MFI-7: Meta-model for Service Registration

Zaiwen Feng, Keqing He, Chong Wang, Jian Wang and Fei He
Wuhan University
2011-05-09
Service Discovery & Orchestration based on MFI

• How can MFI-5, 7, 8 be used to discover (orchestrate) Web services?
  – MFI-5: responsible for service discovery (using P-S relation), and service orchestration
  – MFI-8: responsible for service discovery (using G-P, G-S relation)
  – MFI-9: provide a method to discover services applying relations between RG and P and S
Service Discovery & Orchestration Scenario

- Scenarios:
  - User’s Request: “A user plans to attend a conference in another city”
  - Purpose: Discover services to meet user’s request
  - This scenario is extracted from 19763-9 MFI WD2
    Metamodel for On Demand Model Selection (TR)
    WG2 N1513
User’s Request: A user plans to attend a conference in another city

Goal: to arrange a travel plan

No Match!

Find a Composite Process: “PlanTravelRoute_Process”
Service Discovery & Orchestration Scenario

Composite Process: “PlanTravelRoute_Process”

A new service: PlanTravelRoute_Service
Service Discovery & Orchestration Scenario

• In this scenario:
  – MFI-5: responsible for service discovery (utilize P-S relation), and service orchestration
    • Process05: Book Ticket -> Booking Ticket Web Service (Discovery)
    • Process06: Query Hotel -> Query Hotel Web Service (Discovery)
    • Process07: RentCar -> Rent Car Web Service (Discovery)
    • PlanTravelRoute Web Service can be organized based on realationships between Process05, Process06 and Process07 (Orchestration)
  – MFI-8: responsible for service discovery (utilize G-P, G-S relation)
    • Role_Goal01: ArrangeTravelPlan_Goal -> PlanTravelRoute_Process (Discovery)
    • Role_Goal01: ArrangeTravelPlan_Goal -/-> PlanTravelRoute_Service (Discovery)
  – MFI-9: provide a method to discover services applying relations among RG, P and S
Meta-Model of MFI-7
Function-related Metaclasses(1)

• **Operation**
  – the execution actions of the service.
  – One *Service* can have one or multiple *Operations*

• **Input, output**
  – the link between service and outer space
  – Their type can be depicted as *Message_Type*

• **Message_Type**
  – specifies the type of message that is used, as input variable, out variable.
  – It can be the domain concepts in Ontology
Function-related Metaclasses (2)

- **Precondition**
  - specifies a state that shall exist before a service is invoked

- **Postcondition**
  - specifies a state that shall exist after a service is invoked successfully

Their properties are *name*
Non Function-related Meta-classes

- **Quality_Property**
  - *Quality_Property* is a metaclass that is used to represent a certain Non-functional feature for service, such as availability, response time, etc.
  - It has a property *type*

- **QoSAssertion**
  - *QoSAssertion* is a metaclass that is used to represent an description of a certain Non-functional property for service.
  - It has a property *name*
  - It has one or multiple logical expression which is expressed as *Formula*
Condition-related Metaclasses(1)

- **Formula**
  - Formula is a metaclass that represents logical expression within QosAssertion, Precondition or Postcondition

- **QoSAssertion**
  - *QoSAssertion* have one or multiple Logical Expression represented by *Formula*
  - E.g. The fact that *Response time of service is more than five seconds* can be expressed by *formula* as **Morethan** *(responseTime, 5s)*

- **Precondition**
  - *Precondition* have one or multiple Logical Expression represented by *Formula*
  - E.g. The fact that *balance of VISA card is more than amount of the book to be bought* can be expressed by *formula* as **Morethan** *(balance of VISA card, amount of the book)*

- **Postcondition**
  - Similar as *Precondition*
Condition-related Metaclasses (2)

- **Atom_Formula**
  - Atom_Formula is a metaclass representing a formula that has no sub-formula
  - E.g. There is a condition that the **current date must be before the expiring date of VISA card**

- **Predicate**
  - Predicate is a metaclass representing a function that returns true or false. It contains one argument or links two arguments, which formulates a Atom_Formula

- **Argument**
  - Argument is a metaclass representing variable that is contained or linked by predicate in Atom_Formula

- E.g.
  - For the fact that **the current date must be before the expiring date of VISA card**
  - Atom_Formula is expressed as IsBefore (CurrentDate, Expiring date of VISA card)
  - Predicate is IsBefore, and Argument are CurrentDate and Expiring date of VISA card
• **Composite_Formula**
  
  *Composite_Formula* is a metaclass representing the formula that contains other sub-formulas, which might be either atomic or composite.

• **Logic_Connection**

  *Logic_Connection* is a metaclass representing connections which connect one or multiple *Atomic_Formula* to a *Composite_Formula*.

• **Negation**

  *Negation* is a metaclass representing a kind of *Logic_Connection*. Suppose A is a *Atom_Formula*, *Negation* is used to connect one *Atom_Formula*. *Negation* A is true if and only if A is false.

<table>
<thead>
<tr>
<th>A</th>
<th>Negation A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Condition-related Metaclases(4)

• Conjunction
  – *Conjunction* is a metaclass representing a kind of *Logic_Connection*. *Conjunction* is used to connect two *Atom_Formulas*. Suppose A, B are *Atom_Formula*, Conjunction(A, B) is true if and only if both A and B are true.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Conjunction (A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

• Disjunction
  – *Disjunction* is a metaclass representing a kind of *Logic_Connection*. *Disjunction* is used to connect two *Atom_Formulas*. Supposing A, B are *Atom_Formula*, Disjunction(A, B) is false if and only if both A and B are false.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Disjunction (A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Thanks !
What we agree with...

• Type of Service should be defined such as: entity_service, task_service...
• Service refers to only Web service
• It should be set up *aggregation* relationship among service
• There should reference to MFI Part10
• The definition of Qos Assertion is modified
Definition of Service Type

• Entity Service
  – The *entity service* model represents a business-centric service that bases its functional boundary and context on one or more related business entities.
  – Several of its capabilities are reminiscent of traditional CRUD (create, read, update, delete) methods.

2011.5
Definition of Service Type

• Task Service
  – A task service is a form of business service with a functional context based on a specific business process.

Definition of Service Type

• Utility Service
  – A utility service is intentionally based on a non-business-centric functional context. It typically encapsulates common, cross-cutting functionality that is useful to many service compositions, but which is not related to or derived from existing business models.