Distinction-Based and Verification-Assisted Knowledge Modeling

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

Marc Frappier
Université de Sherbrooke, Canada, Marc.Frappier@USherbrooke.ca
Requirements Engineering

• Build mutual understanding between stakeholders

• Pitfalls – too often assume that:
  – key domain concepts are well understood
  – stakeholders share common definitions

• Basic concept definitions are overlooked
  – deemed too obvious to bother with
RE and Law

• Software engineers are not lawyers
  – Not trivial to translate the intent of a law into specific requirements
    • traceable
    • verifiable

• ex: Consent management in healthcare (Canada):
  – At least five different laws, written at different times, with different objectives, address consent management, privacy and confidentiality of EHR
Knowledge Modeling and Law

• Laws can be modeled, structured and abstracted, using software engineering techniques
  – to simplify domain understanding for software engineers
  – to build a bridge between the legal domain and the software engineering domain

• It is very complex to model the whole text and regulations:
  – We choose to focus on essential knowledge conveyed by Basic Concepts
Distinction-Based Domain Modeling

• Assumptions:
  – Clear-cut distinction is prior to definition
  – Symbols are the shortest mean to connect Meanings with total precision:
    • This is what we call “Formalization”

• Formalization allows engineering of Meanings that are computable by a machine
Calculus of distinctions

• Based on *Laws of forms* - *LoF*, of George Spencer Brown:
  – *LoF* is a formal calculus that can be interpreted as Boolean Logic
  – *LoF* was extended by F. Varela to deal with 3-valued logics
  – We extended *LoF* to deal with elements (numbers, words), bunches of elements, types of elements, and mappings
How to make Distinctions in accordance with LoF?

- A Distinction can be made by instantiating a Distinction Pattern:
  - In a Distinction Pattern,
    - the drawn boundary represents the distinction
    - the 2 drawn mutually exclusive sides represent the 2 indications:
      - The inside represents the indication (atomic)
      - The outside represents the counter-indication
    - the link, encompassing the indication and the counter-indication, identify the Distinction as a whole
Distinction Patterns in action! (1)

Instanciations

- **Language**
  - Language_Natural
    - (= Natural Language)
  - Language_Artificial
    - (= Artificial Language)

  \[ X : (\text{Language\_Natural}) \mid X : (\text{Language\_Artificial}) \]

  A Language is either Natural or Artificial, but not both

- **Object**
  - Object_Natural
    - (= Raw material)
  - Object_Artificial
    - (= Artefact)

  \[ X : (\text{Object\_Natural}) \mid X : (\text{Object\_Artificial}) \]

  An Object is either Natural or Artificial, but not both

Pattern

**DP6: Opposite Attribute Predicate (OAP)**

\[ T \]

\[ T_A \]

\[ T_{[A]} \]

\[ X : (T_A) \mid X : (T_{[A]}) \]

A = Natural; [A] = Artificial; T = Language

A = Natural; [A] = Artificial; T = Object
Distinction Patterns in action! (2)

Instanciations

Living

\[ (U \sqsubseteq (\text{Child}_T) = T \sqsubseteq (\text{Parent}_U) \]

\[ \text{Living} \odot = \text{Living} \]

Note: Identity_{\text{Living}} and Living (Being) are identified

Present

\[ (U \sqsubseteq (\text{Past}_T) = T \sqsubseteq (\text{Future}_U) \]

\[ \text{present} \odot = \text{present} \]

Pattern

DP7: irreflexive Function Inversion (IFI)

\[ X \sqsubseteq (F^\alpha Y) = Y \sqsubseteq (F_X) \]

Note: Identity_M may be identified with M

F = Parent;
F^\alpha = Child;
Id = Id_{\text{Living Being}};

F = Past;
F^\alpha = Future;
T = Present;
Graphical presentation

• We use « UML-like » notations for explaining distinguished words-meaning relations to IT people:
  – « A picture is worth a 1000 words »
Typing modelling

- **Object**
  - **Raw Material**
  - **Artefact**

- **Object**
  - **Object_Natural** (= Raw material)
  - **Object_Artificial** (= Artefact)

X : (Object_Natural) | X : (Object_Artificial)
Inverse Associations modeling

\[ \text{Child} \rightarrow \text{Parent} \]

\[ \text{Id}_{\text{Living}} \]

\[ (U \subseteq \text{Child}_T) = T \subseteq \text{Parent}_U \]

\[ \text{Living} \circ = \text{Living} \]

Note: Identity_{\text{Living}} and Living (Being) are identified
Finally: What is a Distinction?

• A Distinction is a single intentional thought that arrives embodied in two mutually incompatible ideas:
  – Distinction Making is a conscious activity of human beings:
    • It produces a clear-cut and well definable indication in the actor’s language
• A Distinction is mental:
  – It must not be confused with its drawing
Distinction-Based Reasoning

- Describing concepts using formulas and operators (symbols)
- Reasoning about concepts to validate definitions
- Calculus on words-meaning is conducted
  - by substituting and replacing into language constructs
    - well defined indications by the body of their definition
  - By example:

  \[
  \begin{align*}
    \text{father\_bart} & \approx \text{homer} \\
    \text{mother\_bart} & \approx \text{marge} \\
    \text{parent} & \approx \text{father, mother} \\
    (F, G)_X & \approx (F_X), (G_X) \\
    \text{parents\_bart} & \approx (\text{homer, marge})
  \end{align*}
  \]
Calculus of Distinctions
Operators

Boolean expressions

A : B  reads  "A is a B"
A || B  reads  "A and B are disjoint"

Terms (word expressions)

[ A ]  reads  "the opposite of A"
A_B  reads  "A has quality B"
A | B  reads  "A or B"
A & B  reads  "A and B"
Properties of \textit{IsA}

\textit{transitivity}

\[ \alpha_1 : \alpha_2 \land \alpha_2 : \alpha_3 \Rightarrow \alpha_1 : \alpha_3 \]

WorkProduct : Artefact and Artefact : Object

\Rightarrow

WorkProduct : Object
Opposite attributes

natural = [artificial]

natural *is the opposite of* artificial

natural *can be substituted by* [artificial]

and vice-versa
Disjointness

\[ \alpha \parallel \beta \iff \forall x \cdot \neg (x : \alpha \land x : \beta) \]

two types are disjoint iff they have no common subtypes

\[ \alpha \_\beta \parallel \alpha_{[\beta]} \]

having opposite qualities makes two concepts distinct

RawMaterial = Object_natural

Artefact = Object_artificial  \hspace{1cm} \textit{imply} \hspace{1cm} \text{RawMaterial} \parallel \text{Artefact}

natural = [artificial]
Combining qualities

Service = Product_(intangible & nonStorable);

A service is an intangible and non-storable product

Good = Product_(tangible | storable)

a good is a tangible or storable product

Are services and goods distinct?
Application and IsA

• $A_B : A$
  – An A with quality B is an A

• Service = Product_(intangible & nonStorable)
  – A service is a product
Reasoning about combinations

\[ [\alpha \& \beta] = ([\alpha] \& \beta) \mid (\alpha \& [\beta]) \mid ([\alpha] \& [\beta]) \]

Case analysis rule: the opposite of being \( \alpha \& \beta \)
is being at least the opposite of either \( \alpha \) or \( \beta \)

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>[\beta]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>( \alpha &amp; \beta )</td>
</tr>
<tr>
<td>( [\alpha] )</td>
<td>( [\alpha] &amp; \beta )</td>
</tr>
</tbody>
</table>
Why not use plain set theory?

\[ \alpha \cap \beta = \overline{\alpha} \cup \overline{\beta} \]

The most common de Morgan's law in set theory reduces three cases to two overlapping cases.

We simply use a less common law

\[ \overline{\alpha \cap \beta} = (\overline{\alpha} \cap \beta) \cup (\alpha \cap \overline{\beta}) \cup (\overline{\alpha} \cap \overline{\beta}) \]
Complement of qualities

\[ [\alpha_\beta] = \alpha_\beta \]

The opposite of \( \alpha \) having quality \( \beta \) is \( \alpha \) having the opposite of quality \( \beta \)

It is a relative complement
Reasoning on services

[service] = \langle\text{definition of service}\rangle

\[\text{Product} \_\text{(intangible \& nonStorable)}\]

\[\text{Product} \_\text{[(intangible \& nonStorable)]}\]

\[\text{Product} \_\text{( ([intangible] \& nonStorable)}\]

\[\text{Flowware}\]

\[\text{Software}\]

\[\text{Hardware}\]
IsA based on qualities

\[ \beta_1 \& \beta_2 : \beta_1 \]
\[ \beta_1 : \beta_1 \mid \beta_2 \]
\[ \beta_1 : \beta_2 \Rightarrow \alpha \_ \beta_1 : \alpha \_ \beta_2 \]

distinction based on IsA

\[ \alpha : \beta_1 \land \beta_1 \parallel \beta_2 \Rightarrow \alpha \parallel \beta_2 \]
Distinctions on products

- **Product**
  - delivered

- **Good**
  - tangible & storable

- **Service**
  - intangible & nonStorable

  - **Hardware**
    - tangible & storable

  - **Flowware**
    - tangible & nonStorable

  - **Software**
    - intangible & storable
raw materials are natural objects
Set theoretic interpretation

• word = set
• _ = \cap
• [ ] = _ (* complément *)
• & = \cap
• | = \cup
• A : B ⇔ A \subseteq B
• A || B ⇔ A \cap B = \emptyset
Set theoretic interpretation

- Object\_natural = RawMaterial

can be seen as

"the set of objects that are natural are the raw materials"

\[ \text{Object} \cap \text{natural} = \text{RawMaterial} \]
Validation of models using Alloy

- Alloy is symbolic model checker for first-order logic with relations
  - FOF encoded into propositional formula
  - reuses common SAT solvers
  - only two data types
    - signatures (to define basic types)
    - finite subset of the integers
  - Object-oriented in style
What Alloy can do for us

• verify the consistency of models
  – check that definitions contain no contradiction

• check properties of models
  – state properties and check that they are entailed by the definitions
Conclusion

• Calculus of words
  – Words are indications in distinctions of a domain
  – Simple operators intended to represent and manipulate concepts of a domain

• Reason about words
  – Confirm distinctions
  – Check consistency with Alloy
  – Make deductions based on assertions about words