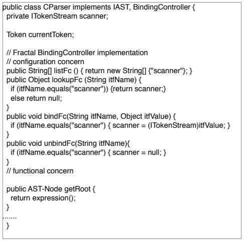
A Smalltalk implementation of EXIL, a Component-based Programming Language

Petr Špaček
in cooperation with
Christophe Dony, Chouki Tibermacine and Luc Fabresse

LIRMM,
University of Montpellier 2
`petr.spacek@lirmm.fr`

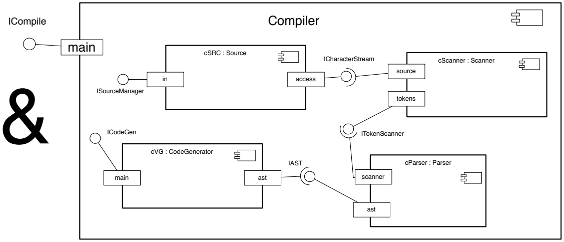
23rd of August, 2011

WHAT WE ARE DOING? - MOTIVATION



```
public class CParser implements IAST, BindingController {
  private ITokenStream scanner;

  Token currentToken;

  // Fractal BindingController implementation
  // configuration concern
  public String[] listFc () { return new String[] {"scanner"}; }
  public Object lookupFc (String itfName) {
    if (itfName.equals("scanner")) {return scanner;}
    else return null;
  }
  public void bindFc(String itfName, Object itfValue) {
    if (itfName.equals("scanner") { scanner = (ITokenStream)itfValue; }
  }
  public void unbindFc(String itfName){
    if (itfName.equals("scanner") { scanner = null; }
  }
  // functional concern

  public AST-Node getRoot {
    return expression();
  }
.......
 }
```

&



- Combine a modeling (architecture description) language and a programming language

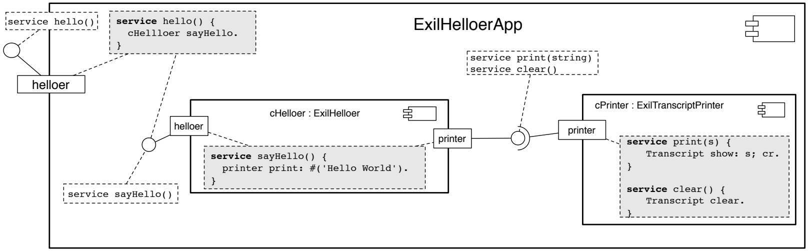
What we are doing? - Approach

- Our approach: components
- Applying component-paradigm into a programming language
- With such a language:
  - design components - design for reuse
  - design applications using components - design by reuse

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"

Szyperski C. Component software: beyond object-oriented programming. 2nd ed., Reading, MA: Addison-Wesley; 2002

IN GENERAL



```
service hello() {
  cHellloer sayHello.
}
```

```
service sayHello() {
  printer print: #('Hello World').
}
```

```
service print(s) {
    Transcript show: s; cr.
}

service clear() {
    Transcript clear.
}
```

- Explicit external contract with an environment
  - requirements - what is demanded from the environment
  - provisions - what is offered to the environment
- Explicit architecture

SCL - OVERVIEW 1

- Scl - Simple Component-oriented language
- Invented by Luc Fabresse (presented in ESUG'06)
- EXIL extends Scl towards to a modeling language

```
(SclBuilder new: #SclHelloer
      category: 'Scl-Examples-HelloWorld')
      requiredPorts: #(#Printer);
      providedPorts: {(#Helloer->#(#sayHello ))}
```

SCL - OVERVIEW 2

- Component
  - Black box
  - Ports described by interfaces
  - Provides and requires services
- Port
  - Unidirectional interaction point
  - Plug
- Service
  - Functionality
  - Like a method or a set of methods
- Interface
  - Describes the valid uses of a port
  - Service signatures sets, protocols, contracts, ...

Exil- Overview

- Component = instance of descriptor
- Reusable interfaces
- Ports
  - described by list of services or by interfaces
  - roles
    - provided
    - required
- Connection
- Internal components

```
interface ICompile {
      compile(source)
}

component descriptor Parser extends AbstractParser {...}

component descriptor Compiler {
  provide {
    main->{compile(source)}
    //or main->ICompile
  }

  require { }

  internalComponents {
    cVG->CodeGenerator;
    cParser->Parser;
    cScanner->Scanner;
  }

  internalConnections {
    connect cParser.scanner to cScanner.tokens;
    connect cVG.ast to cParser.ast;
  }

  service compile(source) {
    (cScanner port: source) setSource: source.
    (cVG port: main) getCode.
  }
}
```

EXIL- NEW FEATURES

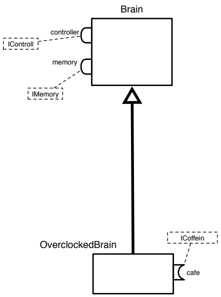
to support modeling

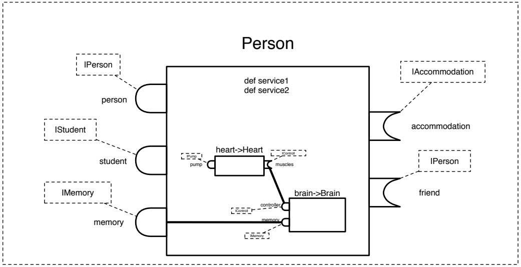
- Explicit architecture
  - extracting architecture from the code
- Inheritance
  - sub-descriptors: a descriptor may extend an another descriptor
  - extension and specialization of:
    - Ports
    - Services
    - Internal components & Connections

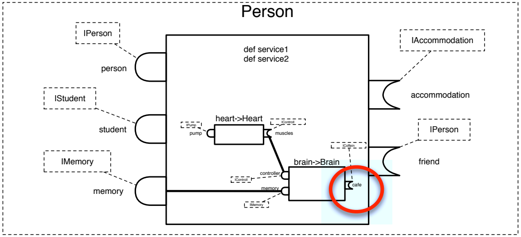
```
interface ICompile {
      compile(source)
}

component descriptor Parser extends AbstractParser {...}

component descriptor Compiler {
  provide {
    main->{compile(source)}
    //or main->ICompile
  }

  require { }

  internalComponents {
    cVG->CodeGenerator;
    cParser->Parser;
    cScanner->Scanner;
  }

  internalConnections {
    connect cParser.scanner to cScanner.tokens;
    connect cVG.ast to cParser.ast;
  }

  service compile(source) {
    (cScanner port: source) setSource: source.
    (cVG port: main) getCode.
  }
}
```

Exil- Inheritance

problem with additional requirements & substitution



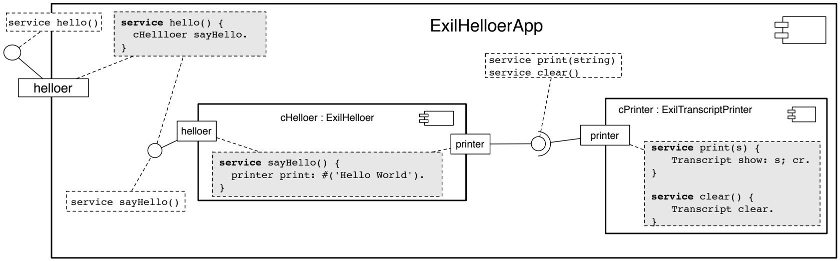




EXIL- IMPLEMENTATION

- EXIL parser uses *PetitParser* framework and *PetitSmalltalk* parser
- Compiler - visitor pattern
- Core
  - `ExilComponent` class
  - `ExilInterface` class
- image can be downloaded here: http://www.lirmm.fr/~spacek/exil (source codes - SqueakSource download is coming)

EXIL- LIVE EXAMPLE



```
service hello() {
  cHellloer sayHello.
}
```

```
service sayHello() {
  printer print: #('Hello World').
}
```

```
service print(s) {
   Transcript show: s; cr.
}

service clear() {
   Transcript clear.
}
```

EXIL- FUTURE WORK

- Reflexivity level - goal = write model analysis and transformations in EXIL
- Architecture constrains
- Visual development

SUMMARIZE

EXIL

- is a component-oriented language
- which merges modeling and programming
- and brings component-paradigm closer to the Smalltalk users

SUMMARIZE

EXIL

- is a component-oriented language
- which merges modeling and programming
- and brings component-paradigm closer to the Smalltalk users

Thank you