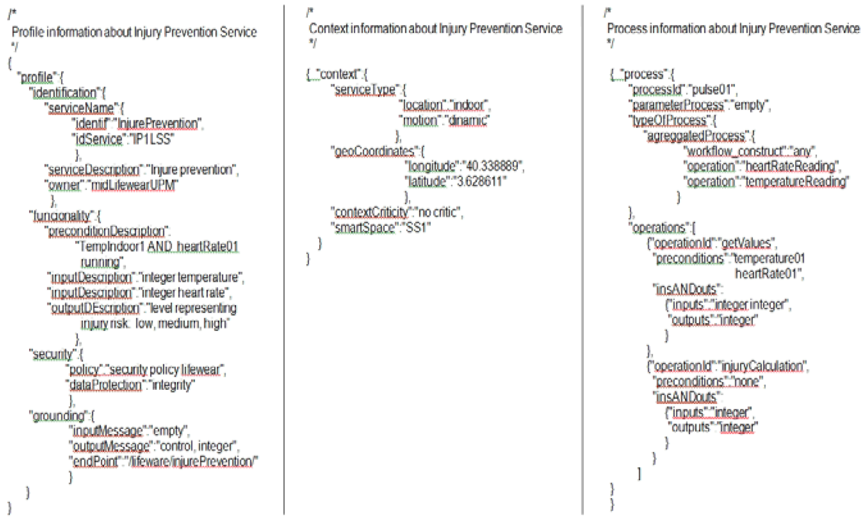
- *Subsumes:* indicates that a location contains other locations (e.g., the location First Floor subsumes the locations Kitchen, Hall and Living Room).
- *Adjacency*: indicates that two locations are physically together (e.g., the Parent Bedroom and the Children Bedroom are adjacent).
- *Mobility*: indicates that two locations are adjacent and there is a way for people to go from one location to the other (e.g., the Hall and the Living Room are adjacent and the Hall has a door to go to the Living Room).

In addition, we propose the term *EnvironmentProperty* to describe the properties (e.g., lighting intensity, presence detection, noise level, etc.) of a certain location.

To describe the system, we propose the terms *Service, ServiceCategory, Operation, Argument, and Process*. The central term is Service, which represents the services (e.g., Lighting, Multimedia Player, Alarm, etc.) that the system provides. Services can be classified into categories and are described by means of the following information:

- *Profile*: the public description of the service. It states the service *identification* (a unique identifier for the service), the service *functionality* (the service operation and their input and output arguments), the *security* profile (description of the security features under which the service will be provided) and *grounding* (protocol used between the service and application that use it).

- *Process:* the logic of the service. The *process* class is refined into *atomic and aggregated/complex processes. An atomic process* takes directly the information generated by sensors and, with the appropriated treatment provides the functionality. In contrary, *aggregated process* providing new functionality that is not directly obtained from sensors by composing several atomic processes. The *aggregated process* can be built using a *sequence*, where the atomic process are executed in a sequential order; or *any_order*, where the order of the atomic process is irrelevant to the aggregated process.
- *Context*: the context conditions in which the service is provided. For instance, if the service is s*tatic,* its functionality is always provided in the same location, or is *mobile*, like in the case of wearable devices where the location of the service can change.



```
/*
 Profile information about Injury Prevention Service
 */
{
  "profile":{
    "identification":{
        "serviceName":{
              "identif":"InjurePrevention",
              "idService":"IP1LSS"
              },
        "serviceDescription":"Injure prevention",
        "owner":"midLifewearUPM"
         },
    "funcionality":{
        "preconditionDescription":
              "TempIndoor1 AND heartRate01
               running",
         "inputDescription":"integer temperature",
         "inputDescription":"integer heart rate",
         "outputDEscription":"level representing
               injury risk: low, medium, high"
              },
    "security":{
            "policy":"security policy lifewear",
            "dataProtection":"integrity"
            },
    "grounding":{
             "inputMessage":"empty",
             "outputMessage":"control, integer",
             "endPoint":"/lifeware/injurePrevention/"
             }
   }
}
```

```
/*
 Context information about Injury Prevention Service
 */

{ "context":{
      "serviceType":{
                  "location":"indoor",
                  "motion":"dinamic"
                 },
      "geoCoordinates":{
                   "longitude":"40.338889",
                   "latitude":"3.628611"
                  },
      "contextCriticity":"no critic",
      "smartSpace":"SS1"
   }
}
```

```
/*
 Process information about Injury Prevention Service
 */

 { "process":{
     "processId":"pulse01",
     "parameterProcess":"empty",
     "typeOfProcess":{
        "agreggatedProcess":{
               "workflow_construct":"any",
               "operation":"heartRateReading",
               "operation":"temperatureReading"
             }
     },
     "operations":[
         {"operationId":"getValues",
          "preconditions":"temperature01
                         heartRate01",
          "insANDouts":
            {"inputs":"integer integer",
             "outputs":"integer"
            }
         },
         {"operationId":"injuryCalculation",
          "preconditions":"none",
          "insANDouts":
            {"inputs":"integer",
             "outputs":"integer"
            }
          }
       ]
 }
 }
```

Fig. 1. Annotated service

To describe the users of the system, we reuse the FOAF SOUPA ontology, which propose the term *Person*. This term is described by a set of properties that include profile information (e.g., *name, gender, birth date*, etc.), contact information (e.g., *email, mailing address, phone numbers*, etc.), and *social* and *professional* relationships (e.g., people that a person knows, relatives, etc.). To properly describe the users, we add the *UserProperty* class, to represent the properties of users, such as user preferences (e.g., preferred music, preferred language, etc.), skills (abilities and disabilities that a person has and may affect to his/her interaction with the system, e.g., computer knowledge, deafness, diseases, etc.) and medical parameters (parameters that determine the user health). With regard to the location in which a person is, we define the *currentLocation* relationship, which relates each person to the location where it is in the current moment. Also, the user is related to the activity that is currently doing and his/her agenda, which describes the user appointments.

A *person* is also associated to *policies*. A policy represents a set of operations and/or services (which group a set of operations) that are permitted for a person. The policy also describes the context information that a person can see and/or modify.

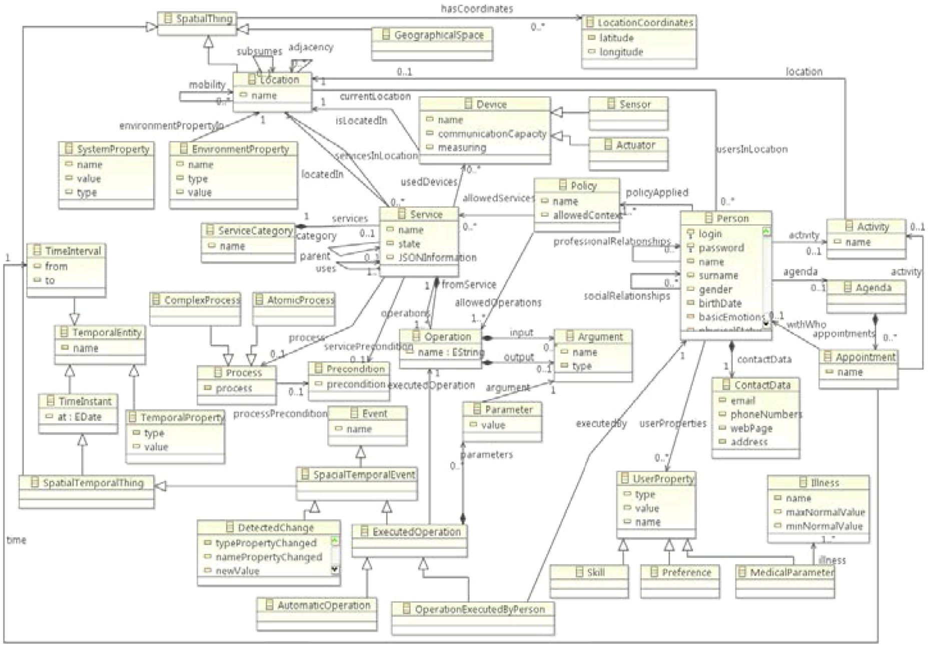


Fig. 2. Class diagram

To describe temporal aspects, we reuse the DAML-Time ontology and the Entry Sub-ontology of Time that SOUPA provides. These ontologies provide us with the term *TemporalEntity*, which is refined into *TimeInstant* and *TimeInterval*. The *TimeInstant* term is defined by using the *at* property that stores the value of time; while the *TimeInterval* term is defined by using the *from* and *to* properties that relate the time interval to the two corresponding time instants. In addition, these SOUPA ontologies provide useful temporal relationships to compare and order to different temporal entities, for instance: *after, before, sameTimeAs, startsLaterThan, startsSoonerThan, startsSameTimeAs, endsLaterThan, endsSoonerThan, endsSameTimeAs*. For avoiding overloading the model, we do not show these relationships in Figure 2. To these classes, we added the *TemporalProperty* class as another refinement of the TemporalEntity class. It represents temporal properties that are not identified as a time instant or a time interval, such as the day of the week, if it is holidays or working days, etc.

To describe the events that happen in the system, we reused the *Event* class proposed by the SOUPA ontology. In SOUPA, an event is a temporal and spacial thing. Thus, SOUPA provides the *SpatialTemporalThing* class, which is the intersection between *TemporalEntity* and *SpatialThing*. In addition, the *SpatialTemporalEvent* class is defined as the intersection of the Event and SpatialTemporalThing classes. The events in our systems can be a change detected by sensors, or can be an operation executed by a person or automated by the system. Thus, in order to better represent the events of our systems, we refine the SpatialTemporalEvent class in the DetectedChange class and the ExecutedOperation class. The DetectedChange represents a change that has been detected by the devices of the system (e.g., the temperature has increased, presence has been detected, the time goes by, etc.). This class is related with the environment or temporal property whose value has changed (e.g., the temperature of the kitchen). The ExecutedOperation class represents an event produced by the execution of an operation (e.g., switching on the light or playing a song). This class is related with the executed operation (e.g., the switch on operation of the lighting service or the play operation of the multimedia player service) and the arguments used for executing the operation. This class is refined in the OperationExecutedByPerson and AutomaticOperation classes. The OperationExecutedByPerson represents the execution of an operation by a person. This class is related with the person that has executed it by using the executedBy relationship. The AutomaticOperation represents the execution of an operation by the system.

To implement the ontology, we use the Web Ontology Language (OWL). OWL is an ontology markup language that greatly facilitates knowledge automated reasoning and is a W3C standard. Using OWL, the classes of the ontology are defined by OWL classes, and the context specific of the system is defined by OWL individuals, which are instances of these classes. In OWL, the properties of each class are represented by attributes whose data type is simple. These properties are *DatatypeProperties*. The relationships with other classes are represented by attributes whose data type is a class. These properties are *ObjectProperties*. For instance, a user named Bob is specified as an individual of the Person class whose *ID* datatypeProperty is *Bob*. Its preferred temperature is specified as an individual of the *Preference* class and added to the *userPreferences* objectProperty (which contains the list of user preferences) of the Bob individual.

4 Test Scenario

To demonstrate the use of the ontology, we designed a test scenario (Fig. 3). In this scenario, a user has a mobile phone with an android application that recommends him/her a sport routine and a diet according to his/her current condition. Regarding the user preferences, the app will prepare a specific daily training plan for losing weight and improving physical condition. The user can see these recommendations in an android device. The connectivity is by means of Bluetooth. Daily, the application should check the historical data of the user profile and adjust the weight loss plan, so the application could advice the client about its achievements and failures and encourage him to follow the plan of action.

At the end of each practice session, the user could view a list of recommendations that will be of interest to recover the biological deficiencies, mineral salts, liquid vitamins and diet-specific menu or could view a calorie intake recommendation. Moreover, the application can analyze the experience of the user and decide whether the practice has been beneficial or detrimental to him/her. In this scenario, the ontology is the key to providing the integration of all needed information and make the services aware of user context.

An OWL context model based on the ontology presented in the previous section is created to manage and integrate all the needed information. In this model, the user information (such as age, tall, diet preferences, favorite sports, diseases, sex, food and environmental allergies) as well as the semantic information about the needed services (profile, context and process) is manually introduced by using Protègè [19].

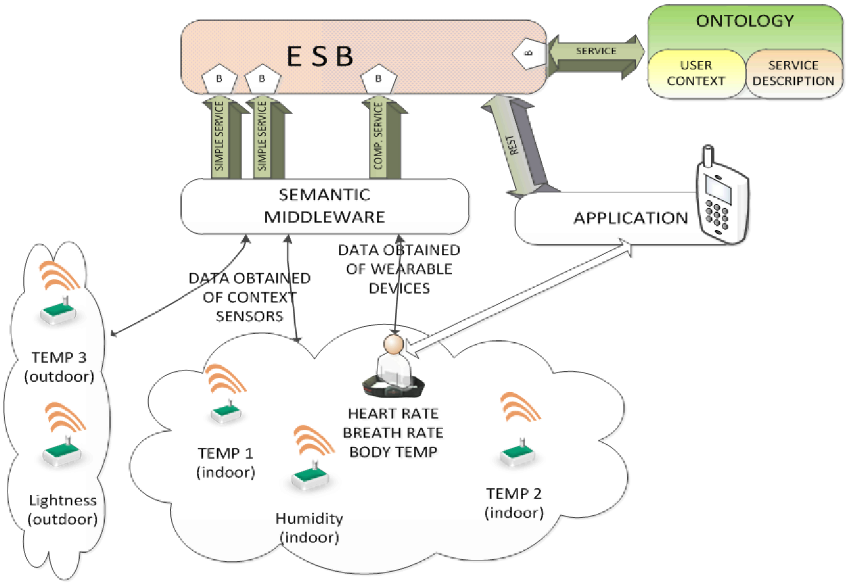


Fig. 3. Test scenario

When the user executes the application in the smartphone android, it uses the services semantically annotated provided by the semantic middleware to recollect data from all sensors and update the context model according to the sensed context.

The simple services provided by sensors are related with the dynamic measures such as heart rate, breathing rate, body temperature, location, indoor temperature in several places, indoor humidity, outdoor lightness or outdoor temperature.

In general, the service profile is static, while its context and process are dynamic. So, the service features can be known to create new composed services by using an orchestrator (dedicated mote). The composed services use and process the context information of the context model to provide the required application functionality.

Thus, the simple services semantically annotated in the context model, are used for the application for updating the sensed information in the context model. Processing this information, the following composed service is provided:

- Muscular Injury Prevention Service:
  a. Context information (provided by simple services) used: indoor temperature, body temperature, heart rate.
  b. Response: Injury levels, including low, medium and high, depending on the value of the context information parameters.

Using this composed service a new composed service can be also created:

- Alarm Muscular Injury Prevention Service: will be activated when the Injury level in the Response of Muscular injury prevention service is high (the gap between indoor temperature and body temperature is excessive, or indoor temperature, or skin temperature or heart rate has exceeded a threshold).

All the composed services are, also, continually updating the corresponding context information in the context model.

So, the ontology-based context model becomes the central point of the system integrating all the knowledge of the application (user context and required services) and allowing services to improve their functionality and provide composed services by taking into account this knowledge.

4.1 Test Scenario Deployment

The following steps have been performed in strict order to integrate the ontology in the ESB:

- Create the corresponding individuals using Protégé 4.1[20].
- Validate the ontology in standalone and using Protégé 4.1 and the Pellet reasoner [21].
- Build an Application programming interface API for managing the ontology at runtime: we used Jena [22] to open the OWL model and manage its individuals, TDB [23] for making the context model persistent and Pellet for providing reasoner. For developing this API we have applied the recommended best practices proposed in [24-25].
- Create a bundle to run the API in the ESB; in this way, services can easily access to the ontology and all the components are integrated.

5 Conclusions and Future Work

In this paper we have proposed an ontology that integrates all the information about user context and services running in a system and becomes a key point to improve the application features. The easiness of the ontology integration is also an important benefit of our proposal. As can be seen in the previous section, the ontology is integrated as a new service in the ESB. In this way, the ontology service can be used by other services that are running in the system.

Furthermore, we have developed a test scenario to demonstrate that the user, from him/her point of view, using only wearable devices, improve the experience of use of the application. What the user doesn't know is that this fact is possible because in the system exist the ontology showed in this paper.

Both the capabilities of the sensors as wearable devices constantly evolving, so, alike, ontology needs too constantly evolving to maintain the capability to represent the abstract content related whit sensors and users profile.

Keeping in mind that it will appear new manufacturers and new users with new profiles, the key point of future work will be to provide to the ontology capabilities of evolution and versioning [26]. To achieve this target it could be used methodologies like DOGMA-MESS [27-28], in a native way or adapting to the global system, like in DiYSE project, to gradually enrich the ontology.

It is also necessary to develop a powerful engine inference, providing the ontology to dynamically create new services based in the orchestration of the existing services.

Another future task is to test the global application in new scenarios, basically of ubiquitous computing, such us e-health, surveillance of young or elderly people, in a non-intrusive way.

In summary, the future work will be directed to adequate the ontology to let the creation of services each time closer to the user.

Acknowledgments.
The work presented in this paper has been partially funded by the Spanish Ministry of Industry, Tourism and Trade in the framework of the European Research Project "LifeWear-Mobilized Lifestyle with Wearables" (TSI-020400-2010-100), which belongs to the ITEA 2 (Information Technology for European Advancement 2) program, and by the Spanish Ministry of Economy and Competitiveness in the AWARE Project (Ref. TEC2011-28397).

References

1. Botts M., Robin A. OpenGis Sensor Model Lenguaje (SensorML) implementation Specification. OpenGIS Implementation Specification OGC 07-000. The Open Geospatial Consortium (2007).
2. Cox S.: Observations and Measurements – Part 1– Observation Schema. OpenGIS Implementation Standard OGC 07-022r1 The Open Geospatial Consortium (2007).
3. Cox S.: Observations and Measurements – Part 2– Sampling Features. OpenGIS Implementation Standard OGC 07-022r3 The Open Geospatial Consortium (2007).
4. The semantic sensor network ontology: a generic language to describe sensor assets. In AGILE Workshop: Challenges in Geospatial Data Harmonization, (2009).
5. Compton M., Henson C., Lefort L., Neuhaus H., Sheth A.: A Survey of Semantic Specification of Sensors. Proc. Semantic Sensor Networks. pp. 17-32 Springer (2009)
6. Chen, H., Finin, T., and Joshi, A.: An ontology for context-aware pervasive computing environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, vol. 18(3), pp197–207 (2004).

7. Chen, H., Finin, T. and Joshi, A. Ontologies for Agents: Theory and Experiences, chapter of The SOUPA Ontology for Pervasive Computing. Whitestein Series in Software Agent Technologies. pp 233-258 Springer Velar, Berlin. (2005)
8. Gu T., Pung H. K. Zhang. D.Q.: A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, vol.: 28(1), pp.1–18, (2005).
9. Tang, Y, Meerman, R., DIY-CDR: An Ontology-based, Do-it-Yourself Components Discoverer and Recommender Theme Issue on Adaptation and Personalization for Ubiquitous Computing, journal of Personal and Ubiquitous Computing, Springer, Z. Yu, D. Cheng, I. Khalil, J. Kay, D. Heckmann (eds.), ISSN 1617-4909, June 21, 2011
10. B. Biegel, Cahill V.: A framework for developing mobile, context-aware applications. 2nd IEEE Conference on Pervasive Computing and Communication. (2004).
11. Dey, A. K. Understanding and Using Context. *Personal Ubiquitous Computing*. (2001)
12. Sheng, Q. Z. and Benatallah B. ContextUML: a UML-based modelling language for model-driven development of context-aware web services. ICMB'05, 11-13 July, pp 206 - 212 IEEE Computer Society, Washington, DC, USA. (2005).
13. Henricksen, K. and Indulska J. Developing context-aware pervasive computing applications: Models and approach. Pervasive and Mobile Computing, 2, pp 37-64 (2006).
14. Ayed, D., Delanote D., Berbers Y., and Berbers Y. MDD Approach for the Development of Context-Aware Applications. CONTEXT 2007, pp. 15-28.(2007).
15. Baldauf, M., Dustdar S., and Rosenberg F. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing,* 2, pp 263-277. (2007)
16. Ye, J., Coyle L., Dobson S., and Nixon P. A Unified Semantics Space Model. LoCA 2007, pp 103-120, Springer-Verlag Berlin Heidelberg. (2007).
17. Chen, H., Perich F., Finin T., and Joshi A. (SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. The First Annual International Conference on Mobile and Ubiquitous Systems 22-26 August, pp 258 – 267. (2005).
18. Roussaki, I., Strimpakou M., Pils C., Kalatzis N., and Anagnostou M. Hybrid context modeling: A location-based scheme using ontologies. Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06), IEEE Computer Society. (2006)
19. Gu, T., Pung H. K., and Zhang D. Q. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, 28, pp 1-18. (2005)
20. The Protégé Ontology Editor and Knowledge Adcquisition, http://protege.stanford.edu
21. Pellet, OWL 2 Reasoner for Java, http://clarkparsia.com/pellet
22. Apache Jena , http://jena.apache.org
23. Apache Jena TDB, http://jena.apache.org/documentation/tdb/index.html
24. How To Design A Good API and Why it Matters. Google. (2007)
25. Guy, M. Report 2: API Good Practice Good practice for provision of and consuming APIs. Tech. rept. UKOLN (2009).
26. Khattak, A.M., Latif, K., Lee, S. and Lee, Y.K.: Ontology Evolution: A Survey and Future Challenges, Communications in Computer and Information Science, vol. 62, pp. 68-75. (2009).
27. de Moor, A., De Leenheer, P., Meersman, R.: *DOGMA-MESS: A Meaning Evolution Support System for Interorganizational Ontology Engineering*, Proc. of 14th ICCS conference, Springer-Verlag, V. 4068, p.189-203, Aalborg, Denmark. (2006)
28. De Leenheer, P. and Debruyne, C.: *DOGMA-MESS: a tool for fact-oriented collaborative ontology evolution*, in proc. Of OTM 2008 workshops, Meersman, Tari and Herrero (Eds.), LNCS 5333, pp. 797-806, Srpinger-Verlag Berlin Heidelberg, (2008).