Business-Specific Languages for Organizational Modeling

João Paulo Pedro Mendes de Sousa Saraiva [1], Alberto Manuel Rodrigues da Silva [2]

1) INESC-ID/Instituto Superior Técnico, Lisboa, Portugal

joao.saraiva@inesc-id.pt

2) INESC-ID/Instituto Superior Técnico, Lisboa, Portugal
alberto.silva@acm.org

Abstract

Nowadays, there are many languages for business modeling, such as UML, BPMN and EPBE, which are well suited to describe, in a generic way, business views that stakeholders consider relevant. However, these languages don't take advantage of the concepts, business-specific, identified during the Information Architecture design. In business user-models, those concepts are usually only present as element names, and thus the modeling tool does not support the semantics associated with those business concepts.

Software Engineering has been facing a similar problem for quite some time, as traditional modeling languages are oriented towards "how the solution is implemented", instead of "what the solution is". DSLs address this issue by raising the abstraction level at which developers operate: instead of handling implementation-level concepts, developers handle concepts from the problem-domain. This could also be used in Organizational Engineering, promoting a better alignment between business models and business concepts.

In this paper, we discuss how information obtained during Information Architecture design can be combined with traditional modeling languages, in order to obtain Business-Specific Languages (BSLs), languages that are better adjusted to the specific business.

Keywords: Organization, Information Architecture, Business-Specific Language.

1. Introduction

There are currently several business modeling languages used for business modeling, such as the Unified Modeling Language (UML) Activity diagrams [UML], the Business Process Modeling Notation (BPMN) [BPMN], the Erikson-Penker Business Extensions (EPBE) [Penker 2000] or the CEO Framework [CEO]; these languages are suitable to describe, in a generic way, business views that stakeholders consider relevant. However, the only semantics that these languages support is the semantics defined in their corresponding metamodel (the language that specifies how users – the business stakeholders – can model their interpretations of the business); semantics associated with the business itself are not supported by the language, and the constraints derived from those semantics are usually specified as *business rules* [BRG]. On the other hand, business entities themselves are usually included in business user-models only as "names", and it is up to the business modeler to assure that the model does not contradict any of the semantics associated with those business entities. However, these business entities and related semantics have already been obtained during the design of the Information Architecture, but they are not actively used when modeling the various business perspectives. This would

make it seem like the defined Information Architecture is not an active first-class citizen during business modeling, but a "weak glue" used only to assist in maintaining the alignment between the Business Architecture and the Application Architecture.

Although it is very important for an organization to have well-defined business processes, a common flaw observed in organizations is in the treatment of *information* itself. Processes are often defined without having a *clear notion of the business information* that these processes manipulate. This is because processes are often treated as first-class citizens and the business information is treated as "something" underlying a process, something that should not be considered outside the context of a single process. A solution to this problem consists in the previous definition of the business information (i.e. the definition of all the concepts with which the organization will deal), so that it can be considered a first-class citizen. Information and processes should be considered as views of an organization, orthogonal and complementary among themselves. Basing systems on information instead of processes leads to a greater stability of the organization and its systems: in the short/mid term, it is much more likely that processes will be changed, and not the information itself.

The Information Architecture [Sousa et al. 2005] is one of the four fundamental components for alignment between Business and IT. It describes the business information that the organization needs to know in order to run its processes and operations, and provides a view of the business information independent of the technological aspects. In the Information Architecture, business information is structured in elements called *Information Entities*, which correspond to the concepts on which the business is based (throughout this paper we will use the term "entities" to refer to Information Entities); the business is responsible for managing and performing operations over entities. Typical examples of these entities could be "Client", "Employee" or "Supplier". An entity must have an identifier, a description and a set of attributes. Attributes are handled by business processes and by applications. An attribute could be the "competences" of an Employee, or the "number of orders" made by a Client.

Although [Sousa et al. 2005] does not explicitly mention the semantics that involve these business entities, we believe that the Information Architecture should also provide an explicit specification of these semantics, as they could be used to continuously assure the integrity of the information available to the business. In short, the task of specifying the Information Architecture should be very similar to the Software Engineering task of specifying the *domain model*.

The Software Engineering industry has been facing a similar problem for quite some time, as traditional modeling languages are oriented towards "how the solution is implemented", instead of "what the solution is". Domain-Specific Modeling is a recent approach that addresses this issue by attempting to raise the abstraction level at which software developers operate: instead of handling implementation-level concepts, developers handle concepts from the problem-domain itself [Kelly 2004]. This attempt at raising the abstraction level is based on the concept of a Domain-Specific Language (DSL) [DSMForum] [MartinFowler], which is a language designed to be used for a specific task (or a set of tasks) in a certain problem-domain, unlike a general-purpose language.

The aim of this paper is to illustrate how we believe that this "DSL line of thought" could also be used in Organizational Engineering (OE), in order to create (or adapt) languages adjusted to a specific business. These languages would promote a better alignment between the business models and the business entities themselves.

This paper is structured as follows. Section 2 describes some important modeling concepts, as well as DSLs and their OE correspondent: Business-Specific Languages (BSLs). Section 3 analyzes how the information obtained during Information Architecture design could be combined with traditional business-modeling languages in order to obtain BSLs. Section 4 concludes the main ideas of this research work.

2. Adapting the language to the context

This section analyzes the relationship between models and metamodels in the context of business modeling, and the emergence of DSLs. These concepts are then extended into OE.

2.1 Models and metamodels

A *model* is an interpretation of a certain problem-domain – a fragment of the real world over which modeling and system development tasks are focused – according to a determined structure of concepts [Silva 2005]. This structure of concepts is defined by a *metamodel*, which is an attempt at describing the world around us for a particular purpose, through the precise definition of the constructs and rules needed for creating models.

It is also important to distinguish the syntax and the semantics of a metamodel. A metamodel's *syntax* consists of the set of *model elements* (graphical representations of *domain elements*) and the relations between those model elements. This is very similar to the definition of "syntax" in the context of linguistics, in which syntax is the study of the way words are combined together to form sentences. A metamodel's *semantics* can be seen from two perspectives: (1) the semantic domain and (2) the semantics of each model element. The semantic domain consists of the whole set of domain elements that the metamodel is supposed to represent (i.e. the concepts that were "captured" during the analysis of the problem-domain). On the other hand, the semantics of a certain model element is determined by the relationship(s) between that model element and one or more domain elements.

2.2 Current business modeling languages

Current languages provide their own semantic, adequate to business modelling, such as the traditional rule that business processes are triggered by specific events, have specific goals, consumes resources and/or information and produces output. However, these languages do not allow the user (i.e. the business modeler) to perform the previous specification of the semantics inherent to the specific business itself. This would allow the configuration of the modeling tool used so that it could only create models that are valid and complete in the context of the business.

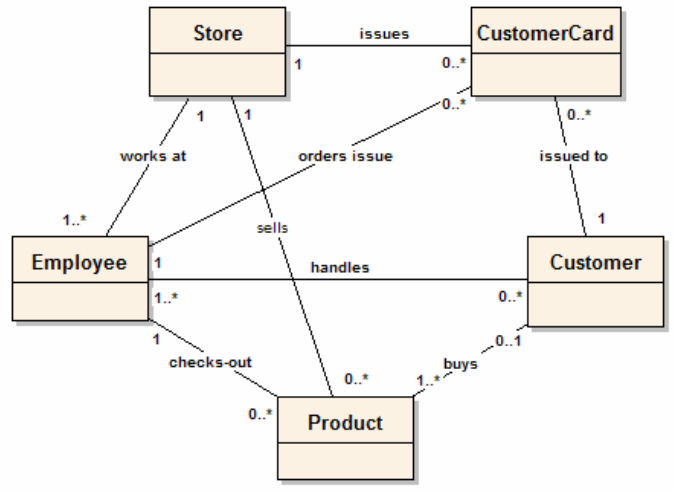

Figure 1 – A simplified view of the business entities of a “grocery store” business.

For example, in the simplified context of a specific "market" business (represented in Figure 1), consider that a Store can issue a Customer Card to a Customer only if that Customer does not already have that Card.

Such semantic specifications must be done either: (1) in the model (as illustrated in Figure 2), by introducing the *choices* necessary to handle all possible states of the manipulated entities; or (2) apart from the visual model itself, by specifying *business rules* (which the modeling tool may or may not support).

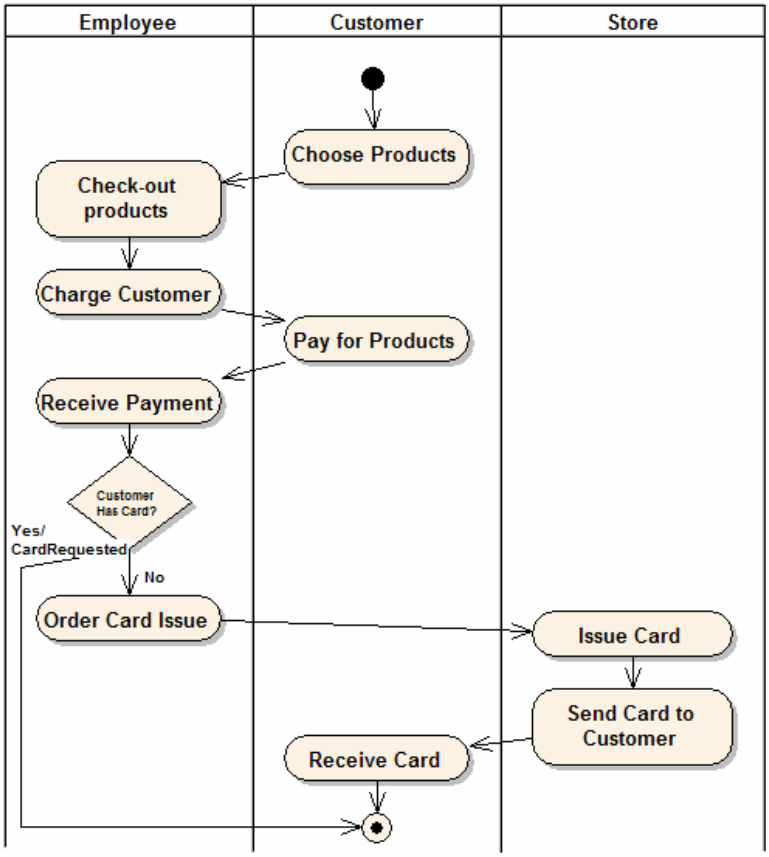

Figure 2 – A simplified UML representation of the "Issue Customer Card" business process.

The problem with the first alternative (handling all possible states directly in the model) is related to the complexity of the entities of the business. Although in businesses with simple entities, this alternative would not complicate the activity of modeling the business, modeling businesses with more complex entities will probably originate situations in which business processes do not contemplate all possible states of these entities; if a business "domain model" is too complex, a business modeler might forget to handle some of those states. This would make business modeling become a "trial-and-error" task, which is clearly counter-productive and would most likely be harmful to the business. The problem with the second alternative (using business rules to assure the complete validation of the model) is that, unless the modeling tool actively supports the input of business rules and the continuous validation of the mode based on those rules, business modeling also becomes a "trial-by-error" task.

These languages are focused entirely on workflows (or business processes) [White 2004] and the specification of "what *actions* the organization *executes*" and "what *information* those actions *manipulate*", without considering the necessary cohesion and the semantics in all the business information that is manipulated. Although business stakeholders often believe a

language for specifying business processes is sufficient, the lack of business-specific information in modeling languages can lead to subtle inconsistencies in modeled business processes, which can lead to disastrous consequences. Nevertheless, the necessary business information is already determined in the Information Architecture, in the form of business entities, attributes and their semantics. However, since the metamodels that define the languages for business modeling are fixed and cannot be altered, this information cannot be incorporated into those languages, and thus user-models cannot be instantly validated from the perspective of the business information.

Software Engineering and Model-Driven Engineering (MDE) [Schmidt 2006] have been facing a very similar problem for a very long time: traditional modeling languages are oriented towards "how the solution is *implemented*" rather than "what the solution *is*". The Domain-Specific Language approach (presented in the next subsection) addresses this issue by raising the abstraction level at which software developers operate: instead of handling implementation-level concepts, developers handle concepts from the problem-domain itself. Today, the topic of Domain-Specific Languages (DSLs) is almost synonymous with its visual complement, Domain-Specific Modeling (DSM). Unlike traditional modeling tools, which had a hard-coded modeling language and restricted users to create their models only within the parameters defined by that language, DSM adds an abstraction layer which enables the configuration of the modeling language with which users will create their models. Note that UML itself, as well as any traditional modeling language, can be seen as a DSL (or a set of DSLs).

2.3 Business-Specific Languages in Organizational Engineering

If we follow the "DSL line of thought" and adapt it to OE, we get "Business-Specific Languages" (BSLs): languages that would be tailored to a specific business domain (for example, a grocery store). Their syntax would be very similar to the syntax of current modeling languages (as most of those languages are standards, and business stakeholders are used to them); nevertheless, the semantics associated with these languages would be very different between them.

The advantages of these languages over traditional modeling languages would include: (1) semantics much better adjusted to the business itself; (2) generated implementations would already include those business semantics; and (3) business models could be continuously validated, while assuring the consistency of the business entities. These validations could be of multiple types, such as: (1) assuring that all possible entity states are handled; and (2) assuring the consistency of operations conducted by business processes on entity. These languages do have the disadvantage of requiring some definition effort, both in syntax and semantics; however, this effort could be considerably shortened by using current modeling languages as *templates* and/or capturing and applying *patterns* of business modeling.

As an example of the first validation type, consider Figure 2 and the choice on whether the customer has a customer card. This choice would normally depend on an attribute "Has Customer Card" of the "Customer" entity that could assume the values "Yes" or "No"; so, it is clear that this choice covers all possible relevant states of the Customer entity. Now, suppose that we alter that attribute so that it could assume another value, "Card Requested"; any business process choices that depend on that attribute would need to be updated to reflect this new value. Although in a simple business model this update would be trivial, in the case of a more complex business model (with many business processes, in which this choice would occur frequently) it would be extremely easy to overlook one of those choices and not update it. If the tool supported this BSL (with the specification of entities and their attributes), it could detect that a certain choice did not consider all possible values of that attribute and alert the user.

3. Specifying a BSL

Just as important as defining the purpose of BSLs is defining the constitution of BSLs and the process of obtaining a BSL that can be used to model a specific business. In this section, we analyze BSLs in terms of their constituents, and describe how to obtain a BSL from a current business modeling language, which could make BSLs more attractive to business stakeholders.

3.1 Components of a BSL

In its essence, a BSL is a metamodel, and therefore it is defined in terms of syntax and semantics, as Figure 3 illustrates.

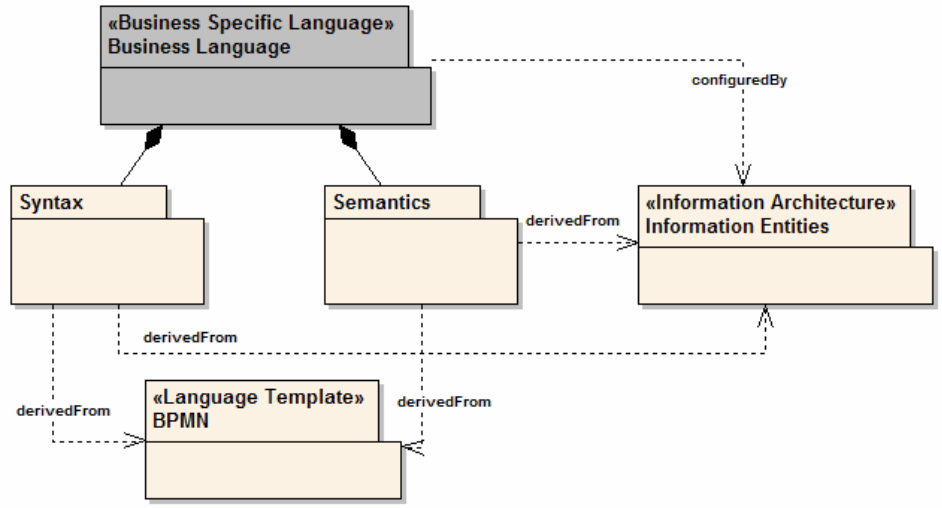

Figure 3 – Components of a BSL, obtained from a modeling language and the Information Architecture.

However, the BSL must cover at least two orthogonal business perspectives, the *information* perspective and the *business process* perspective. This means that, for the syntax specification of a BSL to be complete, it must define the graphical representations of not only the business entities and attributes, but also the graphical representations of every concept necessary to specify business processes and their internal details. And, like the syntax specification, the semantic specification should consider the business entities and attributes and the business processes.

Many of the concepts that are part of a BSL (such as “business process”, “activity”, “event”, etc.) are also shared by many other businesses; in fact, those are usually the concepts that generic business modeling languages focus on. Thus, we can think of those languages as the result of applying “refactoring” techniques on BSLs (i.e. gathering the concepts that are common to BSLs and creating a new language with those concepts). It would be a great advantage to the task of specifying BSLs if the definition of a BSL could be based on such modeling languages, as the next subsection shows.

3.2 Obtaining a BSL from a generic business modeling language

Although the usage of BSLs could be advantageous to business stakeholders, the fact is that the definition of a BSL will probably require some effort, because it involves the definition of both the syntax of the relevant model elements and the semantics of the corresponding domain elements (just like the definition of a typical metamodel). Since this factor could be determinant in preventing most business stakeholders from adopting BSLs, it is necessary to determine how current business modeling languages can be reused in BSLs, so that: (1) the stakeholder’s

transition from a generic language to a BSL can be as easy and straightforward as possible; and (2) the BSL can be defined in a much faster way.

We believe that, from a very simplistic point of view, a BSL can be considered as a combination between a generic language and the business entities defined in the Information Architecture. The IA would provide the business entities and their associated semantics, and the generic language would provide the syntax and some of the semantics that could be applied to the business. Also, current business modeling languages could be used, in the context of BSLs, as *language templates*. These templates would supply most of the syntax (the graphical representations of concepts) of the BSL, as well as some of the generic business semantics. This usage of *language templates* is justified by the fact that current modeling languages already provide concepts that are common throughout any business, such as "business process", "activity", and "event"; since these concepts are commonly used in all business models, a BSL creator would find BSLs a waste of time and effort if he/she was forced to define all those concepts (and their semantics) every time a BSL was created. Nevertheless, the syntax relating to business entities and their attributes should not provided by language templates, as they define no special representation for such elements; instead, the syntax relating to business entities and attributes should be defined manually (as it is specific to the business domain) and the associated semantics should be "copied" from the Information Architecture to the BSL. Therefore, a BSL could be obtained by performing the following steps: (1) making a "copy" of a language template; (2) defining the syntax for the business entities; (3) making a "copy" of the semantics associated with the business entities; and (4) make any adjustments necessary to create a perfect fit between the BSL and the business. Figure 3 illustrates this combination of language templates and business entities; note that, although the template language in the figure represented is BPMN, it could easily be UML or any other business modeling language. Also, the link named "configuredBy" (between the BSL and the Information Architecture) and indicates that the BSL can be considered as "a copy of the language template that is configured (or adapted) to the business information structure" (provided in the Information Architecture).

So, if we consider again the two perspectives of a business model (the information perspective and the business process perspective), we can see that language templates would provide the syntax and semantics necessary for the business process perspective, and the Information Architecture would provide the syntax and the semantics for the information perspective. Nevertheless, the BSL creator should *always* be able to add/change/remove the BSL's syntax and semantics, to create the best possible adjustment between the BSL and the business itself.

3.3 Tight integration between business information and business processes

In order for BSLs to achieve their full potential, they should possess some features that cannot be obtained by "simply" combining generic business modeling languages and business entities. One of these features should be the tight integration between the business process perspective and the information perspective. This integration would require that activities specify the operations they perform on business entities and their attributes, so that the modeling tool could perform consistency checks. Defining those operations would require a taxonomy for all possible operations, such as "access", "set", or "query" (illustrated in Figure 4 and Figure 5); those operations would then be classified, by using a tag called "type". Obviously, only some types of operations would be available to each type of element of a business process.

Figure 4 illustrates a very simple example of a BSL, based on the UML, with some modifications to the "Customer – Customer Card" situation presented in the previous section (the complete specification of this particular BSL is outside of the scope of this paper). This example presents some issues: (1) the manipulated business elements (the "Customer" entity and its "Has Customer Card" attribute) are represented in this model, in the "Information" box; (2) the "Customer has Card?" choice now indicates that it *queries* the "Has Customer Card"

attribute (through the “type = query” text); and (3) some of the process’ activities can perform operations (such as “set”) on that attribute. The types of operations performed on the “Has Customer Card” attribute can help tremendously in the validation of the model. For example, take the first operation (“type = query”) and the corresponding choice (the diamond shape); with this indication, the tool could verify whether the options considered covered all the possible values that the attribute could assume (in this case, it does), and warn the user if a possible value had been overlooked.

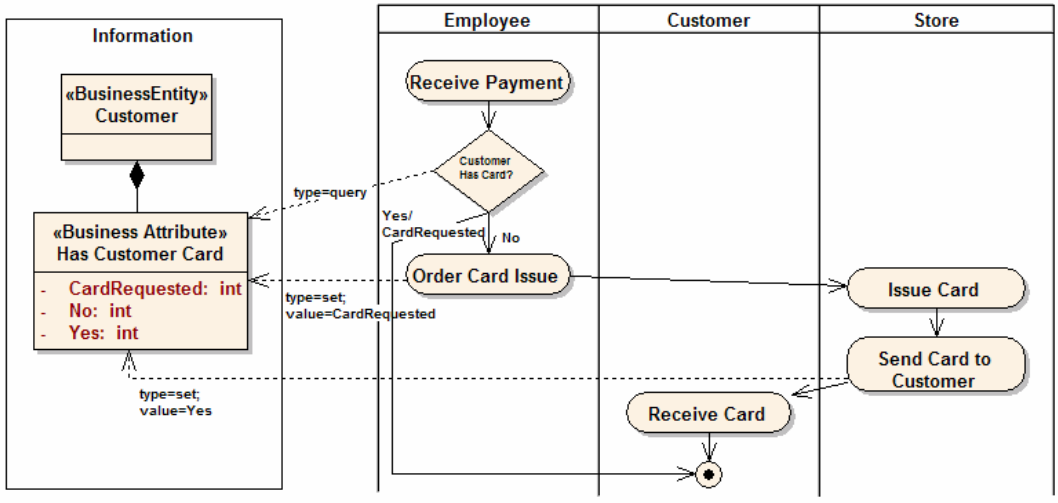

Figure 4 – An example of using business entities in cooperation with a business process.

Finally, Figure 5 illustrates the “access”, “set” and “query” operations, respectively.

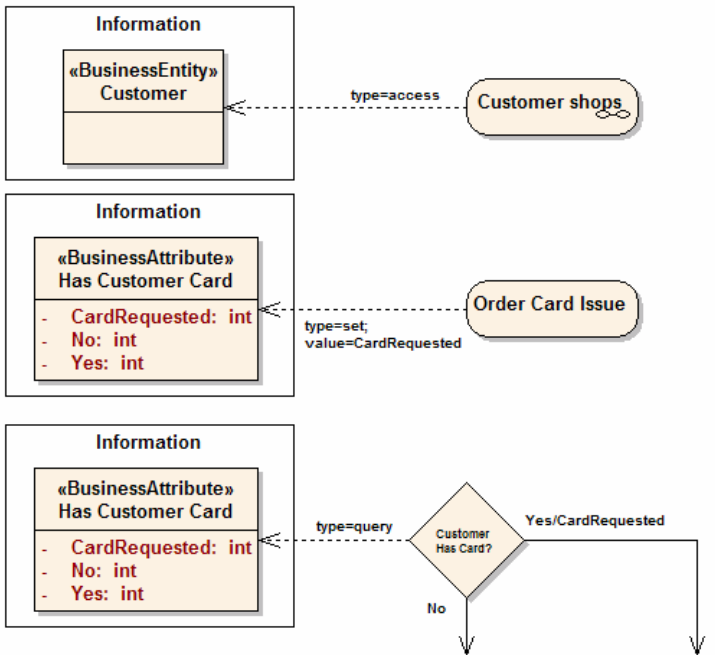

Figure 5 – Examples of the “access”, “set” and “query” operations.

The “access” operation is only allowed for a composite activity or business process; it indicates that its internal activities will access attributes of that business entity. Although optional, the usage of this operation is recommended, as it can help the stakeholder to immediately determine

all the information that the business process must access. The “set” operation is only allowed for an atomic activity; it indicates that the activity will alter the value of the attribute to a new value, which is specified in the tag “value”. Another validation can be done here, as the tool can determine whether the new value is allowed by the attribute. Finally, the “query” operation is only allowed for a choice in a business process; it indicates that the choice queries the current value of the attribute, and takes a certain path depending on that value. It is *highly* recommended that this operation be used, as it can help determine whether all possible values of the attribute were considered (or if the modeler overlooked one of those values by accident).

3.4 Patterns

The definition of the model elements of a BSL should be as oriented towards patterns (such as the ones that can be found in [Silva 2003] or [Penker 2000]) as possible, to keep the BSL’s specification simple and understandable. Pattern application is especially critical in the definition of business entities, as it is very dependant on the “art” and cunning of the BSL creator. Although the application of patterns is also very important in the definition of the concepts of the business process perspective, the fact is that current languages themselves are the results of the application of such patterns.

4. Conclusions

One of the fundamental components for alignment between Business and IT is Information Architecture, which defines the information entities (the business data manipulated by business processes) and attributes relevant to the business, as well as their associated semantics.

As discussed in this paper, current modeling languages do not regard business-specific details as first-class citizens. The problem with these languages is that they provide semantics that apply to all (or almost all) business, but they do not allow the definition of the semantics that are specific to the business itself. The fact that current modeling languages and in particular the involved tools do not regard the semantics of information entities makes very difficult the task of automatically validating created models, as the semantics supported by current tools and languages is too generic.

This paper introduces the concept of Business-Specific Language (BSL), a business modeling language that is tailored to a specific business, in a manner very similar to the DSLs that are currently gaining momentum and support in the Software Engineering community. Their greatest advantage is the possibility of integration between the business-specific entities and the business processes that will manipulate those entities. This allows, among other things, continuous model validation and the detection of many potential problems and pitfalls at model design-time. Also, BSLs can be created based on current languages, in order to maintain the usefulness of already existing business models and their languages. The processes of defining the syntax and semantics of a BSL were briefly explained, as well as the relationships between syntax, semantics, business entities and business processes. Finally, the concepts presented in this paper require extensive validation work through case studies of various natures (i.e. several interesting business-domains), with the purpose of determining the usefulness and the possibilities that the BSL approach can offer to OE and related areas.

5. Acknowledgements

We would like to thank Professor José Tribolet, who lectured the Organizational Engineering course at Instituto Superior Técnico, in the context of which this paper was originally produced.

6. References

[BPMN] BPMN Information Home, http://www.bpmn.org [accessed on 13/07/2006]

[BRG] The Business Rules Group, http://www.businessrulesgroup.org [accessed on 13/07/2006]

[CEO] "CEO – Centro de Engenharia Organizacional", http://ceo.inesc.pt

[Penker 2000] Penker, M., Eriksson, H., Business Modeling With UML: Business Patterns at Work. John Wiley & Sons, 2000.

[DSMForum] DSM Forum: Domain-Specific Modeling, http://www.dsmforum.org [accessed on 13/07/2006]

[Kelly 2004] Kelly, S., "Tools for Domain-Specific Modeling". Dr. Dobb's Journal, http://www.ddj.com/184405819 [accessed on 16/07/2006]

[MartinFowler] Martin Fowler, http://www.martinfowler.com [accessed on 13/07/2006]

[Schmidt 2006] Schmidt, D. C., "Guest Editor's Introduction: Model-Driven Engineering". Computer 39(2), 2006, http://doi.ieeecomputersociety.org/10.1109/MC.2006.58 [accessed on 16/07/2006]

[Silva 2003] Silva, A. R., "Towards the Organizational Engineering Pattern Language. Resources and Roles Based Patterns: The CONTACT, PERSON, ORGANIZATIONALUNIT and ORGANIZATION Patterns", in *Proceeding of the EuroPLoP'2003* (Irsee, Germany, June 2003), http://berlin.inesc-id.pt/alb/static/papers/2003/patterns-oe-part-1.pdf [accessed on 16/07/2006]

[Silva 2005] Silva, A. R., Videira, C., *UML, Metodologias e Ferramentas CASE – 2ª Edição – Volume 1*, Centro Atlântico, Portugal, 2005.

[Sousa et al. 2005] Sousa, P., Pereira, C. M., Marques, J. A., "Enterprise Architecture Alignment Heuristics". Microsoft Architects JOURNAL4, January 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/heuristics.asp [accessed on 13/07/2006]

[UML] Object Management Group – UML, http://www.uml.org [accessed on 13/07/2006]

[UML2Super Spec] Object Management Group, "Unified Modeling Language: Superstructure – Specification Version 2.0", August 2005, http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf [accessed on 15/07/2006]

[White 2004] White, S. A., "Process Modeling Notations and Workflow Patterns", March 2004, http://www.omg.org/bp-corner/pmn.htm [accessed on 14/07/2006]