RBSLA: Rule Based Service Level Agreement Language

Agenda

- State-of-Art: Service Level Agreement
- Approach: Logic Programming (Rules)
- Declarative RBSLA based on RuleML
- Discussion

Adrian Paschke (TUM)
IAWTIC 2005, Vienna, Austria
An SLA contract is a document that describes the performance criteria a provider promises to meet while delivering a service.

It typically also sets out the rights and obligations each person has in a particular context or situation, the remedial actions to be taken and any penalties that will take effect if the performance falls below the promised standard.
Challenges

- **Dependent Rules**
  
  “If the **average availability** falls below 98% then the **mean time to repair** must be less than 10 min.”

- **Graduated Rules**
  
  - Monitoring Schedules

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Time</th>
<th>Availability</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime</td>
<td>8-18</td>
<td>99%</td>
<td>4 sec.</td>
</tr>
<tr>
<td>Standard</td>
<td>18-8</td>
<td>95%</td>
<td>10 sec.</td>
</tr>
<tr>
<td>Maintenance</td>
<td>0-4 *</td>
<td>30%</td>
<td>-</td>
</tr>
</tbody>
</table>

  - Escalation Levels with Role Model

<table>
<thead>
<tr>
<th>Level</th>
<th>Role</th>
<th>Time-to-Repair</th>
<th>Rights / Obligations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Process Manager</td>
<td>10 Min.</td>
<td>Start / Stop Service</td>
</tr>
<tr>
<td>2</td>
<td>Chief Quality Manager</td>
<td>Max. Time-to-Repair</td>
<td>Change Service Levels</td>
</tr>
<tr>
<td>3</td>
<td>Control Committee</td>
<td>-</td>
<td>All rights</td>
</tr>
</tbody>
</table>
Challenges

■ Dynamic Rules

“There might be an unscheduled period of time which will be triggered by the customer. During this period bandwidth must be doubled.”

■ Normative Rules with Violations and Exceptions

“The provider is obliged to repair an unavailable service in $t_{time-to-repair}$. If she fails to do so (violation) the customer is permitted to cancel the contract.”
State of Art

- **Natural language SLAs**
  - Large amounts of contracts
  - Dynamic SOA environment (utility/on-demand computing)
  - Different systems and people (roles) are involved

- **Formal representation languages, e.g. XML based WSLA:**
  - Need interpreter
    - Conventional imperative programming languages, e.g. Java
  - Limited to simple Boolean logic to represent contract rules
  - No variables, no complex terms, no quantifiers, no rule chaining

- **Commercial monitoring tools mainly focus on IT systems/resources**
  - Missing link between technical view and SLA view
  - Contract/Business logic is buried in the code or database tiers
  - Contract rules (logic) are adjusted by parameters
  - Control flow must be completely implemented

⇒ **SLA Representation needs new levels of Flexibility and Automation**

⇒ **Expressive KR which keeps computational tractable**
General Idea: Logic Programming

- Formalisation of contract rules in a logic based rule language

- Derivation Rules:
  
  Body $\Rightarrow$ Head (If Body then Head)

  \[ P_1 \land \ldots \land P_n \land \neg P_{n+1} \land \ldots \land \neg P_m \Rightarrow C \]  
  
  (Horn rules with NaF \( \neg \))

- Example:
  
  If the turnover of a customer is greater then 500$ then the customer is a gold customer.

  
<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate Variable</td>
<td>Predicate Variable</td>
</tr>
<tr>
<td>getTurnover(C)</td>
<td>gold Customer</td>
</tr>
<tr>
<td>500$</td>
<td></td>
</tr>
</tbody>
</table>

  If a customer is a gold customer then the customer qualifies for a discount of 15% on the service base price.

  
<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate Variable</td>
<td>Predicate Variable Constant</td>
</tr>
<tr>
<td>gold Customer</td>
<td>discount Customer 15%</td>
</tr>
</tbody>
</table>
Our Approach

- Management / Control Layer
  - RBSLM
    - Rule Based Service Level Management Tool
  - Dynamic Business / Contract Logic Layer
    - RBSLA
      - Declarative Rule Based Service Level Agreement Language
  - Knowledge Representation Layer
  - Static Execution Layer
    - Mandarax Rule Engine with Prova
    - Java Virtual Machine
  - External System Layer
    - Existing Business Tools / Business Data / Business Objects
      - System and Quality Management Tools etc.
      - EJBs / Web Services / APIs etc.
      - Databases / Datawarehouses / Files etc.

- Contract Log
### Expressive KR: ContractLog

<table>
<thead>
<tr>
<th>Logic</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horn Logic</td>
<td>Derivation Rules (rule chaining) + Negation as Failure, Procedural Attachments, External Data Integration, Typed Logic</td>
</tr>
<tr>
<td>Event-Condition-Action rules (ECA)</td>
<td>Active behaviour (events, actions) + Update primitives for Active Rules</td>
</tr>
<tr>
<td>Event Calculus</td>
<td>Temporal reasoning over effects of events on fluents (contract tracking)</td>
</tr>
<tr>
<td>Defeasible logic</td>
<td>Conflict resolution, default rules and priority relations of rules.</td>
</tr>
<tr>
<td>Deontic logic</td>
<td>Rights and obligations with violations an exceptions of norms.</td>
</tr>
<tr>
<td>Description logic</td>
<td>Contract vocabularies, domain-specific concepts (term typing)</td>
</tr>
</tbody>
</table>
Rule Based Service Level Agreement Language

- Abstract declarative syntax → Simplify authoring/writing of SLAs

- Based on RuleML

- Goals:
  - Machine-Readability and execution in standard LP inference engine ~ rule engine (via Transformation)
  - Tool-Support
  - Interoperability with other (rule) languages

- RuleML
  - Standardization: Open, producer-independent, XML/RDF based web language for rules
  - Rule types:
    - Derivation rules (business rules), e.g. representation with LP
    - Reaction rules (production rules, ECA rules) (not specified yet)
    - Transformation rules, Integrity constraints
  - Currently: Derivation Rules
  - Unitized Structure: Modules for DataLog, HornLog (with Naf), Disjunctive LP, FOL (extended LPs)
  - Since version 0.85: Object-Oriented KR (User-Level Roles, URI-Grounded Clauses, Order Sorted Terms)
  - Not intended to be executed directly, but transformation (e.g. XSLT) into target language, e.g. Prolog.
Predicates (atoms) are n-ary relations defined as an \(<Atom>\).

Derivation Rules \(<Implies>\) consist of one or more conditions \(<body>\) and a conclusion \(<head>\).

Facts are derivation rules with empty bodies and are deemed to be always true: \(\text{Atom}\).

Queries are derivation rules with empty heads: \(\text{Query}\) (body).

Integrity constraints \(<Ic>\) and Transformation rules \(<Trans>\).
RBSLA Language

- **Main extensions to RuleML:**
  - Typed Logic and Procedural Attachments
  - External Data Integration
  - Event Condition Action Rules with Sensing, Monitoring and Effecting
  - (Situated) Update Primitives
  - Complex Event Processing and State Changes (Fluents)
  - Deontic Norms and Norm Violations and Exceptions
  - Defeasible Rules and Rule Priorities
  - Built-Ins, Aggregate and Compare Operators, Lists

- **Note:** Ongoing Work!!! RBSLA v0.1 (described in paper) based on RuleML 0.88

- **RBSLA v0.2 based on RuleML 0.9**
  - Several changes, e.g. events, actions became role tag (can be omitted \(\rightarrow\) compact syntax)

- **Download:** RBSLA project site: [http://ibis.in.tum.de/staff/paschke/rbsla/index.htm](http://ibis.in.tum.de/staff/paschke/rbsla/index.htm)

- **Layered structure (unitized in modules):**
  - Syntax Layers: (striped) RuleML, RBSLA, SLA-specific RBSLA, if-then syntax
RBSLA Interpretation / Execution

- Transformation to ContractLog KR (based on LP)
- Interpretation and execution in Prova/Mandarax rule engine (backward-reasoning)
- Refactoring of rules
  - Removing Disjunctions: $A_1,..,A_n, (B_1\lor B_2), ? C$ becomes $A_1,..,A_n, B', ? C$ and $B_1, ? B'$ and $B_2, ? B'$ (clausal normal form)
  - Removing conjunctions from rule heads: $B ? (H ? H')$ via Lloyd-Topor transformation into $B ? H$ and $B ? H'$
  - Removing classical negation (transformation to normal LPs with Naf, conflict resolution via defeasible logic)
  - Other examples are removing function symbols from rule heads etc.
  - Loop checking
  - Rule set optimization: Sorting, using strictly sequential operators (e.g. cuts) in combination with declarative semantics (well-founded resolution with tabling implemented as meta program)
- Type and Mode Checking
  - Static, e.g. $p(+) \leftarrow p(-)$
- Validation
  - Dynamic via test cases
- Defeasible Compiler (handling incomplete, contradicting knowledge)
  - Translation into LP Metaprogram
- RDFS/OWL Inference Layer (Description Logic Programs)
  - Hybrig approach: Normally DLP inferences, class/individual equivalences with tableau inference (pre-combilation approach)
Advantages

- Rules (contract logic) are separated from the application logic
  - Easier management and maintenance
  - Compact representation via rule chaining

- Logic based formalisation
  - Automation and Execution in rule engine (+extension)
  - Verification and Validation
    - Declarative test-driven validation and verification methods can be applied determining the correctness and completeness of contract specifications against user requirements.
    - Large rule sets can be automatically checked for consistency via static and dynamic structure checks testing types and modes (in-out parameter) of the arguments of rule predicates.
    - Explanatory reasoning chains provide means for debugging and explanation.

- Complex Event Processing

- (Pro-)active Monitoring and Contract State Tracking

- Time and Event-based Rights and Obligations Management

- Automated conflict detection and resolution (e.g. rule prioritization)
References on further Work


- SLA Representation, Management and Enforcement - Combining Event Calculus, Deontic Logic, Horn Logic and Event Condition Action Rules, E-Technology, E-Commerce, E-Service Conference (EEE05), Hong Kong, 2005.

Thank you for attention !!!!

Questions?