ODOG

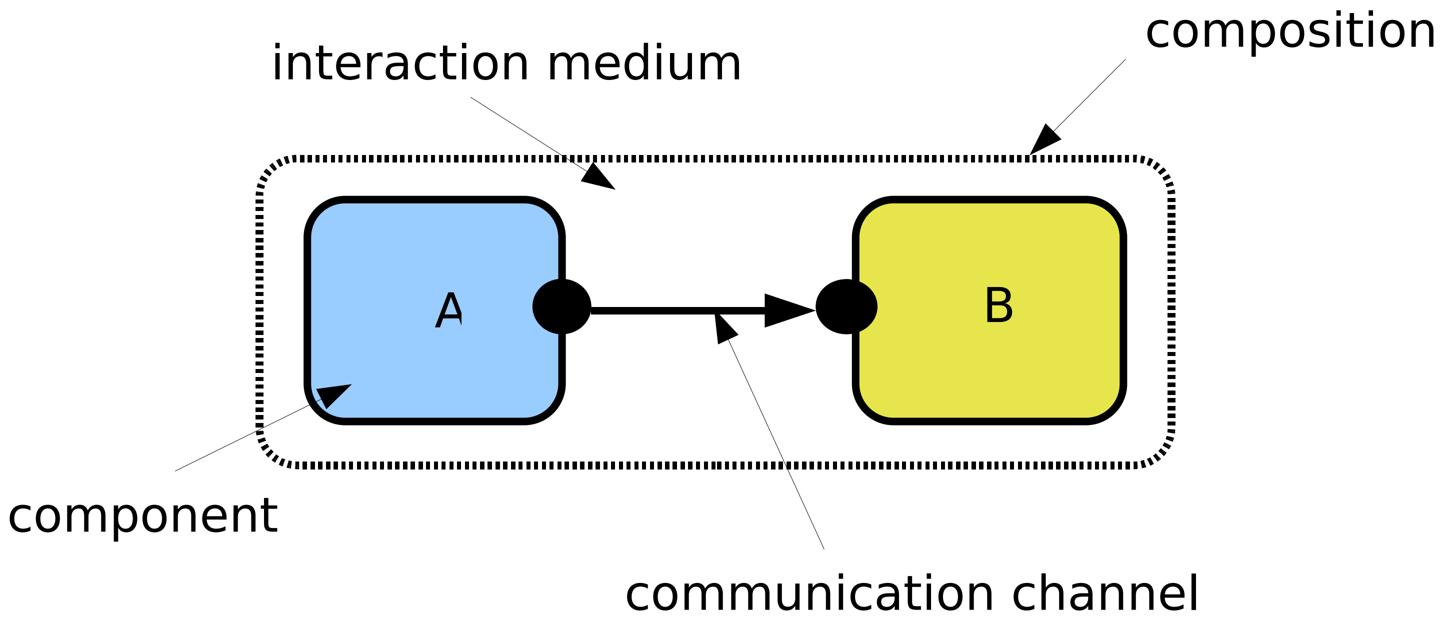
Open source project
for
Programming Concurrent Platforms

odog.sourceforge.net

ivan.jeukens@gmail.com

Concepts

- Component abstraction
- Composition of components



- Two different kinds of languages
  - for the composition (coordination)
  - for the components (host)

Concepts (2)

- Coordination language
  - syntax : block oriented
  - semantics : *interaction semantics*
- Host language
  - programming languages
  - domain specific languages
- What can I do with such a model ?
  - generate an executable code
- “Simulation” or actual code production

Concepts (3)

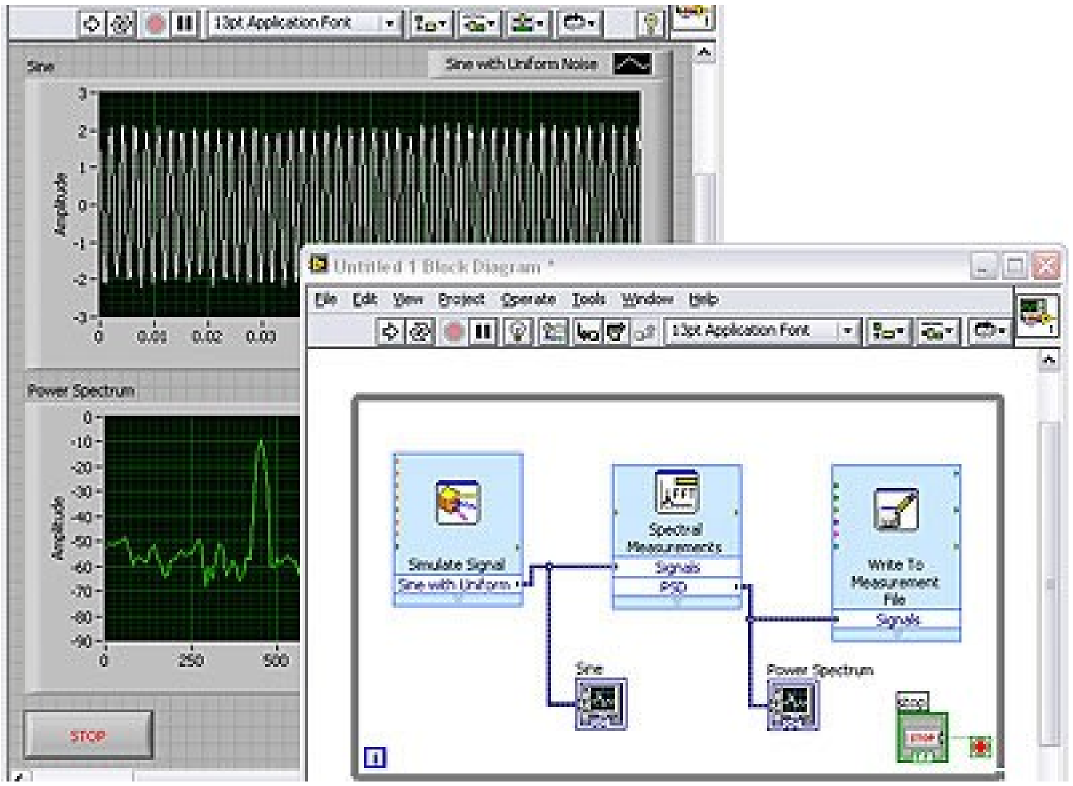
- What are the advantages ?
  - Create programming abstractions more natural to the application domain
    - more productivity
    - better understanding
    - better optimization opportunities
  - Better semantics (no non-determinism, or a controlled one)
  - Possibility to target to any desired platform, hiding the details from the programmer

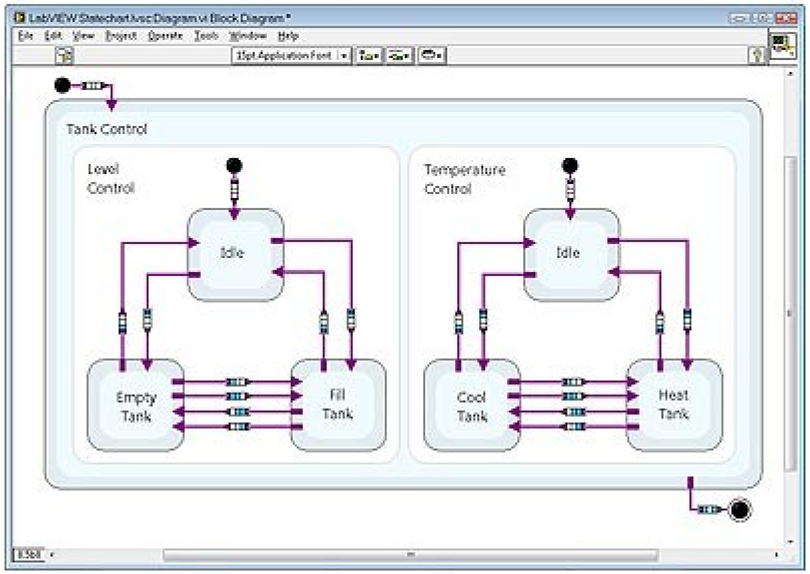
Applications

- Any platform / architecture that has a form of concurrency
  - there are tons....
- Examples
  - interaction with “external” elements
  - multiple entities on the same computation resource
  - multiple computational resources

Related work

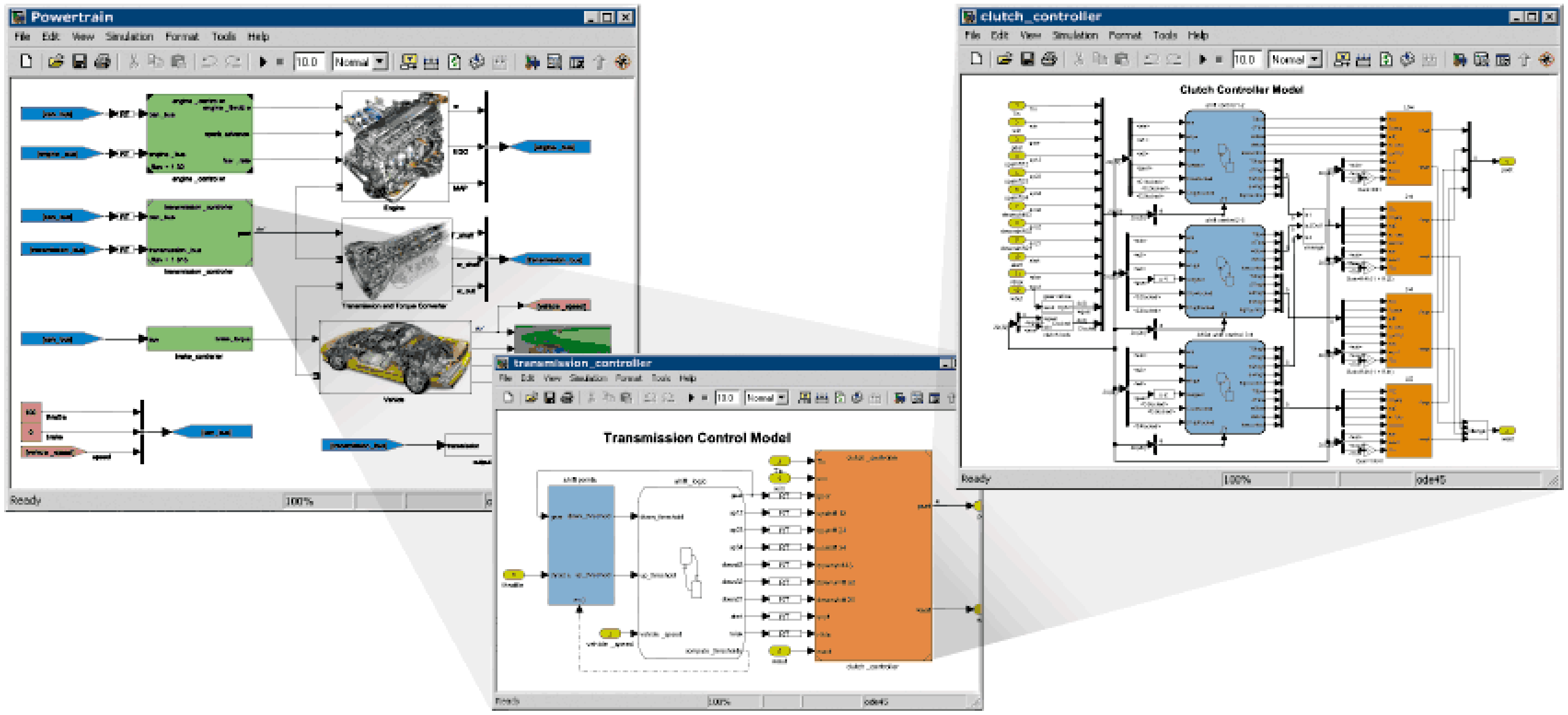
- Is this idea original ? No
- *Labview* - National Instruments




LabVIEW Statechart.lvsc Diagram.vi Block Diagram *
File Edit View Project Operate Tools Window Help
Tank Control
Level Control
Temperature Control
Idle
Idle
Empty Tank
Fill Tank
Cool Tank
Heat Tank

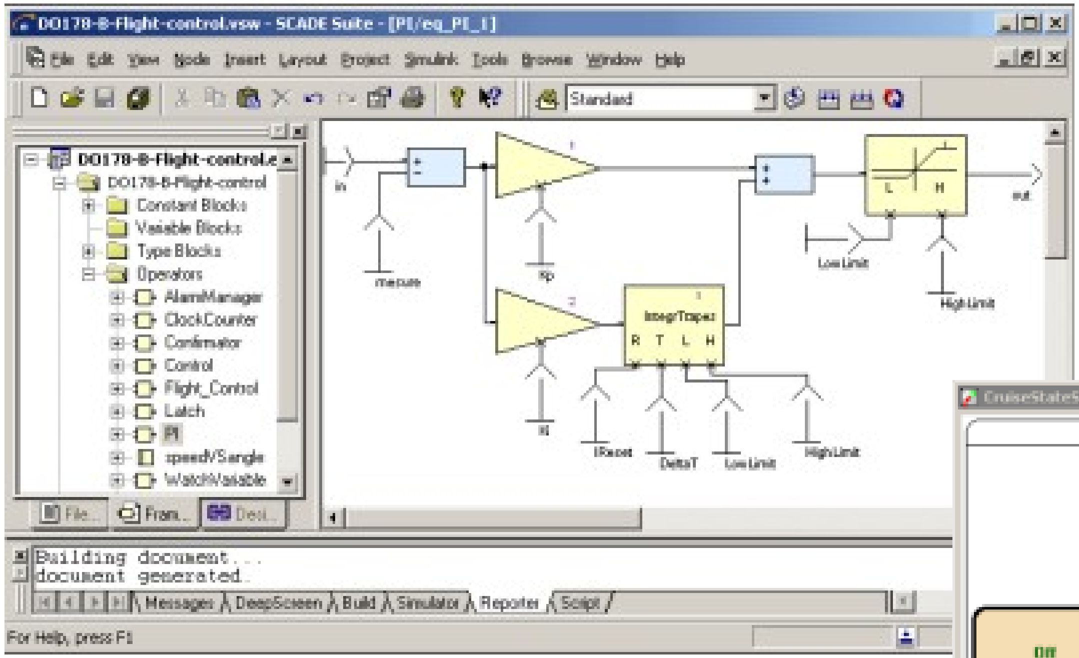
Related Work (2)

- *Simulink* - Mathworks

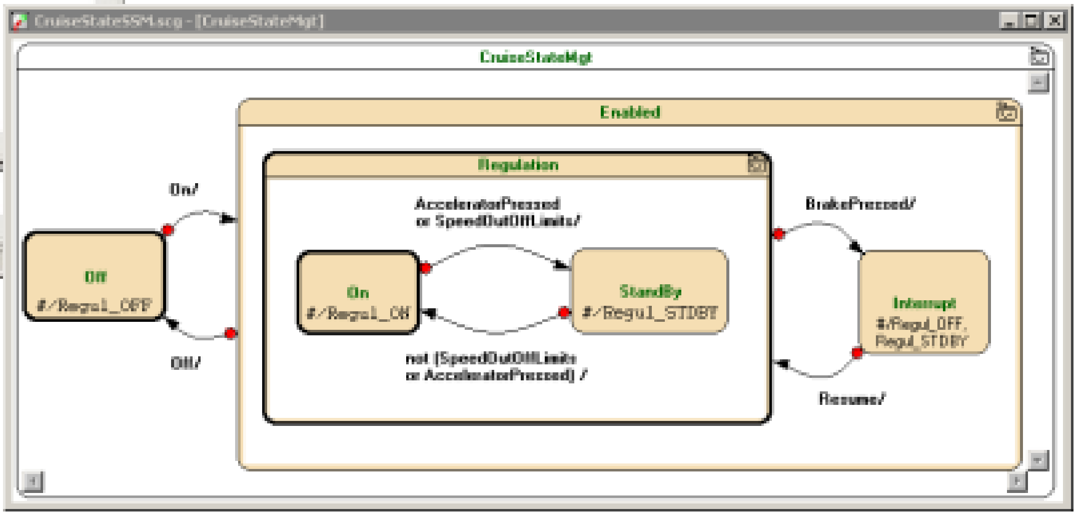


Related Work (3)

- *Scade* – Esterel Technologies

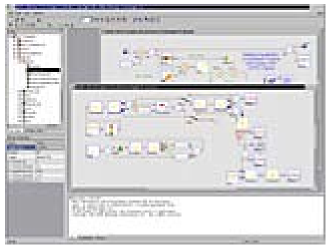
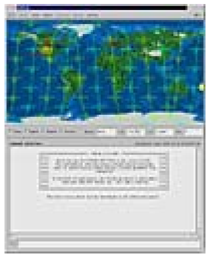
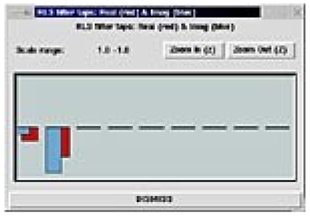


```
Building document...
document generated.
```



Related Work (4)

- *ML Designer* – MLDesign

- … and there are others
- What about academic open source ?
  - several small initiatives, not as a unifying project
  - GME - Vanderbilt (Vanderbilt license)
    - www.isis.vanderbilt.edu/projects/gme/
  - Ptolemy II – UC Berkeley (Berkeley License)
    - ptolemy.eecs.berkeley.edu

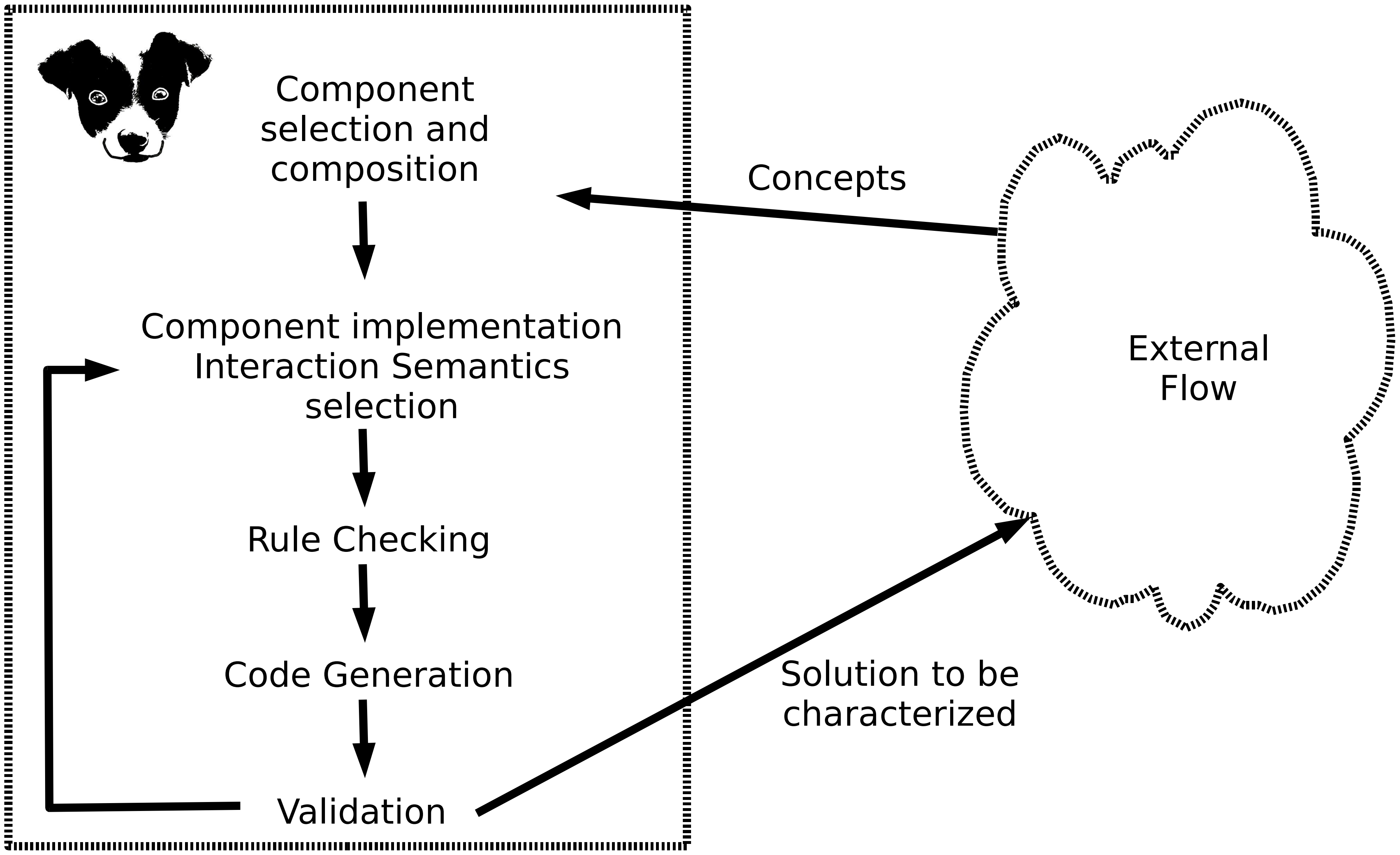
ODOG

- Requirements
  1. support any interaction semantics
  2. support any programming language
  3. easily generate efficient code for the system being described
  4. have a strong validation flow
  5. be embeddable within design methodologies
  6. be open source
- Common solution
  - class libraries for syntax and semantics
  - not good for 2, 3

ODOG's Solution

- Simple
- Tree structure for the abstract syntax
  - XML
- Template based code generation for interaction semantics
  - Avoid the need for (difficult and error prone) code analysis and transformation
  - code “as good as it gets”

Standard ODOG Flow


Component selection and composition
Component implementation Interaction Semantics selection
Rule Checking
Code Generation
Validation
Concepts
External Flow
Solution to be characterized

ODOG v1.0

- 3 Interaction semantics
  - Discrete Events (DE) : simulation
  - Dataflow (DF): data stream processing
  - Synchronous (SR): software based on response to events
- 2 Platforms
  - host : generates code for your machine so you can test your ideas
  - multicore : generates code for DF exploiting parallelism
- GUI for editing and Rule Checker for validation

Applications

- Embedded systems
  - simulation models for the environment
  - implementation and code generation for the software and hardware
- Programming multi-core systems
    - x64 based
    - OMAP architecture (not available due to NDA)
- Planned for the near future
  - Linux module programming
    - device drivers : event-oriented with complex synchronizations

Final Remark

ODOG is customizable

odog.sourceforge.net

ivan.jeukens@gmail.com