Service Request Languages based on AI Planning - II

Alexander Lazovik
CWI
a.lazovik@cwi.nl

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

- Constraint programming as planning algorithm
- Implementation overview
  - Preliminary experimentation
  - Running an example
- XSRL usage scenarios
- Open research issues
The XSRL framework
Constraint Programming

- **Constraint Satisfaction Problem is:**
  - a set of variables $X = \{x_1, \ldots, x_n\}$;
  - for each variable $x_i$, a finite set $D_i$ of possible values (its domain), and
  - a set of constraints restricting the values that the variables can simultaneously take.

- A *solution* to CSP is an assignment to the set of variables such that all its constraints are satisfied.

- CSP is successfully used in
  - resource allocation, scheduling, timetables

- A lot of extensions including:
  - QCSP with quantifiers in constraints;
  - CPNets, reification of constraints to support preferences over variables and constraints;
  - Optimality of the solution.
Why constraints?

- Dealing with numeric values
- Notions of optimal solutions
- Soft constraints to express preferences
- Dynamic addition and removal of assertions

Approach using planning as model checking:

- **Advantages**: effective algorithms for goals over booleans, handling of non-determinism, state together with path properties
- **Disadvantages**: state explosion problem with numerical constraints, non-optimality of the solution
The problem in terms of CP is defined by \( \{\beta, N, \xi, C\} \), where:

- \( \beta \) is a set of boolean variables, representing execution path. Each variable in \( \beta \) represents one execution choice;

- \( N \) is a set of variables ranging over natural numbers, representing repeating executions (cycles) in BP;

- \( \xi \) is a set of non-controlled boolean variables. Each variable represents one non-deterministic outcome;

- \( C \) is a set of constraints, in which
  - (i) if a non-controlled variable occurs in atomic goal then it is universally quantified,
  - (ii) otherwise a value corresponding to non-failure execution is used.
Given a BP, a request and a set of assertions:
- PHASE I: represent the BP as a set of constraint expressions;
- PHASE 2: encode the request and assertions based on goal type and phase I

The encoding stage may be repeated when the synthesized plan is executed and new information is gathered from service invocations (including variable values and new assertions)

When updated, CP problem and its solution may use the results from previous iterations
Phase I: basic idea

\[ \beta a + \beta \leq 1 \]

\[ \beta_1 a_1 + \beta_2 a_2 + \beta_1 + \beta_2 \leq 1 \]
Phase I: basic idea

\[ \beta_1 (a + \beta_2 a_2) \]
\[ \beta_1, \beta_2 \leq 1 \]

\[ \beta (\xi_1 a' + \xi_2 a'') \]
\[ \beta \leq 1 \]
\[ \xi_1 + \xi_2 = 1 \]
<table>
<thead>
<tr>
<th>Domain</th>
<th>Type of action</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>No action</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>(B)</td>
<td>Single deterministic action</td>
<td>$\beta a$</td>
</tr>
<tr>
<td>(C)</td>
<td>Sequence of actions</td>
<td>$\beta_1(a_1 + \beta_2 a_2)$</td>
</tr>
<tr>
<td>(D)</td>
<td>Deterministic branch point</td>
<td>$\beta_1 a_1 + \beta_2 a_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\beta_1 + \beta_2 \leq 1$</td>
</tr>
<tr>
<td>(E)</td>
<td>Nondeterministic branch point</td>
<td>$\beta(\xi_1 a' + \xi_2 a'')$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\xi_1 + \xi_2 = 1$</td>
</tr>
<tr>
<td>(F)</td>
<td>Cycle: state splitting</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>(G)</td>
<td>Cycle: directed cycle</td>
<td>$n\xi(a_1 + a_2 + a_4')$</td>
</tr>
</tbody>
</table>

Table 2. Encoding examples of the service domain.
Phase II: basic idea

- **vital** \((v < 10)\):
  \[ v_0 + \beta a < 10 \]
  \[ \beta \leq 1 \]

- **atomic** \((v < 10)\):
  \[ \forall \xi_2, \xi_2 : v_0 + \beta (\xi_1 a' + \xi_2 a'') < 10 \]
  \[ \beta \leq 1 \]
  \[ \xi_1 + \xi_2 = 1 \]
### Phase II

<table>
<thead>
<tr>
<th>Goal / Assertion</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vital</strong> ( p )</td>
<td>( \xi = \xi^0: c_v \triangleleft v_0 )</td>
</tr>
<tr>
<td><strong>atomic</strong> ( p )</td>
<td>( \forall \xi: c_v \triangleleft v_0 )</td>
</tr>
<tr>
<td><strong>vital-maint</strong> ( p )</td>
<td>( \xi = \xi^0: c_v(t_i) \triangleleft v_0 ), for all encoding steps ( t_i )</td>
</tr>
<tr>
<td><strong>atomic-maint</strong> ( p )</td>
<td>( \forall \xi: c_v(t_i) \triangleleft v_0 ), for all encoding steps ( t_i )</td>
</tr>
<tr>
<td><strong>prefer</strong> ( g_1 ) <strong>to</strong> ( g_2 )</td>
<td>All variables in ( g_1 ) are instantiated before those in ( g_2 )</td>
</tr>
<tr>
<td><strong>optional</strong> ( g )</td>
<td>encoded as <strong>prefer</strong> ( g ) <strong>to</strong> ( \top )</td>
</tr>
</tbody>
</table>
| **before** \( g_1 \) **then** \( g_2 \) | for all \( g^i \in G_1, g^j \in G_2 \), steps \( t_k \):
|                       | \( u_k(g^i) \neq 0 \Rightarrow u_k(g^i) = u_k(g^j) \)            |
| **achieve-all** \( g_1, \ldots, g_n \) | for all \( g^i \in G_i, g^j \in G_j, i \neq j \), steps \( t_k \):
|                       | \( u_k(g_i) \neq 0 \land u_k(g_j) \neq 0 \Rightarrow u_k(g_i) = u_k(g_j) \) |

**Table 3.** Goal and assertion language encodings.
Complex goals: detailed view

**achieve-all** $g_1 \ g_2$

$g_1:$

- $s_2 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_3$

$g_2:$

- $s_2 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_3$

$a_1(g_1) \iff a_1(g_2)$

$a_2(g_1) \iff a_2(g_2)$

**before** $g_1$ then $g_2$

$g_1:$

- $s_2 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_3$

$g_2:$

- $s_2 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_3$

$a_1(g_1) \Rightarrow a_1(g_2)$

$a_2(g_1) \Rightarrow a_2(g_2)$
Complex goals: detailed view

**achieve-all** $g_1 \ g_2$

$g_1$:

- $a_1 \rightarrow s_1$
- $s_2 \rightarrow a_1$
- $s_1 \rightarrow a_2$
- $s_2 \rightarrow s_3$

$g_2$:

- $a_1 \rightarrow s_1$
- $s_2 \rightarrow a_1$
- $s_1 \rightarrow a_2$
- $s_2 \rightarrow s_3$

$\beta_1(g_1) + \beta_2(g_1) = 1$
$\beta_1(g_2) + \beta_2(g_2) = 1$

=>

$\beta_1(g_1) = \beta_1(g_2)$
$\beta_2(g_1) = \beta_2(g_2)$

**before** $g_1$ **then** $g_2$

$g_1$:

- $a_1 \rightarrow s_1$
- $s_2 \rightarrow a_1$
- $s_1 \rightarrow a_2$
- $s_2 \rightarrow s_3$

$g_2$:

- $a_1 \rightarrow s_1$
- $s_2 \rightarrow a_1$
- $s_1 \rightarrow a_2$
- $s_2 \rightarrow s_3$

$\beta_1(g_1) + \beta_2(g_1) = 1$
$\beta_1(g_2) + \beta_2(g_2) = 1$

=>

$\beta_1(g_1) = \beta_1(g_2)$
$\beta_2(g_1) = \beta_2(g_2)$
Preferences: by reification

- prefer vital (v < 10) to vital (v < 20):
  
  \[ v_0 + \beta' a < 10 \quad \text{and} \quad v_0 + \beta'' a < 20 \]

  \[
  (p = 0) \quad \text{and} \quad (v_0 + \beta' a < 10) \quad \text{or} \quad (p = 1) \quad \text{and} \quad (v_0 + \beta'' a < 20)
  \]

  When solving, p is set to 0, if no solution is found, then p is set to 1, ensuring that the second goal is checked ONLY if the first fails.
Sample run

**achieve-all**
- **vital** hotelBooked
- **vital** trainReserved
- **vital-maint** price ≤ 300

Train Provider

Hotel Provider

Flight Provider
**vital** hotelBooked:
0.0 + beta_s1_0*(xi_s1_1*(1.0))=1.0

**vital** trainReserved:
0.0 + beta_s1_0*(xi_s1_1*(beta_s2_0*(1.0)))=1.0

**vital-maint** price ≤ 300
0.0 <= 300.0
0.0 + beta_s1_0 * (xi_s1_1 * (100.0)) <= 300.0
0.0 + beta_s1_0 * (xi_s1_1 * (100.0 + beta_s2_0 * (200.0) + beta_s2_1 * (400.0)))<=300.0
achieve-all:

if \((vital_1\_beta\_s2\_0 + vital_1\_beta\_s2\_1 = 1)\) and 
\((vital_2\_beta\_s2\_0 + vital_2\_beta\_s2\_1 = 1)\)

then 
\((vital_1\_beta\_s2\_0 = vital_2\_beta\_s2\_0)\) and 
\((vital_1\_beta\_s2\_1 = vital_2\_beta\_s2\_1)\)

if \((vital_1\_beta\_s0\_0 = 1)\) then \(vital_1\_beta\_s0\_0 = vitalm_1\_beta\_s0\_0\)
if \((vital_1\_beta\_s1\_0 = 1)\) then \(vital_1\_beta\_s1\_0 = vitalm_1\_beta\_s1\_0\)
if \((vital_1\_beta\_s2\_0 + vital_1\_beta\_s2\_1 = 1)\) 
then \((vital_1\_beta\_s2\_0 = vitalm_1\_beta\_s2\_0)\) and 
\((vital_1\_beta\_s2\_1 = vitalm_1\_beta\_s2\_1)\)
Solution

vital-maint (price <= 300)
  vitalm_1_beta_s1_0 = 1
  vitalm_1_beta_s2_1 = 0
  vitalm_1_beta_s2_0 = 1

vital (trainBooked = true)
  vital_2_beta_s1_0 = 1
  vital_2_beta_s2_1 = 0
  vital_2_beta_s2_0 = 1
  1. bookHotel

vital (hotelReserved = true)
  vital_1_beta_s1_0 = 1
  vital_1_beta_s2_1 = 0
  vital_1_beta_s2_0 = 0
  2. reserveTrain
Implementation and experimentation

- Java, as Eclipse plug-in

For experimentation: domain generator with various parameters, as a result:
- Up to 2000 states with simple goals
- Up to 1000 states with combination of goals on “hard” domains

Reference implementation may be downloaded from [www.dit.unitn.it/~lazovik/xsrl](http://www.dit.unitn.it/~lazovik/xsrl).
Cycle length

% back nondet

% back det

Nondet rate

Det rate

#states

Diff problem

Join allowed

Int effects

B: pos vs. neg

#st no B effect

Cycle length

Diff problem
References on XSRL using CP


XSRL Scenarios

- Pretty much similar to SQL for databases but for FUNCTIONALITY rather than DATA

- Could be used by:
  - Users or user interfaces to break the dependency on the underlying functionality
  - Two processes talking to each other
  - Reference process instantiation

- Nice consequences:
  - Processes are flexible and less depends on used services and other processes
  - Often goal failures may indicate what functionality is missing
  - Same reference process for many clients: domain experts/support teams may be shared
Request-driven process

User has no knowledge of underlying process semantics!
package org.xsrl.bp.freebp;

import org.xsrl.domain.Domain;
import org.xsrl.framework.registry.Registry;

public class FreeBPCompiler {
    public Domain compileBusinessProcess(String name) {
        try {
            SAXFreeBPHandler handler = new SAXFreeBPHandler();
            return handler.parseFreeBP(name);
        } catch(Exception ex) {
            ex.printStackTrace();
            System.exit(1);
            return null;
        }
    }
}
Process lifecycle

Reference process: Design → Scope → Deploy

XSRL requests issued

Monitor not satisfied requests

Executable processes

Process is updated or customized according to monitored request failures

However, reasons may be different:
- incorrect formulation / unintended usage
- VERY special case
- missing behavior
Data inter-dependencies

- No relations between variables or/and between constraints:
  - variables are inter-dependent: hotel price is discounted if user is IPA member
  - operating system chosen implies other software installation: app/web/mail server, etc

- Functional inheritance
  - Process or service MUST support a coordination protocol
  - Protocol can be defined in different forms: from state-transition system to a set of business rules

- Some preliminary ideas may be found in
Non-functional requirements

- XSRL is suitable for expressing non-functional requirements
  - It is usable when non-functional requirement addresses **structural** issues of the business process, where the value for the property depending on the process execution
    - **security**: before any payment service use, authentication service must be called first!
    - **transactional**: payment service must be either successful or corresponding compensation activities are executed!

- Not good for out-of-process properties:
  - examples:
    - the type of underlying protocol or app server used;
    - the service response time if it does not affect other services;

- More scenarios and preliminary ideas may be found in
Balance between offline and online

- When the process is strictly defined:
  - + used as it is, no planning/reasoning phase is needed;
  - + (rather) easy to monitor, steps are clearly defined;
  - - very difficult to maintain, changes are possible but very expensive;

- When the process is built from the scratch for every request:
  - + receive what you need as far as there are necessary atomic services;
  - - tends to be computationally non-tractable for complex cases;
  - - difficult to issue a request to take into account all possible executions – issuer does not have a proper domain knowledge;
  - - expert reference processes/domain knowledge is ignored;

There is no single silver bullet for all domains: some domains need more freedom (traveling, PC assembling), some are more strict (inter-banking communication).
XSRL and AI

- XSRL is more than just an application of AI in SOC:
  - AI Planning:
    - traditional approaches fail in web services with a lot of non-determinism and uncertainty;
    - our framework for interleaving planning and execution is a new hammer for new nails;
  - CP community
    - CP on tree- and graph-like structures;
    - Evolution is modeled in terms of constraints;
Other open issues

- Depends a lot on smart service selection
  - more services -> more important to make a right choice
  - as far as the service is taken it affects many constraints, not easy make an intelligent backtrack!

- Optimality of the solution
  - supported by the CP naturally, but not yet added to the language

- Abstract from SOC standards (BPEL/WSDL/etc):
  - To be successful in SOC nice graphical tools and full standards support must be maintained;
Conclusions

- Service request language (XSRL):
  - with expressive power enough for the user,
  - but, at the same time, computationally tractable;
  - with formally defined semantics;

- Development of a service composition framework:
  - able to deal with non-determinism and uncertainty, typical for web services
  - planning algorithms based on constraint programming;

- Implementation and evaluation of the language and framework:
  - feasibility of algorithms for service business processes;

- Assertion-based monitoring of process execution:
  - providers have more control over execution;
  - applicability of the proposed languages for expressing and monitoring QoS properties.