A Contracting System for Hierarchical Components

Fractal Workshop 2005, Grenoble,

29th november

Philippe Collet* – Roger Rousseau* - Nicolas Rivierre ** - Alain Ozanne **

* Université Nice Sophia Antipolis – I3S – CNRS Centre National de la Recherche Scientifique

** france telecom R&D

Outline

- Problem
- Principles of the *ConFract* system
- Examples of specifications and corresponding contracts
- Contract construction and checking
- Contract negotiation
- Conclusion

Problem: Mastering Complexity ?

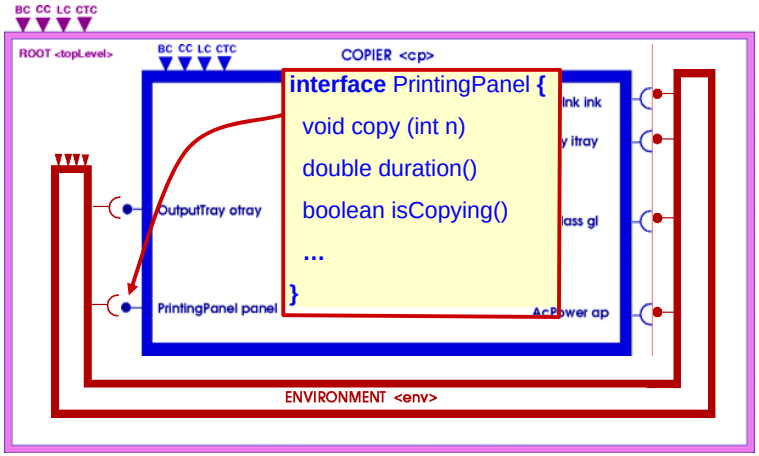
- Usually, contracts are either
  - Implicit between actors or
  - Some interpretation of a specification (responsibility, blame…)
- Adapting programming by contract to hierarchical components
  - Put contracts on bindings and components
  - Support incremental specifications
  - Distinguish specifications from contracts (and their mechanisms)
    - Contracts are configuration and run times objects. They follow dynamic reconfigurations
  - Determine responsibilities on components from those new contracts
    - To handle violations in an appropriate way

The ConFract system

Principle: Contracts are reified at Assembly Time

- Types of contract:
  - Interface contract
  - External composition contract
  - Internal composition contract
- Responsibilities:
  - Each specification becomes a provision of a contract
  - Components play the roles of **participants** in contracts
  - For each provision, participants have a well-defined role :
    - **Guarantor** : capable to act to ensure the provision
    - **Beneficiaries** : can count on the provision
    - **Contributors** : are needed for evaluation, but do not rely on it

External View of the Copier



Interface Specifications

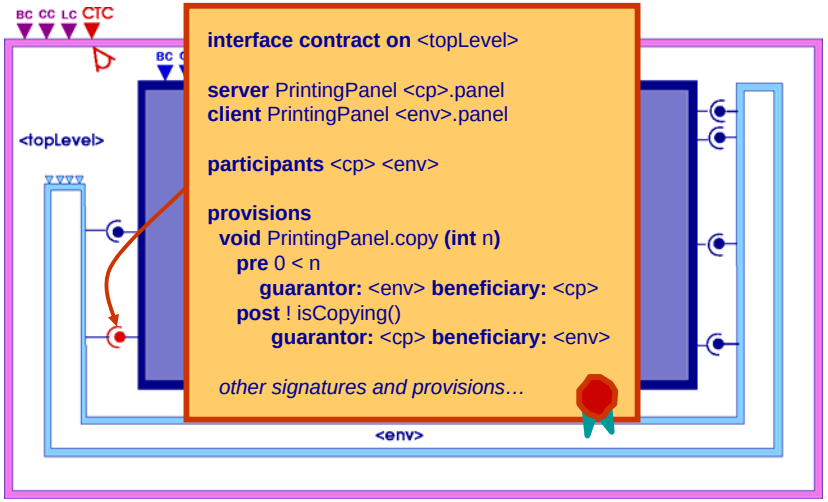
- Interfaces used by Fractal components

```
context void PrintingPanel.copy (int n)
          pre 0<n
          post ! isCopying()

context PaperTray
          inv tray().size() <= capacity()
```

Classic...

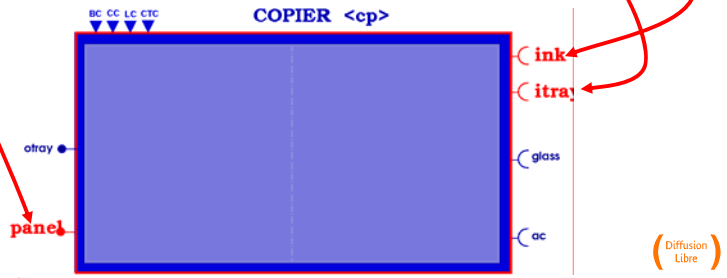
Resulting Interface Contract



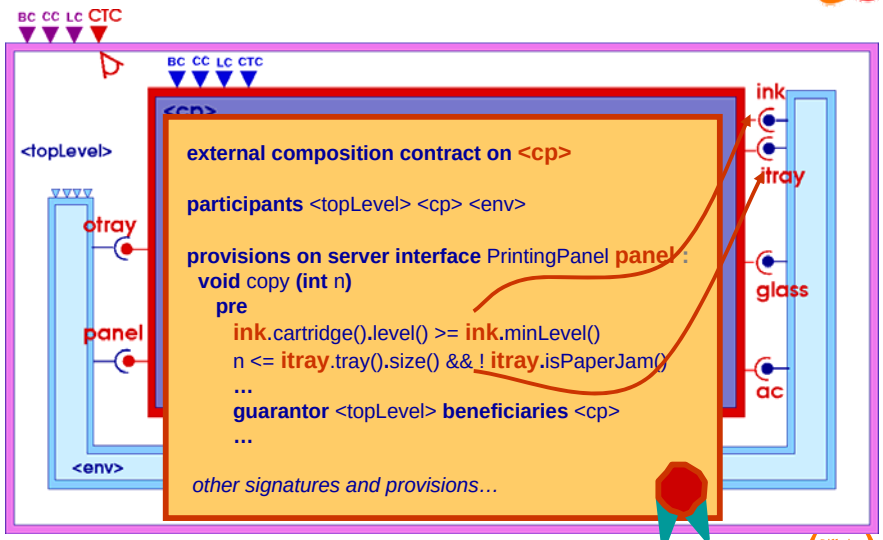
External Specification of a Component

```
on COPIER context void panel.copy(int n)

pre
  ink.cartridge().level() > ink.minLevel()
  n <= itray.tray.size() && ! itray.isPaperJam()
…
```

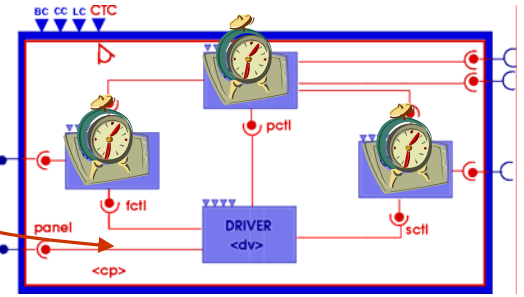


Resulting External Composition Contract



Internal Specification of a Component

```
on <cp> context <dv>.panel.copy(int n)
 post
  let (double thd = <sn>.sctl.duration() +
  n*(<pt>.pctl.duration()+ <fz>.fctl.duration())
    ) in thd-1 <= duration() <= thd+1
end on
```

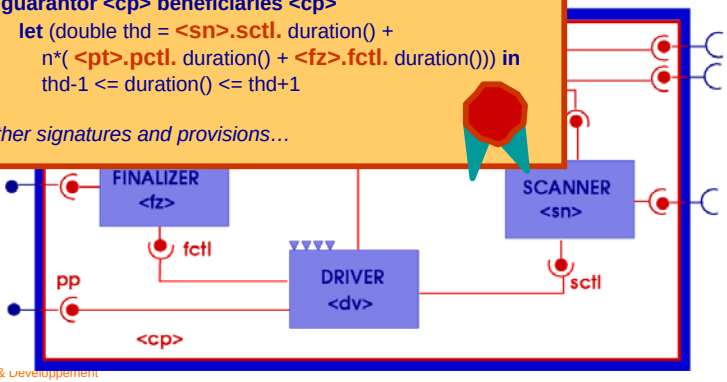


Internal Composition Contract

```
internal composition contract on <cp>

participants <cp> <dv> <fz> <pt> <sn>

provisions on server interface PrintingPanel <dv>.panel :
  void copy (int n)
    post
    guarantor <cp> beneficiaries <cp>
      let (double thd = <sn>.sctl. duration() +
          n*( <pt>.pctl. duration() + <fz>.fctl. duration())) in
          thd-1 <= duration() <= thd+1
...
  other signatures and provisions…
```



Progressive closure of contracts

- **Provisions** created wait for their participants,
- **Responsibilities** are filled in when new components come in,
- When all responsibilities are established, the provision is ***closed*** and ready to be checked
- Contracts **open** when a participant is not available anymore
  - e.g. during dynamic reconfigurations

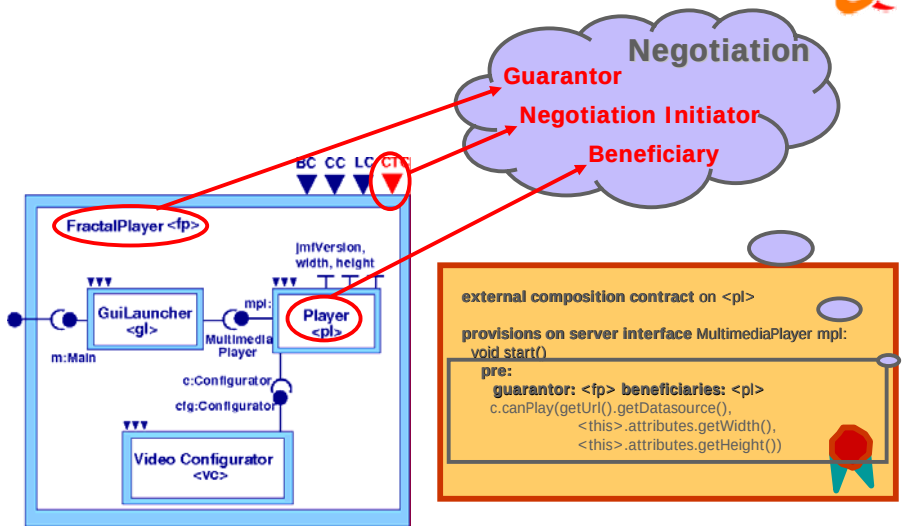
Contract Checking

- **At configuration time :**
  - Component **invariants** are checked (just before the component starts)
- **At runtime, on each Fractal interface:**
  - **Preconditions** are checked at method entry :
    - From the interface contract
    - From the external composition contract of the component receiving the call
    - From the internal composition contract of the surrounding component
  - **Postconditions** and **invariants** are checked in a similar way at method exit

Contract Negotiation

- **Motivation**
  - Handle contract violation,
  - Restore contracts validity
- **Means**
  - Dynamic negotiation,
  - Local to contracting parties,
  - Based on reflexive contracts,
- **Work in progress**
  - Integrate extra functionnal properties,
  - Propagate the negotiation

Contract Negotiation: example



Conclusion

- **ConFract**
  - Dynamic construction of contracts
  - Contracts are extended to compositional viewpoints (internal and external)
  - Responsibilities on each provision allow one to handle violations :
    - Negotiations
    - Dynamic reconfigurations, …
- **Current work**
  - Contract negotiation (concession, effort…)
  - Integration and contracting of different extra-functional properties
  - Integration of several specification formalisms (adapting and extending the underlying metamodel)

Questions

- **Some features not presented in this talk**
  - The dedicated assertion language : CCL-J
  - The integration following separation of concerns : the contract controller
  - Examples of dynamic reconfigurations
  - The metamodel