Bird Technology ™

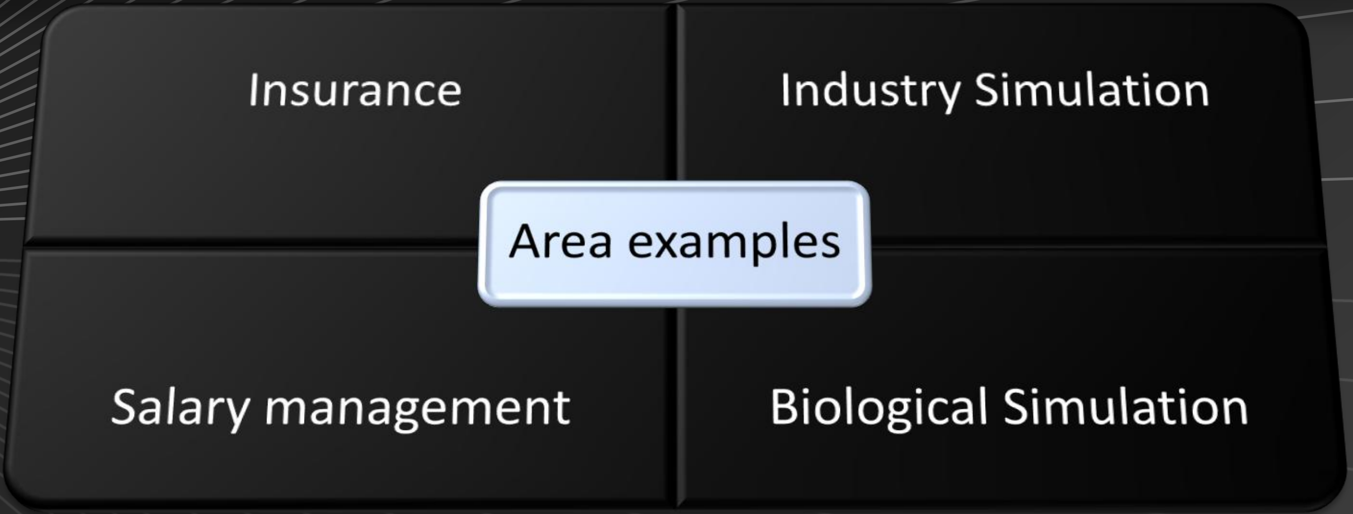
DSL, the absolute weapon for the development

AS2 – Bird Technology – September 2010

Definition

We can identify two types of languages: General Purpose Languages, such as: Smalltalk, Java or UML and Domain Specific Languages DSL.

A Domain Specific Language is conceived with a syntax and a semantics adapted to a category of activity or an approach to manage specific data. It does not have vocation to solve problems apart from this context.



Actual Methodology

Goal

Objective: To industrialize the software production process and maintenance.

Means: methods, technical solutions and tools being able to be applied in a systematic way to the resolution of new problems.

- Analysis Methods
- Modular development
- Advise for write good code
- Application Generator
- Tests, Debuggers, Tracers

Actual Methodology

New Problems

- Market Reactivity,
- Reduce development time,
- Complex Software,
- Reduce time for Debug and maintenance,
- Debug and maintenance more and more complex,
- Aggressive competitors,
- Quality and Reliability,
- Quality process since design at the final delivery.

New Solutions ?

Goal: Specific software from generic software.

Mean: Adapt, almost automatically, software action by provide a lot of parameters.

DSL

The Solution

Definition: Programming language dedicated to a domain or particular problem, providing suitable abstractions and notations as bases construction.

Characteristic

- More expressive compared to traditional languages
- More declaratory than imperative
- The checking of properties allows

Actual Situation

Difficult Development

- Suitable abstractions or notations?
- Declaratory formulation: XML interchange format or palliative of static languages?
- Which orientation to choose: what to make? Versus how to make?

Difficult Reusability

- Library creation or the library already exist?
- Is re-use guided by constructions of the language, convincing with the static languages?
- Domain field captured implicitly by the implementation or explicitly by parameterization?
- Checking: can one prove that the program carries out what I want?

Why use DSL

- Capture expertise on the domain in a really dedicated syntax.
- Easier programming because syntax is reserved for a precise field. Semantics is restricted and fast to memorize.
- Short phase of training, not of bookshop of heavy classes to know.
- What to make? Versus How to make? finally solved, the expression of the need finds a correspondence in semantics.
- Systematic re-use, because semantics contains the expertise.
- Easier and especially possible checking up to the level of the formal proof.
- Separation of the concepts to allow the checking of coherence.

Integration in existing Information System

DSL is not very intrusive in an information system. However thanks to the platform Visualworks and our Semantic Middleware, we can integrate without disturbance the applications, incorporate the existing data according to functional specifications of the domain and create the types of data representing best possible topology of the data of the domain.

The added value of Bird Technology

When we evoke the delivery of a dedicated language, we mean that we are able to deliver an interpreter and a compiler high performance integrating the language dedicated in question.

This is why we speak about R-DSL: Real Domain-Specific Language within Bird Technology. We also have the tools of data bases able to face the whole of the problems of WEB 3.0.

Human Factor - Team Organisation

One can distinguish two categories of developers.
The developers which are pleased to program low level in C/C++, Java.
The developers which prefer more productive syntaxes like 4GL, Visual BASIC, etc…
While thinking only of one DSL oriented Word, one can entrust to the first category the design and the realization of the words at the technical level and the second category the use of the dictionary.

If one follows the evolution of the data-processing languages, true languages, at the conceptual level we can, very schematically, see languages resulting from the theory of the lambda calculus and the oriented objects languages, simplification of the lambda calculus.
In both cases, one can reasonably say that the adoption by the developers is weak. Also, it is to better see these concepts as tools to define DSL which gradually by factorization are transformed into BNL.