SCL : a Simple, Uniform and Operational Language for Component-Oriented Programming in Smalltalk

Luc Fabresse Christophe Dony Marianne Huchard

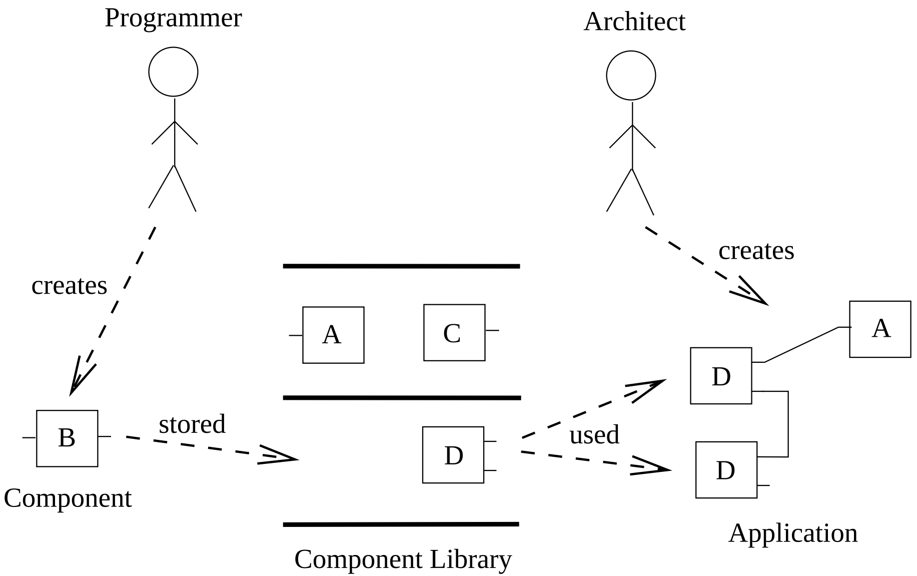
LIRMM
Université Montpellier 2
France

September 3, 2006

Outline

Outline

The High-Level View of CBSD



Current Issues in CBSD

Fundamental Questions

- What is a component? [Szyperski, 1996]
- Component structure?
- Component composition?

Practical Needs

- Models
- Languages
- Tools

Outline

Simple Component Language (SCL)

What is SCL?

A component-oriented programming language

Main purposes of SCL

- Simple, few and only fundamental entities
- Uniform, not an asymmetric extension of an object-oriented language
- Enables **unanticipated composition of independently developed components**
- Synthesis of existing component-oriented languages ideas
- Operationnal, implemented in Smalltalk
- Extensible to experiment new ideas

Basic Entities (1/2)

Component

- Black box
- Ports decribed by interfaces
- Provides and requires services

Port

- Interaction point
- Plug

Basic Entities (1/2)

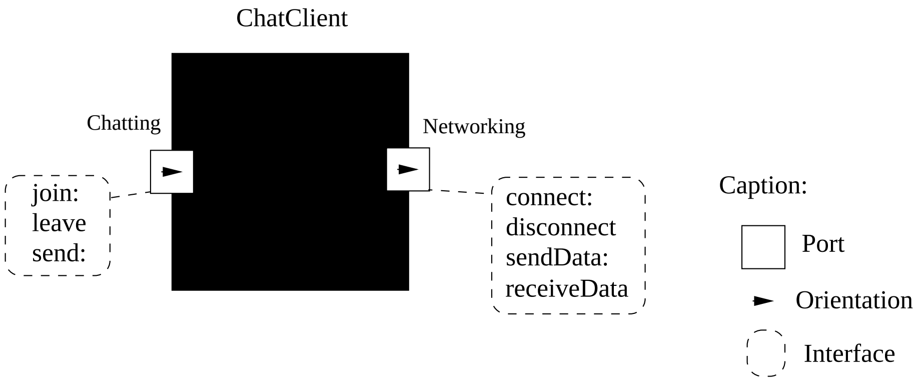
Service

- Functionnality
- Like a method or a set of methods

Interface

- Describes the valid uses of a port
- Service signatures sets, protocols, contracts, ...

Example : a chatclient component



The Connector entity

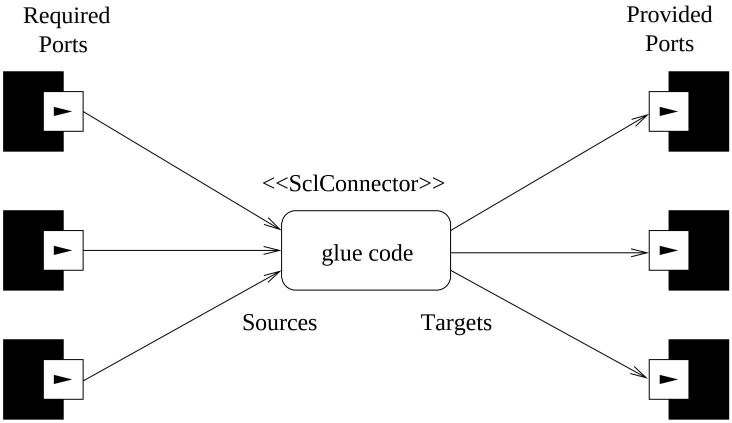
Connector = Reification of Connection

- Better separation
- Non-fixed connection semantics
- Solves adaptation problems
- Possible creation of a reusable library of connectors

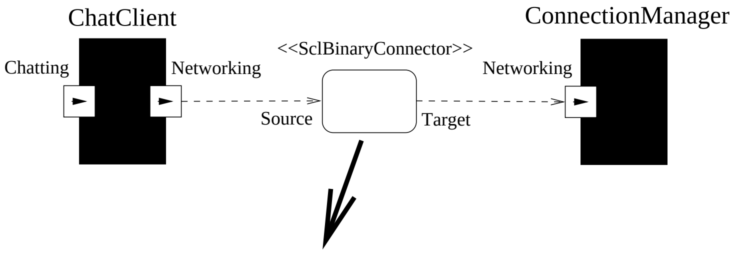
The General Connector Structure

- Sources, a set of ports
- Targets, a set of ports
- Glue code

A Graphical View of a Connector



Example: Connection of two matching components

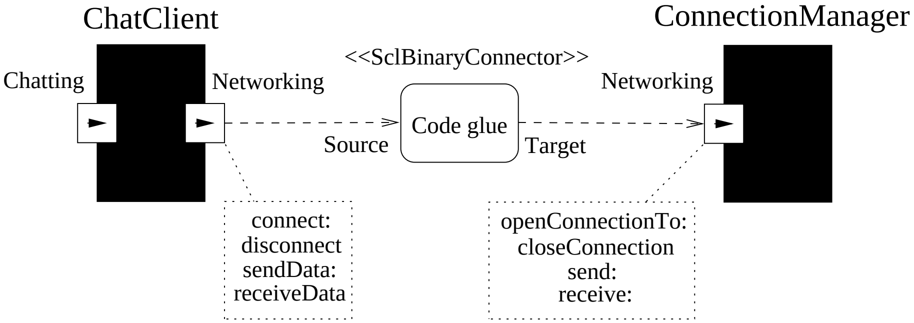


```
SclBinaryConnector new
    source: ( chatClient port: #Networking )
    target: ( connectionManager port: #Networking ) ;
connect
```

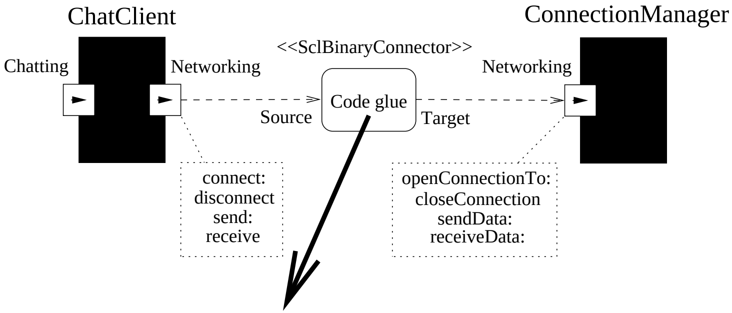
Similar to:

- bindings in Fractal
- connect primitive in Archjava

Example: Connection of two Components with Adaptation



Example: Connection of two Components with Adaptation



```
SclBinaryConnector new
    source: ( chatClient port: #Networking )
    target: ( connectionManager port: #Networking )
    glue: [ :source :target :message |
            (message selector = #connect:) ifTrue: [
                ˆ target openConnectionTo: (message arguments)
            ]
            ... "some more stuff"
          ] ; connect.
```

Mixing AOP and Component Appoaches

Why?

Encapsulate the scaterred code of specific concerns (Log, Transaction,...)

How?

- Asymmetric approaches
- Symmetric approaches

Limited AOP support in SCL

Joint Points on Ports

- before/after/around service invocation
- before/after/around connection/disconnection

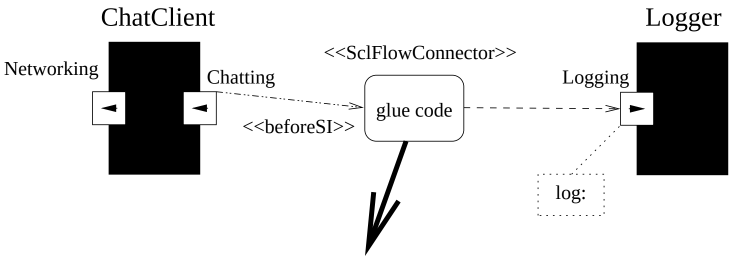
Special Connectors

Source ports are coupled with a keyword (beforeServiceInvocation, ...)

Weaving

Regular connection/disconnection mechanism

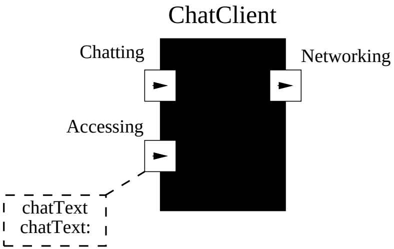
Example: A Logger component

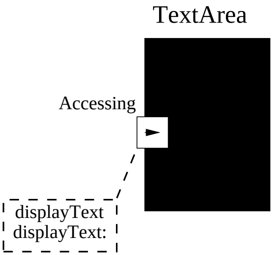


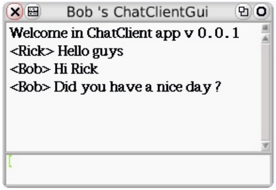
```
SclFlowBinaryConnector new
   source: ( (chatClient port: #Chatting) beforeServiceInvocation )
   target: ( logger port: #Logging )
   glue: [ :source :target :message |
           target log: (message selector asString)
         ] ; connect.
```

Motivating Example (1/2)

How to connect a chatclient component with a GUI component ?



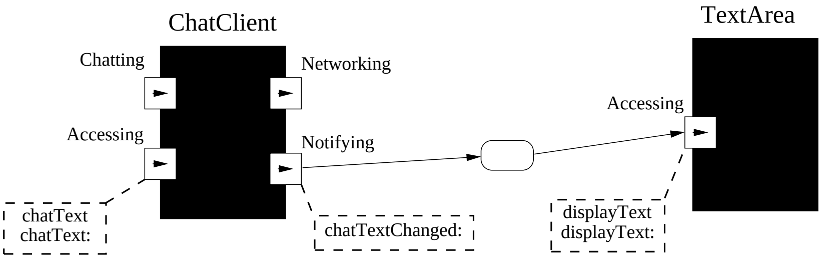


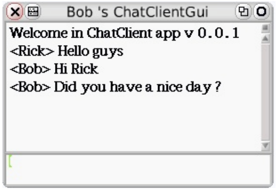


Welcome in ChatClient app v 0.0.1
<Rick> Hello guys
<Bob> Hi Rick
<Bob> Did you have a nice day ?

Motivating Example (2/2)

How to connect a chatclient component with a GUI component ?





```
Welcome in ChatClient app v 0.0.1
<Rick> Hello guys
<Bob> Hi Rick
<Bob> Did you have a nice day ?
```

The limitations with components

Publisher side limitations

- Event signaling (Archjava, CCM, ...)
- Subscribers management (Javabeans, ...)

Subscriber side limitations

- Receivable events (CCM, ...)
- Subscriber reaction

Goal

Extract the application dependent code from components

The SCL Solution Based on Properties

Property

- External state of a component (Javabeans)
- Automatically notifies its value changes if needed
- Only declared by the component programmer

Property Structure:

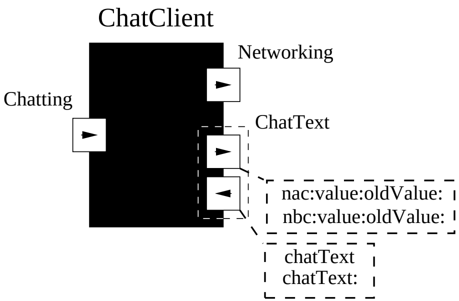
- A name
- An access port provides getter and setter services
- A notification port to invoke notifying services

Notifying services

- Notify Before Change, `nbc:value:oldValue:`
- Notify After Change, `nac:value:oldValue:`
- ...

Example: The chat client component with a ChatText property

```
SCLComponentBuilder create: #ChatClient
  properties: 'chatText nickName'.
  outPorts: 'Networking' inPorts: 'Chatting'.
```



States changes connections

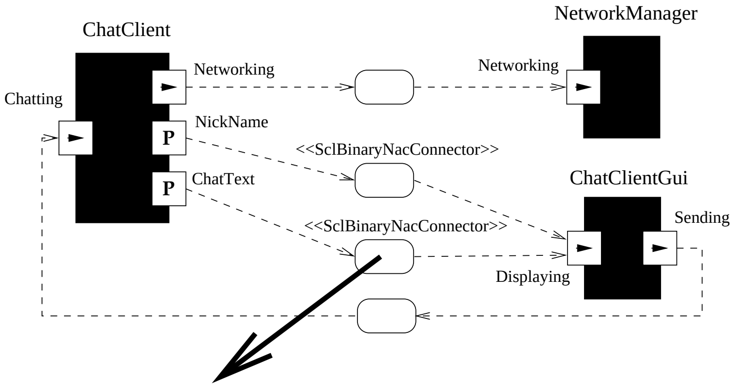
Standard connections

- Based on connectors (ports + glue code)
- Source ports are notifying ports of properties

Features

- Uniformity
- Extensibility

Example: The whole chat client application



```
SclBinaryNacConnector new
    source: ( chatClient notifyPortOf: #ChatText )
    target: ( chatClientGui port: #Displaying )
    glue: [ :source :gui :anInvocation |
            gui displayText: anInvocation arguments second
          ] ; connect.
```

Implementation of SCL

Advantages of Smalltalk

- Prototyping is easier and faster than in statically typed languages
- The meta-level enables message interceptions, addition of new entities (Component, Connector, ...), ...
- Block can be used for representing glue code
- A step to investigate what could be a dynamic component oriented language

Difficulty(ies) with Smalltalk

- Encapsulation

Outline

Conclusions

Scl

- A component-oriented language
- Only one entity of reuse: Component
- Component Composition
- Connections based on Connectors
- AOP support using Connectors
- Component Properties
- Prototyped in Smalltalk

Current Work

- Larger case studies
- Investigate dynamicity
- Improve the prototype
- Tools

Thanks...

Questions?
Suggestions?
Comments?