Algebraic component composition in the UML

Master's thesis presentation

Martin Harbu Bielecki

June 6, 2014

Master's thesis topic

Algebraic component composition in the UML

Master's thesis topic

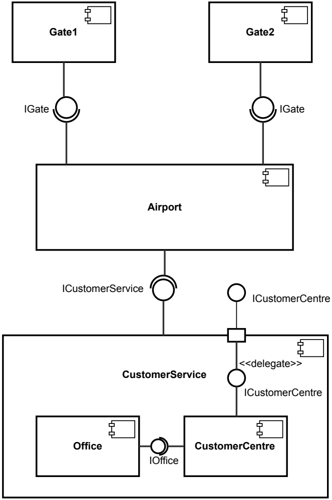
Algebraic component composition in the UML

What are software components?

- Software entity which encapsulates functionality
- A set of provided and required methods described by interfaces
- Major point: composable
- Composition by attaching required and provided interfaces

What are software components?

Component diagram



Master's thesis topic

Algebraic component composition in the UML

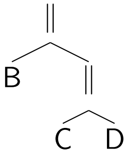
Algebraic component composition

$$\texttt{TextFieldWithButton} = \texttt{TextField} \parallel \texttt{Button}$$

- Define atomic components
- Compose them using an algebraic composition expression
- The result is *always a new composite component* with derived information

Algebraic component composition

The expression: $A = B \parallel C \parallel D$ is represented like this;



Master's thesis topic

Algebraic component composition in the UML

The Unified Modeling Language

- *De facto* standard for OO modeling
- Provides numerous different types of diagrams
- Extension mechanism: UML profile with stereotypes

Forms of UML models

- Two representations of an UML model
- A semantic model and a graphical model
- We say that the graphical model draws the semantic model
- Both will be demonstrated later

Master's thesis topic

Algebraic component composition in the UML

Component models

- Defines a standard and rules for composition
- Ensures compatibility
- Examples: OSGi, CORBA Component Model, rCOS and many others
- My prototype features: algebraic composition, UML modeling, model validation

Why?

Motivation

- The UML supports component modeling in the traditional sense
- Algebraic modeling approach: "take one or two components and apply a composition operator to get a new component"
- Problem: The UML does not support such compositions

The UML problem

- No bookkeeping details, does not store derived information
- "Plain" UML is not enough
- We want to know *which* components were used in a composition (i.e. the left and (right) operand and other arguments

Problem statements

1. How can the Unified Modeling Language (UML), extended with a UML profile, be used to support and represent models of textual algebraic component specifications?
2. What are the benefits of the algebraic component composition approach in component-based modeling?

Methods

- Literature studies
  - Algebraic composition proposed in *rCOS: Defining Meanings of Component-Based Software Architectures* by Dong *et al*.
- Prototype development
- Evaluation
  - Design decisions
  - Modeling examples (airport)
  - Comparing traditional and algebraic approach

The composition operators

The binary operators

- Parallel ($\|$)
- Disjoint ($\otimes$)
- Plugging ($\ll$)

- Variations of taking the union of the methods and the variables of the operands

The composition operators

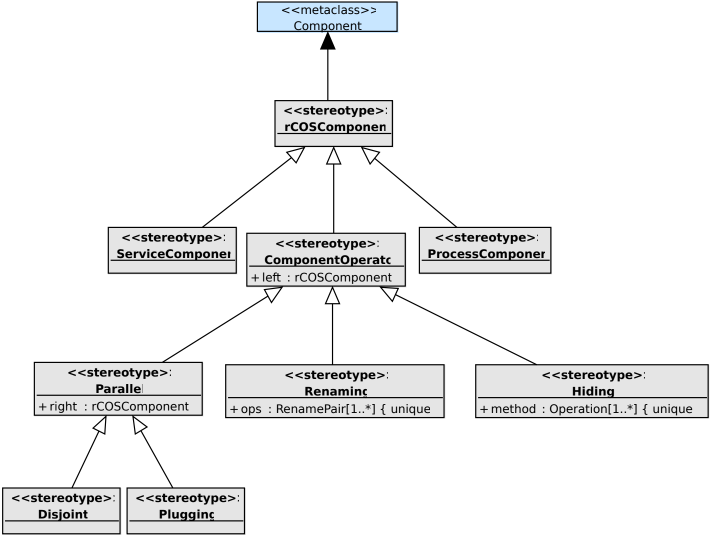
The unary operators

- Renaming ($C[oldname \leftarrow newname]$)
- Restriction/hiding ($C \backslash \{foo, bar\}$)

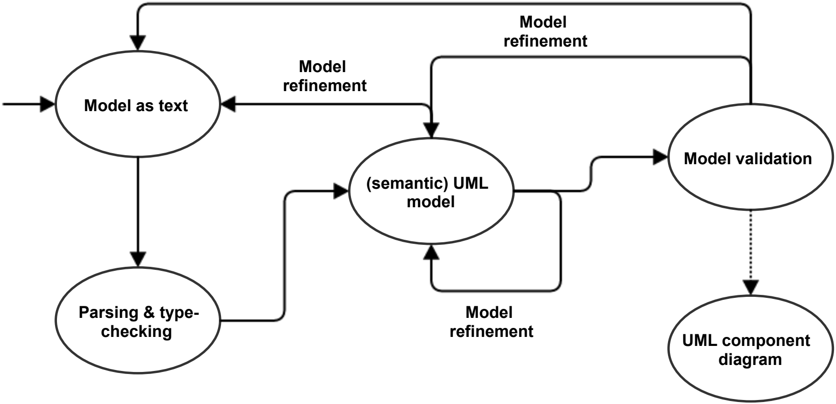
The prototype

- Extension of the rCOS tool
- Defines a DSL, called rCOSPN
- Defines a UML profile
- Can validate UML models using the OCL
- Plug-in for the Eclipse platform

The UML profile



The workflow



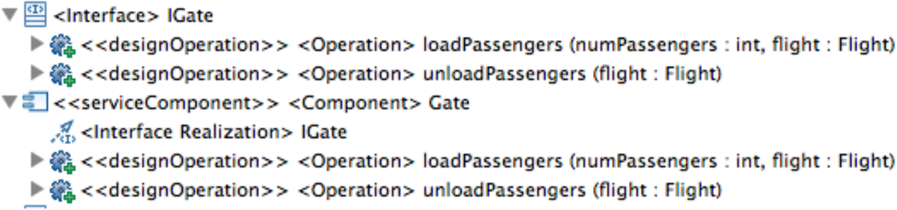
Prototype demonstration

Demonstration

Prototype demonstration

```
interface IGate {
    public loadPassengers(int numPassengers, Flight flight);
    public unloadPassengers(Flight flight);
}
component Gate {
    provided IGate {
        public loadPassengers(int numPassengers, Flight flight) {
            /* Method body */
        }

        public unloadPassengers(Flight flight) {
            /* Method body */
        }
    }
}
```

Prototype demonstration



```
▼ <Interface> IGate
    ▶ <<designOperation>> <Operation> loadPassengers (numPassengers : int, flight : Flight)
    ▶ <<designOperation>> <Operation> unloadPassengers (flight : Flight)
▼ <<serviceComponent>> <Component> Gate
      <Interface Realization> IGate
    ▶ <<designOperation>> <Operation> loadPassengers (numPassengers : int, flight : Flight)
    ▶ <<designOperation>> <Operation> unloadPassengers (flight : Flight)
```

Prototype demonstration

```
interface IOffice {
    public provideEmployee(;string employee);
}

component Office {
    provided IOffice {
        public provideEmployee(;string employee) {
            /* Method Body*/
        }
    }
}
```

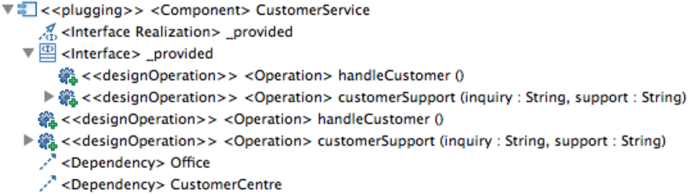
Prototype demonstration

```
component CustomerCentre {

    provided ICustomerService {
        public handleCustomer() {
            /* Method body */
        }
    }

    provided ICustomerCentre {
        public customerSupport(string inquiry; string support) {
            /* Method body */
        }
    }
    required IOffice;
}
```

Prototype demonstration

```
CustomerService = Office << CustomerCentre;
```

Prototype demonstration

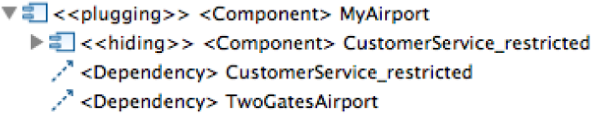


```
▼ <<plugging>> <Component> CustomerService
     <Interface Realization> _provided
  ▼  <Interface> _provided
        <<designOperation>> <Operation> handleCustomer ()
     ▶  <<designOperation>> <Operation> customerSupport (inquiry : String, support : String)
     <<designOperation>> <Operation> handleCustomer ()
  ▶  <<designOperation>> <Operation> customerSupport (inquiry : String, support : String)
     <Dependency> Office
     <Dependency> CustomerCentre
```

Prototype demonstration

```
component Airport {
    required ICustomerService;
    required IGate[2];
}

// COMPOSITIONS ---------------------------------------------------
GateRenamed = Gate[unloadPassengers <- unloadPassengers2, loadPassengers <- loadPassengers2];
OneGateAirport = GateRenamed << Airport;
TwoGatesAirport = Gate << OneGateAirport;
MyAirport = CustomerService \ {customerSupport} << TwoGatesAirport ;
```

Prototype demonstration



```
▼ <<plugging>> <Component> MyAirport
    ▶ <<hiding>> <Component> CustomerService_restricted
      <Dependency> CustomerService_restricted
      <Dependency> TwoGatesAirport
```

Discussion and conclusion

Recall the problem statements

1. How can the Unified Modeling Language (UML), extended with a UML profile, be used to support and represent models of textual algebraic component specifications?
2. What are the benefits of the algebraic component composition approach in component-based modeling?

Discussion

What are the alternatives to profiles?

- Sticking to plain UML
- Creating a new metamodel for UML

Discussion

- The UML profile provides a good solution for our modeling domain
- Simple
- Keeps track of what we need
- Able to interchange industry standard (the UML)
- Was able to build on the rCOS tool

Recall the problem statements

1. How can the Unified Modeling Language (UML), extended with a UML profile, be used to support and represent models of textual algebraic component specifications?
2. What are the benefits of the algebraic component composition approach in component-based modeling?

Discussion

- The algebraic approach is quite different from the traditional approach
- Everything is composable (if it conforms to the rules)
- Emphasizes small, reusable components
- Build composite components from these small building blocks
- The result is always a new independent component computed automatically

Limitations

- Composing two components with the same variable
- No explicit showing of methods that are “attached”

Conclusion

- The algebraic approach have several strengths
- Can be supported by the UML
- Unfortunately also some limitations
- Most people probably want graphical diagrams

Future work

- Extending the tool to graphical modeling
- Method bodies
- Code generation

Thanks for listening!