Abstract: The topic of Learning Objects (LOs) is a central theme in eLearning research. Because of a variety of aspects that relates to LOs, their complexity and development issues is a great challenge. This requires adequate means for specifying and representing of LOs at a higher abstraction level. The introduction of UML into eLearning was not very successful because of the inherent UML’s orientation towards a very specific technological paradigm with specialized concepts such as "concurrency" or "polymorphism". As OWL has the other intention, we propose to use Feature Diagrams (FDs) in eLearning for specification and representation of a learning content. The motivation is a close relationship of FDs to domain ontologies that are treated as a perspective instrument for eLearning. The paper presents the background of FDs, some extensions of FDs related to eLearning, and the extended motivation of benefits and limitations of the use of FDs in eLearning. All these are supported by Case studies.

Introduction

The concept of learning object (LO) is one of the most fundamental terms in eLearning and learning per se. The importance of the concept is well understood in the community; however, the understanding of the essence of the term is still poor. For example, there is a variety of definitions of the term (Rossano et al. 05, IEEE 02, Wiley 00a, Polsani in McGreal 04, Sosteric & Hesemeier in McGreal 04) (the list of references is far from being exhaustive) as well as a variety of taxonomies and standards related to LOs (Rossano et al. 05, McGreal 04). They indicate two things: 1) how complex and 2) how important the domain of LOs is. The main idea behind the LOs is to break down a teaching content into small chunks that can be reused in various learning environments (Wiley 00b, Northrup 07). When reused, such units are combined in various ways leading to a great variability of the learning content.

In general, complexity of LOs originates from our understanding that a LO is a container of knowledge in any teaching discipline, which should be dealt with from different perspectives: course designers, teachers and learners simultaneously taking into account various aspects and perspectives, such as pedagogy, IT capabilities, environment, costs, etc. This extends substantially the number of variants of both knowledge units to be represented and LOs per se. Thus the variability is an important measure of LO complexity. How this complexity can be managed? Currently, LOs (IEEE 02) are a leading technology of choice for eLearning support due to its potential generativity, adaptability, and scalability (Wiley 00b). LOs are also seen as computer-based teaching components that are to be modeled in the development phase in some well-established way. However, the development of LOs remains a vague issue, because there is still no clearly defined and widely adopted LO specification and development methodology as, e.g., in software engineering, where classes and objects are modeled using UML (Fowler 03). There have been several efforts to adopt UML in eLearning domain, e.g., for modeling the interaction between LOs
and a specific Learning Management System (LMS) (Gao et al. 05), or to describe the content and process within "units of learning" in order to support reuse and interoperability (Laforcade 05). However, these efforts have not been entirely successful, because of inherent UML’s orientation towards a very specific technological paradigm with specialized concepts such as "concurrency" or "polymorphism" (Friesen in McGreal 04). Such variability can not be expressed and modeled using UML, which has not adequate means for expressing variants of configurations of a system (LO in this context).

To be delivered to the learners, the teaching knowledge (in the form of LOs) must first be analyzed, understood, specified, and represented, respectively. Specification and representation of LOs play a significant role for the LO development, sharing and reusing (learning). In such a context, we propose to use Feature Diagrams (FDs) for modeling (we use this term in a wide sense here meaning the conceptual modeling) learning content. Originally, FDs were introduced in Feature-Oriented Domain Analysis (FODA) method (Kang et al. 90). FDs were first applied in the context of industrial manufacturing product lines, e.g., for modeling car assembly lines. Later, the idea was extended to software product lines (PLs) (Clements & Northrop 02). The concept, if applied systematically, allows for dramatic increase of software design quality, productivity and provides a capability for mass customization and leads to the “industrial” software design (MacGregor 02).

Based on the success of feature modeling and PL approach in software engineering domain, we propose to use FDs in eLearning domain for specification, representation and structuring of learning content. The motivation is as follows: 1) inadequacy of UML-based approaches for eLearning as it is mentioned above; 2) a close relationship of FDs to domain ontologies that are treated as a perspective instrument for eLearning (Braice & Nejdl 04) and 3) in comparison with UML- or OWL-based (Bechhofer et al. 04) approaches, FDs is a technology-independent and a more intuitive approach allowing represent LOs at a higher abstraction level. The aim of the paper is to provide a more extensive motivation and case-based exploration of the benefits of using FDs in the LOs domain. Our contribution is a conceptual innovation and suggestions of how to extend and adapt FDs to this domain.

The paper is structured as follows. First we describe a background of the FD-based modeling framework. Then we present and analyze the extended motivation of using FDs including relationship with the ontology-based approaches. We also examine our case studies of using FDs as a tool for solving some methodological problems, as well as means for representing knowledge within LOs and managing variability within LOs. Finally we discuss limitations, provide an overall evaluation of the approach, formulate conclusions and outline the future work.

Background of Feature-Based Approach

FDs is a graphical language used for representing and modelling variability at a higher abstraction level, usually at the early design stages, such as formulation of requirements for PL software designs. Below, we present basic definitions and syntax and semantics of conceptualized FDs.

Feature Definitions and Original Context of Use

From the perspective of software engineering, it is commonly accepted that a domain can be analyzed and modelled at a higher abstraction level using feature-based approaches. Informally, a feature is a prominent characteristic of a system, entity or concept in a domain. Since in the software engineering literature there is no consensus on what a feature is, we deliver some definitions of the term. A feature is: (1) End-user visible characteristic of a system or a distinguishable characteristic of a concept that is relevant to some stakeholder (Kang et al. 90, Kang et al. 02); (2) A logic unit of behaviour that is specified by a set of functional and quality requirements (Bosch 00); (3) Qualitative property of a concept (Czarnecki & Eisenecker 00). (4) A functional requirement; a reusable product line (PL) requirement or a characteristic that is provided by one or more members of a software PL (Gomaa 04). All these notions of features are valid and more relevant for different PL methodologies that are dealt with within Software PL engineering which is “a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization” (Pohl et al. 05). The Software Engineering Institute (SEI) defines a software PL as “a set of software-intensive systems sharing a common managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of a core assets in a prescribed way”. By analogy, related LOs can be seen as product families for the eLearning domain.
Feature Model

The intention of a feature model is to represent and model a domain or its part using the feature concept. Specifically, this activity can be seen as a part of the domain analysis process, for example, as it is described by the FODA method (Kang et al. 90). Feature modelling is the activity of modelling the common and the variable properties of concepts and their interdependencies in a domain and organizing them into a coherent model referred to as a feature model (Czarnecki & Eisenecker 00). Therefore a feature model represents the common and variable features of concept instances (sub-features) and the dependencies and relationships between the variable features. The model delivers the intention (usually implicitly) of a concept, whereas the set of instances it describes is referred to as an extension, which narrows the meaning and scope of the concept.

This extension is often referred to as a hierarchy of features with variability (Czarnecki et al. 06). The primary purpose of a hierarchy is to represent a potentially large number of features into multiple levels of increasing detail. Variability defines what the allowed combinations of features are. To organize a hierarchy as an allowed combination of features the identification of feature types is essential. Feature types are the inherent part of the feature model (see next three sub-sections).

Usually a feature model is expressed using FDs that can be supplemented by some additional information such as short semantic description of each feature, rationale for each feature, constraints, default dependency rules etc. Thus FDs are means for the modelling and management of variability, i.e. the commonalities and differences in the applications in terms of requirements, architecture, components, and test artefacts (Pohl et al. 05).

Feature Types

There are three basic types of features: mandatory, optional and alternative. Mandatory features allow us to express common aspects of the concept (usually they are referred to as commonality (Coplien et al. 98)), whereas optional and alternative features allow us to express variability. All basic features may appear either as a solitary feature or in groups. If all mandatory features in the group are derivates from the same father in the parent-child relationship we can speak about the and-relationship among those features (see also Tab. 1). An optional feature is the one which may be included or not if its father is included in the feature model. Alternative features, when they appear in groups as derivates from the same father, may have the following relationships: or, xor, case, etc. The xor-relationship also can be treated as a constraint (Tab. 1). Usually this relationship appears as a constraint when features are derived from different parents. More advanced sub-types of alternative features are group constrains, attributes, cloning and additional constraints (Czarnecki et al. 06).

A sub-feature may be mandatory, alternative, or optional with respect to only the applications, which also enclose its parent feature. If the parent of the feature is not included in the description of the system, its direct and indirect sub-features are unreachable. Reachable mandatory features must be always included in every system instance, while an optional feature may be included or not, and an alternative feature replaces another feature when included. In FODA, selected optional and alternative features are highlighted in the feature diagram for a specific system with the boxes around the name of the selected feature. External composition rules describe additional dependencies (e.g., constraints) between the features within of a FD.

Feature Diagram Definition

The first and seminal proposal to use feature diagrams was introduced by Kang and his colleagues as part of the FODA method back in 1990 (Kang et al. 90). A feature diagram (Fig. 1) is the tree-like or directed acyclic graph (DAG) that consists of a set of nodes, a set of directed edges, and a set of edge decorations. The root represents the top level feature (i.e., concept, entity, system or domain per se). The intermediate nodes represent compound features and leaves represent atomic features that are non-decomposable to smaller ones in a given context. The edges are used to progressively decompose a compound feature into more detailed features. Edges of the graph also denote relationships or dependencies between features.
Feature Diagram Notation

As a result of active research in the field of feature diagram notations and their use, currently there are many similar notations with different syntactic and semantic discrepancies. For more details, see for example (Djebbi & Salinesi 06, Schobbens et al. 06, Sipka 05). In the context of this paper, we use the notation of feature diagrams that is shown in Tab. 1. The presented notation is a result of some adaptation of works (Czarnecki & Eisenecker 00, Djebbi & Salinesi 06, Schobbens et al. 06) as well as our own contribution (case of relation, a function relation between feature values and contextualization of FDs).

![Car diagram]

**Figure 1:** Feature diagram of a simple car (Czarnecki & Eisenecker 00).

<table>
<thead>
<tr>
<th>Feature type</th>
<th>Definition, formalism and semantics of relationships</th>
<th>Graphical notation (syntax)</th>
</tr>
</thead>
</table>
| Mandatory (and–relationship) | Feature B (C, D) is included if its parent A is included: 
  1. If A then B; 
  2. If A then C&D;  
  Father A has son B; or Father A has sons C and D | ![Mandatory diagram](Image) |
| Optional | Feature B (C, D) may be included if its parent A is included: 
  1. If A then B or no feature;  
  2. If A then C or D or no feature;  
  Father A may have son B; or Father A may have sons C&D;  
  or B may be son of father A; C & D may be sons of father A | ![Optional diagram](Image) |
| Alternative: sub-type case-selection | Exactly one feature (B or C or D) has to be selected if its parent A is selected: 
  1. If A then case of (B, C);  
  2. If A then case of (B, C, D) Only one son from the list (B, C) or from the list (B, C, D) is selected; Relationship “case of.” | ![Alternative case diagram](Image) |
| Alternative: sub-type or-selection | At least one feature (B, C or D; or B and C; or B and D; or C and D; or B and C and D) has to be selected if its parent A is selected: 
  1. If A then any of (B, C);  
  2. If A then any of (B, C, D). Relationship any (number) of, i.e. or-relationship | ![Alternative or diagram](Image) |
| Alternative: sub-type xor-selection | If A then (B but ¬C) or (C but ¬B)  
Diffs from case–selection by 1) having two sons only 2) label xor usually written below the father’s node | ![Alternative xor diagram](Image) |
| Context | Describes context of feature relationship (our contribution) | Written at the root of the tree |
| Functional | There can be other relationships: e.g., describe, value of A (V_A) is some function f of the feature B value (V_B) = f(V_B) (our contribution) | ![Functional relation](Image) |
| Constraint exclusive | If I then ¬K and if ¬F then K (here F and K are atomic features derived from different parents);  
Mutual exclusive features (xor relationship between F and K) | ![Constraint exclusive relation](Image) |
| Constraint require | Feature A requires feature B, or shortly: A (require) B | ![Constraint require relation](Image) |

**Note.** Tab. 1 has no link relationship to other FDs, which usually is denoted by tag <reference> <other FD’s name>
Motivation of Using Feature Diagrams (FDs) for LO Domain

There are at least three reasons (they will be demonstrated by our Case studies), why FDs are beneficial for the LO domain:

A. Methodological,
B. Ontology-based knowledge representation within LOs,
C. Managing variability among LOs of similar courses or different variants of LOs within the same course.

A. From the methodological perspective, in the development of LOs, there is the need for specification requirements at a higher abstraction level. Because FDs is a graphical language which is a domain and application independent and also independent upon the implementation technology of LOs, the language can be seen as a tool for specification and representing of LOs. In contrast to the UML notation, syntax and semantics of FDs are simpler and thus can be easily learnt by different stakeholders (course designers, course experts, teachers and learners, etc.). FDs as a means of the unified representation, promote reusability and interoperability in analysis, sharing and distributing of knowledge related to LOs. As it will be shown in our case study 1, we can express on a common language different aspects of known definitions using the feature notation. FDs serve as a tool for a graphical comparison of the analyzed definitions and derivation of a generic definition of LOs. In general, FDs can contribute to the formation of the theory of LOs.

B. From the ontology-based perspective, LOs is a break-down of a teaching content into small chunks that can be reused in various learning environments (Wiley 00b). When reused, such knowledge units of the content are combined in various ways leading to a composition of complex relationships that can be seen as domain ontology. In the eLearning domain as well as in various other domains (e.g., computer science, information science, artificial intelligence, etc.), an ontology is usually conceived as a data model that represents a set of concepts within a domain and the relationships between those concepts. For example, OMG (OMG 07) defines ontology as “the common terms and concepts (meaning) used to describe and represent an area of knowledge”. According to this understanding, ontology can range from Taxonomy to a Thesaurus, or to a Conceptual Model (with more complex knowledge), or even to a Logical Theory (with very rich, complex, consistent and meaningful knowledge).

In the context of sharing and reuse of formally represented knowledge among Artificial Intelligence (AI) systems, T.R. Gruber (Gruber 95) defines ontology, as “an explicit specification of conceptualization”. Specifically, the ontology-based model is a form of knowledge representation about the world or some part of it. It is used to reason about the objects within that domain. Ontology generally describes (or consists of) the following items: Individual, Classes, Attributes, Relations and Events. These terms (except event) have the direct analogs in the feature-based approach (individual ↔ solitary feature; class ↔ grouped features; attributes ↔ variants of a feature, Relations ↔ parent-child relationships and various constraints).

As a feature can be treated as a chunk of knowledge to be learnt in the representation of LOs, FDs contribute to explicit structuring of learning content (chunks of knowledge) at the different level of abstraction. FDs allow expressing the representation of interrelationships between basic knowledge chunks (features) explicitly (see Case study 2), thus they are related to the representation of knowledge and may contribute to better understandability and perception. Knowledge of LO are usually represented using some knowledge-based approach, such as ontology trees of LOs (Bruce & Nejdl 04). Domain ontologies, where domain knowledge is represented as ontology trees, have some syntactic and conceptual resemblance with feature hierarchies represented using FDs. However, FDs have weaker capabilities to express various relationships in representing knowledge (Czarnecki et al. 06). On the other hand, when those capabilities are not enough, FDs can be easily combined with more powerful knowledge representation methods, such as fuzzy logic (Robak & Pieczynski 03).

C. LOs are complex entities entailing many different aspects with a great deal of variants. FDs allow expressing and grasping the common and variable features of LOs explicitly (see Case Study 2). Variability is especially important for representation and development of the so-called generative LOs (Boyle et al. 04).
Case Studies

Case Study 1: Contribution to Domain Analysis and Understanding

In Case study 1, we show how FDs can contribute to the methodology of analysis and understanding of the LOs domain per se. Fig. 2 presents a generic model (a meta-model) represented using a FD that models features of LOs which definitions were taken from the analyzed papers (Rossano et al. 05, IEEE 02, Wiley 00b, Sosteric & Hesemeier in McGreal 04). The model has mandatory features which were mentioned explicitly. The model contains some important relationships among mandatory features or between mandatory and optional features. Features named by ‘Others’ provide the possibility for further extension of the model. Relationships in the model were introduced in ad hoc manner since the analyzed definitions do not provide dependencies among features explicitly.

![Figure 2: A generic model for interpreting the known definitions of LO.](image)

Case Study 2: Contribution to Learning Process and Knowledge Representation

In Case study 2, we provide two interrelated LOs of small granularity taken from the Computer Science domain. The first LO (Fig. 3) is aiming at learning of principles of polynomial calculation effectiveness. Two representation forms of the polynomial are provided: canonical and Horner scheme. Effectiveness of calculation is evaluated by the number of multiplications and additions. All sub-processes are treated as mandatory features (black circles in Fig. 3), except the feature #ADD derived from ‘Canonical form’, which is optional (white circle in Fig. 3). This feature (#ADD) is optional because of the contextual explanation <n is large> (theoretically in this case, effectiveness depends upon #MULT only). In general, the Canonical form requires \((0.5*n*(n+1) + n)\) operations. If we change the context, e.g., by writing <n is small>, than the optional feature should be changed to the mandatory one. Horner scheme requires \(2*n\) operations and is much more efficient.

Therefore, the second LO (Fig. 4) is aimed at learning of how effective the implementation of the Horner scheme can be. The implementation can be done in two domains: in the programming domain, when a specific program instance has to be written, and in the meta-programming domain, when a family of program instances dependent on the predefined contexts has to be written (or, more precisely, generated automatically). Note that a meta-program is a program that creates automatically other programs (in this case, a family of explicitly written Horner schemes).

Again, the FD that describes the implementation possibilities of the LO depends on the context. As the context is broad (<in both domains and predefined contexts>) the FD is solid enough and contains all types of features and relationships (except alternative because there is no or-relationships) and constraints (except <requires>) that were introduced in Tab. 1. Furthermore, the features <Encoding with loop> and <One context only (n=4)> are treated as mandatory. Encoding using the explicit representation of the Horner scheme is more efficient than the implementation by a computer program with a loop because the loop itself requires additional calculations (compare Example 1 and Example 2). This problem is known in Computer Science as loop enrollment.
Example 1: \( n := 4; y := a[0]; \text{for } i := 1 \text{ to } n; \ y := y \ast x + a[i]; \text{end; } \{ \text{Canonical form } \}

Example 2: \( y := ((a[0]\ast x + a[1]\ast x + a[2]\ast x + a[3])\ast x + a[4]); \{ \text{Horner scheme } \}

**Figure 3:** Feature diagram representing effectiveness of polynomial expression calculation.

Though Examples 1 and 2 have different efficiency, however, both represent the *programming domain*. The feature <variety of different contexts> (Fig. 4) is further decomposed in order we could be able to introduce the various contexts and describe a family of the explicit representations of the Horner scheme (in other words, this family represents the *meta-programming domain*). For example, the predefined context is introduced by the following optional features that are *atomic features* (also called *variants*) (Fig. 4):

**Figure 4:** Feature diagram explaining the possible implementation of Horner scheme.

- \( y \) and \( u \) (parent <Left side function>);
- \#1...\#n (parent <Length>);
- *Constants, Array* (parent <Coefficients>);
- \( x, y, z \) (parent <Variable>).
Note that two relationships (case (A, B) and xor (A, B)) are equivalent if features A and B have the same father (e.g., \(<\text{Left side function}>\) or \(<\text{Coefficients}>\) in Fig. 4). In Fig. 4 one can see various types of relationships, including the functional relationship, which is an extension of the FD syntax for the LO domain. Here the erroneous knowledge (relationships between y (parent \(<\text{Left side function}>\)) and y (parent \(<\text{Variable}>\)) is excluded by the constraint using xor-relationship. Example 3 provides a subset of the family, which can be produced according to the selected variants of optional features of the given FD.

**Example 3:**

\[
y := ((a[0]*z + a[1])*z + a[2])*z + a[3])*z + a[4]; \quad \{ \text{variants selected: } y; n = 4; Array; \ z \}
\]

\[
u := ((a[0]*x + a[1])*x + a[2])*x + a[3]; \quad \{ \text{variants selected: } u; n = 3; \ Constant; \ x \}
\]

\[
u := (((a[0]*y + a[1])*y + a[2])*y + a[3])*y +a[4])*y +a[5]; \quad \{ \text{variants selected: } u; n = 5; \ Constant; \ y \}
\]

Now it is clear that Case study 2 (Fig. 4) also shows the role of variability of LOs. The implementation of variability using some technology (e.g., templates or meta-programming techniques) leads to generative LOs introduced first by T Boyle, D. Leeder & their colleagues (Boyle et al. 04, Morales et al. 05).

**Capabilities, Limitations and Problems of Using FDs in LO Domain**

First we evaluate the capabilities of FDs as a modeling language for the LO domain, then we outline limitations of using FDs in LOs domain, and describe how these limitations can be reduced or overcome. We evaluate FDs as a tool for specification, representation and structuring of learning content in the eLearning domain based on a set of general requirements formulated by (Koper and van Es in McGreal 04):

1. **Formalization.** FDs can be seen as a tool for formalization of LOs, though the syntax and semantics of FDs have not become standard yet (see limitations of LOs below).
2. **Pedagogical flexibility.** FDs allow modeling LOs based on different pedagogical theories.
3. **Explicit expression of meaning.** FDs allow explicitly specifying commonalties and variabilities of LOs.
4. **Completeness.** FDs is a complete specification system that can be used to describe all types of LOs, the relationships between different LOs, and the pedagogical activities related to LOs development and usage.
5. **Reproducibility.** The LO specification described using FDs can be used to reproduce learning content.
6. **Personalization.** FDs allow describing personalization aspects with LO as a part of variability management.
7. **Media neutrality.** FDs are independent of LO publishing formats such as web or e-books.
8. **Interoperability and sustainability.** FDs are independent upon implementation of LOs.
9. **Compatibility.** FDs do not contradict using the existing LO standards.
10. **Reusability.** FDs promote the reusability of LOs.
11. **Life cycle.** FDs can be used throughout the lifecycle of the LOs.

The limitations and restrictions of FDs are as follows:

- Although FDs are known for a long time since their syntax and semantics are constantly extending till now (Schobbens et al. 06). FDs are evolving towards domain ontologies (Czarnecki et al. 06).
- The process of creating FDs is time consuming. As the language is graphical, there are some limitations in manipulation and changing of FDs when they are moved into a virtual environment.
- A lack of maturity and experience of using FDs (except the PL development in SW engineering).
- A lack of expressiveness, e.g., the ability to describe domain ontology more comprehensively (in this, case FDs, can be combined with more powerful methods for knowledge representation (Robak & Pieczynski 03). eLearning community is not yet familiar with FDs at a large scale, although similar notation (UML–based) is already accepted (Laforcade 05).
- The dependence of FDs upon the context of using them in the LOs domain (contextualization problem).

**Contextualization problem.** FDs are created by human beings, thus the intent (context) of the interpreter of a given concept is important. In the PL development, the context of describing some concept using FDs is usually expressed implicitly, i.e., by default it is the managing of variability in a domain. In eLearning the context plays a much more important role because the context may be changed depending on the teaching goal. Thus the same concept can be represented using a slightly different shape of a FD. For example, without knowing the intent, there is not always
clear what type of a feature should be selected. Let us consider a FD depicted in Fig. 2. A sub-feature “Effectiveness” derived from the feature “Canonical form” has two attributes: #MULT and #ADD. The first is treated as a mandatory feature, while the second is treated as an optional feature. This judgment is valid only when \( n \) is large enough. Otherwise, the feature #ADD is not to be neglected and should also be identified as mandatory.

To resolve this problem, we suggest along with the basic concept to introduce the explicit statement of context. The intention of the context is to describe the scope of reuse. By changing the context of the concept we can manage the scope of variability explicitly. The context, for example, can be explicitly stated along with the basic concept at the root or apart of the root of the tree. The LO domain is very sensitive to learning context. We call this problem a contextualization of FDs. FDs resemble the ontology trees. FDs have a weaker expressiveness power than knowledge-rich approaches. However, FDs evolve towards richer ontologies and if the power for knowledge representation is yet not enough, FDs can be combined with the approaches that enhance the expressiveness.

Summary and Conclusions

Since Learning Objects (LOs) are entities that 1) contain a variety of attributes with complex relationships (e.g., within LO parts, and with metadata) 2) have to be considered from different contexts (e.g., design, retrieval, learning, etc.) and perspectives (e.g., designer’s, teacher’s, learner’s) LOs can be represented and modeled using the feature-based approach. The approach is based on using Feature Diagrams (FDs), the tree-like graphical language. The root represents a LO, intermediate nodes represent compound features and leaves of the tree represent atomic features. Branches describe various kinds of relationships between features. There are mandatory, optional and alternative features. Using FDs we can identify and model various kinds of ontologies related to LOs. As was shown in our Case studies, we can describe ontology of LOs definitions, ontology of the internal structure for representing the chunks of knowledge within a LO and variability aspects of a LO. We believe that the capabilities of FDs are not restricted by the analyzed Case studies.

FDs have also some limitations (not enough maturity, e.g., FDs syntax and semantics evolve till now, there is a lack of appropriate tools for drawing FDs). The LO domain requires some extension (such as contextualization, and more rich relationships to express various ontologies may be useful). Further research is needed in order to exploit benefits and overcome limitations of FDs. Our further work will be directed to connect the high-level specification of generative LOs with the implementation technology that allow generating on demand LOs instances derived from the LO generative specification.

References


MacGregor, J. (2002). Requirements engineering in industrial product lines. *International Workshop on Requirements Engineering for Product Lines, REPL’02, Essen, Germany. 5-11.*


