# Mapping Features to Context Information: Supporting Context Variability for Context-aware Pervasive Applications

Zakwan Jaroucheh, Xiaodong Liu, Sally Smith
School of Computing
Edinburgh Napier University, UK
{z.jaroucheh, x.liu, s.smith}@napier.ac.uk

***Abstract*—Context-aware computing is widely accepted as a promising paradigm to enable seamless computing. Several middlewares and ontology-based models for describing context information have been developed in order to support context-aware applications. However, the context variability, which refers to the possibility to infer or interpret different context information from different perspectives, has been neglected in the existing context modeling approaches. This paper presents an approach for context-aware software development based on a flexible product line based context model which significantly enhances reusability of context information by providing context variability constructs to satisfy different application needs.**

***Keywords-context-awareness; context variability; feature model; software product line; pervasive applications.***

## I. INTRODUCTION

In the emerging (and challenging) pervasive environments, the context management systems are expected to administer a large volume of dynamically changing context information. Ontologies are a very promising instrument for modeling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques [1]. Thus, we focus on context management employing ontologies as the underlying technology. In pervasive applications, we usually adopt semantic Web technologies which possess pervasive context-aware ability to achieve knowledge sharing, context reasoning and interoperability [2][3]. Common to most of the existing approaches the usage of ontologies (e.g. using OWL) to describe the concepts and properties defining context information in the relevant domain. Obviously the reasoning capabilities of the ontology are of crucial importance to context-aware applications for context knowledge representation and reasoning. However, these approaches suffer from some limitations:

Firstly, in a pervasive environment, as the context manger is expected to administer a large volume of context information represented by RDF triples in the context repository, applying the reasoning capability to infer new context knowledge may have a severe impact on the overall performance of the system.

Secondly, applications use context queries to retrieve the set of context information that adhere to some conditions. Some context queries are too complex to be defined using only general-purpose querying mechanisms (e.g., SPARQL). In addition, the application developer may not have enough knowledge about context semantics, in order to describe queries correctly.

Thirdly, in order of the middleware to serve different types of applications, it should provide context-specific programming abstraction or constructs that model the context variability. Indeed, different context knowledge could be extracted from the context repository by focusing on different views of the context information. For example, in the smart meeting room, a seat may be equipped with light and temperature sensors to reason about its occupation. The seat could be either free or occupied. Two occupation variants may be identified: occupied by object and occupied by a person. These variants represent two facets to the same fact. To the author's best knowledge, the existing approaches do not provide application developers with software constructs through which a view-based customization of the context knowledge could be expressed.

Motivated with these problems and directives in mind, we propose a product line based context modeling approach. Commonality and variability management techniques (e.g., feature model) from software product line can be applied to handle context variabilities for per-application customization.

The paper is organized as follows: in Section 2, we describe the rationale behind the proposed approach. In Section 3 we describe the proposed framework for context management that is illustrated in a simple case study in Section 4. The discussion and concluding remarks end the paper.

## II. RATIONALE OF THE PROPOSED APPROACH

This paper does not intend to discuss the requirements of context modeling which have been already discussed in an earlier paper [4]. Conversely, in this paper we focus on dealing with context variability from the application requirement perspective. The aim is to represent the context information from the requirement perspective. The rationale behind this approach is as follows.

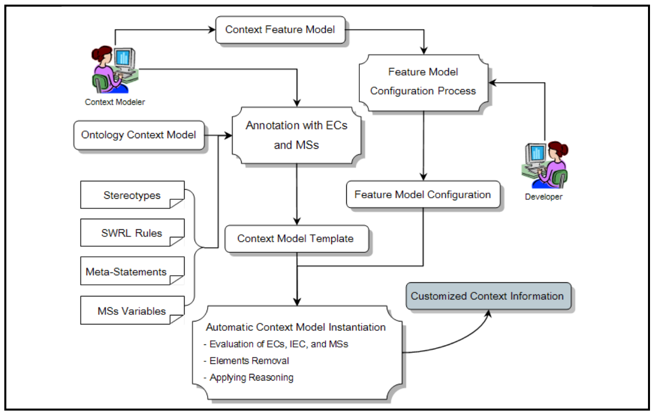
Context Feature Model
Context Modeler
Feature Model Configuration Process
Ontology Context Model
Annotation with ECs and MSs
Developer
Stereotypes
SWRL Rules
Meta-Statements
MSs Variables
Feature Model Configuration
Context Model Template
Customized Context Information
Automatic Context Model Instantiation
- Evaluation of ECs, IEC, and MSs
- Elements Removal
- Applying Reasoning

Figure 1. Overview of the proposed approach

Firstly, in terms of modeling philosophy, in ontology modeling a concept is described by adding its details and implicitly defining in a bottom-up fashion the scope of the concept through the details. Whereas, in feature modeling, a concept is described by first setting its scope and hierarchically adding its details in a top-down fashion [5]. This feature is quite interesting as it allows the context modeler to devise, in a top-down fashion, generic and reusable context features which can be shared among all applications that need to use this context. The relationships between context features express the context variability from the application point of view.

Secondly, according to the context working definition previously presented in [4], we consider that the context knowledge is composed of a set of small contextual knowledge pieces namely *context primitives* which include context entities, attributes, associations, and rules. Each context feature corresponds to a specific set of context primitives. Obviously, considering only the relevant *context primitives* will improve the reasoning performance and reduce response time which is a vital issue in a pervasive environment.

Thirdly, as developers usually do not have full understanding of the context internal semantic, "promoting" the context information using the feature model will enable the contextual knowledge visibility from different views in a top-down fashion. Another advantage is that these context features might be shared between applications which significantly enhances the reusability of context information and reduces application complexity.

## III. CONTEXT AS A DYNAMIC PRODUCT LINE

We import the concepts of features from FODA (Feature Oriented Domain Analysis) [6]. FODA appeals to us because features are essential abstractions that both context consumer and provider understand. Thus, the main concept in the feature description language FODA is the feature itself. Here a feature is a set of context primitives that is relevant to some stakeholder from a specific "focus" point of view.

Indeed, both middlewares and context models are strongly interdependent since the complexity of a context model determines the complexity of context management by a middleware. Coutaz et al. [7] presents this relationship as a conceptual framework that interconnects an ontological foundation for context modeling with the middleware (runtime infrastructure).

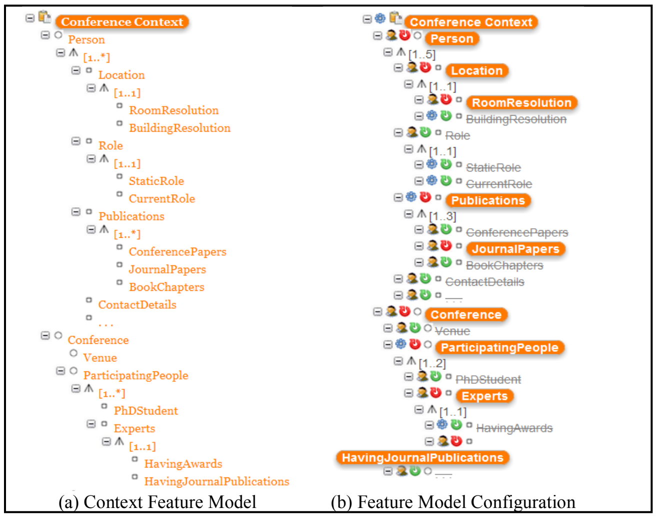

Figure 2. Example of context feature model

### A. *Application-level Context Modeling*

In order to identify which of the context information is eligible for being modeled as feature, we have adopted a simplified list of criteria:

- Identify the context information required by the application adaptation e.g. user location. This should be represented by a generic feature in the feature model.

- Identify the context model transformations or interpretations of the currently available context information in order to be shared by all application instances e.g. room-, floor-, and building-resolution user location information. These interpretations should be represented by different feature variants.

- Regrouping the different identified context features into a logical hierarchy of features in a top-down manner that could be reused by different applications.

The context feature model will be published in a public registry. When an application developer needs to use context information, she reads the XML file representing the context features the context manager is able to deliver to understand the context semantics. Then she is able to configure the feature model and use the middleware services to get the necessary context information.

### B. *Annotated Context Model*

An overview of the proposed approach is shown in Fig. 1. A *context model family* is represented by the context feature model and the ontology-based context model (OCM). The elements of OCM namely the context primitives may be annotated using *existence conditions (ECs)* and *meta-statements (MSs)*. These annotations are defined in terms of features and feature attributes from the feature model, and can be evaluated with respect to a feature configuration. An EC attached to a context primitive indicates whether the primitive should exist in or should be removed from a context product. MS is mainly used to modify or compute

```
<configuration model="Context Feature Model">
   <feature id="RoomResolution">
     <value>1</value>
   </feature>
   <feature id="HavingJournalPublications">
     <minimumJournalRank>350</minimumJournalRank>
     <value>1</value>
   </feature>
    ...
</configuration>
```

Figure 3. Feature model configuration

```
<stereotypes>
<stereotype name="RoomResolution" expression="$RoomResolution ||
$BuildingResolution"></stereotype>
<stereotype name="Paper" expression="$ConferencePapers || $JournalPapers ||
$Experts"></stereotype>
<stereotype name="ExpertHavingAwards" expression="$HavingAwards"/>
...
</stereotypes>
```

Figure 4. Example of available stereotypes

```
<owl:Class rdf:ID="Building">
  <rdfs:subClassOf rdf:resource="#CompoundPlace"/>
  <rdfs:label>BuildingResolution</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Room">
  <rdfs:subClassOf rdf:resource="#AtomicPlace"/>
  <rdfs:label>RoomResolution</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:ID="relatedToJournal">
  <rdfs:domain rdf:resource="#Artefact"/>
  <rdfs:range rdf:resource="#Journal"/>
  <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
</owl:ObjectProperty>
...
```

Figure 5. Example of annotated ontology

```
<metastatements>
 <metastatement name="MS1">
  <expression>
     PREFIX cxt:&lt;http://www.napier.ac.uk/candel#&gt;
     PREFIX xsd:&lt;http://www.w3.org/2001/XMLSchema#&gt;
     DELETE
       { cxt:FMConfiguration cxt:minimumJournalRank "100.0"
        ^^xsd:float }
     INSERT
       { cxt:FMConfiguration cxt:minimumJournalRank
                "$minimumJournalRankVariable" ^^xsd:float }
   </expression>
   <stereotype>ExpertHavingJournalPublications</stereotype>
 </metastatement>
<metastatements>
```

Figure 6. Example of meta-statement

```
<metastatementsVariables>
<metastatementVariable name="minimumJournalRankVariable"
expression="//feature[@id='HavingJournalPublications']/minimumJournalRank"></met
astatementVariable>
</metastatementsVariables>
```

Figure 7. Example of meta-statement variable

```
<swrlrules>
 <swrlrule name="Rule1">
    <expression> PaperPresentation(?p) ^ hasStartDateTime(?p, ?s) ^
hasEndDateTime(?p, ?e) ^ swrlb:currentDateTime(?c) ^ swrlb:beforeTime(?s, ?c) ^
swrlb:beforeTime(?c, ?e) -> PaperPresentationHappeningNow(?p) </expression>
    <stereotype>CurrentRole</stereotype>
 </swrlrule>
 <swrlrule name="Rule2">
   <expression>Researcher(?r) ^ authorOf(?r, ?p) ^ relatedToJournal(?p, ?j) ^
hasRank(?j, ?rank) ^ FMConf(?conf) ^ minimumJournalRank(?conf, ?minRank) ^
swrlb:greaterThan(?rank, ?minRank) -> ExpertResearcher(?r)
   </expression>
   <stereotype>ExpertHavingJournalPublications</stereotype>
 </swrlrule>
...
</swrlrules>
```

Figure 8. Example of annotated SWRL rules

```
<ExpertResearcher rdf:ID="Alice">
   <rdf:type rdf:resource="#Researcher"/>
   <authorOf>
    <Paper rdf:ID="SecondPaper">
      <relatedToJournal>
        <Journal rdf:ID="Journal4">
           <hasRank
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">462.0</hasRank>
           <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>ACM TRANSACTIONS ON COMPUTER SYSTEMS</hasName>
       </Journal>
     </relatedToJournal>
     <biblioReference rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
> Product Line based Context Management </biblioReference>
    </Paper>
   </authorOf>
   <authorOf rdf:resource="#FirstPaper"/>
   <locatedInRoom rdf:resource="#C35"/>
</ExpertResearcher>
...
```

Figure 9. The retrieved context information

the attributes of context model element. This is important for managing context variants as we will see in the case study in Section 4.

An instance of a context model family, which we call *context product (CP)*, can be specified by creating a feature configuration based on the context feature model. Based on the feature configuration, the corresponding context product is generated automatically. The generation process, which is model-to-model transformation, involves evaluating the ECs and MSs with respect to the feature configuration, removing context primitives whose ECs evaluate to false and, possibly doing additional processing such as removing related context primitives.

Obviously, a particularly interesting form of ECs is a Boolean expression over a set of variables each of which corresponds to a feature from the feature model. Given a feature configuration, the value of a feature variable is true if and only if the corresponding feature is included in the feature configuration. In our prototype implementation we use either Boolean expressions in Disjunctive Normal Form (DNF), or more general XPath expressions which can access feature attributes and use other XPath operations, as long as the XPath expression evaluates to a Boolean value. The EC is represented by one or more *stereotypes*. Further, the ECs should be interpreted with respect to the OCM containment hierarchy. In other words, if a context primitive container is removed all the contained context primitives are removed. For example, if entity x is a sub-entity of the entity y, removing y requires removing x as well.

## C. *Implicit Existence Condition (IEC)*

Context primitives that are not explicitly annotated will have implicit EC. The IEC for a context primitive can be provided based on the existence conditions of other context primitives and on the syntax and semantics of the OCM. For example, according to the ontology syntax, an object property requires a class at each of its ends. Thus, a reasonable choice of IEC for an object property would be the conjunction of the ECs of both classes. This way, removing any of the classes will also lead to the removal of the object property. IECs reduce the necessary annotation effort of the user.

## D. *Context Information Generation*

A context information generation process involves computing MSs and ECs, and removing elements whose ECs are false. The complete context product instantiation algorithm can be summarized as follows:

- *Evaluation of MSs and explicit ECs*: The evaluation is done while traversing the OCM containment hierarchy in depth-first order. Children of context primitives whose ECs evaluate to false are not visited because they will be removed.

- *Removal Analysis:* Removal analysis involves computing IECs. The IECs can be computed in a single additional pass after evaluating explicit ECs. In addition, in this step all the individuals and statements whose subjects are included in the elements to be removed are also marked to be removed. For example, if the Room entity is known to be removed, all its individuals and all triples whose subject is of type Room should be marked to be removed.

- *Primitive Removal:* In this step, primitives whose ECs are false are removed.

- *Applying Reasoning:* In order to interpret the remaining context information from the perspective specified by the context feature configuration, it is necessary to apply the corresponding remaining rules. The result of the reasoner will be the context product.

In the implemented prototype we use rule-based inference reasoners. Different rule-based systems provide different logical inference support for context reasoning. To reason about ontologies, a description logic reasoner, namely Pellet is applied. We use the Semantic Web Rule Language (SWRL) on top of OWL for interpreting context using domain specific rules and producing new facts. However, the approach could be extended to use other reasoner types.

## IV. Case Study

This section describes a case study of different applications supporting a conference event. We use some concepts of the SO4PC ontology [2] for expressing context information associated with persons, time, and spaces; and another ontology for describing the research related concepts.

Fig. 2 (a) shows an example of a context feature model that represents different features that could be shared among different applications. For example, if the `Location` feature has been selected, then two mutually-exclusive options are available; either as a room resolution; or as a building resolution. Fig. 2 (b) shows one possible context feature configuration. Each feature may have several attributes. For example, in Fig. 3 that shows a part of the feature model configuration XML file, the `HavingJournalPublications` feature has two attributes: `value` which indicates the selection of the feature or not, and `minimumJournalRank`. This feature allows the retrieval of researchers who have been published in journals whose rank is superior to the `minimumJournalRank` value.

As previously mentioned, in order to link the context feature model to the context primitives, we use stereotypes to annotate ontology elements as well as the SWRL rules. Fig. 4 shows a snippet of the XML files containing the available stereotypes to use for annotation. Each stereotype expression is expressed, as described above, in terms of the features' values of the context feature model. Fig. 5 shows a sample of the annotated ontology elements. We use the Label property to specify the correspondent stereotypes of each element. Further, as mentioned above, MSs can be expressed using XPath. As an example, the MS represented in Fig. 6, uses the SPQRL Update expression to update the datatype property `minimumJournalRank` of the entity `FMConfiguration` by a value retrieved from the variable `$minimumJournalRankVariable` whose value is determined by the XPath expression of the variable `minimumJournalRankVariable` in Fig. 7. Fig. 8 shows a sample set of annotated SWRL rules. For example, Rule1 is used to reason about the paper presentations that are currently taking place. To determine if the researcher is an expert we have two options: by choosing the `HavingAwards` or `HavingJournalPublications` features. The stereotype of the rule is specified by the `stereotype` element. Fig. 9 shows an example of the retrieved context information corresponding to the feature model configuration (Fig. 2 (b)).

## V. Discussion

The proposed approach has several advantages. Firstly, from the context modeler usability perspective, the proposed approach is intuitive; it allows her to think about the context information from different perspectives and use the feature model available tools. Secondly, context feature model allows the context modeler to devise context-specific features that can be shared among all applications that need to use this context. Moreover, retrieving context information using general-purpose query mechanisms remains possible by devising a special context feature.

Thirdly, unlike the reasoning on a one monolithic context information, the proposed approach gives the possibility to provide the context information on arbitrary levels of abstraction thanks to the arbitrary composition of context primitives e.g. inference rules. Fourthly, the use of context-specific features may improve the overall performance of the system, since it might decrease the number of network interactions between an application and the context provider.

## VI. Conclusion

This paper has presented an approach for supporting context-aware applications based on a flexible product line based context model. The proposed approach to model the context information allows the context modeler to specify the context information in a high-level and logical way that regroups context variabilities; and provides application developers with context-specific programming constructs to express their needs. The result is a more intuitive way to represent context and improve overall systems performance. Further work includes extending the proposed approach to the distributed context management architecture.

## References

[1] J. Euzenat, J. Pierson, and F. Ramparany, "Dynamic context management for pervasive applications," The Knowledge Engineering Review, vol. 23, 2008, pp. 21-49.

[2] J. Man, A. Yang, and X. Sun, "Shared Ontology for Pervasive Computing," Lecture Notes in Computer Science, vol. 3818, 2005, pp. 64 - 78.

[3] X.H. Wang, T. Gu, D.Q. Zhang, and H.K. Pung, "Ontology Based Context Modeling and Reasoning using OWL," Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, IEEE CS, 2004, pp. 18-22.

[4] Z. Jaroucheh, X. Liu, and S. Smith, "CANDEL: Product Line Based Dynamic Context Management for Pervasive Applications," International Conference on Complex, Intelligent and Software Intensive Systems (ARES/CISIS 2010), IEEE CS, 2010, pp. 209-216.

[5] K. Czarnecki, C. Hwan, and K.T. Kalleberg, "Feature Models are Views on Ontologies," Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society, 2006.

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Distribution, 1990.

[7] J. Coutaz, J.L. Crowley, S. Dobson, and D. Garlan, "Context is Key," Communications of the ACM, vol. 48, 2005, pp. 49-53.