From Software Engineering to Linguistic Engineering

Laurence Ayache & Pierre Crescenzo & Jean-Louis Paquelin

ALLI
I3S - LIM
CNRS - UNSA

ayache@lim.univ-mrs.fr & Pierre.Crescenzo@i3s.unice.fr & jlp@alli.lpl.univ-aix.fr
Plan

Goals

Focus

Elements of solution
Introduction
Goals: Modelization

- Near from the theory to strictly respect its semantics
- Near from the real world to naturally handle data

\[
\text{plus}(\text{succ}(\text{succ}(\text{succ}(\text{zero}))), \text{succ}(\text{succ}(\text{zero}))) = \text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{zero}))))))
\]

\[
3 + 2 = 5
\]
Goals: Expressiveness (1)

- Higher-level concepts

\[
\begin{align*}
\text{print_integers_tree} & \ (\text{An_integers_tree}) \\
\text{print_reals_tree} & \ (\text{A_reals_tree}) \\
\text{print_trees_tree} & \ (\text{A_trees_tree}) \\
\ldots & \\
\downarrow & \\
\text{print_tree} & \ (\text{A_tree, An_element_printing_routine})
\end{align*}
\]
## Goals: Expressiveness (2)

- **Active constraints**

<table>
<thead>
<tr>
<th>Passive constraints</th>
<th>Active constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X &lt; 4$?</td>
<td>$X &lt; 4!$</td>
</tr>
<tr>
<td><strong>Success:</strong> $X$ is free</td>
<td><strong>Success:</strong> $X &lt; 4$</td>
</tr>
<tr>
<td>$X \geq 2$?</td>
<td>$X \geq 2!$</td>
</tr>
<tr>
<td><strong>Success:</strong> $X$ is free</td>
<td><strong>Success:</strong> $2 \leq X &lt; 4$</td>
</tr>
<tr>
<td>$X = 7$?</td>
<td>$X = 7!$</td>
</tr>
<tr>
<td><strong>Success:</strong> $X = 7$</td>
<td><strong>Failure:</strong> $2 \leq X &lt; 4$</td>
</tr>
</tbody>
</table>
Goals: Expressiveness (3)

- Query language
- Logic relations
- Type relations

A noun is a word.
A verbal phrase is at least constituted of a verb.
Goals: Evolutivity

- To add concepts or to follow linguistic-theories evolutions
- To correct Anima’s imperfections
- To be used as test bed

new language

\[
\begin{align*}
C & \quad \longrightarrow \quad C++ \\
\text{evolution} & \quad \Rightarrow \quad C^2
\end{align*}
\]
Goals: Integration, Introspection and Control (1)

- Tools (debugger, profiler, ...) must be at Anima’s level

<table>
<thead>
<tr>
<th>C++ (source level)</th>
<th>C (target level)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>O = new Circle;</code></td>
<td>if(!(_23 = (T_42 *)malloc(sizeof(T_42) + 15))) {</td>
</tr>
<tr>
<td></td>
<td>_cpp_errno = _NONEW_12;</td>
</tr>
<tr>
<td></td>
<td>_internal(_ERROR_PRINT, _NONEW);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logic (source level)</th>
<th>Prolog (target level)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p(a) ∨ p(b) → p(c)</code></td>
<td><code>p(a) :- p(c).</code></td>
</tr>
<tr>
<td></td>
<td><code>p(b) :- p(c).</code></td>
</tr>
</tbody>
</table>
Goals: Integration, Introspection and Control (2)

- Programmers’ tools have to exist as well as users’ ones

<table>
<thead>
<tr>
<th>Programmer level</th>
<th>User level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger</td>
<td>Inspector</td>
</tr>
<tr>
<td>Profiler</td>
<td>Statistics</td>
</tr>
<tr>
<td>Controller</td>
<td>Tracer</td>
</tr>
</tbody>
</table>
Goals: Integration, Introspection and Control (3)

- Control meta-level is desirable

**Diagram:**

- **External Agent**
- **Meta-level control**
  - forces
  - $X$’s type
- **Processing**
  - $X$
  - unknown type
  - $X$
  - noun
Goals: Software Engineering (1)

- Legibility
  - Natural syntax
  - Users’ knowledge respect
  - Other Software-Engineering principles respect

- Reliability
  - Validity
  - Robustness
Goals: Software Engineering (2)

- Extensibility
  - Easy adaptability on changing
- Reusability
- Compatibility
  - Interfacing ability
- Performance
Goals: Software Engineering (3)

- Portability
- Verifiability
  - Easier collection procedures
  - Easier guarantee procedures
  - Easier proving
  - Easier testing
  - Easier debugging
Goals: Software Engineering (4)

- Integrity
  - Components (programs, data, documents, ...) protection
- Usage simplicity
  - Easy Anima learning and operating
  - Easy results interpretation

→ Cause and consequence of all other Software-Engineering principles
Focus: Prolog

- Not very legible but NLP-widely-used syntax
- No type system and sometimes no constraint solver
- Not very configurable and extensible
- No appropriateness with recent grammatical theories
- No control meta-level
- In relation to software engineering: not very maintainable, a bit reusable (but not reused), reliable
Focus: LIFE

- Complex and not very legible syntax
- Good expressiveness and higher-level concepts
- Existing type system and constraint solver
- Not very configurable and extensible
- Enough appropriateness with recent grammatical theories
- Disused and not very performant interpreter and compiler
- No control meta-level
- In relation to software engineering: not very maintainable, a bit reusable (but not reused), not reliable
Focus: Oz

- Disconcerting but clear syntax
- Good expressiveness and higher-level concepts
- Existing inadequate type system but good constraint solver
- Configurable and extensible but at C++ level
- No appropriateness with recent grammatical theories
- No control meta-level
- In relation to software engineering: enough maintainable, a bit reusable (but not reused), reliable
Elements of solution: Modelization

- Good expressiveness
  depends on higher-level concepts
  The most important: constraints

- Encapsulation
  provides good control and autonomy

→ Constrained objects
Elements of solution: Expressiveness

Good expressiveness relies on

- Higher-level concepts
- Active constraints
- Query language
- Logic relations
- Type relations (Inheritance link, Cliente link, ...)

and is implemented by

- Objects
- First-order discipline
Elements of solution: Evolutivity

- Meta-Object Protocol
  - Basic-mechanisms transparent box
  - Basic-mechanisms modification ability
Elements of solution: Integration, Introspection and Control

Analysis + Time + Work

= 

Debugger, Profiler, 

\[ \downarrow \]

Integration + Introspection + Control
Elements of solution: Software Engineering (1)

- Legibility
  - Encapsulation
  - Overloading
  - Polymorphism
  - Modularity

- Reliability
  - Specification definition and implementation
  - Designers’ experience and appraisal
Elements of solution: Software Engineering (2)

- Extensibility
  - Modularity
  - Autonomy

- Reusability
  - Of existing languages
  - Relies on object technology
Elements of solution: Software Engineering (3)

- Compatibility
  - Objects as exchange standard
  - Autonomy
- Performance and Portability
  - Well-known solutions
- Verifiability
  - Well-known and well-tried algorithms
  - Helped with good tools (debugger, profiler, . . .)
Elements of solution: Software Engineering (4)

- Integrity
  - Robustness
  - Designers’ talent

- Usage simplicity
  - Application-domain (NLP) knowledge

But few direct control
Conclusion and Future

Many existing partial solutions but no integration

Integration under control of Software-Engineering through a coherent multi-paradigm language: Anima