A Dependency Markup Language for Web Services

Robert Tolksdorf

Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto:research@robert-tolksdorf.de
http://www.robert-tolksdorf.de

Overview

- Current description mechanisms of Web-Services are *not adequate*
- *Dependencies* of activities can be used do describe composite services better
- *Dependency Markup Language* proposed

Proposals for the description of Web Services

Lang.	External	Internal
WSFL	functional interface specified	data- and control flows specified
XLANG	interface specified with WSDL	control flow specified, event handling
WSCL	allowed interactions (conversation) specified by interaction-transition net	not specified
DAML-S	functional interface specified	control flow specified by imperative constructs
ASDL	allowed usage specified by state-(conditioned) transition net	control flow specified
WSMF	functional interface specified with pre- and postconditions	not specified
BPSS	interactions specified by message exchanged	state/transition net
BPML	interactions specified by message exchanged	control flow specified by imperative constructs

Visiting a restaurant

- Restaurants offer a service which consists of taking an order, preparing food, serving it and finally collecting
- Interface (free style notation):
  {wallet!=empty}
  void visitRestaurant(Money wallet, Order whatToEat)
  {repleted=TRUE}
- But:

Restaurant	Service flow
Full service	take order–cook–serve–collect
Fast food	cook–take order–collect–serve
Buffet	cook–take order–serve–collect
Church supper	collect–take order–cook–serve

(after [WL95])

Problem

- Qualities of services:
  - External interface
  - But also: Internal behaviour of interest
- Current Web Service descriptions cannot capture that
  - Internal behaviour is not specified
  - Internal behaviour is specified too low level
- Needed:
  - Express "visiting a restaurant" as an *abstract* process that is implemented by various concrete ones
  - Express concrete behaviour at right semantic level (eg: cooked freshly after my order)

Coordination theory/1

- "Coordination is management of dependencies"[MC94]
- Kinds of dependencies and how they are managed by coordination mechanisms [Cro91,Del96]:

Coordination mechanism	Dependency managed	
Resource allocation		Shared resources
Notification	Prerequisite	
Transportation	Transfer	Producer/Consumer
Standardization	Usability	
Synchronization		Simultaneity
Goal selection Decomposition		Task/Subtask

Coordination theory/2

- [RG00,RG01]:

Dependency class	Dependency
Temporal	Strict sequence
	Real-time
Causal	Loose sequence
	Data dependency
	Resource dependency
Abstraction	Generalization/ Refinement
	Aggregation

Coordination Mechanism	Execution order
Sequence	Sequence
Alternative	Choice
	Conditioned choice
Iteration	Loop
	Conditioned loop
Concurrency	Accidental
	Enforced
	Prohibited

- Dependencies more adequate than control flows (a.b and b.a manage exclusive operation of a and b)
- *We model behaviour by dependencies*

Abstraction

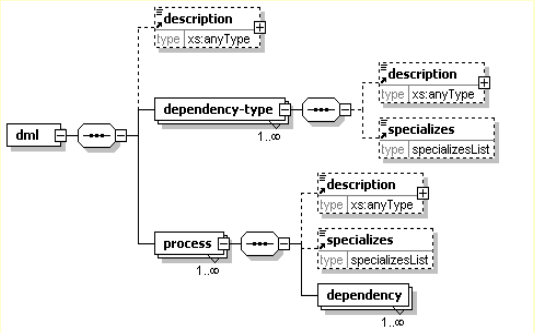
- Different granularities:
  - concrete processes in restaurants
  - abstract spec. of "freshly cooked" (cook depends on order)
  - most abstract notion "restaurant visit"
- All behaviours above are equivalent wrt. interface and post-condition of service
- Not all abstractions are useful ("do something")
- Automatic classification of behaviour is difficult
  - Typing processes [MCL+99]
  - Generalizing existing processes/deriving specializations [WL95]
- *We relate abstractions and specializations explicit*

Typing

- Port- and service-types similar to interfaces in OMG/CORBA. There:
  - Interface-types related by specialization / generalization plus formal notion of a contravariant subtyping of interfaces
  - Relations used by a trader in service discovery
- *Proposal: dependencies used as additional information about the internal workings of a service*
- Service dependency typing:
  - Clients can express abstract expectations on the dependencies ruling the workings of the service.
  - Level of detail in the description can depend on how much information the service provider is willing to disclose.
  - Abstraction and specialization express a relative semantic of what the services do.

DML

- *Dependency Markup Language* DML
  - Dependency types are defined
  - Dependency types can be related
  - Processes are defined by a set of dependencies
  - Processes can be related



Dependency types

```
<dependency-type id="unizh-dependency" specializes="any"/>
<dependency-type id="temporal"
 specializes="unizh-dependency"/>
<dependency-type id="strictSequence">
 <specializes>temporal looseSequence</specializes>
</dependency-type>
<dependency-type id="realTime" specializes="temporal"/>
<dependency-type id="causal"
 specializes="unizh-dependency"/>
<dependency-type id="looseSequence" specializes="causal"/>
<dependency-type id="dataDependency" specializes="causal"/>
<dependency-type id="resourceDependency"
 specializes="causal"/>
<dependency-type id="abstraction"
 specializes="unizh-dependency"/>
<dependency-type id="generalization"
 specializes="abstraction"/>
<dependency-type id="refinement" specializes="abstraction"/>
<dependency-type id="aggregation" specializes="abstraction"/>
```


Processes/1

```
<process id="restaurantVisit"
 name="Visit to a restaurant">
 <description>An abstract description of a
  restaurant visit where only cooked food is eaten.
 </description>
 <dependency type="looseSequence"
  from="cook" to="serve"/>
</process>
```

```
<process id="freshlyCooked"
 specializes="restaurantVisit">
 <description>An abstract description where
  things are cooked after an order.
 </description>
 <dependency type="looseSequence"
  from="takeOrder" to="cook"/>
</process>
```

Processes/2

```
<process id="fullService" specializes="freshlyCooked">
 <dependency type="strictSequence"
  from="takeOrder" to="cook"/>
 <dependency type="strictSequence"
  from="cook" to="serve"/>
 <dependency type="strictSequence"
  from="serve" to="collect"/>
</process>
<process id="fastFood" specializes="restaurantVisit">
 <dependency type="strictSequence"
  from="cook" to="takeOrder"/>
 <dependency type="strictSequence"
  from="takeOrder" to="collect"/>
 <dependency type="strictSequence"
  from="collect" to="serve"/>
</process>
```


Coordination environment

- Coordinating Web Services
  - Idea: Coordination services bind themselves to the dependencies they manage
  - Coordination service generates specific schedules with their help
- DML based Service discovery
  - Idea: Specific control flow is specialization of abstract process
  - Existing Web Services can be classified and traded
- Automatic classification
  - Idea: Calculate specialization
  - Hard, limited

Outlook / Summary

- Implement coordination environment
- Consider multiparty dependencies
- Perhaps better build on RDF than use a separate markup language
- Build dependency catalogue

- Current description mechanisms of Web-Services are *not adequate*
- *Dependencies* of activities can be used do describe composite services better
- *Dependency Markup Language* proposed

- www.robert-tolksdorf.de/dependencies