Task-Oriented Business Requirements
Elicitation for Web Services

Stephen Gorton

IST-FET IST-2005-16004

Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH
Web Service Architecture

- Composition often the top layer;
- Composition and orchestration still a blur!
- Little regard for a more abstract requirements layer.
Current Solutions 1

- Approach 1: Composition as Requirements
  - BPEL:
    ```xml
    <process name="tset">
      <partnerLink name="client"/>
      <partnerLink name="service1"/>
      <partnerLink name="service2"/>
    </process>
    ```
    
    - DAML-S
      ```xml
      <daml:Class rdf:ID="tset">
        <daml:property rdf:resource="Process#ComposedOf"/>
        <daml:property rdf:resource="Process#Components"/>
      </daml:Class>
      ```

Code snippets taken from Milanovic and Malek: Current Solutions for Web Service Composition. IEEE Internet Computing, Nov/Dec 04
Current Solutions 2

• Approach 2: Specialised Requirements Language
  - BPMN:

  ![BPMN Diagram]

  - UML:

  ![UML Diagram]
Wedding Example

- Business goal $g = \text{“plan wedding”};$

- Broken down into objectives (composite tasks):
  - $ct_1 = \text{plan pre-wedding celebrations};$
  - $ct_2 = \text{plan preparations};$
  - $ct_3 = \text{plan legalities};$
  - $ct_4 = \text{plan ceremony};$
  - $ct_5 = \text{plan post-ceremony celebrations};$
  - $ct_6 = \text{plan honeymoon}.$

- Tasks are arranged according to result timeline, not according to execution timeline!
  - e.g. ceremony and post-ceremony celebrations often planned in parallel.

- Policies:
  - The entire event should not cost more than £10k;
  - The ceremony and post-ceremony celebrations should be on the same day;
  - The honeymoon should be booked through a known and trusted travel agency.
Flows:
- Control runs from start to finish;
- Solid lines indicate control flow routes;
- A task is executed when control reaches it;
- Control proceeds when the task has finished.

Flow Split:
- FS: in -> OUT;
- Control proceeds down each output simultaneously;
- No limit on number of output flows;
- Parallel split workflow pattern

Conditional Merge:
- CM: IN -> out;
- Forces synchronisation;
- Mandatory and optional flows;
- Specifies minimum number of flows;
- Discriminator workflow pattern.
**Strict Preference:**
- SP: \( in \rightarrow out; \)
- Input is a set of pairs \( \{t, n\} \)
  - \( t \) is a task;
  - \( n \) is a priority rating;
- New workflow pattern.

**Flow Merge:**
- FM: \( IN \rightarrow out; \)
- Incoming set of control flows contains only one active flow;
- No synchronisation issue;
- (Multiple) Merge workflow pattern.

**Random Choice:**
- RC: \( in \rightarrow out; \)
- All tasks invoked;
- When a first gets to a “commit”, all others are cancelled;
- New workflow pattern.
Other Notation

- Flow Junction Operator:
  - FJ: \( in \rightarrow \{out_1, out_2\} \);
  - Left output is primary;
  - Output flow chosen according to a test;
  - Exclusive choice workflow pattern.

- Bounded cycles allowed:
  - For both composite and atomic tasks;
  - Can be modelled with flow junction and flow merge.
  - (since we only allow one control flow input, a flow merge function should be used).
Summary

• Current notations not appropriate:
  – UML has some merits but does not support many workflow patterns;
  – BPMN is the nearest to a complete solution;
  – None allow for the expression of all requirements.

• A simple graphical notation:
  – Describing process flows;
  – Scope for core and non-core (non-functional) requirements;
  – Offers the context in which policies are used.

• Further work:
  – Workflow patterns (data and resource patterns);
  – Policies and policy framework at the business level;
  – A workbench.
Thank you.

Any Questions?