The Object Constraint Language (OCL)

- OCL is used for adding important information to UML models
- OCL is used
  - To *add constraints*, restrictions on values that formed part of an object-oriented model
  - To *state conditions* on execution of a system
  - To *define business rules*
  - To *define queries on the model* – so questions about a model can be answered automatically by executing an OCL query
- OCL2.0 is a new improved version.

Modelling with OCL

- Used for queries and constraints over UML2 models
- OCL syntax covers the following aspects of models:
  - *Initial values*
  - *Contracts*
  - *Constraints*
- If you read the reference (the Warmer and Kleppe book from http://www.klasse.nl/ocl/), these aspects are detailed for class diagrams, not component diagrams; however the approach is similar
- To understand how OCL can be used to provide a formal semantics of components

Context of OCL Expressions

- An OCL expression annotates a particular model entity (class, interface, datatype, component or entire application) – this is called the *context of the expression*
- Annotation takes the following form:
  ```
  Context ModelEntity
  [init:/pre:/post:] [inv:]
  ```
- OCL expression about ModelEntity
  - The context acts as a namespace for the OCL expressions
  - The OCL expression can refer to aspects of the context:
    - the attributes and methods of the class
    - the provided interfaces of the component
    - the required interfaces of the component
Interface Semantics

- Recall that we should always include a behavioural semantics before using an interface within an architecture.
- So far we have settled with using English to define these behaviours.

<table>
<thead>
<tr>
<th>Interface name</th>
<th>List of operations (all public)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speciation of behaviour</td>
<td></td>
</tr>
</tbody>
</table>

"sendMessage connects to the SmtpHost and sends the message msg."

Interface Specifications – Initial Values

- Initial value rules are associated with an interface context.
- Initial value rules define how a component interface implementation must behave at initialization.
- Can make reference to getter (side-effect-free) methods of the interface context.

Context: interface name
Init: OCL Expression about side-effect-free interface methods

<table>
<thead>
<tr>
<th>Interface name</th>
<th>List of operations (all public)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of behaviour</td>
<td></td>
</tr>
</tbody>
</table>

Example:
context ITransportable
init: getSmtpHost = "localhost"

Contracts in OCL

- Contract for an interface operation has the following form:
  - Context: interface name::method name
  - pre: OCL precondition expression
  - post: OCL postcondition expression

- Context consists of the interface name followed by the operation name (demarcdated by scope :: symbol).
- Pre- and post-condition can make use of any getter (side-effect-free) operations of the interface.

Example: interface sends SMTP compliant messages and adds them to a sent mail folder.

<table>
<thead>
<tr>
<th>Interface name</th>
<th>List of operations (all public)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of behaviour</td>
<td></td>
</tr>
</tbody>
</table>

Let isSMTP(s) returns true when s is an SMTP compliant message, and getSentMail() returns a folder of mail that has been sent.

context: ITransportable::
sendMessage(msg:Message)
pre: isSMTP(msg)
post: getSentMail() = getSentMail()@pre>union(Set(msg))

Example: interface sends SMTP compliant messages and adds them to a sent mail folder.

<table>
<thead>
<tr>
<th>Interface name</th>
<th>List of operations (all public)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of behaviour</td>
<td></td>
</tr>
</tbody>
</table>

The @pre Symbol

- The @pre symbol permits a postcondition to refer to the value a getter method returned at the beginning of the method call
- So, in
  \[ \text{post: getSentMail()} = \text{getSentMail()}@pre -> \cup(\{\text{msg}\}) \]

  the **LHS** `getSentMail()` refers the value of `getSentMail()` after executing the method, while the **RHS** `getSentMail()`@pre refers to `getSentMail()` prior to executing the method.

  \[ \text{getSentMail()} = \text{Set\{cs3sad, mum, csi\}} \]

  Initially, the value of `getSentMail()` is referred to in postcondition as `getSentMail()`@pre.

  \[ \text{ITransportable::sendMessage (msg:Message)} \]

  After executing, we get a new value of `getSentMail()`, because a message was added to send mail folder.

  \[ \text{getSentMail()} = \text{Set\{cs3sad, mum, csi, pdiddy\}} \]

Messaging in Postconditions

- OCL contracts are used to define side-effects of interface operations
- OCL expressions are not permitted side-effects – and therefore generally only refer to getter methods of interfaces
- It is possible to say that a particular side-effect-producing method can be called (e.g., assert that a message has been passed), using the \(^\wedge\) operator: `\text{b}^\wedge\text{m}()` asserts that the method \(\text{m}\) has been called on element \(\text{b}\)
- If an interface has a `sendMsg` method that should result in a subsequent call to a `passMsg` method, this can be specified as:

  \[ \text{context: ITransportable::sendMsg(msg:Message)} \]

  \[ \text{post: self^passMsg()} \]

  **Note:** `self` is like this in Java

Components as Substitutable Units

- **Substitutability:** Replace a component with an alternative or updated version without breaking the systems in which the component is used.

  New system = old system with Hotel reservation component replaced.

  HolidayRes 3.0  HolidayRes 2005

  Hotel reservation  Hotel reservation 2
  Air reservation  Air reservation
  Car reservation  Car reservation

  Holiday reservation session  Holiday reservation session

  Hotel reservation

Laws of Substitutability

- A component \(B\) can be substituted for component \(A\) if, and only if,
  - \(A\)'s provided interfaces are preserved by \(B\) (that is, signature and expected behaviour preserved)
  - \(B\)'s required interfaces are the same as or fewer than \(A\)'s (that is, \(B\) cannot require more)

- These constraints must be satisfied in order to avoid system breakage.
Substitutability

- That definition was vague about “expected behaviour” being preserved.

- If the interfaces have OCL contracts, then we will define the formal laws of substitutability.

- Component B can be substituted for component A if, and only if,
  - A’s provided interfaces are preserved by B
  - B’s required interfaces are the same as or fewer than A’s (that is, B cannot require more)
  - Preconditions of B’s provided interfaces are weaker than (are entailed by) preconditions of A’s interfaces
  - Postconditions of B’s provided interfaces are stronger than (entail) postconditions of A

Substitutability Constraints

Can be replaced by

isHTTP1.0 entails isHTTP1.1 & secure and isHTML entail isHTML → “Webserver2 is substitutable for Webserver”