Service Request Languages based on AI Planning

Marco Aiello
Instituut voor Wiskunde en Informatica - Rijksuniversiteit Groningen
aiellom@cs.rug.nl

IPA Lentedagen on Service-oriented Computing
Conference center Kapellerput, Heeze,
April 3-5, 2007
Summary

- AI Planning
- Planning approaches to service composition and orchestration
- The XSRL service request language
- XSRL via model based planning
- XSRL via constraint satisfaction
- Open issues
Artificial Intelligence: Planning

- A branch of artificial intelligence concerned with the generation and execution of a set of actions to achieve some goal.

- Usually, a state transition system is used to model the environment.

- Key definitions: goal, plan, environment, actions, initial state, external events, state transition function.
AI Planning: types of planning

- **Domain specific**: most widely used in practice

- **Domain independent**: most research focus on this, though hard to have an effective completely domain independent planner. Usually, some extra assumptions are necessary:
  
  - finite system / fully observable / deterministic / static / attainment goals / sequential plans / implicit time / off-line planning (classical planning)

- **Configurable**: the engine is domain independent, but the input includes domain information on how to drive the search (hierarchical task networks HTN, control formulas)
Why AI Planning for services?

- Planning becomes a useful if not indispensable tool when acting in complex and dynamic domains (e.g., robots, satellite operation, intelligent agents)

- Service based environments have the characteristics of being open, complex, non-deterministic, highly dynamic and democratic (non centralized control)

- Therefore, issues of service composition, discovery, orchestration and interaction led themselves nicely to be modeled and solved as AI planning problems

- Many approaches have been proposed in the literature: from fully on-line composition of services to HTN for specific orchestration tasks. Let’s review the most relevant ones.

AI Planning approaches to services

• The common ingredient is the assumption that some extra semantic information is available and is used to model the domain.

• The difference lies in when the composition takes place: from off-line planning to run-time planning. The extreme is that the environment is considered as a set of independent services, a goal is provided and these are composed at run time to satisfy the goal.

• Let us review the most representative approaches.
An approach using Golog

• Web resources are semantically described and all accessible to the planner

• The world is described in the situation calculus

• World description and goals are then translated in Golog: a second order programing language for dynamic domains which handles non-determinism and probability

The assumption of full semantic description of the services and description of the world in terms of the situation calculus made this approach unfeasible.
Automata based composition

- Services are described as automata
- A composed goal service also, then planning consists in finding atomic services that composed together create the goal service
- Via this formal approach it is possible to draw conclusions on the complexity of composition
- The assumption of having automata descriptions of every service is also strong

Regression planning based composition

- The web service environment is modeled using the PDDL language taking into account incomplete knowledge about the domain
- Then regression planning is used

```
(define (problem bb-probl)
  (:domain www-agents)
  (:objects amazin.com wabash.com - Merchant)
  (:facts (web-agent amazin.com)
    (web-agent wabash.com)
    (freevars (pid - Product-id))
    (expected-reply amazin.com
      (query-in-stock ?pid)
      (reply-in-stock true)))
  (freevars (sid - Message-id pid - Product-id)
    (expected-reply wabash.com
      (query-in-stock ?pid)
      (reply-in-stock false)))
  (:goal (know-val-is (in-stock amazin.com (prodid "P001")) true)))
```

Template based composition: HTN

• Hierarchical task networks represent plan templates that control execution, reasoning can be done via constraint satisfaction, and user interaction is easily included in the execution process

• In [2,1] automated service composition is achieved by modeling services as web information sources (exposed by automated web-site wrapping software) for which a common data model is known in advance

A semantic web approach

- In [1] OWL-S Process Models are encoded as SHOP2 Domains. With this approach:
  - the methods correspond to recursively composable workflows
  - the planning scales well to large numbers of methods and operators
  - Support complex precondition reasoning
  - Support for user interaction

Service Request Languages via AI Planning

• It is clear that extra semantic information is needed, but where should it go?

• What are reasonable assumptions with respect to what happens at design time and what at run-time?

• We introduce the notion of a service request language which works based on AI Planning

• The assumption is that a set of services populating an electronic marketplace is available and that these follow standardized business process, then we want to equip the requester and service providers with instruments to operate, abstracting from marketplace’s details and avoiding reimplementation as much as possible. These instruments are service request languages
The XSRL approach

- XSRL (XML Service Request Language) is a goal language based on the extension of temporal logics
- The business process represents the planning domain
- A service message is a planning action
- Web service implementations are the world

Extra knowledge is placed in the transformation of a business process (e.g. BPEL) into a planning domain. But once this is done, there is no need to redesign the process or reimplement services. Just changing the request will suffice

The XSRL framework
XSRL distinctive features

- The expressive power of the request language which include constructs to express preferences, to order subgoals, to refer to the execution properties of achieving the goals

- The domain representation captures, among other things, non-deterministic behavior

- The possibility of obtaining optimal solutions with respect to one or more parameters
XSRL: Sample request

- Given a business process for a domain in the travel industry, user requests:
  
  - reserving one night trip to Paris with hotel
  
  - avoiding to travel by train, if possible
  
  - spending overall amount less than 300 euros
  
  - spending less than 100 euros per hotel, if possible, and 200 at most
  
  - receiving, finally, a confirmation of payment
XSRL: language constructs

- XML Service Request Language, based on an extension of temporal logic (EaGLe and CTL):

  - achieve-all, vital, atomic (reachability)

  - before...then (sequencing)

  - prefer...to, optional (preference)

  - vital-maint, optional-maint (maintainability goals)

  - =,<,> (linear constraints)
<table>
<thead>
<tr>
<th>Goal / Assertion</th>
<th>Where satisfied</th>
<th>Type of goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>vital $p$</td>
<td>In a state where $p$ holds to which there is a path from the initial state modulo failures</td>
<td>reachability</td>
</tr>
<tr>
<td>atomic $p$</td>
<td>In a state where $p$ holds to which there is a path from the initial state despite failures</td>
<td>reachability</td>
</tr>
<tr>
<td>vital-maint $p$</td>
<td>In a state to which there is a path from the initial state modulo failures. $p$ must hold in all states along the path</td>
<td>maintainability</td>
</tr>
<tr>
<td>atomic-maint $p$</td>
<td>In a state to which there is a path from the initial state despite failures. $p$ must hold in all states along the path</td>
<td>maintainability</td>
</tr>
<tr>
<td>prefer $g_1$ to $g_2$</td>
<td>In states where $g_1$ is satisfied, otherwise the satisfiability of $g_2$ is checked</td>
<td>preference</td>
</tr>
<tr>
<td>optional $g$</td>
<td>States where $g$ is satisfied are checked first, otherwise the goal is ignored</td>
<td>preference</td>
</tr>
<tr>
<td>before $g_1$ then $g_2$</td>
<td>In states, to which there is a path from the initial state, such that, states along these path where $g_1$ is satisfied precede those where $g_2$ is satisfied</td>
<td>sequencing</td>
</tr>
<tr>
<td>achieve-all $g_1, \ldots, g_n$</td>
<td>In states, to which there is a path from the initial state, such that, there are states along the path where $g_i$ are satisfied</td>
<td>composition</td>
</tr>
</tbody>
</table>

Table 1. Goal and assertion language constructs.
XSRL: Sample request

achieve-all
before
achieve-all
prefer vital-maint hotelPrice < 100
to vital-maint hotelPrice < 200
optional-maint ¬ trainBooked
vital confirmed ^
location = “Paris” ^
hotelReserved
then
atomic final
vital-maint price < 300
The XSRL framework
XSAL: the provider’s perspective

- An assertion is a condition that is verified during the execution of a business process
- In any given state an assertion is either satisfied or not
XSAL examples

• Provider level:
  • Use VISA card for payment, if possible
  • Preserve a positive balance on a bank account

• Role level:
  • All flights to a health risk countries must have a special medical insurance purchased
  • Travel package entity evolution is checked

• Process level:
  • Special offer for people with fidelity card (OneWorld)
  • Process must be atomic: always reaching a final state
XSAL examples implemented

• Provider level:
  - `optional` (*cardType = VISA*)
    *Use VISA card for payment, if possible*
  - `optional-maint` (*balance > 0*)
    *Preserve a positive balance on a bank account*

• Role level:
  - `vital` (*healthRisk -> insuranceTaken*)
    *All flights to a health risk countries must have a special medical insurance purchased*

• Process level:
  - `optional-maint` (*royaltyCard -> (roleType = acceptsRoyalty]*)
    *Special offer for people with fidelity card (OneWorld)*
  - `atomic` `final`
    *Process must be atomic: always reaching a final state*
XSAL: Assertions classification

Actor assertions (based on ownership)
How does the planning actually take place?

- The domain is a state transition system that can be seen as a formal model
- The goal is an extended temporal logic expression
- Model checking the formula on the model not only provides a yes or no answer but also a path in the model from the initial state to a goal state or a counterexample, respectively.
- The successful path is in fact a plan to satisfy the formula
- Given the fact that the system is non-deterministic and that the formula is able to express preferences and path condition, some extra care is necessary (the MBP planner based on the nuSMV model checker was used)
Model based planning for XSRL

• The advantages of this approach lie in having a sound and complete framework for planning in non-deterministic domains.

• The main problems are that

1. There is no notion of solution enumeration or optimality

2. It is hard to deal with numeric values (planning at the knowledge level or booleanizations are necessary)

3. Experimentation shows that the approach does not scale well

• A better approach with constraint programming will be seen next