Context Awareness in the Networked Enterprise: Methodological and Technological Issues

Fabio A. Schreiber, Letizia Tanca
PEDiGREE (*PE*rvasive *D*atabas*e* *GR*oup of *E*ngin*E*ers)

Dipartimento di Elettronica e Informazione
Politecnico di Milano
{schreibe,tanca}@elet.polimi.it

Abstract. The formidable amount of heterogeneous information, accessed by the networked enterprise through all the available channels, makes it difficult for users to find the right information at the right time and at the right level of detail. Contextual meta-data about the system and the users can be used to reduce this plethora of information, providing high-quality, focussed knowledge to users and applications at all decision-making points. We propose context-aware system design methodologies and techniques exemplified within the wine production scenario, where several classes of users access the networked-enterprise data sources, the sensors used for monitoring the productive cycle, and external sources of different nature.

Keywords: context-awareness, data integration, data management, design methodology, sensor networks language

1. Introduction

Data and information constitute the main assets of an information system, and must be conveniently exploited to provide the enterprise with the appropriate knowledge, at each decision level and for each business need. This fact is particularly true when the enterprise itself co-operates with fellow-enterprises to amplify its prospects and opportunities, constituting a Networked Enterprise. In such an enterprise, the available data sources may have different natures, varying from simple relational data to semantically annotated knowledge, to data coming from sensors. The knowledge about the current user's context enables the system to interpret, integrate and filter (tailor) the available information in order to: 1) provide the user or the application with the appropriately tailored set of data or services, thus eliminating information noise, 2) match devices' physical constraints, in particular in mobile applications, 3) appropriately tailor sensor queries.

The ArtDeco project [1] focuses on the problems of finding, extracting, representing and formalising knowledge, in all various forms in which it may be embedded, in order to build a context-aware, semantic model of the business domains of networked enterprises. After a quick introduction to the architecture of the ArtDeco Web Portal, we concentrate on context-aware knowledge elicitation and querying.

The applicability of the proposed context-aware design model and techniques is referred to a scenario where several classes of users access the networked-enterprise internal data sources, external sources of different nature and the peripheral pervasive devices (e.g., RFID tags, sensors, WSNs) used for monitoring the production cycle. These physical data, besides being used for monitoring the production process, also concur in determining the state of the context.

2. System Architecture

Fig. 1 depicts the general architecture of the ArtDeco Web Portal, which provides five main functionalities: taxonomy-driven word tagging and tag-based querying; knowledge extraction from natural language sources and concept-based natural language querying; knowledge extraction from applications and sensor networks; ontology extraction from structured data sources; and collection of data from enterprise processes. Such modules rely on information stored in the Domain Model, describing the domain by means of a Taxonomy, a Domain Ontology, a Semantic Network and a Mapping Model.

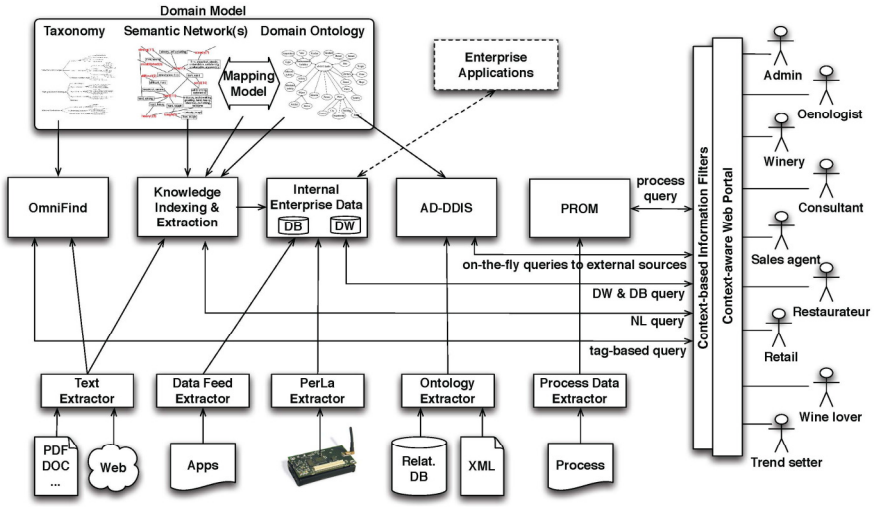

Fig. 1 The ArtDeco Context-Aware Web Portal

The *Taxonomy*, compiled by a domain expert, provides a hierarchy of tags that associate labels to the words that lexicalise them. This simple model is then used by the OmniFind module to provide taxonomy-driven word tagging and support user tag-based queries. The *Domain Ontology*, a knowledge base defined by means of Description Logics, contains the concepts relevant for the application domain and is defined by a domain expert, starting from, and extending, the *Taxonomy, Domain Ontology, Semantic Network and Mapping Model* are used by the *Knowledge*

Indexing & Extraction module to provide knowledge extraction from natural language sources and support concept-based natural language queries.

The *Extractors* gather data from heterogeneous sources, such as textual documents, applications, sensor networks, database, XML files, and processes. While the OmniFind and the Knowledge Indexing & Extraction modules analyse and index the content of textual documents, data coming from application and sensor networks are given to the *Internal Enterprise Data* module, while the ontologies extracted from structured repositories, such as relational databases and XML files, are used by the AD-DDIS module to allow on-the-fly access from external applications.

Users interact with the system by means of the Context-aware Web Portal. The portal provides different users with different context-aware views on the database or the data warehouse. Each view corresponds to a different working context of the ArtDeco Web Portal and it is determined on the basis of a context-model and a methodology for Context-Aware View Design [2], and queries coming from the Web Portal will be answered by means of the views associated to the current context instead of resorting to the whole database or data warehouse that may contain unnecessary information. Enterprise Applications can exchange information with the Internal Enterprise Data database, in order to take advantage of the data collected by the system, or interact with the portal to gain information from the other data sources, possibly filtered on the basis of the context.

The information management sub-system of the ArtDeco project is devoted to providing a uniform, ontology-based semantic access to information coming from heterogeneous data sources. In ArtDeco, we chose to keep a centralised repository, in the form of a Relational Database with an associated Data Warehouse used to provide also for context-aware analytical queries, temporal trend analyses and similar mining tasks to the aim of supporting the product design and innovation processes.

The internal relational database is constituted by two sets of tables: the enterprise database (the so-called Enterprise Tables), and a set of domain-independent tables used to structure the information coming from the extractors (the Web-Search Tables). These tables contain the findings in web documents that are considered "relevant" by the extraction engine and are linked to the semantically relevant enterprise tables. As an example, the following sentence appeared in a blog entry of a wine-expert titled "Barbera 2010: Pride in Simplicity?": *"[…] so she poured us the unoaked wines, a fantastic Langhe Nebbiolo and a Barbaresco.[…]"*. From this sentence we can derive the interesting fact that "Langhe Nebbiolo" and "Barbaresco" are "unoaked wines" and the fact that the author of the entry considers them "fantastic". Each of these findings corresponds to tuples in Web-Search Tables which are linked to the entries of the enterprise tables that store internal information about Nebbiolo and Barbaresco.

While the centralised repository is useful for analytical query processing, mediated on-line access to the original data sources (e.g. XML documents and legacy and current databases from the networked business partners) is supported by AD-DDIS, the integration component of the ArtDeco framework [1]. Sensors data are handled as relational databases since we rely on the PerLa Language (see below). The Domain Ontology is used by AD-DDIS as Global Schema and later mapped to a set of ontologies, each describing the semantics description of one data source and extracted by means of domain-aware wrappers. Data source ontologies are (semi-) automatically mapped to the domain ontology and used to enable query distribution.

3. The Context Model

In this section we informally introduce the *context model* [2] adopted to represent the various contexts the enterprise members are working into. A set of *context dimensions* is used to capture different characteristics of the environment; each dimension can assume different *values* (a.k.a. *concepts*): we use black nodes for the dimensions and white nodes for the concepts. In Figure 2 we present the graphical representation of a simplified *context schema* (the *Context Dimension Tree (CDT)*) for a wine production monitoring application. In the wine production process CDT of Fig. 2, the **`Role`** dimension describes the "actors" involved: **`Farmer`**, **`Oenologist`** and **`Driver`**; the **`Phase`** dimension describes the phases of the wine production process and can assume the concepts **`Growth`**, **`Ageing`** and **`Transport`**; the third dimension is related to the risks to be kept under control, with the two concepts of **`Overheating`**, owing to exposure of wine bottles to sun- light, and **`Disease`** which can affect the grapes. A *context instance* is then a conjunction of propositions:

$$Context \equiv \wedge_{i,j}(Dimension_i = Concept_{i,j}).$$

We can now define the *Transport_ Monitoring_context*; the bottled wine, in fact, must not be kept under direct sunlight for more than a certain amount of time to avoid overheat and a consequent alteration of the wine flavour: *Transport_Monitoring* ≡ *(Role = Driver)* ^ *(Phase = Transport)* ^ *(Risk = Overheat)*. It is worth noticing that not all possible sets of concepts are valid contexts: for instance the dimension **`Role`** cannot assume simultaneously the **`Driver`** and **`Farmer`** concepts (the children concepts of a dimension are always to be instantiated in mutual exclusion). Invalid contexts are thus ruled out by appropriate constraints [2].

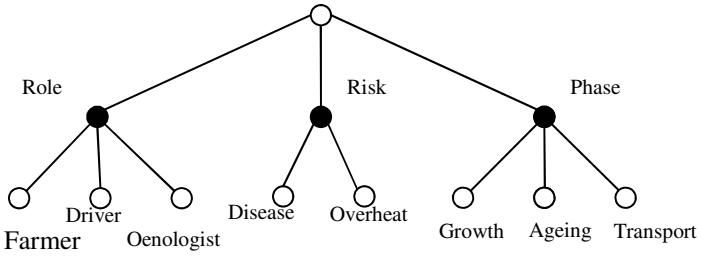

Fig. 2 The CDT schema (for the wine production process)

The context model [2] allows the representation of context in terms of *observable entities*, which have a *symbolic* representation within the system (e.g. the **`Overheat`** risk concept) and possibly a *numerical* value gathered from the environment sensors. Gathering context data from the environment requires a simple interface, possibly based on a declarative approach, which, on the one side, interacts with the network of highly heterogeneous physical devices and, on the other, is correctly interfaced with the internal, symbolic representation of context. Then, it is possible to analyse how symbolic observables can be mapped to *numeric* observables (e.g.: temperature ranges), which are instantiated by retrieving them from the pervasive system. The

PerLa system [4,5], presented in the next section, allows to perform this important task effectively and efficiently.

4. Managing Context Through PerLa

We illustrate now the middleware layer and a specific language permitting to install context-aware queries on, and extract data from, peripheral pervasive devices (e.g., RFID tags, sensors, WSNs) allowing seamless integration of such data within the rest of the information at the enterprise's disposal. As extensively presented in [4], PerLa is a framework to configure and manage modern pervasive systems. Adopting a data-centric approach, it relies on a query language using an SQL-like metaphor. PerLa queries allow to retrieve data from the pervasive system, to prescribe how the gathered data have to be processed and stored and to specify the behaviours of the devices. A typical PerLa query, deployed on a group of sensors, and used to gather data from the field is shown below:

```
…….
SELECT ID , temperature , humidity , location_x , location_y
SAMPLING EVERY 1 m
EXECUTE IF EXISTS ( temperature ) AND is_in_vineyard (location_x , location_y)
REFRESH EVERY 10m
```

PerLa is based on a middleware whose architecture exposes two main interfaces: a high-level interface which allows query injection, and a low-level interface that provides plug&play mechanisms to handle devices. Moreover, the PerLa language supports the definition and the management of context [6] through *CDT Declaration* and *Context creation*:

CDT Declaration

```
CREATE DIMENSION <Dimension Name>
    [CHILD OF <Parent Node >]
    { CREATE CONCEPT <Concept Name>WHEN <Condition>
    [EVALUATED ON <Low Level Query >]}*
```

The *CREATE DIMENSION* clause is used to declare that a new dimension must be added to a CDT, possibly as a child of a concept node (*CHILD OF* clause). Once a dimension has been declared, it is possible to specify the values it can assume, using the *CREATE CONCEPT/WHEN* pair. For each pair the designer must specify the name and the condition for assuming the specified values by means of *numeric* observables that can be measured from the environment. We postpone the explanation of the *EVALUATED ON* clause to the next Subsection, where it plays a fundamental role. The CDT of Figure 5 is specified by the following set of statements:

```
CREATE DIMENSION Role
    CREATE CONCEPT Farmer WHEN get_user_role ( )=' farmer '
    CREATE CONCEPT Oenologist WHEN get_user_role ( )=' Oenologist '
    CREATE CONCEPT Driver WHEN get_user_role ( )=' driver '
CREATE DIMENSION Risk
    CREATE CONCEPT Disease WHEN get_interest_topic ( )=' disease '
    CREATE CONCEPT Overheat WHEN temperature > 30 AND brightness > 0.75;
CREATE DIMENSION Phase
    CREATE CONCEPT Growth WHEN get_phase ( )='growth '
    CREATE CONCEPT Ageing WHEN get_phase ( )=' ageing '
    CREATE CONCEPT Transport WHEN get_phase ( )=' transport '
```

In this CDT, the **`get_user_role()`**, **`get_phase()`** and **`get_interest_topic()`** functions are employed to retrieve context information that cannot be deduced from sensors readings, but have to do with other aspects of the application. This information is typically gathered from some external XML source or database. This clearly highlights how PerLa supports the passage from *symbolic* to *numeric* observable: the **`Overheat`** *symbolic* value is in fact defined in terms of the **`Temperature`** and **`Brightness`** physical quantities (*numeric* observables) that can be sampled from the environment using very simple queries.

Context creation

```
CREATE CONTEXT <Context Name>
ACTIVE IF <Dimension>=<Value> [AND <Dimension>=<Value>]
ON ENABLE <PerLa Query>
ON DISABLE <PerLa Query>        /*one-shot only */
REFRESH EVERY <Period>
```

The *CREATE CONTEXT* statement is used to create a context instance in PerLa and allows to associate a unique name to it. The *ACTIVE IF* statement translates the $Context \equiv \curlywedge_{i,j}(Dimension_i = Concept_{i,j})$ statement of Section 3 into PerLa. This statement is fundamental for the middleware in order to decide if a context is active or not. The actions that must be performed in both these situations must be specified using the *ON ENABLE* clause and are expressed using any type of PerLa query . The *ON DISABLE* clause can be coupled only with “one-shot" queries, that is, queries that are executed only once upon deactivation of a context, and thus do not create conflicts with the queries enabled by the next active contexts. The middleware will also perform the necessary controls according to the condition specified in the *REFRESH* clause that completes the syntax. In the next example we show how context management statements and queries/actuation commands on the target system are uniformly mixed in order to achieve a context-aware behaviour. For the *Transport_Monitoring* context we can use the following statements:

```
CREATE CONTEXT Transport_Monitoring
ACTIVE IF Phase = ' transport ' AND Role=' driver ' AND Risk=' overheat '
ON ENABLE:
    SELECT temperature , gps_latitude , gps_longitude
    WHERE temperature > 30
    SAMPLING EVERY 120 s
    EXECUTE IF  location = ' truck_departing_zone '
SET PARAMETER ' alarm ' = TRUE;
ON DISABLE:
    DROP Transport_Monitoring ;
SET PARAMETER ' alarm ' = FALSE;
REFRESH EVERY 24 h ;
```

In this example, after creating the context, a very short query is issued: the *SELECT* clause specifies that both temperature and GPS location must be retrieved every two minutes (*SAMPLING EVERY* clause), while the *WHERE* clause allows to filter the sampled values. The *EXECUTE IF* finally deploys the query only on those devices located into the vineyard truck departing zone. This query features also an *actuation query* introduced by the *SET PARAMETER* clause and is used to activate an alarm if the risk of overheat becomes real.

The internal structure of the PerLa middleware also supports the *Context Language* (CL). A *Context Manager (CM)* is in charge of: 1) creating and maintaining the CDT;

2) detecting which contexts are active or not in a precise moment; 3) performing the correct actions expressed by the user according to context statuses. In the following we analyse these steps.

Creation of the CDT During this phase all the necessary numeric observables (declared using the CREATE CONCEPT/WHEN clauses) are retrieved, and the EVALUATED ON clause becomes important. In fact, as long as this clause is unemployed, the CM executes a series of independent queries in order to retrieve the necessary information from the pervasive system. The designer could be interested in modifying this default behaviour, especially when the environment changes rapidly and the same observable is employed in different concepts (leading thus to some inconsistencies using different queries). This clause is useful also to introduce some optimisations (e.g.: discarding some unwanted devices). For example, on the Overheat dimension:

```
CREATE CONCEPT Overheat WHEN temperature > 30 AND brightness > 0.75 ;
    EVALUATED ON:
    SELECT temperature , brightness
    EXECUTE IF  location=' truck_departing_zone ' AND battery > 0.7
```

In this example the observables **`temperature`** and **`brightness`** are sampled simultaneously using one single query (instead of two independent queries). Moreover the query is executed only on those devices that are located in the truck departing zone and whose battery power is enough to operate efficiently (*EXECUTE IF* clause); notice that functional and non-functional data are collected in the same way. Once all the results are available (independently of the presence of the *EVALUATED ON* clause) the system can create a series of tables (one for each dimension with concepts nodes) that contain a column for every attribute expressed in the *CREATE CONCEPT/WHEN* clauses. The table reports also the IDs of the devices that were taken into account during the retrieval phase. Every table entry then represents the actual value (sampled from the environment) and the device that physically produced it. If we consider again the **`Overheat`** dimension and supposing that the computation of the relative *EVALUATED ON* returned only the 1,3,4 IDs, a table for this dimension could be the following:

ID	temperature	brightness
1	28	0.60
3	31	0.71
4	33	0.80

Fig. 3 Table for the Overheat dimension

Once all the necessary information has been gathered it is possible to evaluate every condition expressed by the *WHEN* clauses used during the CDT declaration. In particular, simply looking up every table, the CM assigns to a CDT concept node the ID(s) of those devices whose sampled values satisfy the condition expressed by the *WHEN* clause of the concept definition. When this phase is concluded the system knows which devices are in the situation described by the concepts of every dimension of the CDT. For example, referring to the **`Overheat`** in Figure 3, the CM can deduce that only sensor number 4 is detecting the risk of overheat since both temperature >30 and brightness >0.75 conditions are true simultaneously, while this is

not the case of sensors number 3 and 1. With similar computations the CM also selects the concepts that correspond to the results calculated by the static functions, such as **`get_user_role().`**

Context detection A context is active if the dimensions that define it assume the values specified by the $Context \equiv \wedge_{i,j}(Dimension_i = Concept_{i,j})$ statement. Considering also the results of the static functions, the system recognises as active all the contexts whose CDT concepts contain a not-empty device list. In fact, from the CM point of view, if one ID has been associated with a concept it means that, for at least one device, a CDT dimension is currently assuming that value. If this situation is true for every < *Dimension* > = < *Concept* > used to define a context C then the environment is exactly in the situation expressed by C, and C is considered as active.

Performing context actions Once a context has been recognised as active, the CM simply injects the query specified by the *ON ENABLE* clause into the middleware dedicated components. At this point the execution flow equals the one of any other query that is manually injected into the system, and is thus completely controlled and managed by the middleware dedicated components.

5. Conclusions

In this paper we have proposed a model for the organization and exploitation of context-awareness in a networked Enterprise. Context-aware thinking is useful for structuring systems in a way that clearly separates concerns into what is strictly functional to the applications and what concerns the environment in which they act. More advantages of this approach are described in [3]

References

1. Anastasi G., Bellini E., Di Nitto E., Ghezzi C., Tanca L., Zimeo E. (Eds.): Methodologies and Technologies for Networked Enterprises. Springer Verlag (to appear 2012)
2. Bolchini C., Curino C.A., Quintarelli E., Schreiber F.A., and Tanca L.: Context information for knowledge reshaping. Intl. Journal of Web Engineering and Technology, 5(1):88–103, (2009).
3. Bolchini C., Orsi G., Quintarelli E., Schreiber F. A., Tanca L.: Context Modeling and Context Awareness: steps forward in the Context-ADDICT project. Bulletin of the IEEE Technical Committee on Data Engineering, 34(2), pp. 47-54, (2011)
4. Schreiber F.A., Camplani R., Fortunato M., Marelli M., Rota G.: PerLa: A Language and Middleware Architecture for Data Management and Integration in Pervasive Information Systems. IEEE Transactions on Software Engineering, (2011). http://doi.ieeecomputersociety.org/10.1109/TSE.2011.25
5. PerLa website: http://perlawsn.sourceforge.net/index.php
6. Schreiber F.A., Tanca L., Camplani R., Vigano' D.: Towards autonomic pervasive systems: the PerLa context language. Proceedings of the 6th International Workshop on Networking Meets Databases, Co-located with SIGMOD 2011, Athens, (2011)