Using Ontologies in the Domain Analysis of Domain-Specific Languages

Robert Tairas[1], Marjan Mernik[2], Jeff Gray[1]

[1] University of Alabama at Birmingham, Birmingham, Alabama, USA
{tairasr,gray}@cis.uab.edu

[2] University of Maribor, Maribor, Slovenia
marjan.mernik@uni-mb.si

Abstract. The design stage of domain-specific language development, which includes domain analysis, has not received as much attention compared to the subsequent stage of language implementation. This paper investigates the use of ontology in domain analysis for the development of a domain-specific language. The standard process of ontology development is investigated as an aid to determine the pertinent information regarding the domain (e.g., the conceptualization of the domain and the common and variable elements of the domain) that should be modeled in a language for the domain. Our observations suggest that ontology assists in the initial phase of domain understanding and can be combined with further formal domain analysis methods during the development of a domain-specific language.

Keywords: Domain Analysis, Domain-Specific Languages, Ontology

1 Introduction

The development of a Domain-Specific Language (DSL) requires detailed knowledge of the domain in which the language is being targeted. Paradigms such as *Generative Programming* [3] and *Domain Engineering* [5] also require an understanding of the target domain, which is done through a process called domain analysis that produces a domain model. An important theme in the domain analysis used by both paradigms is the need to determine elements that can be reused. The reusable components or software artifacts form the building blocks for developing new software systems. In DSL development, in addition to the overall knowledge of the domain, the domain model can reveal important properties that will influence the way the language is shaped. In particular, the search for reusability in domain analysis can be translated into realizing the *commonalities* and *variabilities* of a domain. This information can assist in pointing out elements in the domain that can be fixed in the language and those that must provide for variabilities; hence, domain analysis has the potential to be beneficial if used during DSL development. However, clear guidelines for the use of established domain analysis techniques in the process of DSL development are still lacking [11].

Ontology development is one approach that has contributed to the early stages of domain analysis [5]. This paper investigates the use of ontology during domain analysis in DSL development and how it contributes to the design of the language. The rest of the paper is organized as follows: Section 2 describes the potential connection between ontology and DSL development. Section 3 provides a case study on the use of ontology in the development of a DSL for air traffic communication and Section 4 provides some observations on ontology in DSL development based on the case study. Related work, a conclusion, and future work are described in Sections 5 and 6.

2 Early Stage DSL Development

Chandrasekaran et al. [2] propose two properties related to ontologies: the first is a representation vocabulary of some specialized domain. This vocabulary represents the objects, concepts, and other entities concerning the domain. The second is the body of knowledge of the domain using this representative vocabulary. This knowledge can be obtained from the relationships of the entities that have been represented by the vocabulary. Ontologies seek to represent the elements of a domain through a vocabulary and relationships between these elements in order to provide some type of knowledge of the domain.

An interesting connection can be observed between ontology and DSL design. As it relates to DSL development [11], a domain model is defined as consisting of:

- a domain definition defining the scope of the domain,
- the domain terminology (vocabulary, ontology),
- descriptions of domain concepts, and
- feature models describing the commonalities and variabilities of domain concepts and their interdependencies.

Not only is an ontology useful in the obvious property of domain terminology, but the concepts of the domain and their interdependencies or relationships are also part of the properties of an ontology [2]. The knowledge of the commonalities and variabilities of the domain concepts can further provide crucial information needed to determine the fixed and variable parts of the language. This part is a more open question as to the potential of finding commonalities and variabilities through information obtained from the ontology.

As it relates to the DSL development process as a whole, the insertion of ontology development in the early stages of DSL development can potentially provide a structured mechanism in the part of DSL development that is still lacking attention. The early stages of DSL development (i.e., domain analysis) have not received as much attention compared to the latter stages of development (i.e., language implementation). Various DSL implementation techniques have been identified in [11], including interpreter or compiler development and embedding in a General-Purpose Language (GPL). In contrast, only four out of 39 DSLs evaluated in [11] utilized a more formal domain analysis, such as FAST [14] and FODA [8]. These formal approaches have shown to result in good language design, but their use is still

limited and it has yet to be seen how well they will be adopted by the community. The question is whether other less formal approaches, such as Object-Oriented Analysis (OOA) or ontology, can be reused in the early stages of DSL development. In order to promote interest in the domain analysis stage of DSL development, this paper advocates the use of ontology in DSL development, which is observed through a case study of a DSL for air traffic communication.

3 Case Study

Ontology development to assist in the design of a DSL is described through a case study in this section. Section 3.1 provides a summary of the air traffic communication problem domain. The ontology and its related competency questions are given in Sections 3.2 and 3.3. The development of a class diagram, object diagram, context-free grammar, and sample program related to the DSL and obtained from the ontology is given in Section 3.4.

3.1 Air Traffic Communication

A case study was selected to apply the ontology development process and observe its usefulness in domain analysis related to DSL development. The case study selected focuses on the communication that occurs between the air traffic control (ATC) at an airport and the pilot of an aircraft. More specifically, the communication is between the ground controller that is responsible for the traffic between the runways and the ramps containing gates in an airport, and the pilots of an aircraft that has just arrived or is in the process of departure. The purpose is to develop a DSL that can standardize the language for the communication between the two individuals. English is the standard language in this domain, but more often the controllers or pilots of non-English speaking countries may experience problems communicating in English. A DSL that standardizes the communication can be translated into the native tongue of the controller or pilot for better comprehension. A separate functionality could check and verify the path that is given to a pilot by a ground controller. An example communication sequence that highlights the potential communication problem is given in Listing 1. The controller is asking the captain to hold short of taxiway "MikeAlpha," but the pilot continually assumes it is taxiway "November."

Listing 1. Example of air traffic communication

```
  ATC:  Make the right turn here at Juliette. Join Alpha. Hold short
        MikeAlpha.
Pilot:  Right on Juliette hold sh ... Taxi Alpha. Hold November [...] Can
        we taxi now?
  ATC:  Make the right turn here at Juliette. Join Alpha. Hold short of
        MikeAlpha.
Pilot:  Roger, join right Juliette. Join Alpha. Hold short November.
  ATC:  OK, I'll say it again. Hold short of Mike Alpha "M" - "A"
        MikeAlpha, not November.
Pilot:  OK, hold short of MikeAlpha.
```

3.2 Ontology Development

Following the ontology development process outlined by Noy and McGuinness [13], competency questions are selected that serve as the purpose of the ontology. In order to obtain a domain model as defined in Section 2, two competency questions were selected: "What are the concepts of the domain and the interdependencies of these concepts?" and "What are the commonalities and variabilities of the domain?"

Both the Ontolingua[1] and DAML[2] ontology libraries were searched for existing ontologies related to the domain in this case study, but no related instances contained the vocabulary necessary for the domain. Although a new ontology is needed for this case study, the availability of an existing ontology in other cases provides a head start to the development of a domain model as the important terms and relationships have been determined already for the domain and can be used toward the subsequent steps of DSL development.

Table 1. Listing of classes and associated slots

Class	Description	Slots		
		Name	Description	Values
`Aircraft`	Arriving or departing aircraft	`Airline ID`	Name of the airline	Two letters
		`Flight Number`	Flight Identification	Integer
`GroundControl`	Controller in charge of airport ground traffic			
`Tower`	Controller in charge of take-offs and landings			
`Runway`	Available take-off and landing locations	`Runway Number`	Runway Identification	1 – 36 (i.e., runway heading 10° – 360°)
		`Runway Orientation`	To distinguish parallel runways	Class `Left` or `Right`
`Taxiway`	Paths connecting runways to ramps	`Taxiway Name`	Taxiway Identification	One or two letters (digits)
`Ramp`	Aircraft parking area	`Ramp Name`	Ramp Identification	String
`Gate`	Passenger embarkation and disembarkation	`Gate Letter`	Gate Identification	One letter
		`Gate Number`	Gate Identification	Integer
`Turn`	Command to turn	`Direction`	Turning direction	Class `Left` or `Right`
		`Taxiway`	Taxiway Identification	Class `Taxiway`
`HoldShort`	Command to hold short of a runway or taxiway	`Runway`	Runway Identification	Class `Runway`
		`Taxiway`	Taxiway Identification	Class `Taxiway`
`Contact`	Command to contact a separate controller	`ATC`	Controller to contact	Class `Tower` or `GroundControl`
`Follow`	Command to follow behind another aircraft	`Aircraft`	Aircraft Identification	Class `Aircraft`

[1] Ontolingua Ontology Library, http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html
[2] DAML Ontology Library, http://www.daml.org/ontologies

Utilizing the tool introduced by Noy and McGuinness [13] called Protégé 2000[3], the ontology for the case study was developed. The terms in Protégé 2000 are stored as classes. This allows for terms to be considered subclasses of other terms. In addition to classes, Protégé 2000 also contains slots and instances. Slots are the properties and constraints of the classes. Slots define the properties of classes and also determine the values that can be set for each property. Instances are actual instances of the classes in the ontology. These can be used to determine how well the ontology is representing a domain.

Table 1 contains a selection of classes and slots of the ontology that was developed in Protégé 2000 for the case study. In addition to the classes and slots in Table 1, instances of these classes were also determined. These instances are based on the information from a simplified diagram of the Birmingham International Airport (BHM) as shown in Figure 1. For example, instances of the `Runway` class are 6, 24, 18, and 36. Instances of the `Taxiway` class are A, B, F, G, H, M, A1, A2, A3, A4, B1, G1, H2, and H4. The `Ramp` class consists of Cargo and Terminal.

3.3 Competency Questions Revisited

The usefulness of the ontology in Table 1 can be measured by how well the ontology assists in answering the previously specified competency questions from Section 3.2. Regarding the first question, the ontology provides the concepts of the domain through the classes. The interdependencies between the concepts can be derived from the constraints of the slots of the classes. For example, the `HoldShort` class is dependent on either the `Runway` or `Taxiway` classes, as this command is always followed by the location in which the pilot is to hold short.

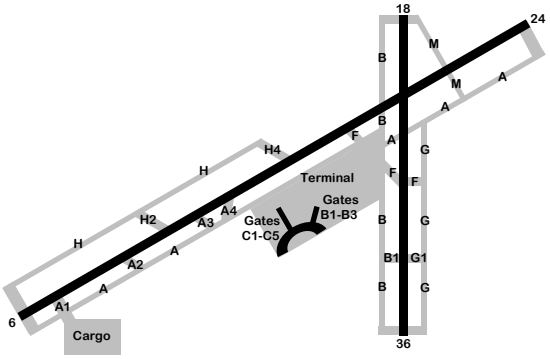

Fig. 1. Simplified Diagram of Birmingham International Airport (BHM)

Answering the second question related to commonalities and variabilities is less evident if observing only the ontology's structure of classes and slots. Information regarding the variabilities can be extracted by including the instances of classes, such

[3] Protégé 2000, http://protege.stanford.edu

as the instances from BHM. Classes `Runway` and `Taxiway` consist of many instances, which could mean these classes have the variabilities property. Moreover, instances that represent airports other than BHM will also contain different values for these classes, which could also be interpreted as containing variabilities. The classes not containing instances, such as most of the commands (i.e., `Turn`, `HoldShort`, and `Contact`), could be interpreted as common concepts in all instances. These commands are common in the ATC domain and represent standard commands that are used in all airports. However, the specific airport elements (i.e., collection of instances of runways and taxiways) may change depending on the airport.

3.4 Conceptual Class Diagram

The ontology process is similar to the process of object-oriented analysis [1]. However, one distinction is that ontology design is mainly concerned with the *structural* properties of a class, whereas object-oriented analysis is primarily concerned with the *operational* properties of a class [13]. The focus here is a methodology that can assist in determining the domain concepts for DSL development by reusing an approach from general software engineering.

Figure 2 presents a conceptual class diagram that was manually generated from the structural information of the classes in the ontology from Table 1. In this case, the development of the class diagram has been assisted by the information obtained from the ontology. In Figure 2, similar classes are grouped together. For example, classes `Gate`, `Ramp`, `Runway`, and `Taxiway` represent physical structures in the airport. Such groupings identified the need for a generalized class for each group. A generalized class was included in the diagram for `Runway` and `Taxiway`, because from the slot properties of class `HoldShort`, two possible values can be used (i.e., `Runway` and `Taxiway`). In the diagram, this is represented by abstract class `Way`. The classes at the bottom of the diagram represent communication commands. These are associated with other classes through their respective slot properties. Generalizations such as `Command` and `Way` were not part of the original ontology and were only introduced during the development of the class diagram. These classes in turn can be used to update the ontology to further improve the structure of the ontology. This can be seen as part of the process of iteratively refining the ontology to better represent the domain.

From the class diagram in Figure 2, an initial context-free grammar (CFG) for the DSL can be generated, as shown in Listing 2. This CFG was manually obtained from the conceptual class diagram to CFG transformation properties defined in [12]. Relationships such as *generalization* and *aggregation* in the class diagram are transformed into specific types of production rules in the CFG. For example, a generalization where classes `Runway` and `Taxiway` are based on class `Way` is transformed into the production rule `WAY ::= RUNWAY | TAXIWAY`. An aggregation where class `Gate` is part of class `Ramp` is transformed into the production rule `RAMP ::= GATES`. In this case the non-terminal `GATES` is used, because the cardinality of this aggregation is zero or more gates on a ramp (i.e., 0..*). An additional production rule is generated to represent this cardinality (i.e., `GATES ::= GATES GATE | ε`).

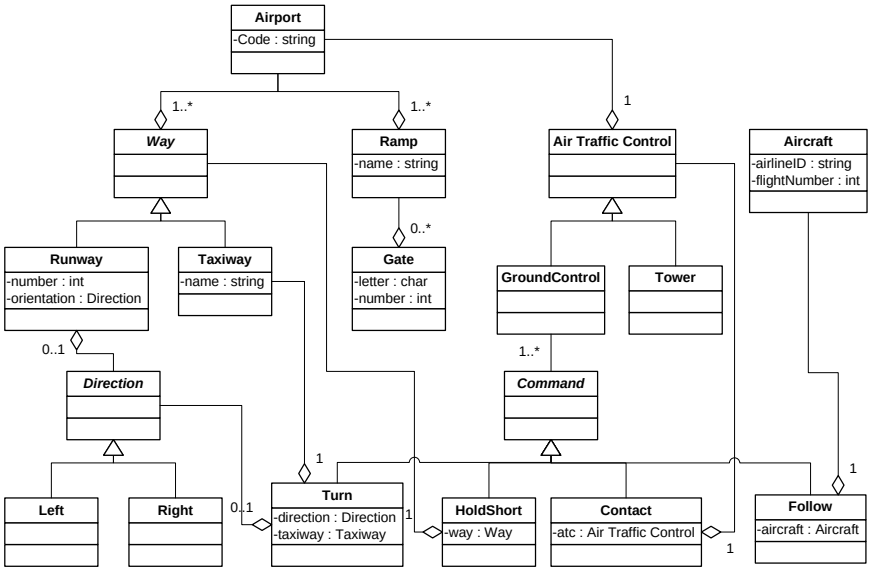

Fig. 2. Conceptual class diagram obtained from the ontology

Listing 2. Transformation of conceptual class diagram to context-free grammar

```
AIRPORT       ::= WAYS RAMPS ATC
WAYS          ::= WAYS WAY | WAY
WAY           ::= RUNWAY | TAXIWAY
RUNWAY        ::= number DIRECTION
TAXIWAY       ::= name
RAMPS         ::= RAMPS RAMP | RAMP
RAMP          ::= name GATES
GATES         ::= GATES GATE | ε
GATE          ::= letter number
ATC           ::= GROUNDCONTROL | TOWER
GROUNDCONTROL ::= COMMANDS
COMMANDS      ::= COMMANDS COMMAND | COMMAND
COMMAND       ::= CONTACT | FOLLOW | HOLDSHORT | TURN
CONTACT       ::= ATC
FOLLOW        ::= AIRCRAFT
HOLDSHORT     ::= WAY
TURN          ::= DIRECTION TAXIWAY
DIRECTION     ::= LEFT | RIGHT | ε
AIRCRAFT      ::= airlineID flightNumber
```

The transformation of the class diagram into the CFG above, albeit manual, followed a predefined collection of transformation rules. The manual transformation of the ontology into the class diagram is less formal, but was done by connecting the properties of the classes in the ontology with the graphical representation of the class diagram. In order to provide a more automated transformation between the ontology and the class diagram, developing a transformation between an Web Ontology Language (OWL) instance for the ontology and a textual representation of the class diagram could be considered. Related to this, UML-based ontology development has been proposed [6]. Specifically for this case, the transformation between an XML-based OWL file into a class diagram represented in XMI could assist in the

automation of the ontology to class diagram step. After the transformation to a CFG, some keywords have been added to the CFG for easier human parsing, as shown in Listing 3.

Listing 3. Addition of keywords and production refactoring

```
AIRPORT       ::= WAYS RAMPS ATC
WAYS          ::= WAYS WAY | WAY
WAY           ::= runway RUNWAY | taxiway TAXIWAY
RUNWAY        ::= number DIRECTION
TAXIWAY       ::= name
RAMPS         ::= RAMPS RAMP | RAMP
RAMP          ::= ramp name GATES
GATES         ::= GATES GATE | ε
GATE          ::= gate letter number
ATC           ::= GROUNDCONTROL | TOWER
GROUNDCONTROL ::= COMMANDS
COMMANDS      ::= COMMANDS COMMAND | COMMAND
COMMAND       ::= CONTACT | FOLLOW | HOLDSHORT | TURN
CONTACT       ::= contact ATC
FOLLOW        ::= follow AIRCRAFT
HOLDSHORT     ::= hold short WAY
TURN          ::= turn DIRECTION on TAXIWAY
DIRECTION     ::= left | right | ε
AIRCRAFT      ::= airlineID flightNumber
TOWER         ::= tower
```

An example of a program written in this DSL is shown in Listing 4 and is based on the CFG of Listing 3. Even from this simple DSL for ground control, it can be seen that some simple verification of aircraft path control can be checked. The development of the DSL has been aided by the ontology that was initially produced, which in turn assisted in the generation of a class diagram. This provided a means to understand the domain in the early stages of DSL development, which provided input to the subsequent structure of the DSL, as seen in the grammar in Listing 2.

Listing 4. An example program

```
// description of BHM airport
runway 6 runway 24 runway 18 runway 36
taxiway A taxiway A1 taxiway A2 taxiway A3 taxiway A4 taxiway B  taxiway B1
taxiway F taxiway G  taxiway G1 taxiway H  taxiway H2 taxiway H4
ramp Cargo
ramp Terminal gate B1 gate B2 gate B3 gate C1 gate C2 gate C3 gate C4 gate C5

// commands from Ground Control
turn right on A
turn left on M
hold short runway 18
contact tower
```

An object diagram of the example program in Listing 4 is illustrated in Figure 3. Airport-related structures such as gates, ramps, runways, and taxiways are represented by multiple objects that will differ among various airports. However, the types of commands issued by the ground control remain the same. The specific attributes of the command objects are based on the objects of the structures of a particular airport, e.g., taxiway A and M, and runway 18. As described in Section 3.3, evaluating the instances of the classes provides information regarding the elements of the domain that are common (or fixed) and those that are variable.

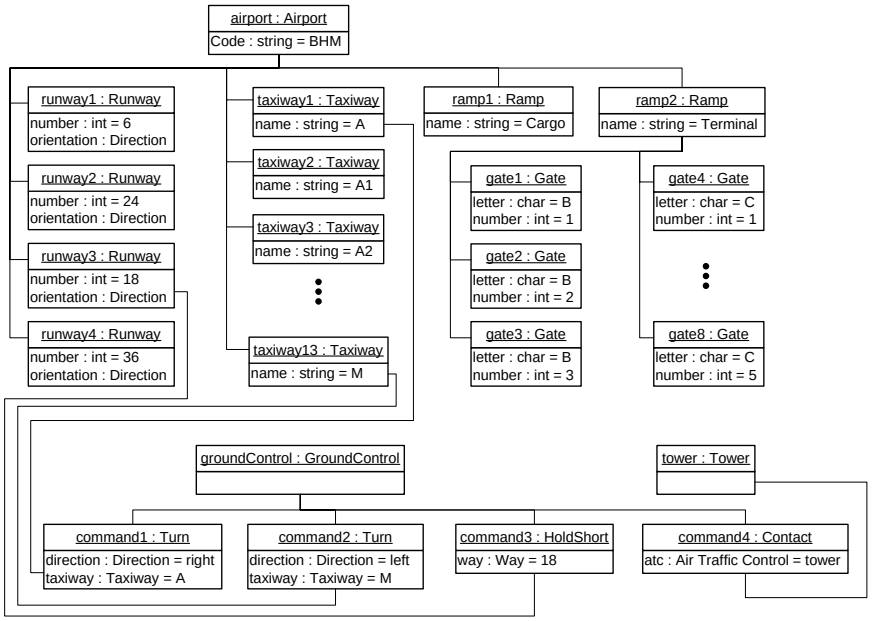

Fig. 3. Object diagram from example program

4 Ontologies in DSL Development

Section 3 summarized the development of a preliminary ontology using the standard development process as seen in literature using a well-known tool called Protégé 2000. The usefulness of the ontology was measured by answering several competency questions that were selected to match the goals of domain analysis. Domain concepts and their interdependencies were determined. In addition, commonalities and variabilities as they relate to the DSL can be determined by observing the instances of the classes in the ontology. It should be noted that the ontology and class diagram went through several iterations before reaching the state described in Section 3. However, further refinements may help to provide more satisfactory answers to the competency questions. The ontology was then used to manually generate a conceptual class diagram, which in turn produced an initial context-free grammar for the proposed DSL.

The case study has shown the potential usefulness of ontology in the development of a DSL specifically during the early stages of development. An ontology can provide a well-defined and structured process to determine the concepts of a domain and the commonalities and variabilities for a DSL, which can result in the generation of a class diagram and subsequently a CFG from the information. Two further observations highlight the benefits of an ontology-based approach. First, if an ontology is already available for a domain, then this existing ontology can be used to

initiate the development of a DSL without the need to start from scratch. This was not the case for the air traffic communication domain described in Section 3, but ontologies for other domains could already exist and be utilized in the DSL development for those domains. Second, the entire process outlined in Section 3 could be used as an alternative to a more formal domain analysis technique such as FODA. In a separate direction, the ontology alone can be combined with formal domain analysis techniques (e.g., proposed by Mauw et al. in [10]) and be used as a supplier of a well-defined input of domain concepts and relationships for further analysis.

5 Related Work

De Almeida Falbo et al. describe the use of ontology in domain engineering that has the purpose of developing software artifacts for *reuse* [5]. A more recent publication demonstrates the use of ontology in engineering design requirements capture [4]. Both cases propose methodologies of utilizing ontology in terms of providing the knowledge about a specific domain, albeit more in a general case of engineering. However, the utilization of ontology in domain analysis in these works translates well to the similar effort in DSL development. Guizzardi et al. associate ontology with the development of Domain-Specific Visual Languages (DSVL) [7]. The ontology is used to assist in developing a representative language for a specific domain that is correct and appropriate. Similarly, our initial investigation described in this paper utilizes ontology as part of the main goal of developing a representative language for a domain such as air traffic communication. However, in addition to this, the common and variable elements of the domain are also considered through the ontology in order to determine the structure of the domain-specific textual language (i.e., fixed and variable language constructs).

Gašević et al. describe efforts to associate the two technical spaces of Model-Driven Architecture (MDA) and ontology, which include the utilization of MDA-based UML in ontology development [6]. We follow a similar approach where a connection is made between the ontology in Table 1 and the UML class diagram in Figure 1. However, in addition to this association, we perform manual transformations on the class diagram to obtain a context-free grammar for the DSL.

6 Conclusion and Future Work

An initial investigation of the usefulness of ontology in domain analysis in DSL development was described in this paper. A case study demonstrated the insertion of ontology development in the DSL development process, where a class diagram was obtained from the ontology and subsequently a CFG was produced. The ontology assisted in answering questions related to the domain, such as the main concepts and their interdepencies, and the common and variable parts related to the DSL. The ontology also provided a structured input to the subsequent stages of DSL development. Continued exploration of ontology-driven domain analysis may provide further proof of effectiveness in the analysis of domains for DSL development.

The class diagram in Figure 2 that was generated from the ontology can also serve as the basis for creating a metamodel. Slight adaptations of this diagram could represent the metamodel for a tool like the Generic Modeling Environment (GME) [9], which provides a domain-specific modeling language that has a concrete syntax that resembles concepts from the domain. Thus, the results of the domain analysis and the observed ontology can inform technologies of both grammarware and modelware. This direction will be explored as future work. In addition, the transformations that were performed were done manually based on predefined transformation properties. A possibility for a more automated step is the transformation of the Web Ontology Language (OWL) representation into a Backus-Naur Form (BNF) representation for the DSL. Such a transformation may map similar elements and perform some alterations between the representations. This direction will also be considered in future work.

Acknowledgments. This project is supported in part by NSF grant CPA-0702764.

References

[1] Booch, G.: Object-Oriented Development. IEEE Transactions on Software Engineering 12, 211--221 (1986)

[2] Chandrasekaran, B., Josephson, J., Benjamins, V.: What Are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems 14, 20--26 (1999)

[3] Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston (2000)

[4] Darlington, M., Culley, S.: Investigating Ontology Development for Engineering Design Support. Advanced Engineering Informatics 22, 112--134 (2008)

[5] De Almeida Falbo, R., Guizzardi, G., Duarte, K.: An Ontological Approach to Domain Engineering. In: International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 351--358, Ischia, Italy (2002)

[6] Gašević, D., Djurić, D., Devedžić, V.: *Model Driven Architecture and Ontology Development*. Springer, Berlin (2006)

[7] Guizzardi, G., Ferreira Pires, L., van Sinderen, M.: Ontology-Based Evaluation and Design of Domain-Specific Visual Modeling Languages. In: International Conference on Information Systems Development, Karlstad, Sweden (2005)

[8] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)

[9] Lédeczi, Á., Bakay, Á., Maróti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: Composing Domain-Specific Design Environments. IEEE Computer 34, 44--51 (2001)

[10] Mauw, S., Wiersma, W., Willemse, T.: Language-Driven System Design. International Journal of Software Engineering and Knowledge Engineering 14, 625--664 (2004)

[11] Mernik, M., Heering, J., Sloane, A.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37, 316--344 (2005)

[12] Mernik, M., Črepinšek, M., Kosar, T., Rebernak, D., Žumer, V.: Grammar-Based Systems: Definition and Examples. Informatica 28, 245--255 (2004)

[13] Noy, N., McGuinness, D.: Ontology Development 101: A Guide to Creating Your First Ontology. http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf.

[14] Weiss, D., Lay, C.: *Software Product Line Engineering*. Addison-Wesley, Boston (1999)