Consistency of Service Composition

1. **External**: between interface specifications
2. **Internal**: between interface specification and implementation
Example: Car Rental Service

Matching *provider* and *requestor* specification within *registry* must ensure compatibility of:

- **Data types**
  - Does Customer have the same meaning for requestor and provider?

- **Operation signatures**
  - Can provider operation be supplied with suitable parameters from a call of requestor operation?

- **Behavior**
  - Does provided operation actually carry out what is expected by a requestor?

Data Types and Signatures

- **Data types**: parties use common domain model (ontology)

Behavior: Operation Contracts

**Pre-condition:**
Customer provides rental info and selects car

**Effect:**
Car is reserved for customer

Required
- Formal specification (logic, graph transformation, ...) for automatic matching
- Integration into mainstream SW development methods (UML) for wider applicability

Outline
- Contracts as graph transformation rules
- Semantics of rules
- Semantic / syntactic compatibility, soundness

Contracts as Graph Transformation Rules

**Signature:**

reservCar(c:Customer, my_car:Car, ri:RentallInfo)

**Behavior:**

GT rule

**Data types:**
type graph

Typed DPO
[Corradini et al 96]
What is the right notion of compatibility? That depends on how services should interact:
1. Requestor guarantees $\text{pre}_R$
   → Provider assumes $\text{pre}_P$
2. Provider guarantees $\text{effect}_P$
   → Requestor assumes $\text{effect}_R$
... a contravariant relation.

what it should mean, that:
- an assumption is correct
- a guarantee is fulfilled
... a question about the semantics of contracts.

Operational Semantics: The DPO Approach

- $L$ is embedded into graph $G$.
- The elements of $G$ matched by $L - I(K)$ are removed.
- The elements matched by $R - r(K)$ are added to $D$. The changes to $G$ are exactly those specified by the rule.
Loose Semantics of Contracts

Requestor has only loose idea of behavior of the other service

Provider has complete info, but may prefer not to publish everything

Contracts are incomplete specifications of service behavior

Formally: Double-Pullback (DPB), allows unspecified

Deletion: at least elements of G matched by L - I(K) are removed

Creation: at least elements matched by R - r(K) are added to D

(faithful) transition vs. transformation

Contracts as Rules, revisited

→ Positive Application Conditions

**Precondition:** what must be present before, no matter what happens later

- deleted
- preserved
- created
What is the right notion of compatibility? That depends on ...

how services should interact:
1. Requestor guarantees $\text{pre}_R$
   $\Rightarrow$ Provider assumes $\text{pre}_p$
2. Provider guarantees $\text{effect}_p$
   $\Rightarrow$ Requestor assumes $\text{effect}_R$

... a contravariant relation.

what it should mean, that:
- an assumption is correct
- a guarantee is fulfilled

... a question about the semantics of contracts.

Semantic Compatibility

1. $\text{pre}_R \Rightarrow \text{pre}_p$: applicability of requestor rule implies applicability of provider rule
2. $\text{effect}_p \Rightarrow \text{effect}_R$: transition via provider rule is also transition via requestor rule.
Semantic Compatibility

\[
\begin{align*}
R: & \quad c:Customer \\
& \quad r:RentalInfo \\
& \quad L_p: my\_car:Car \\
& \quad R_p: my\_car:Car \\
& \quad p: Customer \\
& \quad L: car:Car \\
& \quad r: RentalInfo \\
& \quad L_p: car:Car \\
& \quad R_p: car:Car \\
& \quad c: Customer \\
& \quad r: RentalInfo \\
& \quad car:Car \\
& \quad p: Customer \\
& \quad r: RentalInfo \\
& \quad car:Car \\
& \quad p: Customer \\
& \quad r: RentalInfo \\
& \quad car:Car \\
& \quad p: Customer \\
& \quad r: RentalInfo \\
& \quad car:Car \\
\end{align*}
\]

Semantic Compatibility: formally

\[
(p_1, \hat{L}_1) \text{ semantically matches } (p_2, \hat{L}_2), \text{ in symbols } (p_2, \hat{L}_2) \models_{\text{match}} (p_1, \hat{L}_1), \text{ iff:} \]

(i) for all graphs \( G \), if there exists \( d_{L_1} : \hat{L}_1 \rightarrow G \) s.t. \( d_{L_1} := d_{L_2} \circ \hat{l}_1 \) satisfies \( IC \) of \( p_1 \), then there exists \( d_{L_2} : \hat{L}_2 \rightarrow G \) s.t. \( d_{L_2} := d_{L_1} \circ \hat{l}_1 \) satisfies \( IC \) of \( p_2 \), and

(ii) for all spans \( t : (G \xrightarrow{g} D \xrightarrow{h} H) \), if there exists a transition \( G \xrightarrow{p_2/d_2} H \), then there exists a transition \( G \xrightarrow{p_1/d_1} H \) using the same bottom span \( t \).
What do we have?

Semantic compatibility relation $|=\quad$
- quantified over all graphs and transitions
- cannot be verified directly

**Objective:** syntactic matching relation $|--$
- **Soundness:** $p_2 |-- p_1$ implies $p_2 |= p_1$
- **Completeness:** $p_2 |= p_1$ implies $p_2 |-- p_1$

---

**Syntactic Matching Relation**

$R:$

- $c\cdot$Customer
- $n\cdot$RentInfo
- $my\_carCar$

$P:$

- $c\cdot$Customer
- $n\cdot$RentInfo
- $carCar$

$\pre_R \Rightarrow \pre_P: \text{ requestor must provide all information necessary for the execution of the provider operation,}$

$\text{effect}_p \Rightarrow \text{effect}_R: \text{ effect of the provided operation must include those expected by the requestor.}$
Syntactic Matching: formally

\[(p_1, \hat{L}_1) \text{ syntactically matches } (p_2, \hat{L}_2), \text{ in symbols } (p_2 : s_2, \hat{L}_2) \vdash_{\text{match}} (p_1 : s_1, \hat{L}_1), \text{ iff:} \]

(i) there exists an injective graph homomorphism \(h_L : \hat{L}_2 \rightarrow \hat{L}_1\) s.t. \(h_L \circ \hat{I}_2\) satisfies IC of \(p_2\), and

(ii) there exist graph homomorphisms \(h_L : L_1 \rightarrow L_2\), \(h_K : K_1 \rightarrow K_2\), and \(h_R : R_1 \rightarrow R_2\) s.t. the diagrams (a), (b), and the diagram on the left commute, and the diagrams (a) and (b) represent a faithful transition.

What do we have?

- **Semantic compatibility**: relation \(\models\)

- **Syntactic matching**: relation \(\models\)

- **Soundness**: \(p_2 \models \Rightarrow p_1\) implies \(p_2 \models = p_1\)

- **Completeness**: \(p_2 \models = p_1\) implies \(p_2 \models \Rightarrow p_1\)
Consistency of Service Composition

- **External**: between interface specifications
- **Internal**: between interface specification and implementation

Internal Consistency

Service Description:
- Class diagram
- Operation signatures
- Operation contracts

Operation annotations: JML assertions

Implementation

model

JML Compiler

executable binary code with run-time tests for contracts

Requestor

Provider

✔

JML from Graphical Contracts

Semantic idea: Assume rule \( r \) specifying method \( m \).
- If \( r \) is applicable to \( G \), then \( m \) invoked in \( G \) (with appropriate parameters) terminates without exception.
- If invocation yields \( H \), there exists a graph transition from \( G \) to \( H \) via \( r \).

After manually refining the models (business \( \rightarrow \) analysis view), translate
1. class diagram \( \rightarrow \) Java class frames
2. rules \( \rightarrow \) JML
   - patterns
   - rules

Class diagrams \( \rightarrow \) Java class frames

UML attributes \( \rightarrow \) private attributes with access methods
UML associations \( \rightarrow \) pairs of attributes, mutually consistent

private int orderNo;
pUBLIC int getOrderNo(){...}
pUBLIC void setOrderNo(int no){...}

private Customer buyer;
pUBLIC void setBuyer(Customer c){...}
pUBLIC Customer getBuyer(){...}

private TreeSet revBuyer;
pUBLIC void addRevBuyer(Order o){...}
pUBLIC void removeRevBuyer(Order o){...}
pUBLIC boolean hasRevBuyer(Order o){...}
contracts -> jml

public class ShopImplementation {
    ...
    /*
     * @ public normal_behavior
     * @ requires JML-PRE;
     * @ ensures JML-POST;
     */
    public boolean addProductToOrder(
            int productNo, int customerNo, int orderNo) {...}
    ...

    @
    addProductToOrder(productNo, customerNo, orderNo)

    @ public normal_behavior
    @ requires (\exists Product p;
    @ \exists Order o;
    @ \exists Customer c;
    @ \exists OrderNo oNo;
    @ \exists CustomerNo cNo;
    @ \exists ProductNo pNo;
    @ p.getProductNo() == productNo;
    @ c.getCustomerNo() == customerNo;
    @ o.getOrderNo() == orderNo;
    @ o.containsProduct(p) == false));

contracts -> jml: patterns

- starting at this navigate to as yet unbound objects, check attributes and links and bind them
- select navigation paths to achieve earliest possible failure
Contracts → JML: Rules

@ public normal_behavior

@ old Product p = getProductByNo(productNo);
@ old Customer c = getCustomerByNo(customerNo);
@ old Order o = c.findOrderByNo(orderNo);
@ requires p != null;
@ requires c != null;
@ requires o != null;
@ requires o.getCustomer() == c;
@ requires o.containsProduct(p) == false;
@ ensures p != null;
@ ensures c != null;
@ ensures o != null;
@ ensures !not_modified(p, c);
@ ensures o.getCustomer() == c;
@ ensures o.getProducts().contains(p);

Non-deterministic Matching

Solution: store all possible bindings and check that at least one satisfies post-condition
**Consistency of Service Composition**

- Visual representation of contracts based on GT with loose semantics
  - External: syntactic characterization of service compatibility
  - Internal: mapping of contracts to JML

**Open questions**
- relation between business-level and analysis-level contracts
- verification of mapping GT → JML
- implementation and evaluation

---

**Papers**

With A. Cherchago, M. Lohmann: *A Formal Approach to Service Specification and Matching based on Conditional Graph Transformation*, ICGT 2004 in Rome

With M. Lohmann: *Model-Driven Development of Reactive Information Systems: From Graph Transformation Rules to JML Contracts*, to appear in *STTT*