On the role of Domain Ontologies in the design of Domain-Specific Visual Modeling Languages

Giancarlo Guizzardi, Luis Ferreira Pires, Marten J. van Sinderen

Centre for Telematics and Information Technology,
University of Twente
P.O. Box 217, 7500 AE, Enschede,
The Netherlands
{guizzard, pires}@cs.utwente.nl,
sinderen@ctit.utwente.nl

Abstract: Domain-Specific Visual Modeling Languages should provide notations and abstractions that suitably support problem solving in well-defined application domains. From their user's perspective, the language's modeling primitives must be intuitive and expressive enough in capturing all intended aspects of domain conceptualizations. Over the years formal and explicit representations of domain conceptualizations have been developed as domain ontologies. In this paper, we show how the design of these languages can benefit from conceptual tools developed by the ontology engineering community.

1. Introduction

In recent years, an increasing attention has been paid to the development of domain-specific modeling languages. It is believed that these languages can lead to an increase in productivity in the modeling activity and contribute to the production of models that are more flexible, reusable and easier to maintain than models produced by using general-purpose modeling languages.

Notwithstanding, in order to be effective, a domain specific modeling language must be defined taking into account the needs of its client users. From their perspective, the use of the language should be intuitive and satisfactory in the following terms:

(i) The semantics of the produced models should be clear, i.e., it should be easy for a model designer to recognize what language constructs mean in terms of domain concepts;

(ii) The language should be sufficiently expressive to represent all domain concepts that should be captured by the intended models.

For these reasons, there is a demand for techniques that support the construction of explicit models of domain conceptualizations. Additionally, there is a need for concrete and precise guidelines for selecting which domain concepts should be represented as language constructs and how.

Formal and explicit specifications of domain concepts have been developed for many years as domain ontologies, in application areas, such as, mechanical engineering, medicine, aeronautics, ceramic materials, tourism, agriculture, software process, media on demand management, telecommunications, steel factoring, port logistics, software quality, among many others. Moreover, the ontology engineering community has developed a significant body of knowledge w.r.t the systematic development, reuse and integration of domain conceptualizations. Currently, the proliferation of formal ontologies is growing faster due to the special attention dedicated to the W3C Semantic Web initiative.

The question this paper aims to address is the following: How can we use available formal domain ontologies as a starting point for the design of domain-specific visual languages? In section 2, we present the different elements of languages design, namely, syntax, semantics and pragmatics. In section 3, we discuss the subject of formal ontologies and its relation to each of these elements. In section 4, we outline the main steps for an ontology-based methodology for the design domain-specific visual modeling languages. Section 5 presents some conclusions and points for further investigation.

2. Elements of Language Design

One of the main success factors for a domain specific language is its ability to provide to its user a set of modeling primitives that correspond to relevant domain abstractions. Domain abstractions are constructed in terms of concepts, i.e., abstract representations of certain aspects of entities that exist in a given domain. An abstraction of a real world entity expressed in terms of a set of domain concepts is termed *architecture* in our work [5,27]. An architecture may comprise structural but also behavioral aspects of the modeled entities.

Architectures have to be documented, communicated and analyzed. This implies that a language is necessary for representing them in a concise, complete and unambiguous way. Figure 1 depicts the distinction between architecture and its representation, and their relationship with the domain conceptualization and the representation language. Architectures are abstract conceptual entities that must be expressed in a representation language to generate concrete artifacts (representations). In the scope of this article these representations are called models and the representation languages used on their creation are called modeling languages.

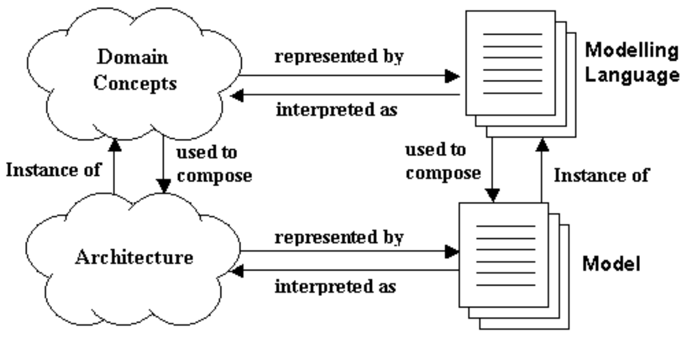


Figure 1- Relation between domain conceptualization, architecture, modeling language and model

According to Morris [1], a language comprises three parts: syntax, semantics and pragmatics. Syntax is devoted to ‘the formal relation of signs to one another’, semantics to ‘the relation of signs to real world entities they represent’ and pragmatics to ‘the relation of signs to (human) interpreters’. In the following subsections we elaborate upon these three definitions.

2.1 - Syntax

Harel et al. [2] discusses the distinction between the puristic notion of information and its syntactical representation as data, which is the medium used to communicate and store information. Data is essential to give a concrete and persistent status to some information, but it is in itself, however, vacuous in terms of meaning. Thus, to extract the information behind a piece of data, it is necessary an interpretation that assigns meaning to it. The same piece of information may be represented as different data (e.g., "December 31st, 2002" and "the last day of the year 2002" refer to the same entity). Likewise, the same piece of data may represent different things for different people at different points in time. The difference between information and data resembles the difference between a language syntax and semantics.

In order to communicate, agents must agree on a common communication language. This fixes the sets of signs that can be exchanged (syntax) and how these signs can be combined in order to form valid expressions in the language (syntactical rules).

In sentential languages, the syntax is first defined in terms of an alphabet (set of characters) that can be grouped into valid sequences forming words. This is called a lexical layer and it is typically defined using regular expressions. Words can be grouped into sentences according to precisely defined rules defined in a context-free grammar, resulting in an abstract syntax tree. Finally, these sentences are constrained by given context conditions.

In diagrammatic (graphical) languages, conversely, the syntax is not defined in terms of linear sequence of characters but in terms of pictorial signs. The set of available graphic modeling primitives forms the lexical layer and the language abstract syntax is typically defined in terms of a meta-model (case of OMG's MDA) or an abstract visual graph [3]. Finally, the language meta-model is enriched by context-conditions given in some constraint description language, such as, OCL or first-order logic (FOL). In either case, context conditions are intended to constrain the language syntax by defining the set of correct (well-formed) sentences of the language. Some of these constraints are motivated by semantic considerations (laws of the domain being modeled) while others are motivated by pragmatic issues (discussed in section 3). Nevertheless, a meta-model is a description of the language's abstract syntax since it defines: (i) a set of constructs pragmatically selected for the purpose of performing a specific (set of) task(s) and, (ii) a set of well-formedness rules for combining these constructs in order to create valid models in the language.

2.2 – Semantics

Besides agreeing on a common vocabulary, participants in a communication process must share the same meaning for the syntactical constructs being communicated, i.e., they must interpret in a compatible way the expressions of the communication language being used. Without a proper way to extract information, the data being exchanged is worthless. Semantics deals with the meaning of signs and symbols or, in other words, with the information behind a piece of data.

In general terms, a semantic definition for a language $\mathcal{L}$ consists of two parts: (i) a semantic domain S; (ii) a semantic mapping from the syntax to the semantic domain, formally $\mathcal{M}: \mathcal{L} \rightarrow S$ [2]. The semantic mapping tells us about the meaning of each of the language's expressions in terms of an element of the semantic domain.

The relation between the semantic domain and the domain conceptualization (in Figure 1) can be interpreted according to two different stances to semantics. In AI research, semantics denote "some form of correspondence specified between a surrogate and its intended referent in the world; the correspondence is the semantics for the representation" [4]. In other words, semantics is a mapping (interpretation) from the language vocabulary to concepts that represent entities in the real world. Conversely, approaches with heritage in other traditions of programming consider semantics without a commitment to a specific real-world conceptualization. In these approaches, the term “semantics” is used to denote rules for program compilation or automated interpretation. According to these approaches, the semantic domain is a mathematical or formal domain not necessarily related to a real-world conceptualization. In the AI approach, the semantic domain is a material domain, such as, engineering, business, medicine or telematics. In [5], these are called formal and architectural semantics, respectively. From now on, we use the term *domain conceptualization* when referring to a real-world conceptualization and the term formal conceptualization otherwise.

In [5], it is argued that designers should concentrate on the elaboration of models, using the modeling language merely as a vehicle for the representation of design characteristics. A modeling language can only be a useful general purpose language if its vocabulary, syntax, semantic and pragmatics, are defined based on the needs of those who are supposed to use the language in the elaboration of models. This view, supported here, emphasizes the precedence of real-world concepts over formal concepts in the design of a domain-specific modeling language or, in other words, the precedence of architectural semantics over formal semantics.

In section 3 we discuss the relevance of concrete, formal and explicit representations of domain conceptualizations for domain-specific visual languages design. In the context of this work, a formal and explicit specification of a domain conceptualization is called a domain ontology.

2.3 –Pragmatics

While the abstract syntax defines the rules for the creation of well-formed sentences of a given language, the vocabulary, or concrete syntax, provides a concrete representational system for expressing the elements of the domain conceptualization.

A representation may be considered to be a collection of objects and some relations between these objects. The type (visual or otherwise) of a particular representation, and more generally of a language, is determined by the characteristics of the ‘symbols’ used to express these objects and relations [7]. Throughout this discussion we use the term symbol in a broad sense. Thus, our symbol definition includes visual features (e.g., morphological, geometric, spatial and topological relations) used by diagrammatic languages to represent objects and relations.

In sentential languages, there is a clear separation between vocabulary, syntax and semantics. The syntactic rules which permit construction of sentences, may be completely independent of the chosen vocabulary, and may be clearly distinguished from a definition of semantics. For example, let P be a propositional logic whose vocabulary consists of the propositions p and q, and the symbols $\wedge$ and $\neg$ representing the logical connectives ‘and’ and ‘not’, respectively. The syntactic and semantic rules for P tell us, respectively, how to construct and interpret formulas using this vocabulary. However, we may substitute the symbols $\{p,q, \wedge,\neg\}$ for $\{X,Y,\&,\sim\}$ throughout P to produce a logic which is effectively equivalent to P. Alternatively, we could retain the vocabulary and syntax of P, while altering the semantics to produce a vastly different logic [7].

The same does not hold for graphical languages, in which vocabulary, syntax and semantics are not clearly separable. For example, a graphical vocabulary may include shapes such as circles, squares, arcs and arrows, all of differing sizes and colors. These objects often fall naturally into a hierarchical typing which almost certainly constrains the syntax and, furthermore, informs the semantics of the system.

Likewise, spatial representing relations, such as inclusion, are part of the vocabulary but clearly constrain the construction of potential diagrams and are likely to be mapped onto semantic relations with similar logical properties [7]. This idea is illustrated by Figure 2 below, in which two different languages are used to express logical syllogisms. The sentential language of (Fig.2-a) and the graphical language of (Fig.2-b) are semantically equivalent. Despite of that, the inference step that culminates with conclusion (iii) is performed in a much more straightforward way in the language of Euler's circles (fig.2–b). This classic example shows how semantic information can be directly captured in a visual symbol. Here a sequence of valid operations is performed which cause some consequence to become manifest in a diagram, where that consequence is not explicitly insisted upon by the operations. This is due to the fact that the partial order properties of the set inclusion relation are expressed via the similarly transitive, irreflexive and asymmetric visual properties of proper spatial inclusion in the plane, i.e. the representing relation has the same semantic properties of the represented relation. This argument is given a formal account by [8] where immediate inferences like in (fig.2-b) are termed "free-rides".

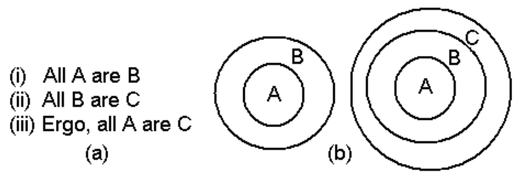
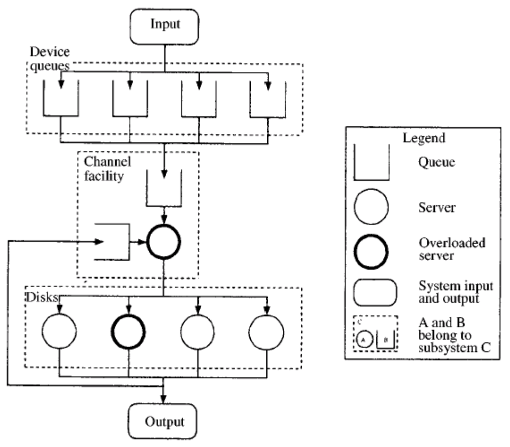


Figure 2- Logical Syllogism represented in a sentential language (a) and in the visual language of Euler's Circles (b)

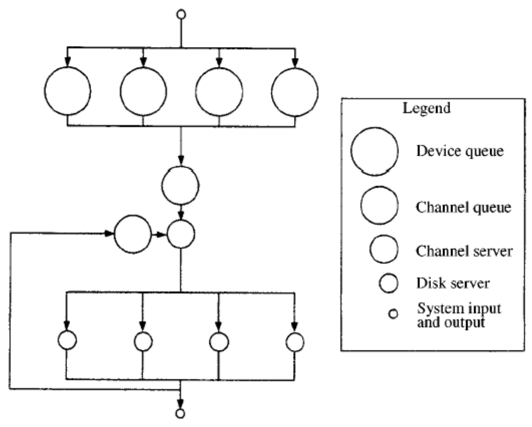
In sentential languages the relationships between symbols are necessarily captured in terms of the concatenation relation, which must then be interpreted by some intermediate abstract syntax [7]. In visual languages, intrinsic properties of the representation system can be systematically used to directly correspond to properties in the represented domain. Gurr [7] use the term *directness* to denote the correspondence between properties of representations and that of which they represent and, the term *systematicity* to denote the systematic application of directness in the design of concrete syntaxes.

When correctly employed, systematicity can result in major increases in the effectiveness of the diagram for performing specific tasks. For instance, Hahn et al. [9] shows how three semantically equivalent languages, namely, UML's activity, sequence and collaboration diagrams, can differ drastically in terms of effectiveness when employed for model integration. In contrast, whenever misused or neglected, directness can communicate incorrect information and induce the user to make incorrect inferences about the semantics of the domain.

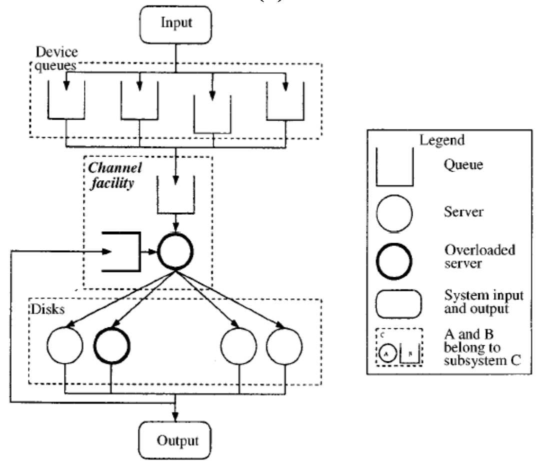
To illustrate this idea, we show in figure 3 three versions of the same diagram. This example is cited by [7] and has been taken from [10]. Figure 3-a represents a computer's disk subsystem. The visual concrete syntax also expresses relations and intrinsic properties of represented domain entities via either icons or other visual features such as the use of spatial inclusion within a box drawn with a dotted line denoting membership of a subsystem. In fig.3-b, the inherent ordering of graphical symbols used to represent the nodes can be incorrectly interpreted that all nodes in the represented network fall into a single conceptual category, in which they are similarly ordered. In fig.3-c we can find several examples of directly inferred incorrect information: (i) one of the device queues in the upper section of the diagram has been laid out irregularly which can be interpreted that this queue must be different than the others somehow; (ii) in the lower section of the diagram the disk-symbols are organized as two separated groups, wrongly implicating that this division reflects some grouping in the domain; (iii) the line-width used for the lower queue in the channel facility differs from the one used for all other queues in the diagram; (iv) a different font has been used for the channel facility's text label, implying that it must have different subsystem status.



(a)



(b)



(c)

Figure 3- Different diagrams representing the same architecture (a computer's disk subsystem).

All three diagrams of Figure 3 faithfully portray all the features of the assumed network model, i.e., diagrams (b) and (c) cannot be considered misleading through purely semantic considerations. The problematic implications of these diagrams do not follow from the incorrect use of the symbols but rather by various aspects of its semantics which are not explicitly specified, but that can be misleading [7].

While examples of Figures 3-b and 3-c illustrate the potential traps of ignoring pragmatic aspects of diagrams, other research has shown that the efficacy of diagrams for communicating information can be increased by the correct usage of such aspects. Studies by Petre and Green [11] of engineers using CAD systems for designing computer circuits, demonstrated that the most significant difference between novices and experts is in the use of layout to capture domain information. In such circuit diagrams, the layout of components is not specified as being semantically significant. Nevertheless, experienced designers exploit layout to carry important information, e.g. by grouping together components that are functionally related. By contrast, certain diagrams produced by novices were considered poor because they either failed to use layout or, in particularly ‘awful’ examples, were especially confusing through the misuse of common layout conventions informally adopted by experienced engineers [7].

In [7] and [12], Gurr presents a framework to formally evaluate the relation between the properties of a representation system and the properties of the domain entities they represent. According to Gurr, visual representations are more or less effective depending on the level of isomorphism between the algebras used to model the *representing* and the *represented* world. To put in different terms, let M_D and M_S represent the meta-models of the domain and the language’s concrete syntax, respectively. If M_D and M_S are isomorphic, the implication for the human agent who interprets the diagram is that their interpretation correlates precisely and uniquely with the world being represented. By contrast, when the correlation is not an isomorphism then there may potentially be a number of different target worlds that would all match the interpretation.

Despite of the major contributions introduced by this framework, we claim that many additional benefits would arise from a more suitable representation of the domain conceptualization than the algebra used in [7,12]. In this work we argue that the more we know about a domain the better we can design pragmatically effective languages. As we show in the next section, there are important meta-properties of domain entities that are not captured by ontologically-neutral mathematical languages. The failure to consider these properties hinders the possibility of accounting for other direct aspects of visual syntaxes.

3. Ontologies and Domain Conceptualizations

The Webster dictionary defines the term "ontology" as “a particular theory about the nature of being and the kinds of existents”. In the philosophical tradition, ontology refers to a particular system of categories that seeks to study and structure a certain portion of reality. Since Aristotle’s theory of substance (objects, things and persons) and accidents (qualities, events and process), they have been used to create domain theories. In Computer Science, however, ontologies are accounted as engineering artifacts whose purpose is to make explicit the semantic distinctions existent in a domain conceptualization, i.e. “a formal and explicit specification of a conceptualization [6]”.

Hayes [13] introduced the use of ontologies in Computer Science (more specifically in Artificial Intelligence). Since then, they have been employed in areas such as conceptual and information modeling, information integration, software engineering, database theory, computational linguistics and knowledge representation and engineering. In addition to that, ontologies have been developed in material domains such as medicine, e-commerce, product catalogue integration, mechanical engineering, knowledge management and geo-information systems, among several other examples [14].

More recently, a discipline called *Ontological Engineering* [15] has emerged, with the purpose of creating languages, methodologies and techniques to systematically develop domain conceptualizations as well as to reuse and integrate existing ontologies. In addition to that, the Ontological Engineering discipline congregates a spectrum of techniques that have been developed in areas such philosophical logic, cognitive science, naïve metaphysics and linguistics for many years. We claim that conceptual modeling for the purpose of language design can greatly benefit from these techniques.

A scenario where ontologies play an important role is when comparing models produced by different languages that share the same domain conceptualization. Suppose we have two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ that share the same conceptualization C_D. Assume the two languages have been developed to be used in different problem-solving tasks and intend to model different (aspects of) entities of the domain conceptualization C_D. As a consequence, the languages have different syntaxes and semantics. One question that arises is: how to relate (compare, translate) models written in $\mathcal{L}_1$ and $\mathcal{L}_2$?

Since they have different syntaxes and semantics, $\mathcal{L}_1$ and $\mathcal{L}_2$ should not be compared only in syntactical terms. Languages can have equal syntaxes with radically different semantics as much as they can have different syntaxes with equivalent mappings to a common semantic domain. For the case of automatic translation, the formal semantics of $\mathcal{L}_1$ and $\mathcal{L}_2$ cannot guarantee consistency in the results either. For example, if C_D refers to a domain of family relations, the expression `(x related-to y)` in $\mathcal{L}_1$ can be wrongly translated to an expression `brother(a,b)` in $\mathcal{L}_2$, since the *related-to* relation satisfies all the formal axioms of brotherhood (irreflexive, symmetric, transitive).

Semantic comparability is a challenge in enterprise engineering due to the lack of interoperability among different (yet coexistent) process models. Many manufacturing engineering and business software applications use process information, including manufacturing simulation, production scheduling, manufacturing process planning, workflow, business process reengineering, product realization process modeling, and project management. Since each of these applications utilizes process information in a different way, process information is also represented differently in each application. This problem is even more critical than it looks like, since enterprise systems must manage the heterogeneity inherent to its various sub-domains, by integrating different models into coherent frameworks (e.g., enterprise models for processes, structure, goals, deontic assignments).

This problem has been currently addressed by the NIST PSL (Process Specification Language) Project[1] and the community working on the Semantic Web [16]. For instance, Ciocoiu and Nau [17] provide a formal definition of ontology-based semantics for enabling automated model translation between different first-order declarative languages. In [18], this approach is exemplified via the translation between process models written in ILOG and IDEF 3. This idea is illustrated in Figure 4. The solution creates logical interpretations π_1 and π_2 that relate the formal semantics of $\mathcal{L}_1$ and $\mathcal{L}_2$ to a common domain ontology O_D. The assumption here is that O_D is general enough to subsume the parts of C_D targeted by $\mathcal{L}_1$ and $\mathcal{L}_2$. More than that, O_D is a formal logic theory defining the semantics of the conceptualization shared by $\mathcal{L}_1$ and $\mathcal{L}_2$. Unlike the meta-models of $\mathcal{L}_1$ and $\mathcal{L}_2$, O_D is not biased towards a selection of concepts in C_D for the solution of specific tasks. It is a specification of domain-knowledge perceived as important by a community in order to address the needs of possibly several different applications.

Since a domain ontology is also a concrete artifact, it must be represented in some specification language $\mathcal{L}_O$. To be consistent with the position defended here, the ontology representation language (ORL) $\mathcal{L}_O$ should also be based in a domain conceptualization, in this case, a meta-conceptualization.

[1] http://ats.nist.gov/psl/

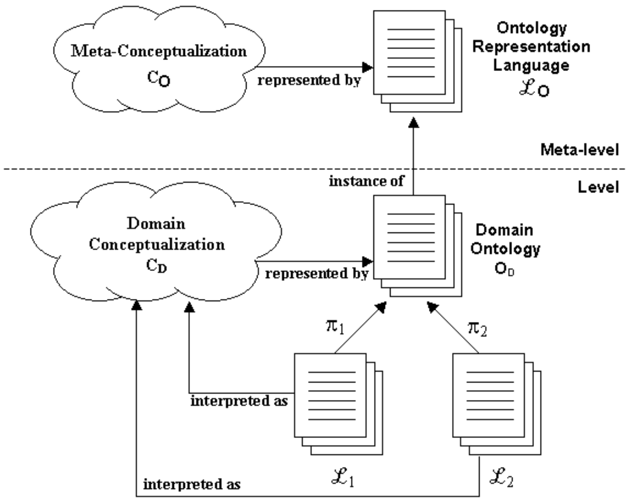


Figure 4- Defining the semantics of two DSL via a shared domain ontology

In the PSL approach aforementioned, the language used as $\mathcal{L}_O$ is FOL (first-order logic). Although, it is sufficient for the purpose of automatic translation and machine processing, FOL is neutral in terms of a meta-conceptualization. In a logical specification, we are only concerned with the predicates necessary to represent the concepts of our domain and with evaluating the truth of this predicates for certain individuals. For instance, suppose we want to state that a red apple exists, we would write down something like $\exists x\ (apple(x) \wedge red(x))$.

If we want to impose a certain structure to our domain representation we could write $\exists x{:}apple.red(x)$ as well as $\exists x{:}red.apple(x)$. In other words, we can consider that there are instances of apples that can posses the property of being red or, or there are instances of red things that can have the property of being apples. Both these many-sorted logic formalizations are equivalent to the previous one-sorted axiom, but each one contains an implicit structuring choice.

According to Strawson, the difference between the two predicates lies in the fact that *apple* "supplies a principle for distinguishing and counting individual particulars which it collects", while *red* "supplies such principle only for particulars already distinguished, or distinguishable, in accordance with some antecedent principle or method"[20]. Informally we can say that the instances of apple have an identity characterized by an identifying property (e.g., spatial-temporal location). The same is not true for instances of red things, since we cannot differentiate red things only by their property of being red. From another perspective, we can say that things such as instances of color (weight, height, tune and so on) do not have independent material existences, i.e., they only exist on the material entities, which are their bearers.

The meta-properties of sortality (ability to carry or supply an identity) and existential dependence (ability to exist independently of other entities) are well known in the philosophical literature and are (informally) reflected in the meta-models of languages popularly used in conceptual modeling, such as, UML and E-R. For instance, in UML, sortal/independent universals can be represented by classes (Person, Cow, Airplane), sortal/dependent by roles (Employee, Student, Professor) while non-sortal universals are represented by attributes (name, age, color).

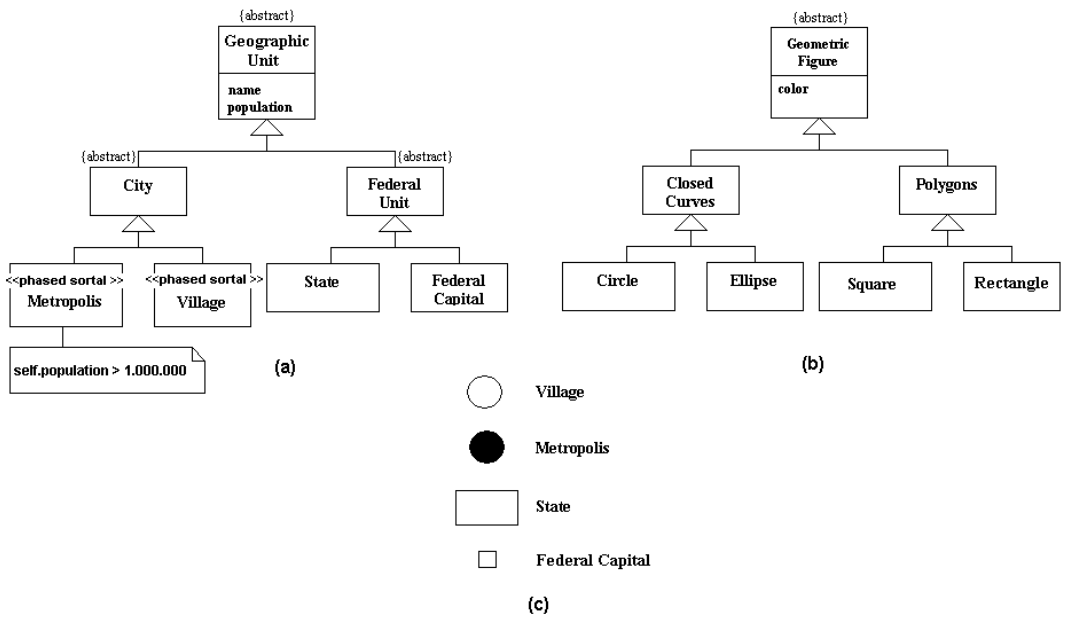


Figure 6- Defining a concrete syntax for a domain representation

Imposing a structure to the domain representation is necessary when this representation is meant to be used for communication, reasoning and problem-solving tasks involving humans. For instance, the process of developing an ontology involves the participation of people with different backgrounds and requires a structured and, preferably, visual communication language to be used between domain experts and ontology engineers. Moreover, the explicit representation of conceptual categories involving domain concepts is also necessary for the development of pragmatically efficient concrete syntaxes. Figure 6 presents an example of a domain representation (a) and a concrete syntax for its conceptual entities (c). The homomorphism between the representation system (b) and that which it represents (a) is exploited in order to built directness into the language's concrete syntax (c). There are other things that should be noticed: (i) only concrete classes in (a) are represented as symbols; (ii) Whilst the subclasses of *Federal Unit* are represented by two different symbols, the "subclasses" of City are represented by variations of a property (color) of the same symbol. This design decision can be motivated by observing another meta-property, namely rigidity. The intention of a rigid concept applies to all its instances in every possible world. In other words, an instance of a rigid concept cannot cease to be an instance of this concept without loosing its identity. Examples of rigid concepts are *Person*, *Dog*, and *Planet* while examples of anti-rigidity are *Student*, *Pet*, *Husband*, and *Butterfly*. Some anti-rigid concepts can only exist in the scope of a relation (e.g. student, husband) while others denote phases in the existent of an entity, called here phased-sortals [21]. These concepts are typically called *role* and *state* by the OO community, respectively [22].

The ontological meta-properties of the modeling primitives of $\mathcal{L}_O$ are outside the scope of this paper, as well as definition of guidelines for the representation of these primitives in terms of a concrete syntax. The objectives aimed by this section are: (i) to argue that an ontology representation/conceptual modeling language should provide concrete and precise means for deciding how domain entities should be modeled; (ii) to show that the failure to do so will have impact on the communicability and reasonability of the domain models and on the pragmatics of derived domain-specific visual languages.

For a complete formalization of the meta-properties of sortality, dependence, rigidity and unity, please refer to [21]. For an extension of UML, based on an ontologically well-founded meta-conceptualization please refer to [23,24].

4. From a domain ontology to a Domain-Specific Visual Language

Section 3 has shown how the semantics of different domain-specific languages $\mathcal{L}_1$ and $\mathcal{L}_2$ could be defined in terms of a shared domain ontology. This situation, exemplified in Figure 4, could be formally defined as follows: let C_{D1} and C_{D2} be subparts of the conceptualization C_D whose concepts define the architectural semantics of the constructs of languages $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. Now let, O_{D1} and O_{D2} be subparts of O_D that represent conceptualizations C_{D1} and C_{D2}, respectively. Finally, let $\oplus_1$ and $\oplus_2$ be two operations such that when applied to O_D they generate O_{D1} and O_{D2}, respectively. We claim that, taking a proper domain ontology O_D as starting point, it is possible to conceive operations $\oplus_i$ such that the ontology O_{Di} generated from O_D is a suitable starting points for the design of a domain-specific language $\mathcal{L}_i$.

To investigate the nature of $\oplus$ we must define the characteristics of the product this operation is supposed to generate. The meta-model M of a language must be able to: (*i*) identify which elements of the conceptualization must be made part of the incorporated in the syntax of the language; (*ii*) define the rules between syntactical elements that prescribe a criteria for the specification of valid statements in that language, i.e., well-formedness rules.

In order to select the concepts that will be part of a language's abstract syntax one must consider the purpose of the models that will be created using the language, what kind of tasks users of the language intend to perform, what answers a user expects to extract for a model using the language. One way to determine the elements of the models that are necessary to allow the users to perform their tasks is to represent the cognitive steps that the user will probably take towards the construction of the model. In principle, models used for Human-Computer interface design (e.g. GOMS [25]), task-ontologies [26], and design algebras for the selection of concepts in the design of domain frameworks [27] are strong candidates.

In [28], we apply the approach introduced here on the development of a simple domain-specific visual language $\mathcal{L}$ from an ontology O_{family} of family relations. In this case study, we formally defined an algorithm α in terms of set theory, such that α represents possible cognitive steps taken by users of the language in order to build the intended models. Moreover, the intermediate solutions sets generated by the execution of α highlight which elements of the domain ontology are necessary in the model construction process. The meta-model resulting from this activity correctly defines the abstract syntax of the language, whose elements are taken from O_{family}, and serves as a suitable reference for the derivation of a homomorphic model for the language's concrete syntax. Additionally, the axioms of O_{family} that refer to the elements present in the abstract syntax are also added to the language's meta-model. This axiomatization represents the language's well-formed rules and its logic-based semantics.

The remainder of this section outlines the main steps of a methodology for ontology-based domain-specific modeling language design:

1. **Select or Develop a Domain Ontology:** We initially assume here that there exist a suitable formal ontology representing the conceptualization of the domain we are interested. In many situations, this is not the case. For instance, a shared-ontology can be developed to serve as means for integration of different applications models in the same domain, e.g., integration of different applications dealing with manufacturing process [18]. There is a significant amount of knowledge about how to engineer (model, formalize, reuse, integrate, etc.) ontologies. For an ontological engineering methodology we refer to [29].

2. **Extract from the Domain Ontology the language's initial meta-model:** In order to do this, one must define a procedure ⊕ for selecting the concepts (and corresponding axioms) of the domain ontology that are relevant to produce the intended models in the language.

3. **Refine the language's meta-model:** If the meta-model produced in step 2 lacks sufficient axioms to define the language's semantics, the designer should provide logical interpretations [17] for its abstract syntax, thus defining the language's semantics in terms of the axiomatization of the original ontology.

4. **Develop the language's concrete syntax:** Once the meta-model is stable, it can be used to design a pragmatically efficient representational system to be used as the language's concrete syntax. Pragmatic efficiency is increased when a representation exhibit the same properties of the abstract thing it represents. For that reason, the meta-model should explicitly represent the axioms constraining domain relations (e.g., irreflexivity, asymmetry, transitivity) as well as the meta-properties of domain concepts (dependence, rigidity, sortality). Since the meta-model is extracted from the domain ontology provided in step 1, this ontology should, thus, be modeled in suitable representation language. The language should make explicit the commitments made by the ontology designer towards a meta-conceptualization, i.e., it should be clear to which of the ontological categories the ontology concepts belong (e.g., classes, roles, attributes, phased-sortals, etc.).

5. Final Considerations

In this paper we argue that formal domain ontologies are suitable as starting point for the design of domain-specific visual languages. We present the different elements of languages design and discuss the relation between ontologies and each of these elements. Additionally, we motivate the need for an ontologically well-justified representation language for the purpose of ontology specification and language conceptual meta-modeling.

A number of formal ontologies have been developed during the years, in several important application areas. Motivated by the Semantic Web, we have seen a fast growth on the number of implemented ontologies as well as a diversification of their application domains. We demonstrate that ontology engineering can make important contributions to the area of domain-specific visual modeling languages by providing: (i) a sound vocabulary and semantics for these languages; (ii) a precise way to compare (evaluate, extend) different languages in the same domain; (iii) a precise account for designing and evaluating the pragmatic effectiveness of language concrete syntaxes.

In particular, we are interested in the development of appropriate concepts and languages for modeling structural and behavioral aspects of business process and telematics systems.

In section 4, we outline the main steps for an ontology-based methodology for the design domain-specific visual modeling languages. Some of these steps must be further elaborated. For example, we intend to future investigate alternative techniques for extracting a language's initial meta-model from a domain ontology (some of these techniques are briefly mentioned in section 4). Additionally, we aim at defining precise guidelines for the design and evaluation of concrete syntaxes for domain-specific visual languages.

References

1. Morris, C.W. Foundations of a theory of signs. In: International Encyclopedia of Unified Science (O. Neurath, R. Carnap & C. Morris, eds), Chicago University Press, Chicago, pp. 77-138. 1938.
2. Harel, D.; Rumpe, B. Modeling Languages: Syntax, Semantics and All That Stuff
   technical paper number MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, 2000.
3. Erwig, M. Visual Graphs, 15th IEEE Symposium on Visual Languages, 122-129, 1999.
4. Davis, R., Shrobe, H. & Szolovits, P. 1993. What Is a Knowledge Representation? AI Magazine, Spring 1993.
5. Ferreira Pires, L. Architectural Notes: A Framework for Distributed Systems Development, PhD Thesis, Centre for Telematics and Information Technology, University of Twente, The Netherlands, 1994.
6. Gruber, T. R. Toward Principles for the Design of Ontologies used for Knowledge Sharing. Int. J. Human-Computer Studies, v. 43, n. 5/6, 1995.
7. Gurr, C.A. Effective Diagrammatic Communication: Syntatic, Semantic and Pragmatic Issues, Journal of Visual Languages and Computing, 10, 317-342, 1999.
8. Shimojima, A. Operational constraints in diagrammatic reasoning. In: Logical Reasoning with Diagrams. Oxford University Press, New York, pp. 27-48. 1996.
9. Hahn, J.; Kim, J. Why are some Diagrams Easier to Work With?: Effects of Diagrammatic Representation on the Cognitive Integration Process of Systems Analysis and Design. In: ACM Transactions of Computer-Human Interaction, 6/3, Sept.1999.
10. Marks, J.; Reiter, E. Avoiding unwanted conversational implicature in text and graphics. In: Proceedings of AAAI-90, pp. 450-456. 1990.
11. Petre, M.; Green, T.R.G. Requirements of graphical notations for professional users: electronics CAD systems as a case study. Le Travail Humain 55, 47-70. 1992.
12. Gurr, C.A. On the isomorphism, or lack of it, of representations. In: Theory of Visual Languages, Chap. 10 (K. Marriot & B. Meyer, eds), Springer, Berlin. 1998.

13. Hayes P. The Naive Physics Manifesto, Expert Systems in Microeletronics age", D. Ritchie Ed., Edinburgh University Press, 1978, pp 242-270.
14. Uschold, M.; Gruninger, M. *Ontologies: Principles, Methods and Applications,* Knowledge Engineering Review; Volume 11 Number 2, June 1996
15. Devedzic, V. Undestanding Ontological Engineering, Communications of the ACM, 45/4, pp.136-144, April, 2002.
16. Euzenat, J., Towards a principled approach to semantic interoperability. In: Proceedings of the IJCAI-01, Workshop on Ontologies and Information Sharing,
    Seattle, USA, August 4-5, 2001.
17. Ciocoiu, M., Nau D., Ontology-Based Semantics. In: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2000), Breckenbridge, Colorado, April 12-17, 2000.
18. Ciocoiu, M.; Gruninger, M. Ontologies for Integrating Engineering Applications, Journal of Computing and Information Science in Engineering 1(1): 12-22, 2000.
19. Lutters, E.; Manufacturing Integration based on Information Mangement, PhD Thesis, University of Twente, The Netherlands, 2001.
20. Strawson, P. F. Individuals. An Essay in Descriptive Metaphysics. London and New York: Routledge, 1959.
21. Guarino, N.; Welty, C. A Formal Ontology of Properties, In: Proceedings of 12th Int. Conf. on Knowledge Engineering, and Knowledge Management, Lecture Notes on Computer Science, Springer-Verlag, 2000.
22. Odell, J.Advanced Object-Oriented Analysis and Design using UML, Cambridge University Press, 1998.
23. Guizzardi, G.; Herre, H.; Wagner G.. "Towards Ontological Foundations for UML Conceptual Models", 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02), Lecture Notes on Computer Science, Springer-Verlag, 2002.
24. Guizzardi, G.; Herre, H.; Wagner G.. "On the General Ontological Foundations of Conceptual Modeling", 21st International Conference on Conceptual Modeling (ER-2002), Lecture Notes on Computer Science, Springer-Verlag, 2002.
25. John, B. E.; Kieras, D. E. The GOMS family of analysis techniques: Tools for design and evaluation. Carnegie Mellon University School of Computer Science Technical Report No. CMU-CS-94-181, 1994.
26. B. Chandrasekaran, J. R. Josephson and V. R. Benjamins, Ontology of Tasks and Methods, Banff Knowledge Acquisition Workshop. 1998.
27. B. Tekinerdogan, Synthesis Based Software Architecture Design, Ph.D. thesis, University of Twente, The Netherlands, March 2000.
28. Guizzardi, G.; Ferreira Pires, L.; van Sinderen, M.J.; An ontology-based approach for the design of Domain-Specific Visual Languages, Centre for Telematics and Information Technology Technical Report, University of Twente, The Netherlands, 2002.
29. Fernández, M.; Gomez-Perez, A. Pazos, A.; Pazos, J. Building a Chemical Ontology using METHONTOLOGY and the Ontology Design Environment IEEE Expert: Special Issue on Uses of Ontologies.January/February 1999. PP: 37:46