Ontology-based information tailoring

Carlo Curino, Elisa Quintarelli, and Letizia Tanca
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32—20133 Milano (Italy)
curino,quintare,tanca@elet.polimi.it

Abstract—Current applications are often forced to filter the richness of datasources in order to reduce the information noise the user is subject to. We consider this aspect as a critical issue of applications, to be factorized at the data management level. The Context-ADDICT system, leveraging on ontology-based context and domain models, is able to personalize the data to be made available to the user by “context-aware tailoring”. In this paper we present a formal approach to the definition of the relationship between context (represented by an appropriate context model) and application domain (modeled by a domain ontology). Once such relationship has been defined, we are able to work out the boundary of the portion of the domain relevant to a user in a certain context. We also sketch the implementation of a visual tool supporting the application designer in this modeling task.

I. INTRODUCTION

Nowadays content and services are available at different sources and places, thus a user can be seen as an integral part of numerous applications in which he/she interacts with service providers, product sellers, governmental organizations, friends and colleagues. Information access needs thus to be appropriately personalized, in order for the user not to be taken aback by the huge amount of available data. Our approach addresses these issues with particular attention to the problem of mobile data tailoring, which consists of the exploitation of knowledge about the user, the adopted channel and the environment, altogether called context, to the end of reducing the amount of information imported on mobile devices.

Tailoring is needed because of two main reasons: one is the need to keep information manageable, in order for the user not to be confused by too much noise; the second reason is the frequent case that the mobile device be a small one, like a palm computer or a cellular phone, in which condition only the most relevant information must be kept on board.

The Context-ADDICT (Context-Aware Data Design, Integration, Customization and Tailoring) project [1] aims at the definition of a complete framework which, starting from a proposal of a methodology for the early design phases, supports mobile users through the dynamic hooking and integration of new, available information sources, until an appropriate, context-based portion of data is delivered on their mobile device.

The whole process is widely based on an ontological representation of the application domain and of datasource contents. An ontology is used to represent the context model as well, the so called Context Dimension Tree, which plays a fundamental role in tailoring the ontology of the target application, according to the user information needs. We have chosen the OWL language [2] for the representation of the ontologies since it has become a well formalized, well supported and widely-accepted standard.

In this paper we discuss how the concepts contained in the context model, describing the possible contexts, may be connected to the domain ontology, in order for the tailoring process to be performed dynamically, whenever the user’s context changes.

In particular, this activity is part of the design phase, when, after the definition of the context model, the next task for the designer is to specify such connection.

Target of this task is the definition of the chunks, which basically define the portions of source data to be materialized on each portable device [3], [4]. The chunks, in turn, are defined based on the analysis dimensions, which are the key concepts of our context model, and introduce the different perspectives the mobile device is viewed from.

A dimension captures an aspect of a context or of a user’s current context. We propose some intuitive dimensions, which can be integrated with additional ones if needed, or dismissed when not appropriate:

- **Holder**: the various categories of device users involved; for example “students”, “families”, or “elders”, in a tourist application.
- **Interest Topic**: the various topics of interest for the possible users of an application. For example, in a mobile application for tourists, interest topics could be “restaurants”, “entertainments”, “museums”.
- **Situation**: different points of view for a same interest topic, for example “hospitalized” and “at-home” for a personal medical care application.
- **Space**: based on the place where the user is currently located (it might be GPS coordinates or any other location information), its granularity may vary depending on the application. For example a museum’s “room” or a whole “city” or a “region”.
- **Time**: a temporal indication based on the current time; its granularity may vary. For example, one could choose the “current month” or “last year”, but the reference point is always now.

The chosen dimensions are used at different steps in the methodology to tailor the data to be stored on the devices. They are represented at design time in a hierarchical structure,
actually a tree, which shows also further decompositions that can be used to define the user context.

Figure 1 shows the Context Dimension Tree of an application supporting campus life in a university. For each dimension, the designer defines a set of admissible values called Dimension values, defined as follows: a dimension value is a concretization of a certain dimension concept, an indication capturing a precise value of an aspect of the context, e.g., in the university context, values for the dimension Holder may be “student” or “professor”, while the dimension Space may have as values the different space granularities, such as “city” or “university”.

The Context Dimension Tree is the context model of the Context-ADDICT project. An array of dimension values, one for each dimension, called chunk configuration, fully defines a precise user profile and context; thus, by means of dimensions and dimension values the Context Dimension Tree can describe all the contexts relevant to the specific data tailoring task.

We can so represent a single chunk configuration of the example of Figure 1 by a positional array of the type: (holder, interest_topic, situation(subdimen of the situation), space). An instance of chunk configuration is: <professor, research, research_act(prj_proposal), university>

Each of the chunk configurations must be then related to a set of relevant concepts of the application domain, corresponding to a set of relevant data in the datasources. We will refer to this set of relevant data as a Chunk. In this paper we show how chunks are designed, by stating the connection between chunk configurations and chunks, and how they are used as queries to tailor the data to be dynamically kept on the mobile device.

Many approaches have been proposed to problems strictly connected to the ones included in our proposals. A lot of works in the literature have designed and implemented adaptive applications by introducing the notions of user profile and contexts [5]–[11]. However, these approaches often provide specific solutions to personalize data that mainly consider either user preferences [5] or device features [7], [9]. In this work we introduce a very general notion of context that can include, at different levels of detail, aspects related to users and usage of an application. On the other hand, our notion of context is a data-centric one, in the sense that the features (dimensions) taken into account in the context model designed for a specific application are based on the data the mobile device has to be endowed with in order to be autonomous in a given context.

In the data integration context a lot of proposals have been introduced, as examples we can cite OBSERVER [12] and MOMIX [13], [14], which explore integration of ontology-based data sources, SEMEX [15], which offers personal data integration capabilities supported by a domain and user model, and INFOMIX [16], a general solution using data source wrappers and a global schema to integrate different sources. In the current work we do not elaborate further the data integration task, rather we focus on the possibility of consider context-awareness to select mainly the structure (and as a consequence also the data) of relevant informations.

The paper is organized as follows: in order to keep the paper self-contained, Section II will briefly summarize the main aspects of the Context-ADDICT project; then, Section III will present a formal definition of the Context Dimension Tree, and its representation in Description Logic and OWL, together with a brief description of the domain model we adopt. Section IV will show the formal relationship between chunk configurations and chunks; such connection is declaratively defined as a (set of) query(ies) specified in Description Logic, defined on the domain ontology. Section IV-B will present the tool which supports the designer in the choice of the ontology concepts to be related to the chunk configurations, and produces the actual queries which define the chunks expressed in OWL, while Section V will draw the conclusions and discuss future work.

II. THE CONTEXT-ADDICT PROJECT AND DESIGN METHODOLOGY

In this section we briefly sketch the Context-ADDICT system architecture and the corresponding methodology, an extension of the Very Small Data Base (VSDB) Methodology [3], [4].

A. An overview of Context-ADDICT

The Context-ADDICT system, depicted in Figure 2, has been designed to support the integration and tailoring of not-known-in-advance datasources in a distributed mobile environment.

We can functionally divide the overall system architecture in three main subsystems:

- design-time,
- run-time schema level,
- run-time data level.

The first subsystem, which is typically used by an application designer, supports the context and domain description based on ontology formalisms, and the definition of their relationship: this is the main issue we discuss in this paper.
As shown in Figure 2, the designer of a Context-ADDICT application will interact with a set of tools to: (1) define the context model, represented as a Context Dimension Tree, (2) adapt an existing domain ontology (or build it from scratch if needed) to represent the main characteristics of the application domain, and (3) define their relationship, supported by the tool named Context Integrator. The result of these operations is an ontology-based model called Enriched Domain Ontology, which contains all the information about the domain, enriched with a set of metadata describing their relations to the contexts represented in the Context Dimension Tree. These operations are performed at design time.

The run-time schema level subsystem is responsible for datasource ontology extraction, integration and tailoring, thus the operations described below are performed at run-time when actual datasources become available. The datasources we consider might be of different types, such as Ontologies, Relational Databases, XML documents, Web services, Web pages, data from sensor networks, data from peers of a P2P network, etc. To manage the actual integration we need to choose a common representation format, and we have chosen to represent the datasources as ontologies for two main reasons: to leverage on the results obtained in literature on ontology integration, and to enable new, smart, reasoning-based applications which are starting to appear in the field of the Semantic Web [17]. Some of the datasources may already offer an ontological representation of their data, while we need to perform semantic extraction to enable the integration of datasources presenting only native interfaces.

For each datasource, the semantic extraction process will produce an ontology, possibly along with a set of metadata useful for query processing: later, the required data, appropriately tailored, may be copied locally, via a synchronization process, or just queried when needed, depending on data characteristics (e.g. it is useless to keep current stock values materialized on the device, since they need to be always up to date).

The result of this process is that for each available datasource we will obtain a uniform representation: a datasource ontology. The datasource ontologies will be integrated, via a mapping procedure, in the Enriched Domain Ontology producing what we have called the Merged Schema. The Merged Schema is the Context-ADDICT equivalent of a dynamically built Global Schema for a distributed database, although enriched with all the information about the context, which will be used for context-aware data tailoring. The Merged Schema will be a very rich ontology, containing all the information on the datasources (no instances yet), the domain description, the context description and their relationship. As already mentioned, the Merged Schema, exploiting the built-in context model, is now ready to be tailored to produce local, context-aware portions of it, containing all the information needed by a given user in a given context, the so called Chunks.

The third subsystem, the data level subsystem, takes care
of data loading and chunk management on the user devices. The data chunks chosen for the current context are selectively retrieved via the datasources wrappers, by using the Merged Schema, which contains the bindings between the domain ontology and the datasource ontologies, and various kinds of metadata, which enable the system to query datasources by using the appropriate data properties.

As one can imagine, in such a distributed and dynamic environment the deployment of such a system is a particularly challenging task; depending on the application scenario different strategies may be chosen for components, from centralized, static ones, to distributed, dynamic ones.

The next section is devoted to a better understanding of the design phase, which is the main subject of this work.

B. The Methodology

The Very Small Data Base (VSDB) Methodology [3], [4] provides guidelines for the design of context based databases for very small devices. To cope with the scenario we have quickly reviewed, this methodology must be extended from a static situation, where the very small database is derived as a set of views over a Global Database Schema, to a fully dynamic one, where the small, mobile database receives the appropriate portion of data from many, time-variant datasources whose schemata may be unknown in advance.

The design methodology can be divided into the following main phases, performed by the application designer on the structures detailed in the next sections:

- **Context Dimension Tree definition**: during this phase the designer will select the needed analysis dimensions and define all the possible values (hierarchically organized) of each dimension. The navigation of the Context Dimension Tree determines the possible chunk configurations.

- **Domain Ontology choice and enrichment**: during this phase the designer takes an already existing Domain Ontology and adds all the missing concepts for the foreseen application scenario; if no Domain Ontology is available, the Domain Ontology is designed from scratch.

- **User Identification hooks definition**: both the Context Dimension Tree and the Domain Ontology are enriched with a set of concepts representing all the possible user identifications, which will allow the data level subsystem to know the appropriate handles which link the device users to the datasource data pertaining to them; for instance, if the Merged Schema contains a concept student, the identification handles might indicate that, in general, good identifiers for students are the student code or the social security number. Such information will be used at run-time, as information for the datasource wrappers, to tailor the instances more effectively, i.e. only those data relevant for the specific student will be retrieved.

- **Chunk definition**, further divided into:
  - **Context-Domain integration**: in this phase the designer connects the Context Dimension Tree to the Domain Ontology, by selecting the domain concepts that are relevant for each chunk configuration of the Context Dimension Tree;
  - **Tailoring Metadata definition**: the Domain Ontology is enriched with metadata aimed at instructing the Tailor Module on how to operate over the Merged Schema at run time. For example, given two domain ontology concepts \( C_1 \) and \( C_2 \), the metadata may specify that the datasource portion to be loaded on the device should contain, besides the concepts corresponding to \( C_1 \) and \( C_2 \), also all the potential relations - yet unknown at design time - between these two.

The final output is the so called *Enriched Domain Ontology*, a domain ontology enhanced and instrumented on the basis of our context and user interest model: the Context Dimension Tree.

III. THE CONTEXT AND DOMAIN ONTOLOGIES

In this section we discuss and formalize our context model, the Context Dimension Tree, and the domain model, the Domain Ontology, formalizing their representation.

A. Context Dimension Tree

According to the above described scenario, the Context-ADDICT context model is a tree whose nodes are dimensions...
and subdimensions, and whose leaves are the dimension values that are to be combined to produce chunk configurations. In this paper, the analysis dimensions and the chunk configurations are formalized as an ontology [18], [19] as follows.

A **Context Dimension Tree** is an ontology \( \langle C, R, I, A \rangle \) such that:

1) \( C \) is the set of concepts and \( C \supseteq \{ \text{Thing}, \text{Dimension}, \text{Chunk} \} \); that is, the set of concepts contains the special concepts Thing, Dimension and Chunk.
2) \( R \) is a set of roles.
3) \( I \) is the set of concept values (i.e. instances of the concepts in \( C \)).
4) \( A \) is a set of axioms.
5) \( C, R, I, A \) are pairwise disjoint;
6) Dimension \subseteq \text{Thing} and \text{Chunk} \subseteq \text{Thing}; that is, the Dimension and Chunk concepts are in relation with the Thing concept.
7) \( \forall C_1, C_2 \in C \) such that \( \neg \exists C_3 \subseteq C_1 \) and \( \neg \exists C_4 \subseteq C_1 \) then \( C_1 \cap C_2 = \emptyset \); that is, leaf concepts are pairwise disjoint sets of individuals.
8) \( \forall a \in I \) such that \( a \in \text{Chunk} \) and \( \forall C \subseteq \text{Dimension} \), \( \exists R_C \in R \) such that \( R_C(a,i) \) with \( i \in C \); moreover, if \( R_C(a,i') \) holds then \( i' = i \): that is, each Chunk instance is in relation with one instance, and only one, of each Dimension sub-concept.

Consider for example the Context Dimension Tree in Figure 1. Since this is a working example, we have kept it quite simple and have not inserted very detailed classes of concepts; for example, the holder dimension is composed by the concepts professor and student, and is not further specialized in subclasses such as full professor or associate professor. The Context Dimension Tree for the university ontology can also be expressed in Description Logic (DL) [20] as in the boxes below; the first describes the Taxonomy of the Context Dimension Tree, the second the relations with their domain, range and cardinality, instances, and the last their disjointness axioms:

**Fig. 4.** Domain Model: graphical representation of the OWL-based domain ontology.
Thus, the ontology contains as classes all the nodes of the Context Dimension Tree of Figure 1; the set of arcs in the tree are formalized in the ontology by the is-a relation. Moreover, the classes Chunk and ValidChunk are introduced: the first has some relations used to group one and only one value for each dimension in each chunk configuration, whereas the ValidChunk class is a subset of Chunk as defined below. The corresponding OWL-based ontology is in Figure 1.

The chunk configurations can be easily derived from this representation by instantiating the selected analysis dimensions; i.e. all dimension values conceived by the context designer are considered, and combinatorially combined. For example, for each holder value, all the possible chunks are built by combining it with all the values of all the other dimensions.

Since this process would produce a very large amount of chunk configurations - if \( n_i \) is the number of possible values of dimension \( i \), the final number of configurations is \( n_1 \ast n_2 \ast \ldots \ast n_k \) - the derivation must be done with an eye to the configurations’ actual significance; indeed, only some of the possible combinations of dimension values make sense in the real world, and some of the combinatorially generated chunk configurations must be discarded. In order to reduce the number of valid chunk configurations, the designer can define a set of semantic constraints, further discussed in [21], each represented by a logical implication of the form:

\[
dimension_i = \text{value}_i \rightarrow \dimension_j = \text{value}_j
\]

This means that a chunk configuration which has the value of the dimension \( i \) “makes sense” if and only if the dimension \( j \) is instantiated with value, or with a value in the sub-tree rooted in value. More complicated constraints can be specified by using boolean operators in the antecedent or in the consequent of the logical implication.

The constraints excluding useless chunks in the university example are:

\[
\begin{align*}
\text{holder} = \text{student} & \implies \text{situation} = \text{class} \lor \text{situation} = \text{exam} \\
\text{holder} = \text{student} & \implies \text{interest_topic} = \text{courses} \lor \text{interest_topic} = \text{eating_out} \\
\text{interest_topic} = \text{administration} & \implies \text{space} = \text{university} \\
\text{situation} = \text{exams} & \implies \text{space} = \text{university}
\end{align*}
\]

Such constraints eliminate, for example, the meaningless chunk configuration:

\[
< \text{professor}, \text{administration}, \text{exams}, \text{city} >
\]

The chunk is meaningless because, in the application we are considering, the “interest_topic” administration is supposed to be limited to the university reality.

These constraints allow us to reduce the number of valid chunk configurations of our example from the 64 to 34, leading to a reduction of the designer’s labour to about 53% of the original. The number of valid chunks can be further reduced, by isolating and merging equivalent, thus redundant, chunk configurations. For example, the chunk configurations characterized by holder=student, interest_topic=eating_out, space=university, and any situation value, actually correspond to the same context. This means that the situation dimension is redundant for this set of chunks, thus we can group them in an equivalence class and rewrite the chunk configuration, by using the “*” wildcard, as <student,eating_out,*,university>, and operatively define only one chunk, saving effort.

Given a set \( \mathcal{F} \) of constraints, the Valid Chunk concept is specified by adding the following properties to the Context Dimension Tree definition at the beginning of this section:

- ValidChunk \( \subseteq \) Chunk and ValidChunk is defined by \( \mathcal{F} \); that is, the Chunk concept has the ValidChunk subconcept defined by the set \( \mathcal{F} \).
- \( \forall i \in \text{Thing} \text{ then } \exists C \in C \setminus \{\text{Thing}\} \text{ such that } i \in C; \text{ that is, each instance is a member of a concept different from Thing.} \)

The DL specification of the set \( \mathcal{F} \) of constraints on the university Context Dimension Tree is as follows:

\[
\begin{align*}
\text{Valid}_{\text{Chunk}} & \subseteq \neg \exists \text{holder_rel.student} \sqcup \exists \text{situation_rel.class} \sqcup \exists \text{situation_rel.exam} \\
\text{Valid}_{\text{Chunk}} & \subseteq \neg \exists \text{holder_rel.student} \sqcup \exists \text{interest_topic_rel.courses} \sqcup \exists \text{interest_topic_rel.eating_out} \sqcup \exists \text{interest_topic_rel.research} \\
\text{Valid}_{\text{Chunk}} & \subseteq \neg \exists \text{interest_topic_rel.administration} \sqcup \exists \text{space_rel.university} \\
\text{Valid}_{\text{Chunk}} & \subseteq \neg \exists \text{situation_rel.exams} \sqcup \exists \text{space_rel.university}
\end{align*}
\]

while the chunk configurations for

\[
\text{chunk1} = < \text{student}, \text{courses}, \text{classes}, \text{university} >
\]

can be specified in DL in the following way:

\[
\begin{align*}
\text{holder_rel(chunk1,student)} \\
\text{interest_topic_rel(chunk1,courses)} \\
\text{situation_rel(chunk1,classes)} \\
\text{space_rel(chunk1,university)}
\end{align*}
\]

A more articulated chunk as:

\[
\text{chunk5} = < \text{professor, research, research_act(prj_proposal), univ} >
\]
The complete DL formalization includes a set of axioms as the examples presented for chunk1 and chunk5 for each valid combination of the analysis dimensions.

B. Domain Ontology Representation

In Context-ADDICT, the application domain concepts are represented by a domain ontology \( O_d(\{C_d, \mathcal{R}_d, I_d, \mathcal{A}_d\}) \) which may be defined from scratch by the designer or, in particular for those fields in which commonly-agreed ontologies already exists (e.g. medicine, biology etc.), be simply edited and completed by the designer. This choice is based on the assumption that, with the advent of the Semantic Web, commonly-agreed standard domain ontologies will become more and more frequent. Figure 4 shows a graphical representation of an OWL domain ontology about the university campus life domain. For performance reasons the considered domain ontologies should not have instances: the instances should be retrieved from the datasources after the integration phase, and their absence during the early steps of integration and tailoring improves the performance of both, since most of the integration complexity depends on the size of the input ontologies; we plan to investigate further in this direction.

IV. CONTEXT-DOMAIN RELATIONSHIP

In the previous sections we have shown how we model domain and context as ontologies; in this section we present how the relationship between context and domain models can be naturally rendered by a third ontology. The objective of this phase is to capture, for each context, represented by a (set of) chunk configuration(s) extracted from the Context Dimension Tree, which portion of the domain is relevant, e.g. which set of elements (concepts and relations) of the domain ontology is interesting for the user in that particular context. As already stated, thanks to the integration phase that will be performed at run-time, this relevance relation is automatically extended over the datasources (both at schema and instance level), and also define which data extracted from each datasource are actually relevant for the given user context. The following subsections describe the way we have chosen to capture this idea of relevance and the tool we are developing to support the designer in the definition of the Context-Domain relationship.

A. Relationship definition

Let us start from the domain ontology presented in Section III-B and the Context Dimension Tree described in Section III-A. Table I defines in Description Logic the Context-Domain Relationship, using the array-based notation introduced above to specify the chunk configurations: each chunk configuration has a corresponding set of DL axioms describing the portion of domain ontology relevant for it.

The axioms for a chunk definition are DL expressions that define the relevant role for the specific chunk. They are realized imposing the equivalence between two sets of individuals; the left hand side has the form:

\[ \exists \text{relevant} . \{ \text{chunk}_n \} \]

whereas the right hand side declares the list of objects that are relevant for the chunk and is a disjunction of conjunctive expressions, containing concept names and existential quantified formulae on roles, such as:

\( (\text{Concept1} \sqcap \exists \text{role}. \text{Concept2}) \sqcup (\text{Concept3} \sqcap \exists \text{role2}. \text{Concept4}) \)

It is necessary to specify such axioms for each of the valid chunk. The first two examples of Table I are simple: all the instances of a concept, e.g. Restaurant or FastFood, are relevant for a given chunk configuration. They highlight how, given the same interest topic, different values for the holder dimension produce different chunks; i.e. professors are interested in restaurants to eat out, while students are interested in fast food. The third axiom shows a more articulated example, where not all the instances of a concept are relevant, but only those that satisfy a particular path condition, including the selection of a particular research topic; in this examples $\text{PAR}$ is a parameter to be evaluated at run-time, whose value is used to set a proper filter on the instances. Moreover, not only one concept is relevant to chunk30, but a set of concepts specified in the right-hand side. The fourth rule is used as an example of how different chunk configurations may insist over the same concept, but selecting different sets of instances: e.g. the Faculty concept is considered relevant like in the third axiom, although the set of retrieved instances differs in the two cases. In fact, in rule 3 only those faculty members who are part of a research group involved in projects in the $\text{PAR}$ research area are taken, while in rule 4 all the Faculty instances are retrieved.

Thus, given a Context Dimension Tree ontology \( O_c(\{C_c, \mathcal{R}_c, I_c, \mathcal{A}_c\}) \), a Domain Ontology \( O_d(\{C_d, \mathcal{R}_d, I_d, \mathcal{A}_d\}) \), and a set of Chunk Definition Rules \( CDR \), their relationship is an ontology \( O_r(\{C_r, \mathcal{R}_r, I_r, \mathcal{A}_r\}) \) such that:

1) \( C_r = C_c \sqcup C_d \)
2) \( \mathcal{R}_r = \mathcal{R}_c \sqcup \mathcal{R}_d \sqcup \{ \text{relevant} \} \)
3) \( I_r = I_c \sqcup I_d \)
4) \( \mathcal{A}_r = \mathcal{A}_c \sqcup \mathcal{A}_d \sqcup \mathcal{CDR} \)
5) \( C_r, \mathcal{R}_r, I_r, \mathcal{A}_r \) are pairwise disjoint;

where \( \sqcup \) represents the disjoint union.

The axioms capturing the idea of relevance will be used to tailor the domain ontology (and by extension the datasource ontologies). To perform this task they are exploited at run-time at two different levels:

- At the schema level: they are used as specifications of the set of concepts and roles which are relevant for the given context; in this way we can exclude, from the local schema on the device, the irrelevant concepts and relations (i.e. those not mentioned by the axioms), by
a process of sub-ontology factoring similar to the one described in [22].

- At the instance level: they are directly applied as DL queries, to select which instances of a concept should be retrieved to be stored in the user device. Since the actual implementation will be based on OWL, this procedure will exploit a general purpose reasoner such as Racer [23] or Pellet [24].

The portion of domain ontology identified as relevant for a chunk configuration will be, at run-time, mapped to the datasource schemas the two might be connected via different paths which where not present in the Domain Ontology. Tailoring metadata can express rules like: “take every path shorter than X hops among them”. As a result we are able to predicate topologically about relations and concepts which are unknown at design-time. The tailoring metadata are part of our ongoing research activity.

The context model, the domain model and their relationship are now represented by three different OWL DL [2] ontologies, physically stored in three different XML/RDF files, two of which contain the domain and context models, while the third imports (via builtin OWL constructs) the others, and captures in OWL the information about context-domain relationship described in Table I.

<table>
<thead>
<tr>
<th>Chunk Configuration</th>
<th>DL Chunk Definition Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>chunk10:</td>
<td>$\exists_{\text{relevant}} { \text{chunk10} } \equiv \text{FastFood}$</td>
</tr>
<tr>
<td>$&lt; \text{student.eating_out}, *, \text{university} &gt;$</td>
<td></td>
</tr>
<tr>
<td>chunk17:</td>
<td>$\exists_{\text{relevant}} { \text{chunk17} } \equiv \text{Restaurant}$</td>
</tr>
<tr>
<td>$&lt; \text{professor.eating_out}, *, \text{university} &gt;$</td>
<td></td>
</tr>
<tr>
<td>chunk30:</td>
<td>$\exists_{\text{relevant}} { \text{chunk30} } \equiv$</td>
</tr>
<tr>
<td>$&lt; \text{professor.research}, \text{research_act(prj_proposal)}, \text{university} &gt;$</td>
<td></td>
</tr>
<tr>
<td>$\text{Publication} \sqcap \exists_{\text{publicationResearch}} { \text{Research} \sqcap \exists_{\text{topic}} { \text{SPAR} } } \sqcup \text{ResearchGroup} \sqcap \exists_{\text{researchProject}} { \text{Research} \sqcap \exists_{\text{topic}} { \text{SPAR} } } \sqcup \text{Faculty} \sqcap \exists_{\text{workFor}} { \text{ResearchGroup} \sqcap \exists_{\text{researchProject}} { \text{Research} \sqcap \exists_{\text{topic}} { \text{SPAR} } } }$</td>
<td></td>
</tr>
<tr>
<td>chunk43:</td>
<td>$\exists_{\text{relevant}} { \text{chunk43} } \equiv \text{Faculty}$</td>
</tr>
<tr>
<td>$&lt; \text{professor.administration}, *, \text{university} &gt;$</td>
<td></td>
</tr>
</tbody>
</table>

TABLE I
CHUNK DEFINITION RULES EXPRESSED IN DESCRIPTION LOGIC

Although by applying the constraints over the Context Dimension Tree we have already reduced the number of chunk configurations to be manually defined, this is still a labour intensive step of our methodology. For this reason we have paid some attention to make the designer-tool interaction as intuitive as possible, and further analysis are ongoing to make this phase as automatic as possible.

The task to be fulfilled is the definition, for each valid chunk configuration, of the set of concepts, relations and instances that must be part of the chunk. Figure 5 shows the simple graphical interaction\(^1\) provided by the tool to build the chunk definition rules of Table I. The interaction we have designed allows the creation of simple paths over the domain ontology graphical representation, which express the right-hand side of the axioms. The chunk definition is

\[^1\] We enable also skilled users to manually define their own axioms, which will be automatically checked for consistency.
practically performed by three mouse-based interactions with the graphical representation of the domain ontology:

- **Full node click (left-button):** can be performed on any ontology node; the corresponding concept and all its instances are selected to be part of the chunk (i.e., the set of relevant data for the given chunk configuration). This is used to produce axioms such as

  \[ \exists \text{relevant}. \{ \text{chunk}X \} \equiv \text{Concept}1 \]

- **Edge click (left-button):** can be performed on edges departing from previously selected node; the arrival node is included in the chunk, and only those instances of the edge destination concept which take part to the relation represented by that edge, are included in the chunk. This is used, as shown in Figure 5, to build path conditions such as

  \[ \exists \text{relevant}. \{ \text{chunk}X \} \equiv \text{Concept}2 \land \exists \text{role}^- \cdot \text{Concept}1 \]

- **Filtered node click (right-button):** can be performed on any node, the corresponding concept is chosen. Its instances are taken only if they match a set of conditions the designer will specify on the concept attributes (DataTypeProperties). The condition may involve parameters, such as the user-id or spatio-temporal coordinates to be evaluated at run-time. It is used to produce rules like:

  \[ \exists \text{relevant}. \{ \text{chunk}X \} \equiv \text{Concept}2 \land \exists \text{role}. \{ \text{SPAR} \} \]

Exploiting a combination of these three interactions the designer will easily build the proper set of chunk definition rules.

Let us now use as an example the definition of chunk30 of Table I:

1) by using the menu on the left the chunk30 is selected, thus, the left-hand side of the axiom is set to \( \exists \) relevant.\{chunk30\}

2) with an **filtered node click** operation over the node representing the concept of Research and the selection of the topic attribute as filtering attribute, the following portion of axiom is written: “Research\(\land\exists\text{topic}.\{\text{SPAR}\}”

3) with an **edge click** over the edge publicationResearch the following portion of axiom is added to the right-hand side of the axiom (and the previous value of the axiom is surrounded by brackets): “Publication\(\land\exists\text{publicationResearch.}”. This is the first disjunct.

4) the designer pushes the “new rule” button to create the next disjunct.

Step 2 is repeated with concept Research, step 3 is repeated by clicking ResearchProject and the second disjunct is ready. The same process is repeated to create further disjuncts. At the selection of the next chunk to be defined, the rule just built is saved in OWL format.

This procedure can be easily improved by allowing the user to select not only simple paths, but also a general tree, which is part of the entire graph-based. The definition of the tailoring metadata, currently under development, will be supported in a similar way.

V. CONCLUSIONS

In this paper, after the introduction of the Context-ADDICT system and its methodology, we have focused on the definition of the domain and context models, and the description of their relationship. The Context Dimension Tree has been formally defined and examples in DL and OWL have been provided. We have shown how the context-domain relationship can be naturally rendered in DL (or equivalently in OWL) by a set of axioms which express relevance of portion of the domain ontology for a specific chunk configuration. A tool, that supports the designer in the various phases of the methodology, has been presented, showing how a simple graphical interaction can be used to define the context-domain relationship. As an ongoing research we are working to increase the automatism of all the steps of the methodology, in particular the context-domain relationship definition.

ACKNOWLEDGMENT

We like to thank Cristiana Bolchini, Alessandro Giusti, and Fabio Schreiber for the precious collaboration in the Context-ADDICT project, and Mario Arrigoni Neri for the useful discussions.

REFERENCES


