Distinction-Based and Verification-Assisted Knowledge Modeling

Philippe Michelin
Aebis, Paris, France
pmichelin@aebis.com

Marc Frappier
GRIL, Département d'informatique
Université de Sherbrooke, Canada
Marc.Frappier@USherbrooke.ca

***Abstract*—This position paper presents a lightweight approach for knowledge modeling centered around the notion of distinction. Concept models are represented using UML class diagrams. Distinctions between concepts are established by using attributes and stating properties of the attributes using the @L-is calculus on words. This calculus relies on simple laws to manipulate relationships between words. We show how to represent these concept models in ALLOY, a symbolic model checker for first-order logic, which allows one to verify the consistency and completeness of a concept model.**

***Keywords*-Concept model, distinction, @L-is, ALLOY, model checking, consistency, completeness.**

I. INTRODUCTION

Requirement engineering requires mutual understanding between customer, end users, and suppliers, of what is intended to be engineered; this means transferring tacit knowledge, about the product and the project, between human actors. Too often people assume that they are working with well-defined basic concepts, and that they share a common understanding of these concepts. However, it is our experience that basic concept definitions are frequently overlooked, deemed too obvious to bother with. Issues arise from misunderstandings between project stakeholders. Conversely, heavy, cumbersome and self-referential definitions of basic concepts (*ie*, concepts defined from themselves) are sometimes provided and don't help understanding.

When main requirements are written with great detail, their compliance with regulations and standards raise the same type of question, namely: Is there a real compatibility between "what is intended to be engineered" and the intent of the legislator, expressed in regulatory texts? Most regulatory texts are not written to be plain and applicable but to respect internal objectives of the Legal Authorities that provide them. As a result, written regulations and standards are frequently incomprehensible, not only to common readers, but to lawyers themselves.

In this paper, we outline our approach to build domain models. Our approach is pragmatic and based on very simple principles, to the point that it may seem "too simple" for a researcher in domain modeling. This emphasis on simplicity arises from practical experience with domain modeling. When a domain modeling activity is necessary, it is usually because the domain is poorly understood. Thus, the domain modeling notation and methodology must not add complexity to an already complex situation. One must use something very simple in order to focus on the essential elements of the domain to model. Simplicity is not in opposition with precision and formality, so our notation is also formal and supports reasoning to verify model consistency and check properties.

Our approach is based on concept models (our personal view of it), which we choose to represent using a variant of UML class diagrams, with traditional features like inheritance, attributes, associations and constraints. Constraints are first modeled using a calculus on words called @L-is, which allows one to express simple properties and relationships between words, and reason about them using intuitive laws. This model can be translated into first-order logic (FOL) and checked with Alloy [4], a symbolic model checker for FOL with relations. So our approach is very similar to ontologies and description logic [1], except that we want to essentially manipulate words like domain experts do. Reasoning in terms of sets and relations is a typical computer scientist reaction which we want to avoid in the first steps of modeling, to be closer to the natural way of representing domain knowledge by non computer scientist domain experts. The translation to ALLOY allows one to refine a model, to use the power of set theory and FOL to state constraints, and to check consistency and completeness.

Our methodology is similar to the one presented in [6], except that we stress a critical point, the notion of *distinction*, which we find central in identifying relevant elements of a domain model. We identify patterns of distinction to help the modeler in building a model. In [3], Alloy is used to represent OWL ontologies and to check their consistency, subsumption of classes, instantiation, and instance properties. Classes are represented as elements of a single Alloy signature, and similarly for property of classes, which are represented as elements of signature Property. Our translation of concepts (*ie*, ontology classes) is simpler: we directly represent concepts as Alloy signatures, which makes it simpler to model properties of concepts and reason about them.

The rest of this paper is structured as follows. Section II describes our methodology, which is illustrated by a simple case study of product management. We conclude with some remarks in Section III and providing an outlook on how

we plan to apply our methodology for a requirements engineering project on consent management in health care, that will be subject to several laws and regulations.

II. Methodology

A. Step 1: Identify the purpose, objectives and users of the model

This most important step determines the trajectory of the modeling activity and its scope. When issues arise during modeling phases, one should always come back to this initial step and determine which concepts are *relevant*, and which characteristics of these concepts are *essential*, with respect to the purpose, objectives and users of the model. The purpose denotes the primary intended use of the concept model. Our technique is well adapted for abstract modeling, and it is not restricted to the software development context. For instance, one could use it for domain understanding as a prerequisite for requirements analysis in systems development (software, hardware). It could also be used for knowledge management and documentation when critical employees are retiring or leaving an organisation, for business process improvement, standards development, and so on. The choices made when modeling a concept strongly depend on the purpose, objectives and users. Completely different models of the same concept can be produced depending on the scope identified.

For our case study, let us look at Wikipedia, which is often a good example of community knowledge, sometimes not fully disambiguated and free from contradictions, to find out general information about what a product is [8]: "a product is defined as a thing produced by labor or effort or the result of an act or a process". It then proposes several views and ideas, among which are the following:

- "A marketing view, in which a product is anything that can be offered to a market that might satisfy a want or need."
- "A manufacturing view, in which products are purchased as raw materials and sold as finished goods."
- "A project management view, in which products are the formal definition of the project deliverables that make up or contribute to delivering the objectives of the project."

For the sake of illustration, we decide to integrate these three views as our objective, our purpose being to gain a general understanding of the central concepts of product management.

B. Step 2: Build a concept model

A *concept* represents a collection of instances which share common structural characteristics (attributes, relationships and constraints on these attributes and relationships). Hence, a model of a concept describes its structure, and concept instances are examples of a concept which satisfy its structure. Sometimes, a concept has no instance, for instance artificial and natural can be seen as two concepts with no instance. They are used mainly as *qualities* to describe other concepts and they often arise from adjectives in natural language. For instance, objects are either artificial (man-made, *eg*, a car) or natural (available as-is in nature, *eg*, iron ore, bauxite, trees). Here, we use artificial and natural as qualities to distinguish two objects. Another viewpoint is to consider a car as an instance of artificial, and iron ore and trees as instances of natural. If the purpose of our model is to build an understanding of product management, then we prefer to see cars as instances of artefact rater than instances of artificial. Concepts with no instance are useful to establish distinctions between concepts. One can distinguish raw materials from artefacts using these two qualities (natural vs artificial).

Where does one find these concepts? What are the good qualities for the good concepts? It is best to start from a solid basis, such as standards, textbooks, regulations and interview with practitioners. Otherwise, the probability of success is very low. These sources provide the "raw material" for conceptual modeling, but they must be distilled to extract relevant concepts. For our case study, we reuse the terminology proposed in the glossary of the Capability Maturity Model Integration (CMMI®) [2], published by the Software Engineering institute (SEI℠). The CMMI is an integrated process improvement approach that transcends disciplines and provides organizations with the essential elements of effective processes. Although the CMMI product suite is a very strong foundation for process improvement, it is not without terminological problems that may lead to misunderstanding. In accordance with the CMMI for Services V1.2 glossary, the definition of "service" starts with "a product that is intangible and non-storable". Wikipedia mentions "tangible and intangible products". Two questions arise: How to model adjectives "tangible", "intangible" and "non-storable"? How to model the conjunction word "and"? In English, the word "and" has several meanings. It sometimes means a conjunction, as in the definition of service in the CMMI. It sometimes means a sum of two disjoint parts, as in the Wikipedia excerpt for product.

This leads one to realize that even from a small number of concepts, seemingly trivial questions arise, but they are not so trivial in the end. One rarely finds the proper set of concepts right from start.

Figure 1 illustrates the relevant concepts selected from the CMMI glossary and the Wikipedia definition according to our purpose (marketing, manufacturing and project management). Of course, this is not the model that we have first drawn. It has been successively refined, adding concepts, disambiguating them using distinctions, finding commonalities. The top part of the model (RawMaterial and artefact) originates from the manufacturing view-point. In this model, the *IsA* relationship (depicted using the UML convention) is the typical back bone. We reuse the traditional meaning of *IsA*: attributes, relationships and constraints of the generic concept are inherited by the sub-concepts. Sub-

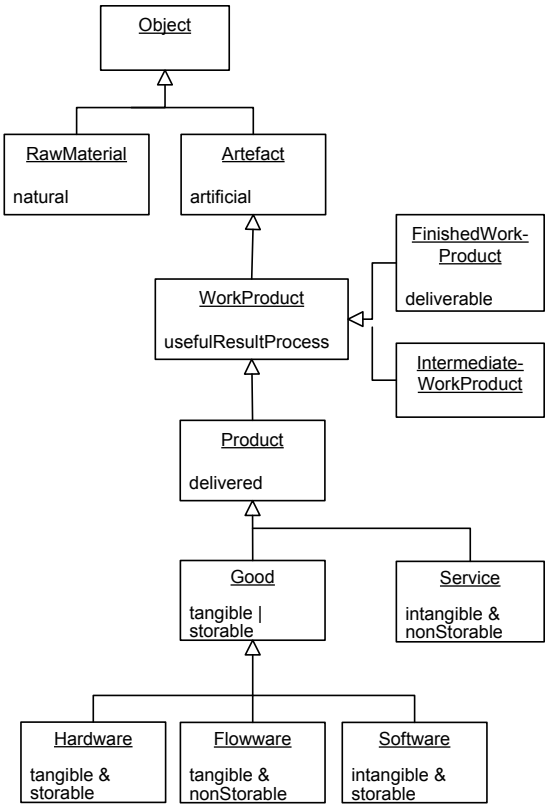

Figure 1. Concept diagram for product

concepts used to establish a distinction are *disjoint* when an instance of the generic concept is an instance of at most one of the sub-concepts. Sub-concepts are *covering* if an instance of the generic concept belongs to at least one sub-concept. A *partition* is a set of sub-concepts which are both disjoint and covering. We use a shared arrow head to denote disjoint sub-concepts in our concept diagram. In Figure 1, concepts RawMaterial and Artefact are disjoint, similarly for FinishedWorkProduct and IntermediateWorkProduct. Concept WorkProduct also has another sub-concept, Product, which is not disjoint, justifying the use of an unshared arrow head for each.

Attributes are often used to mark the distinction between concepts. Attributes of a concept are usually qualities, similar to adjectives in natural language. They do not have a value of some type for each given instance, as one expects for an attribute in a UML class diagram. Instead, a concept attribute denote a quality that each instance of the concept possess. In Figure 1, attribute natural and artificial mark the distinction between concepts RawMaterial and Artefact: an object cannot be natural and artificial at the same time. Hence, these two qualities are mutually exclusive and can also be used to partition instances of Object. Qualities are sometimes connected by "and" and "or", which are denoted respectively by "&" and "|", as in concepts Good and Service.

C. Step 3: Formalize Distinctions in @L-is

The concept model defined in the previous has been made more precise by providing distinctions using a simple calculs of words called @L-is, inspired from *Laws of Form* [7] by George Spencer-Brown. This calculus is supported by an interpreter [5] which can perform computations by using definitions as rewriting rules. @L-is is intended to manipulate words as semantic units in natural languages. It shares some similarities with logic and set theory, but it is essentially about words, because that is what we deal with in the first step of a conceptual model. In the next step of our methodology, we will show how these definitions can be translated into FOL and set theory, in order to exploit their expressiveness.

Distinctions are formalized in @L-is under several forms. The first form of distinction is the *IsA* relationship, which is represented by the typing operator ":"'. For instance, we have

$$\mathsf{RawMaterial} : \mathsf{Object}$$

and

$$\mathsf{Artefact} : \mathsf{Object}$$

to represent the first two sub-concepts of Figure 1. Typing is transitive:

$$\alpha_1 : \alpha_2 \wedge \alpha_2 : \alpha_3 \Rightarrow \alpha_1 : \alpha_3 \tag{1}$$

Instantiation is also represented with ":", such that we have EiffelTower : Artefact. This is justified by the fact that in natural language, one uses the same relationship, "is a", for both forms (instance and subtyping).

The distinction of objects as being either RawMaterial or Artefact can be made more precise by expressing what makes them distinct. As mentioned before, raw materials are natural objects, whereas artefacts are artificial objects. This is represented by the application operator "_" as follows:

$$\mathsf{Object\_natural} = \mathsf{RawMaterial}$$

$$\mathsf{Object\_artificial} = \mathsf{Artefact}$$

The application operator introduces sub-typing, by the following law:

$$\alpha\_\beta : \alpha \tag{2}$$

This law applied to the definition of RawMaterial allows us to deduce that

$$\mathsf{RawMaterial} : \mathsf{Object}$$

We can state that raw materials are distinct from artefacts by defining that natural and artificial are mutually exclusive (*ie*, opposite) using operator "[ ]"

$$\mathsf{natural} = [\mathsf{artificial}]$$

This means that something cannot be both natural and artificial, and the equality means, as usual, that natural can

be substituted by [artificial], and vice-versa. Two types are *disjoint*, noted $\alpha \parallel \beta$, when the following condition holds:

$$\alpha \parallel \beta \Leftrightarrow \forall x \cdot \neg(x : \alpha \wedge x : \beta)$$

The following law states that opposite attributes make disjoint sub-types of a common super-type.

$$\alpha\_\beta \parallel \alpha\_[\beta] \tag{3}$$

It allows one to conclude that

$$\mathsf{RawMaterial} \parallel \mathsf{Artefact}$$

There are relationships between attributes that do not entail disjointness. For example, what is delivered should be deliverable. We could also model these relationships in @L-is. We may not necessarily want to use sub-typing for that purpose.

Service is distinguished from Good as follows. A service is an intangible and non-storable product, whereas a good is a tangible or storable product. The word "and" is represented by operator "&", meaning "being both", and "or" by "|", meaning "being one or the other". So a service is defined as follows:

$$\mathsf{Product\_}(\mathsf{intangible} \,\&\, \mathsf{nonStorable}) = \mathsf{Service},$$

and similarly for a good:

$$\mathsf{Product\_}(\mathsf{tangible} \mid \mathsf{storable}) = \mathsf{Good}.$$

To show that goods and services are distinct, we can use some laws relating "&", "|", "_" and ":", and some basic distinctions:

$$\mathsf{tangible} = [\mathsf{intangible}]$$

$$\mathsf{storable} = [\mathsf{nonStorable}]$$

The following law states that the opposite of being α and β is to be at least one of the opposite of α and β.

$$[\alpha \,\&\, \beta] = ([\alpha] \,\&\, \beta) \mid (\alpha \,\&[\beta]) \mid ([\alpha] \,\&[\beta]) \tag{4}$$

The following law states that the opposite of having some quality is having the opposite of that quality.

$$[\alpha\_\beta] = \alpha\_[\beta] \tag{5}$$

Sub-typing is related to "&", "|" and "_" as follows:

$$\beta_1 \,\&\, \beta_2 \quad : \quad \beta_1 \tag{6}$$

$$\beta_1 \quad : \quad \beta_1 \mid \beta_2 \tag{7}$$

$$\beta_1 : \beta_2 \quad \Rightarrow \quad \alpha\_\beta_1 : \alpha\_\beta_2 \tag{8}$$

Operators "&" and "|" are also associative, commutative and idempotent. Sub-typing preserves disjointness:

$$\alpha : \beta_1 \wedge \beta_1 \parallel \beta_2 \Rightarrow \alpha \parallel \beta_2 \tag{9}$$

Using these laws, we deduce

$$\begin{array}{ll}
& [\mathsf{service}] \\
= & \langle \text{definition of service} \rangle \\
& [\mathsf{Product_}(\mathsf{intangible} \,\&\, \mathsf{nonStorable})] \\
= & \langle (5) \rangle \\
& \mathsf{Product_}[(\mathsf{intangible} \,\&\, \mathsf{nonStorable})] \\
= & \langle (4) \rangle \\
& \mathsf{Product_}(\\
& \quad ([\mathsf{intangible}] \,\&\, \mathsf{nonStorable}) \\
& \quad \mid \\
& \quad (\mathsf{intangible} \,\&[\mathsf{nonStorable}]) \\
& \quad \mid \\
& \quad ([\mathsf{intangible}] \,\&[\mathsf{nonStorable}]) \\
&)
\end{array}$$

This last type is a subtype of Good, by laws (6), (7) and (8). We can also decompose it into three subtypes using the same laws.

$$\mathsf{Product\_}([\mathsf{intangible}] \,\&\, \mathsf{nonStorable}) = \mathsf{Flow}$$

$$\mathsf{Product\_}(\mathsf{intangible} \,\&[\mathsf{nonStorable}]) = \mathsf{Software}$$

$$\mathsf{Product\_}([\mathsf{intangible}] \,\&[\mathsf{nonStorable}]) = \mathsf{Hardware}$$

By transitivity (law (1)), each of these sub-types is also a sub-type of Good. They are all disjoint from service, by laws (9) and (3).

We can provide a set-theoretic interpretation of the @L-is operators and laws, to justify their consistency.

- A concept is represented by a set. A quality is also represented by a set.
- The application operator "_" is represented by intersection "$\cap$".
- Operators "&" and "|" are respectively represented by "$\cap$" and "$\cup$".
- Operator "[ ]" is represented by set complement "$\overline{}$", with a subtle distinction for $[\alpha\_\beta]$, which is mapped to a complement relative to α, that is, $\alpha \cap \overline{\beta}$.
- The type operator ":" is represented by set inclusion "$\subseteq$".

The laws provided are then easily mapped to classical laws of set theory.

One may ask why we do not directly use set theory instead of @L-is? This is a methodological choice, which can be illustrated by law (4), which was used to uncover the concepts Good, Service, Hardware, Software and Flowware. The representation of $[\alpha\&\beta]$ in set theory is $\overline{\alpha \cap \beta}$. The law most commonly used for this expression in set theory is deMorgan's:

$$\overline{\alpha \cap \beta} = \overline{\alpha} \cup \overline{\beta}$$

Unfortunately, this law decomposes our opposite expression into two terms which are not disjoint, thus masking distinctions. Of course, the representation of law (4) in set theory is also valid, but it is far less common or natural. There are other represenstations of "&" and "|" which make this distinction more explicit. For instance, one can use the

Cartesian product ("×") for "&" and disjoint sum ("+") for "|". In fact, this is closer to the operational definition used in @L-is. There are several other interesting interpretations. A Cartesian product of two qualities (each denoted by a set of two elements) naturally produces a set with four elements, denoting the four cases to consider. @L-is was designed to force the emergence of distinctions during the modeling process. Set theory does not naturally enforce this. Thus, @L-is is our choice to manipulate words, and we use conventional mathematical theories to check its consistency. We must shield the modeler from the peculiarities of set theory, Cartesian product, disjoint sum, and let him focus on the bare essentials for modeling the concepts of a problem.

D. Step 4: Formalize Concepts in Alloy

An ALLOY specification consists of a set of *signatures*, noted `sig`, which basically define sets and relations. Constraints, noted `fact`, are axioms which condition the values of sets and relations. The declaration `sig X {r : Y -> Z}` declares a set `X` and a ternary relation `r` which is a subset of the Cartesian product $X \times Y \times Z$. Operator `->` denotes Cartesian product. The first component of a relation (*ie*, `X` for `r`) is always induced from the signature where the relation is declared. ALLOY supports usual operations on relations, like union (`+`), intersection (`&`), difference (`-`), join (`.`), transitive closure (`^`). Alloy can make two kinds of verification on a specification. The `run` command checks that the axioms (*ie*, `fact`) of a specification are consistent, *ie*, that there exists a model (in the model-theoretic sense of logic). The `check` command verifies that a property (*ie*, a formula) is valid in all models, *ie*, that there exists no counterexample for the formula. These two commands take as input upper bounds for the number of instances for each signature. Thus, the validity of the verification is limited to bounded models, as usual in model checking.

There is a quite simple mapping between our concept diagrams, @L-is constraints, and ALLOY. Each concept is mapped to a signature. There are two possible strategies to map *IsA* relationships: `extends` and `in`. They both entail subsumption and inheritance of attributes and constraints. In addition, extensions are disjoint. If one wants to check that qualities and distinctions defined in the concept model are sufficient to entail disjointness of two concepts, then *IsA* is mapped to `in`, and command `check` is used to verify that the two concepts are indeed disjoint in all possible models defined by the qualities and distinctions. Qualities (*eg*, artificial) are defined using a generic attribute, called `quality`, defined in signature `Object`. Then, we use a `fact` to state that a concept, represented by a signature, has a quality.

```
sig Object {
  quality : set QualityValue
}

sig RawMaterial in Object {}
fact {
  hasQuality[RawMaterial,Natural]
}
```

Predicate `hasQuality` is defined as follows:

```
pred hasQuality[o:set Object,
                v:set QualityValue]
// Each o has at least one quality of v
{
  all a : o | some (v & a.quality)
}
```

ALLOY operator "&" denotes set intersection. The fact that two qualities are distinct, as defined with the @L-is operator "[...]", is also stated as a `fact` using predicate `opposite`, as follows:

```
fact{ opposite[Natural,Artificial] }

pred opposite[q, q':QualityValue]
{
  no(quality.q & quality.q' )
}
```

The expression `quality.q` returns all objects having quality `q`.

We use ALLOY to check mainly five types of properties:

1) Consistency of definitions, by using the `run` command. This finds errors like contradicting qualities in a hierarchy of concepts, or the impossibility of satisfying constraints written in FOL.
2) Distinction among concepts, as determined by qualities and their distinctions. This finds errors like forgetting qualities of a sub-concept to distinguish it from another sub-concept. It amounts to checking the completeness of our definitions. For instance, this verification finds that IntermediateWorkProduct is not distinguished from FinishedWorkProduct in Figure 1, because we have forgotten to mention quality nonDeliverable for IntermediateWorkProduct. ALLOY does so by finding a model where FinishedWorkProduct and IntermediateWorkProduct share a common instance.

   ```
   check
   {no FinishedWorkProduct &
       IntermediateWorkProduct
   } for 10
   ```

3) Subsumption among concepts determined by qualities. For instance, we may wish to check that what is delivered is indeed deliverable. This verification fails for the model of Figure 1, because we haven't said that Product is a sub-concept of FinishedWorkProduct. This is again a completeness check.

```
check
{  quality.Delivered in
    quality.Deliverable
} for 10
```

4) Instantiation of a concept. This checks that an instance, defined as a one element signature (denoted by keywords `one sig`), is an instance of a concept. In the example below, the first `run` command, which states that Bauxite satisfies the qualities of RawMaterial, succeeds, but the second `run` fails, because it states that Bauxite is an instance of Good, which is inconsistent with goods being artificial, an opposite of natural.

```
one sig Bauxite in Object {}
fact {
  hasQuality[Bauxite,Natural]
  hasQuality[Bauxite,Deliverable]
  hasQuality[Bauxite,Storable]
  hasQuality[Bauxite,Tangible]
}
run {Bauxite in RawMaterial} for 10
run {Bauxite in Good} for 10
```

5) Ad hoc constraints can be checked, thanks to the power of FOL.

ALLOY provides a graphical view of models found by the `run` and `check` commands, which is quite handy to debug specifications. It also provides a text-based evaluator which can evaluate any term or formula on the found models, to further facilitate debugging and validation.

III. CONCLUSION

We have presented in this paper a lightweight approach for knowledge modeling. Our work is indeed similar to ontologies and description logic; we definitely need to further investigate the *distinctions* (`;-`) with our approach. We started this work with the idea of manipulating words and distinguishing them in simple ways, because that is what domain experts use when they talk about their domain of expertise: simple words representing key concepts. The versatility of @L-is and ALLOY allow us to quickly adapt our notation, rules and laws to deal with unexpected domain structures. The systematic use of distinction is crucial to pinpoint ambiguities, redundancies and contradictions. Mathematical concepts like sets and relationships are indeed useful, but end-users do not reason in terms of sets and relations; usually only computer scientists do. This is why we introduce them in the last steps, to benefit from the expressiveness of first-order logic and its model checking tools. This is a key distinction between our approach and description logic and ontologies.

Establishing distinctions between words is our first guiding principle for building concept models. These distinctions can be reasoned about in two ways. First by using laws of the @L-is calculus on words; this can be simulated using the @L-is interpreter; then a first-order logic, set-theoretic view of concept models can be developed and checked using ALLOY. Such a distinction-based and verification-assisted knowledge modeling process seem to be a good trade-off between formalism and pragmatism.

Based on these method and tools, we are current initiating the development a consent management system in health care services in the Sherbrooke area in Québec. Electronic health records (EHR) are seen as a key element for improving the cost, efficiency and quality of health services. Patient *consent* is required by law before health records can be stored and exchanged between health actors. Each time a component (*eg*, a family doctor or an hospital) needs access to a patient's health record stored in another component, it must obtain consent from the patient before the information is transferred. At least five different laws, written at different times, with different objectives, address consent management, privacy and confidentiality of EHR, at both the federal and provincial level in Canada. We plan to distill key concepts of consent management expressed in these laws, then to determine which ones are manageable within the consent management system and which ones must be managed using a proper medical process. For instance:

1) A patient's written consent is required before using his health record for research studies; this consent must be "free and enlightened": how to precisely define these two key concepts?
2) The generic concept patient is insufficient to deal with emergency treatment for which a written consent is not required from patients (and in several other cases in fact).
3) How to distinguish between implicit consent and explicit consent, different levels of granularity in consent management, and so on.

REFERENCES

[1] Baader, F. *et al* (Eds): *The Description Logic Handbook*, Cambridge University Press (2007).

[2] CMMI for Services, Version 1.2, Software Engineering Institute, Carnegie Mellon University, USA.

[3] Hai H. Wang *et al*: Reasoning Support for Semantic Web Ontology Family Languages using ALLOY, *Multiagent and Grid Systems* 2 (2006) 455–471.

[4] Jackson, D.: *Software Abstractions*. MIT Press (2006)

[5] Maynier, J.: Modélisation des liens sémantiques des mots d'un langage métier, Master thesis, Université de Sherbrooke, Faculté des sciences, Sherbrooke, Canada, December 2003.

[6] Noy, N. F., McGuinness, D. L.: Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, March 2001.

[7] Spencer-Brown, G.: *Laws of Form*, Julian Press, New York, 2^{nd} édition (1972).

[8] Wikipedia: `http://en.wikipedia.org/wiki/Product_(business)`