Classification and Ontological Aspects in Software Engineering

María del Pilar Romay[1] and Carlos E. Cuesta[2] *

[1] Departamento de Sistemas Informáticos
Escuela Politécnica Superior, Universidad Europea de Madrid
Villaviciosa de Odón, 28670 Madrid (Spain)
pilar.romay@gmail.com
[2] Grupo Kybele, Departamento de Lenguajes y Sistemas Informáticos
Escuela Sup. Cc. Experim. y Tecnología, Universidad Rey Juan Carlos
Móstoles, 28933 Madrid (Spain)
carlos.cuesta@urjc.es

Abstract. The organization of objects into classes and categories is an essential task in the process of forming concepts. Within computer science, this classification activity must be supported by a well-founded representation system; among the alternatives, *ontologies* appear as a particularly suitable choice. Classification has therefore an ontological nature, as it both defines a system of categories, in the philosophical sense, and is founded by some ontology, in the computational sense. However, there is an implicit duality; classes are both conceived within an ontological hierarchy and expressed as a linguistic structure. These often conflict. Our proposal is to tackle this problem by defininining two orthogonal dimensions, namely an *ontological* one and a *metalingistic* one, to separate these perspectives. But once dimensions have been defined, some other proposals appear. For instance, different ontological hierarchies are perceived from different viewpoints and domains; a *subjective* dimension is therefore provided to include them. Similarly, the evolution of elements though the software lifecycle suggests to classify the outcomes of different phases; then a time-aware *evolutionary* dimension is also defined. These four coordinates provide a precise and flexible classification framework. We suggest that recent advances in Software Engineering, particularly model-driven approaches, require the definition of such a framework.

1 Introduction

The Software Engineering discipline has the commitment to find out how to build software in a systematic way. This goal must be achieved by using several different methods, techniques and concepts. Many of these are not strictly new, but have their origins deeply rooted in other fields of knowledge, including Philosophy. For example, consider two closely related but different notions, namely *classification* and *ontology*. Coming from a long philosophical tradition, their adoption in Computer Science is still relatively recent; yet they have an increasing importance within the field. This paper discusses the interdisciplinary nature of these two concepts, and its consequences.

* Author is partially funded by Project MCYT-TIC2003-07804-C05-01 (DYNAMICA-PRISMA).

Ontology, as the representation of a reality, has a fundamental role in reasoning, and therefore in automatic reasoning. To be able to reason about this reality, it is first necessary to describe it. This means to represent it in the form of *models* which separate relevant information, thus making easier to interpret it. These models are composed of *concepts*, abstractions obtained by grouping individual objects into *classes* or categories where common features are shared. These concepts are in turn organized into complex structures, which provide the ontological frame of reference where to define every separate subjective *viewpoint* about the original reality.

Therefore representation implies a categorization process. The process itself consists of classifying knowledge to provide a certain conception of the world, and hence the generic name of *classification*. Classifying provides (and requires) the ability to reason about categories; whether automated or not, this reasoning is still an information processing activity. Again, this must be preceded by a modeling activity, in which a suitable representation is built. In summary, they bear a mutual dependency.

During the last decade, Software Engineering has been supported by classification techniques. The most important problem to solve is the classification of software elements, that is, the identification and definition of conceptual classes or categories, and the organization of those elements within these categories. In a Software Engineering context, this would make possible to group the many by-products of the software development process. Classification manages this kind of data, covering application themselves, their models and every autonomous fragment, but also the associated documentation and even a lot of process-specific information.

In the following sections, we discuss the role of classification within Software Engineering. Then we explore the concept of ontology with regard to this problem, and the current form it adopts in Computer Science. Supported by this context, we suggest that a classification system which simultaneously comprises several orthogonal dimensions would better suit the needs of Software Engineering. We delve further into this proposal by considering four such dimensions, which cover a significant part of the spectrum. We finish by briefly discussing the expressive power of this approach.

2 The Classification Problem in Software Engineering

To understand a complex reality, the first step is to organize the information, identifying properties which make us able to group elements into classes of common features. But the notion of *class* has been used in many ways, within the context of Software Engineering. The best known of them is related to the object-oriented paradigm, where the class is usually used as the basic modeling element for software. This has still the connotation of being located at the implementation level, but in fact the organization process can be considered at a higher level of abstraction.

Therefore there are not only programming-level classes, but the rich spectrum of software elements created during the development process. These are valuable elements which should be uniquely identified, so that they could be reused in another development process, or perhaps retrieved to later evolve into a new software element.

In Software Engineering, the nature of the elements to be classified is quite different from that of elements in other fields, such as Biology or Biblioteconomics. When

classifying software elements, part of the relevant information is related to the target environment, the so-called application domain. But these are intertwined with other structural, functional and behavioural aspects. In short, the nature of software properties is complex, as they blend together several concerns.

For this reason, different classification systems have been proposed. These proposals vary as different kinds of properties are selected to be used in different representation schemes. The choice is made according to their intended purpose, which includes topics such as domain analysis, information retrieval, quality evaluation systems, support for reuse within CASE tools, and many others.

The classification problem comprises two activities; the building of a sufficiently wide category system, which should be able to accommodate software elements from any conceivable domain, and the definition of a flexible organization process, which should be able to find the right category for every such element. Therefore, software classification is an ontologic problem, using the term in the broad sense. But within the specific context of knowledge representation, the ontological approach is just an option, one strategy which could help in tackling the problem. In the next section we will briefly discuss the consequences of adopting an ontological approach.

2.1 Classification Systems in Software Engineering

The more ancient classification systems were related to Biology. The primitive man began by finding similarities between objects in the world, particularly plants and animals; and with time he learned how to abstract their differences and group them in classes or categories, which made later possible to infer less apparent similarities. Classification would gradually evolve towards a formal system, a *taxonomy*, where names of beigns are organized. Theophrastus, an Aristotle's disciple, wrote the first taxonomy in History, a compilation about Botanics.

The modern age of classification is considered to have started two millennia later, again in Botanics, with the works of Linnaeus. However the most important innovations in this direction come from Biblioteconomics. Consider that according to their contents, many books could fit in several categories, and there are also several organization strategies. But even more important, these are applied on library *registers* rather than physical books themselves. Software classification is somewhat similar: it's not a question of organizing elements, but their *metadata*.

In the context of Biblioteconomics, Ranganathan is considered as the pioneer to propose the idea to simultaneously use different dimensions to classify, which was a substantial advance in the field. That kind of model has already surpassed the limits of this discipline, extending its influence to many other fields of knowledge. For instance, these systems have been used in Software Engineering to support software classification, within the context of Systematic Software Reuse.

Among these library-inspired systems, probably the best known one is the *facet*-based approach. Facets were introduced into software classification by Prieto-Diaz [15], and they define which is probably the most evolved technique in the field. This approach has been widely tested and studied; it is included also as a part of many other proposals, which complement it by providing adequate specification standards and retrieval methods, together with several facet-definition mechanisms [11].

Apart from facets, many other classification techniques have been considered within Software Engineering, in the context of software reuse and reverse engineering. They use a wide range of approaches and are supported by many foundations, including category theory, clustering systems, and diverse methods and heuristics from the fields of Artificial Intelligence and Natural Language Processing.

Classification systems can be divided in two different groups, namely *controlled vocabulary* systems, such as facet-based techniques, and *non-controlled vocabulary* systems. Some controlled vocabulary proposals have been highly criticised because, contrary to what was expected, they have not achieved better results than non-controlled vocabulary techniques [12]. In fact, this only happens when a classification system has been conceived from the restricted point of view of a single designer, which may not coincide with the point of view of some other users. More adequate classification systems could be designed by studying the domain in advance, and by including both coincidences and variation points in the description. That's the purpose of some of the most popular techniques of Domain Engineering nowadays [8].

However the use of domain analysis techniques during classification systems' design is not enough to avoid the problem of having different viewpoints of the same domain, and they do not consider the evolution of components either. Therefore it is necessary to define a framework where the different perspectives and dimensions which appear when classifying information –particularly software-related information– can be acommodated. In this paper we propose such a framework, where classification is conceived as unfolding on different dimensions, where ontological, meta-linguistic, subjective and evolutionary concerns are evaluated. This framework must be shaped such that knowledge about both elements to classify and their environment can be represented. Then it can be used to reason about these elements, or to adapt the classification system to this changing environment.

3 Ontology in Software Engineering

A correct representation of information is of utmost importance to achieve an adequate classification. For this reason, many approaches to software classification use complex knowledge representation techniques, related to and originated in the context of Artificial Intelligence. *Ontologies* are considered among the most expressive and powerful of these. As they are themselves conceived as a system of categories, they are particularly useful for classification purposes.

Ontologies, in the specific computational sense, have evolved recently to a very expressive and generic linguistic form, supported by Description Logics [3], which includes even behavioural aspects. Also, the concept of ontology itself has become more general, somewhat approaching to its original philosophical meaning.

3.1 Defining Ontology in the Context of Philosophy

Ontology is usually defined as the theory of *what is*, the branch of Metaphysics which reasons about *being*. In classic terms, it deals with essence and existence, with objects themselves and their relationships; in short, with *reality*. Building on this, Mario

Bunge [4] defines Ontology as the branch of Philosophy with studies the prevailing aspects of reality, such as existence, change, time, hazard, mind or life.

An immediate consequence of the study of "being and the object itself" is the effort to build up a system which supports analysis and abstraction of properties. That kind of system takes usually the form of some category structure and a theory of universals. For this reason, ontology is often related to the design of category systems, to the extent that the word has acquired this additional meaning. That's the context where we are able to state, perhaps abusing the language, that *classification* is an ontologic activity.

But now we must distinguish *the* ontology from *an* ontology; that is, *general* Ontology from *special* Ontology. The first term refers to the discipline itself; when it defines a system of categories, they are those of existence itself, such as object, state or change. The second term refers to the ontologic perspective of *a part of* reality, and it encompasses the categories of a branch of knowledge, such as Physics or Biology.

For the same reason, there is a classic confusion between Ontology and two related branches, *Mereology* and *Epistemology*. The former one is the study of the relationship between parts and the whole, and it is related to the definition of classes and sets. Today Mereology is understood as an independent discipline, and not as a subset of Ontology. The second on is, by definition, the theory of *what is known*, the branch of Philosophy which deals with *knowledge*. It studies both the cognitive process and the conceptual structures that the mind builds to grasp the reality. The similarity between Ontology and Epistemology is due to the fact that both of them are devoted to define category structures, and they are also strongly related. Ontologic structures intend to describe the reality, while epistemologic ones describe just the way we understand this reality. They should be identical a priori, but there is no way to state that correspondence, as we actually only know what is known. This problem is even more apparent in the context of Computer Science, and in particular within Artificial Intelligence, where we usually asume that what we know is all that exists (closed-world axiom).

A crucial moment in the History of the discipline is the establishment of the branch known as *Formal Ontology*, which can be traced to the early works of Edmund Husserl and, particularly, to Stanislaw Leśniewski [16]. Formal Ontology can be briefly defined as the result of combining the intuitive method of classic Ontology with the symbolic method of modern Logic, to finally identify both as two perspectives of the same science. This definition is due to Nino Cocchiarella [4].

From a pragmatic point of view, a formal ontology is often described as a category system with a logic support. We should emphasize that they are both necessary. Too often ontology has been identified just with the category system, as the traditional philosophical approach seems to suggest. But this is a mistake, and sometimes it has appeared too when transferring these ideas to Computer Science.

From this point of view, an ontology is sometimes identified to a taxonomic hierarchy of classes. Then it is described as a structure defining classes, which are related by inclusion or subsumption abstractions. This approach is inadequate, as an ontology needs not to be constrained to adopt such a form. The only actual requirement is that it must be able to constrain the range of possible interpretations [5].

There is a clear relationship between a formal ontology, as described, and the current concept of ontology within Computer Science, which justifies the use of the term. A

computing system could be conceived as a logic reasoning system, governed by well-defined rules of behaviour, which operates on a certain information base. When this base is shaped as a structured representation of the world, it is describing the reality, as the system conceives it. When this representation is coherent, the logic engine of the system can be used to reason about reality. Therefore, the information base is serving as the fragment of reality supporting a logic theory, and from this point of view it can be considered as an ontology, in the broad sense. In summary, we could define an ontology, in Computer Science, as a certain representation of reality, structured in categories, and supported by a logic system able to reason about it.

3.2 The Role of Ontologies in Computer Science

The best-known and most widely cited definition of ontology, within the Computing field, is Tom Grüber's [5], originally developed in the context of Artificial Intelligence. But nowadays the scope of this definition has been extended to cover every area of Computer Science. Actually, it has influenced not only theoretical approaches, but even the most informal understanding of the concept. According to him, an ontology is the specification of a conceptualization. This definition is quite satisfactory, as it summarizes several aspects of the concept in a short sentence. This sentence can be understood at different levels. Informally, it can be read as a generic definition, which is both coherent with the traditional philosophic intuition and formal ontology principles. But a more specific reading provides also a precise description of the concept within Artificial Intelligence and Computer Science.

Even the choice of terms has proven to be quite fortunate. On the one hand, the reference to *specification* clearly states the fundamental limitation of the concept within an information system: the ontology must be articulated using a language. On the other hand, it also explains why the notion is relevant in the Computing context. Just consider that several agents could have different *conceptualizations*, that is, diverging perceptions of the world and different knowledge bases.

However this definition has been challenged by some authors, and in particular by Nicola Guarino [4], which rejects the specific use of the term "conceptualization". He states that Grüber is borrowing the term from Genesereth and Nilsson's work, where it is conceived to mean an extensional set of relationships outlining a certain situation. But the intuitive meaning of the word is much wider, as it refers to the psychological process of creating a set of concepts. Guarino suggests that Grüber's definition is commonly accepted because most people assumes this term is used in the wider sense. The difference is subtle anyway, and in general terms we can simply ignore Guarino's objection, as it doesn't really affect the way the notion is used.

One could argue that those intended ontologies are actually epistemologies, as they are not describing the reality itself, but the knowledge base of a reasoning agent: the way it perceives, knows and conceives this reality. However the use of the term is not just a consequence of the traditional metaphysical confusion between them, but it has also something to do with the very essence of information systems. From their point of view, information is the only matter; it defines a virtual world in which only what can be described exists. Therefore an ontology is not just representing knowledge, but the only reality the system can manage.

Curiously enough, this is also the philosphical distinction between formal ontology and material ontology [4]. The first one does not consider the actual relationships between objects in the material world, but those between the corresponding concepts inside a formal structure. The mapping between this conceptual structure and the reality is the subject of the second one. But in a computational system, only the first one matters; what it needs to represent is just the conceptual network used by an information system, and supported by a formal category structure.

4 Dimensions for Classification

To organize the information in classes is a difficult task, as we are not usually able to identify a single classification framework. This framework should be defined by selecting relevant properties in the elements to classify, but this is far from trivial. There are many variables to consider, such as the different natures of those elements, the alternate views of the same reality, or the relationships between seemingly independent aspects. Most of the existing classification systems decide to adopt a rigid scheme, in which each and every element must be uniquely classified.

Though this strategy seems to solve the problem, in fact it just moves the complexity into the organization process. Instead we propose the adoption of a different approach, in which every such factor is made explicit as a *dimension*, a separate *viewpoint* to accommodate separate *perspectives*, composing a general classification framework. This way, we are not trying to create an unified scheme, but to harmonize many different schemes. Using this strategy, at least four different, core dimensions are almost immediately identified. We devote the rest of this document to briefly discuss them.

First, we must provide a dimension to encapsulate the traditional approach, which uses a suitable representation system to define the relevant categories and their hierarchies; these set up our *ontological dimension*. But here, a different though related perspective appears too. The need for a representation system is just the general need to articulate the information in some kind of language. But this language has an inner structure itself, comprising several levels of abstraction, which could affect and even distort the representation. This might influence the classification process, and therefore a *meta-linguistic dimension* is required to accommodate this effect. There is always an implicit tension between ontological hierarchy and linguistic abstraction [2]; by maintaining them separate as diverging axis, this tension is somehow relieved.

Initially, this onto-linguistic binomial defines a static scheme; but our aim is to model a world in constant change. In fact, not only the world, but the model itself evolves in time: a temporal dimension is therefore required. Milestones in this dimension describe different stages in the model's evolution. Then, this axis receives the name of *evolutionary dimension*, as its essence is related to change, more than to time.

Finally, the fourth dimension is related to the point of view which the model is being described from. One lesson which Philosophy (and even Physics) can teach Software Engineering is that any description of a reality depends on the observer, inasmuch as it depends on the reality itself. Within software development, modeling has been mostly focused on the object, and only occasionally the subject is considered. Every observer can provide a different perspective or viewpoint; this means that a *subjective dimension*

is required to accommodate them all. However, once this feature has been provided, it can be used to distinguish among not only observers, but their intentions. A different perspective can be provided by the same observer with a different concern. Therefore a *subject* can refer both to a viewpoint or an aspect.

By considering those dimensions in a classification system would provide it with enhanced flexibility for scheme specification and category definition. Of course, the classification and recovery process itself becomes also more complex.

4.1 Ontological Dimension

Classification is essentially an ontological activity – and therefore the ontological approach has been widely used. In fact, when only this dimension is considered, the result is comparable to tradiditonal classification systems. This has nothing to do with choosing ontologies as the representation language; our endeavour is just to classify an element within an explicit or implicit category system, according to its meaning.

Though this is not always obvious, philosophical discussions about ontology are often still relevant in Computer Science. For instance, Roberto Poli made the following remarks [14] about formal ontology, which can be directly transferred to the computing context. Moreover, they could be used to support the main thesis in this paper, which has been conceived from a Software Engineering perspective.

- An ontology *is not* a catalogue neither a taxonomy, but a general framework or structure in which catalogues and taxonomies could be organized.
- An ontology *is not* an epistemology, as it is not reduced to study knowledge. While the former deals with the objective side of reality ("objects"), the latter describes the subjective side ("subjects").
- Nothing precludes the existence of *several* ontologies, in plural.

The first point remarks the generic nature of an ontology, which is not just a taxonomy but something more complex, at a higher level of abstraction. The second point intends to clarify a classic confusion, specifying the essential difference between an ontology and a knowledge representation system. Besides it exposes the basic distinction between objective and subjective viewpoints, the same one can find in recent extensions to modeling concepts [6, 7]. Finally, the third point excludes the requirement of having a single knowledge structure, and even implies the need to reconcile several ontologies, which need not be orthogonal.

In fact, this is also our basic argument when proposing to consider a *subjective* dimension. This will be used to develop several ontological hierarchies, which then will be synchronized by a *weaving* process, just like in other areas within Software Engineering. Therefore, the same element might be classified using concepts from different ontologies, corresponding to many perspectives or domains. Actually there's no need to combine them to define an unified structure; the only requirement is to ensure that their contents are not strictly contradictory.

This is somehow similar to the Husserlian approach to ontology, which is found among the origins of phenomenology. Husserl conceives the existence of a reality, which is differently perceived by different subjects. His *intentional* ontology is therefore the objective substrate which is shared by different subjective acts of conscience. In

Computer Science terms, there are several (subjective) knowledge-based systems trying to describe the same reality. We are not actually able to know this reality, but we must assume some kind of consistency between those systems, though not always apparent, which reflects the common reality they are trying to describe.

4.2 Meta-Linguistic Dimension

Besides influencing classification systems, ontological aspects pervade modeling environments. There is a common conflict in both cases: though classification intends to define an abstract system of categories, this system must be described using a concrete language. The implicit tension between ontological and linguistic concerns is even more apparent in visual languages, like those in UML.

The nature of the problem is revealed as a *dual classification* conflict [1]. Every software element can be classified in two different hierarchies, according to either its internal meaning –ontological viewpoint– or the role it is assigned by the language's syntax –metalinguistic viewpoint–. Both perspectives are relevant for software classification. For instance [2], the term "dog" designates, from an ontological point of view, a certain animal species; but from a meta-linguistic perspective, it could refer to a certain class in the object-oriented model of our chosen language. This information is also pertinent to achieve an accurate classification.

Though it could seem less important, the linguistic representation of a software element is actually essential to classify it. The same term could refer to an object, a class or a metaclass in different contexts. Sometimes this helps to distinguish either an analysis-phase or a design-phase artifact. Each one implies a different linguistic role; and depending on it, the term could even be assigned to a different ontology.

Even when some modeling languages, like UML itself, provide the facilities to define linguistic families, like the MOF, or elements with variant levels of abstraction, like *stereotypes*, those are not enough to adequately define an ontology [2]. This means that an ontological relationship cannot be deduced from the corresponding linguistic representation; there is not a direct mapping between them.

Some authors, including Atkinson and Kühne [1, 2], have suggested the separation of these viewpoints in two dimensions, which respectively define an *ontological* hierarchy and a *metalinguistic* hierarchy. This makes possible to hold all the relevant information, while at the same time avoids the need to combine both perspectives. Every software element is thus classified under two different categories, but these are located in orthogonal dimensions. In summary, every element has now two "coordinates".

This approach is both simpler and more flexible than the fusion of hierarchies. Moreover, it can be generalized; it needs not to be restricted to just two dimensions. An extended approach would make possible to describe *several* different ontologies in separate dimensions –now related to domain-level concerns and temporal roles– within the same multi-dimensional classification system.

4.3 Subjective Dimension

Once that multiple dimensions are being considered, the separation between ontological and metalinguistic concerns doesn't seem enough. For instance, when the classification

process comprises several different domains, the simplest approach is to describe each one in a separate ontology, instead of combining them all in a single structure. This defines an orthogonal domain-oriented axis, the *subjective dimension*, which partitions the classification scheme according to different contexts or themes; but it can also be used to describe different viewpoints. The common thread is that every such coordinate locates, all in all, a different perspective.

The same element could play different roles depending on the way it is used, or could be conceived from a different point of view, which depends on the observer. For instance, the same tree is the home for an eagle, and just wood for a logger [6]. In short, the same concept could have a different relationship to different *subjects*. Therefore, it could be classified under a different category from each perspective. That was the central idea of Harrison's subject-orientation [6].

Indeed, a subject's target is another subject's context: the same notion, approached from two points of view. As a consequence, multiple ontologies appear. Though they are essentially hierarchic and independently conceived, some of their concepts seem to coincide by chance. In fact, often they should have more than one point in common, as logically they would use several related terms, even under different perspectives.

The interest and true complexity of this approach is revealed when such points of contact appear between those *a priori* orthogonal hierarchies; that is, the same concept is conceived from several viewpoints, or a software element must be classified under several categories. This makes possible to define partial mappings between those hierarchies, which define a sort of heterarchy.

On the other hand, this could be conceived just as an organization problem, related to the location of a concrete software element. Our first approach suggests to use a feature-oriented organization [8], where *features* are the properties which describe a software element, and simultaneously, which are considered relevant for some perspective. In fact, every subject (or viewpoint) is outlined by the set of features it defines, and which in turn define the subject itself.

In the end, a classification system is composed of several ontologic hierarchies with many *join points*, instead of trying to define an universal ontology, which would be impossible to manage. This approach tries to provide the best ontology in every domain, to later consider their contact points. The final purpose is to automatically deduce the relationship between these ontologies, starting from those join points.

This approach could seem cumbersome and perhaps unrealistic when transferred to Computing, but actually it is beign applied with growing success in Software Engineering, both in early phases of development such as requirements and architecture, and at the implementation level with so-called "aspect-oriented" technologies.

Among these, the *multidimensional separation of concerns* by Ossher *et al.* [13] stands out for it bears a certain resemblance to our approach. This is the heir to subject-orientation [6], and the most important among symmetric models of aspects [7]. There are also asymmetric models, like exemplified by AOP [9]. Though their original conception is different, these provide a similar technological platform, which could also be used to implement our approach. Nevertheless, both models use the same basic notion, that of a *weaving* process. This is the automatic processing of relationships between

"aspects" from their join points. This is essentially a dependency resolution, but allows us to maintain separate models, diferring their combination.

In summary, our approach intends to use this background to realize some kind of semi-automatic dependency resolution between ontologies. This requires that our representation language is precise, accurate and expressive enough; for this reason, we have selected certain advanced Description Logics [3] as the *formal* foundation for ontology description. Their weaving is supported, in turn, by Category Theory techniques.

4.4 Evolutionary Dimension

Probably ought to the taxonomic nature of traditional classification schemas, inspired in biologic and bibliographic metaphors, existing classification systems have been mostly static. However this has nothing to do with the essentially *dynamic* nature of software. In particular when considered as a part of the software development process, the lifecycle of any software element must be conceived as an evolution in time.

Software elements hold changing information. At a given moment, they could be a part of a finished software project, or stay in some intermediate stage of the development process. Their representation has been changing along time, and this includes a bunch of process-oriented meta-information which could still be of interest in the future, even at a different stage from some other software project.

Recent research remarks the relevance of such process-related information, which describes in detail the evolution of software elements, and affects their organization within a sensible classification system. This is even more important in the context of model-driven development processes, just like MDA [10]. This approach requires a precise definition of every stage in the process itself, but also of their transformations. These definitions are provided in the same modeling language considered for the rest of the process, and therefore they are also keen to be classified and reused. This is in fact one of the reasons to use such a complex approach.

But this evolutionary nature has only recently been considered, in some of the most modern classification systems. They take the form of complex structures in which the evolution of a software element can be *traced* all through its lifecycle, by describing the relationship between the different incarnations of this element at different stages of the development process. Again, we propose to separate this additional information in an *evolutionary dimension*. Apart from being useful, this approach makes possible the integration of the classification system and the Software Engineering process, particularly within any model-driven context.

5 Conclusions

Traditional classification systems have either ignored the concerns in previous sections, or considered them apart from each other. Our purpose, when defining them as dimensions, is to make possible to gather them all in a single classification framework, such that their information can be used altogether. We suggest that only that kind of joint system provides the required expresiveness to introduce the benefits of classification in the context of modern Software Engineering.

Consider that an adequate classification system must be evolutionary, at least partly; it should be able to be extended as it is used. Of course, it should still have an ontological nature, especially considering recent advances in ontology-based languages, which have become even able to express behaviour. And the ability of combining several separate contexts will be increasingly important, because software development itself is already unfolding into many viewpoints and perspectives.

Such a classification system, when supported by a solid foundation, as provided by Description Logic languages [3], would make possible to actually integrate classification processes and ontological aspects into the Software Engineering mainstream.

References

1. Colin Atkinson and Thomas Kühne. Rearchitecting the UML Infrastructure. *ACM Transactions on Modeling and Computer Simulation*, 12(4):290–321, 2002.
2. Colin Atkinson and Thomas Kühne. Model-Driven Development: A Meta-Modeling Foundation. *IEEE Software*, 20(5):36–41, September 2003.
3. Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.
4. Raul Corazzon. What is Ontology? In *Ontology. A Resource Guide for Philosophers*. http://www.formalontology.it, February 2005.
5. Tom R. Grüber. A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.
6. William Harrison, Harold Ossher, and Peri Tarr. Subject-Oriented Programming – A Critique of Pure Objects. In *Proc. 4^{th} Conf. Object-Oriented Programming Systems, Languages and Applications (OOPSLA'93)*. ACM Press, 1993.
7. William Harrison, Harold Ossher, and Peri Tarr. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. Technical Report RC22685 (W0212-147), T.J. Watson Center, IBM Research, December 2002.
8. K. Kang, Sholom Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasability Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.
9. Gregor Kiczales, John Lamping, Cristina V. Lopes, James Hugunin, Erik Hilsdale, and Chandrasekhar Boyapati. *Aspect-Oriented Programming*. U.S. Patent # 6.467.086, 2002.
10. Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, Boston, 2003.
11. Ali Mili, Rym Mili, and Roland T. Mittermeir. A Survey of Software Reuse Libraries. In *Annals of Software Engineering*, volume 5, pages 349–414. Baltzer Science, 1998.
12. Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Harold Mcheick. Another Nail to the Coffin of Faceted Controlled-Vocabulary Component Classification and Retrieval. In *ACM Symposium on Software Reusability (SSR'97)*, pages 89–98, 1997.
13. Harold Ossher and Peri Tarr. Multi-Dimensional Separation of Concerns and the Hyperspace Approach. In *Software Architectures and Component Technology: the State of the Art in Software Development*. Kluwer, 2000.
14. Roberto Poli. Ontology for Knowledge Organization. In *Knowledge Organization and Change*, pages 313–319. Indeks, 1996.
15. Ruben Prieto-Diaz. Implementing Facet Classification for Software Reuse. *Communications of the ACM*, 34(5):88–97, 1991.
16. J.T.J. Srzednicki, V.F. Rickey, and J. Czelakowski, editors. *Lesniewski's Systems: Ontology and Mereology*. Kluwer/Ossolineum, 1984.