How to simplify software development with high level programming languages ?

Pierre-Alexandre Voye - ontologiae@gmail.com

Projects structures

- Both in proprietary and open source project, steps are the same. The difference is usually in the centralized or decentralized approach :
    * Needs analysis
    * Requirements and architecture
    * Detailed design
    * Implementation
    * Unit Test
    * System verification and validation

Recurrent troubles in software development

- The key problem in open source and proprietary project is the frequent back action steps due to troubles which appears in different steps.
- Reason, the key problem is the language semantic.

The architectural design usually take in mind the paradigm of the language which will be used : procedural, object oriented, functionnal, logic, ...

Detailed design

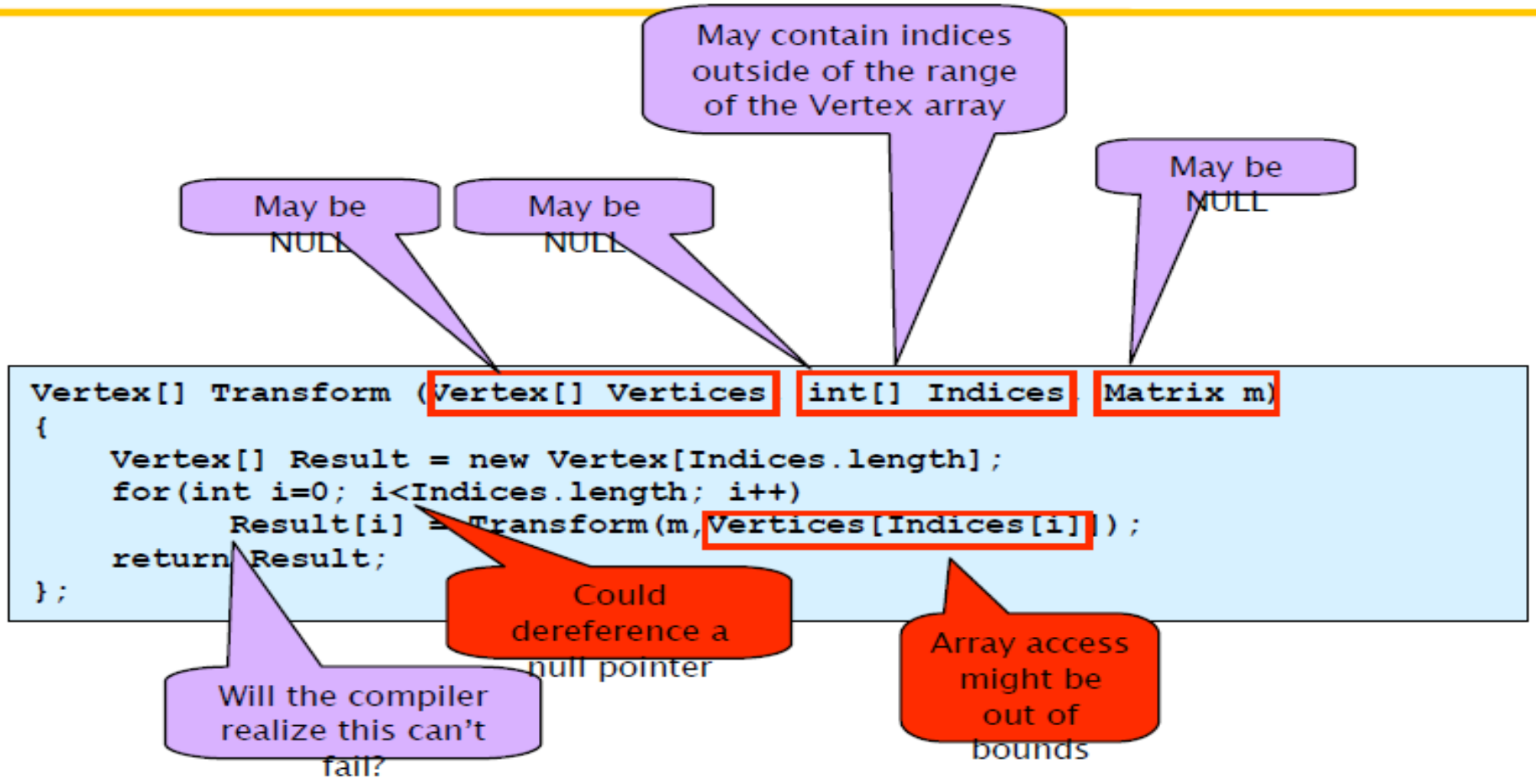
- Explicit the "business rules" or internal logic schemas.
- Procedural and object oriented language are both operational semantic language, which modify the current state of the memory of the computer, step by step.
  - So they have very few intrinsic abality to detect wrongs.
    - For instance for interraction between data structures and rules.
- There's usually troubles which will be discover in following steps, sometimes at the end of the cycle.

Implementation

- KEY IDEA : MORE WE HAVE CODE, MORE WE HAVE BUG

- Implementation : the big and true problems usually come at this step.
  For instance :
  - How implement a machine state ?
  - How making a valid specification when querying datas ?

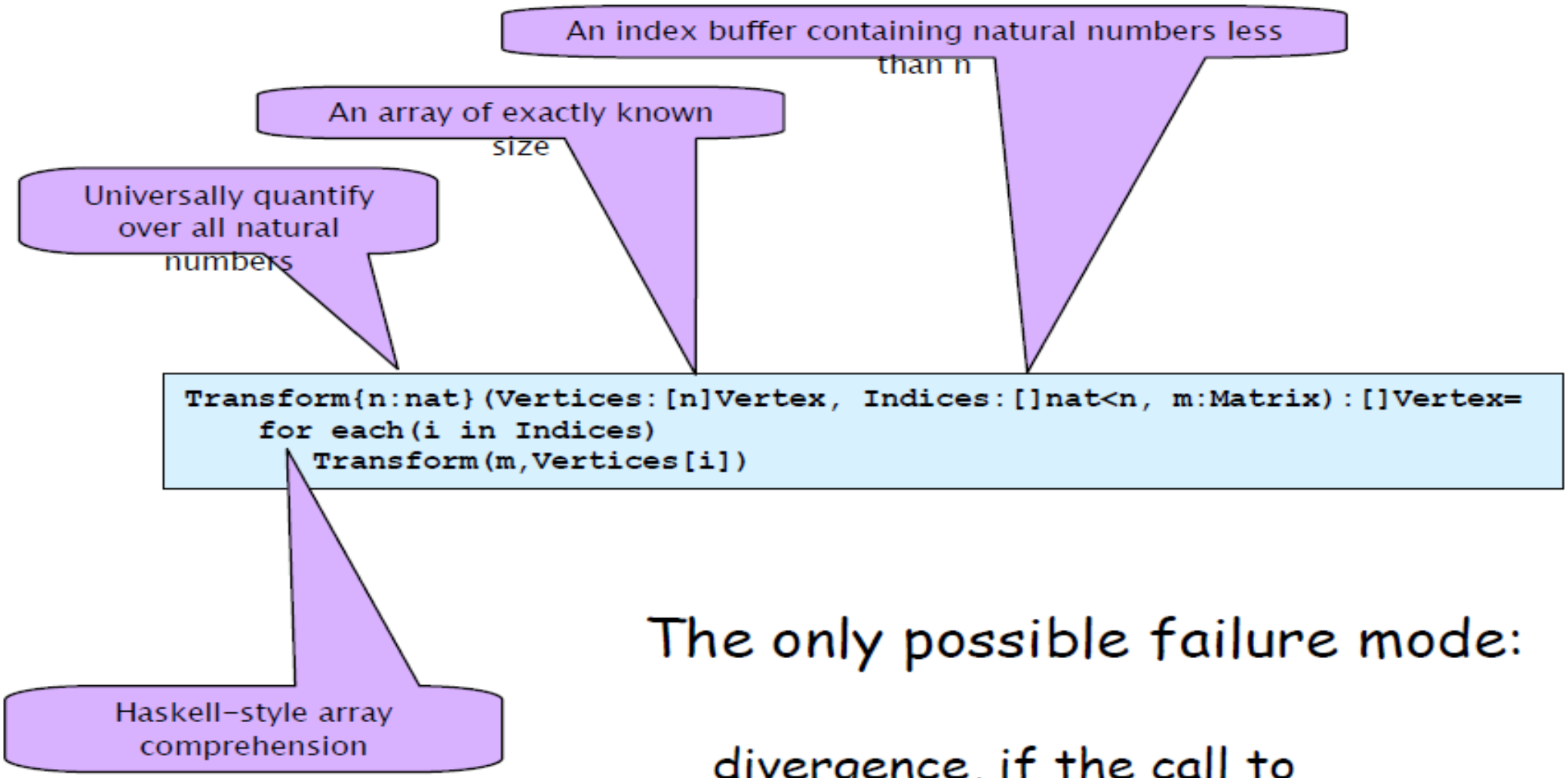
Dynamic Failure in Mainstream Languages



Our code is littered with runtime failure cases,

Yet the compiler remains silent!

What we would like to write...



The only possible failure mode:

divergence, if the call to Transform diverges.

Intrinsic datas constraints

- How to build in internal data constraint ?:
  - Sum type, in functionnal language like OCaml, haskell :
    type tree = Sheet of string | Children of (tree list);;
    - It's a pain when you map an xml grammar...
  - Datas are determined by others
    data1 = data2 + data3 ;
    We never write directly in data1

- Ok there are frameworks, but...

Example with Low/High level language for low level programing :
Lisaac and C

Low level constraints

- Speed
- Memory management
- Data structure mapping, interruptions, ...
- Usually no debug messages (not enough memory for that, or too low level nature of language used : segfault, etc...)...

C problems

- Big functions because of the procedural design => more bug
- Casting is extremely error prone, and debugging it isn't easy since we have no clear message
- Lot of troubles with pointer : String management, each complex data structure involve pointer which implies potential problems
- No complex data structure : no linked list, no hashmap, etc...
- No call on null detected at compile time (but it's true we have tools)
- Programmers have to be extremely aware about memory managment : a little error may have extremely catastrophic consequences.
- Making low level optimization makes the code unreadable.

An example with Lisaac, you can low level programming but :

- Complex data structures
- We can create our control structure (readability, smaller code size)
- We can return several values, which doesn't force the programer to code n function, but one.
- Magic container (to be implemented) : the compiler choose the best container for the job, giving constraint (performance, size, ...)
- Genericity (in the object and in the method (C++ template))
- Ability to include C and thus ASM code in Lisaac
- 4 type of inheritance : more flexibility.
- Multiple and dynamic inheritance

An example with Lisaac, you can low level programming but :

- Extremely strong typed language : the compiler is extremely rigorous about type security, like OCaml. So : NO CAST !! And thus information at compile time.
- No manual memory management, so less bugs.
- The compiler optimize itself the memory (mark and sweep GC).
- 30~40 % line code less, so less bugs
- Contract programming (specification and debug).
- Readability : keyword in function
- **So code is less error prone**

Future, paradigm change needs ?

Detailed design

Prolog is an old logic paradigm language, but unhappily only constraint centric.

It's emerging from less than ten years with a new approach :
Rules based programming.
More interresting when combined with agent approach.

Implementation

Data filtering : integrated query language (C# LINQ, Jaque : http://code.google.com/p/jaque/)

```
r = from(data,
        where( { Integer i => i > 5  },
                    orderBy( { Integer i1, Integer i2 => i1-i2 },
                                groupBy( {Integer i => "a"},
                                   orderBy( { Group<String, Integer> g1,Group<String, Integer> g2
                                                => g1.getKey().compareTo(g2.getKey()) },
                                            select( { Group<String, Integer> g => 4 } )
                                               )
                                          )
                                )
                    )
        );
```

Implementation

Agent and rules oriented programming :

- Datas which are contraint by others
- Verification of status
- Integration of time dimension.
  - (F-Logic, TOM, FLORA2)

But how to be accessible for the average programmer ?

Conclusion

The meaning of the History is to give more and more tasks to the compiler, which is becoming an AI.

Questions ?


ISAAC
Object Prototype


Rencontres Mondiales du Logiciel Libre