Rule Based Service Level Agreement Language

RBSLA

29.11.2005

IBIS, TU München
+49 89 289 17534
http://ibis.in.tum.de

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

Service Level Agreement (SLA)

- *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

Schedule	Time	Availability	Response Time
Prime	8 -18	99%	4 sec.
Standard	18-8	95%	10 sec.
Maintenance	0-4 *	30%	-

  - Escalation Levels with Role Model

Level	Role	Time-to-Repair	Rights / Obligations
1	Process Manager	10 Min.	Start / Stop Service
2	Chief Quality Manager	Max. Time-to-Repair	Change Service Levels
3	Control Committee	-	All rights

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* $t_{time\text{-}to\text{-}repair}$*. 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 → Head (If Body then Head)

  P1 ´ … ´ Pn ´ ~Pn+1 ´ … ´ ~Pm → C (Horn rules with NaF „~“)

- Example:

  *If the **turnover of a customer is greater then 500$** then the **customer is a gold customer**.*

Prerequesite		
Predicate	Complex Term	Constant
>	getTurnover(C)	500$

→

Conlusion	
Predicate	Variable
gold	Customer

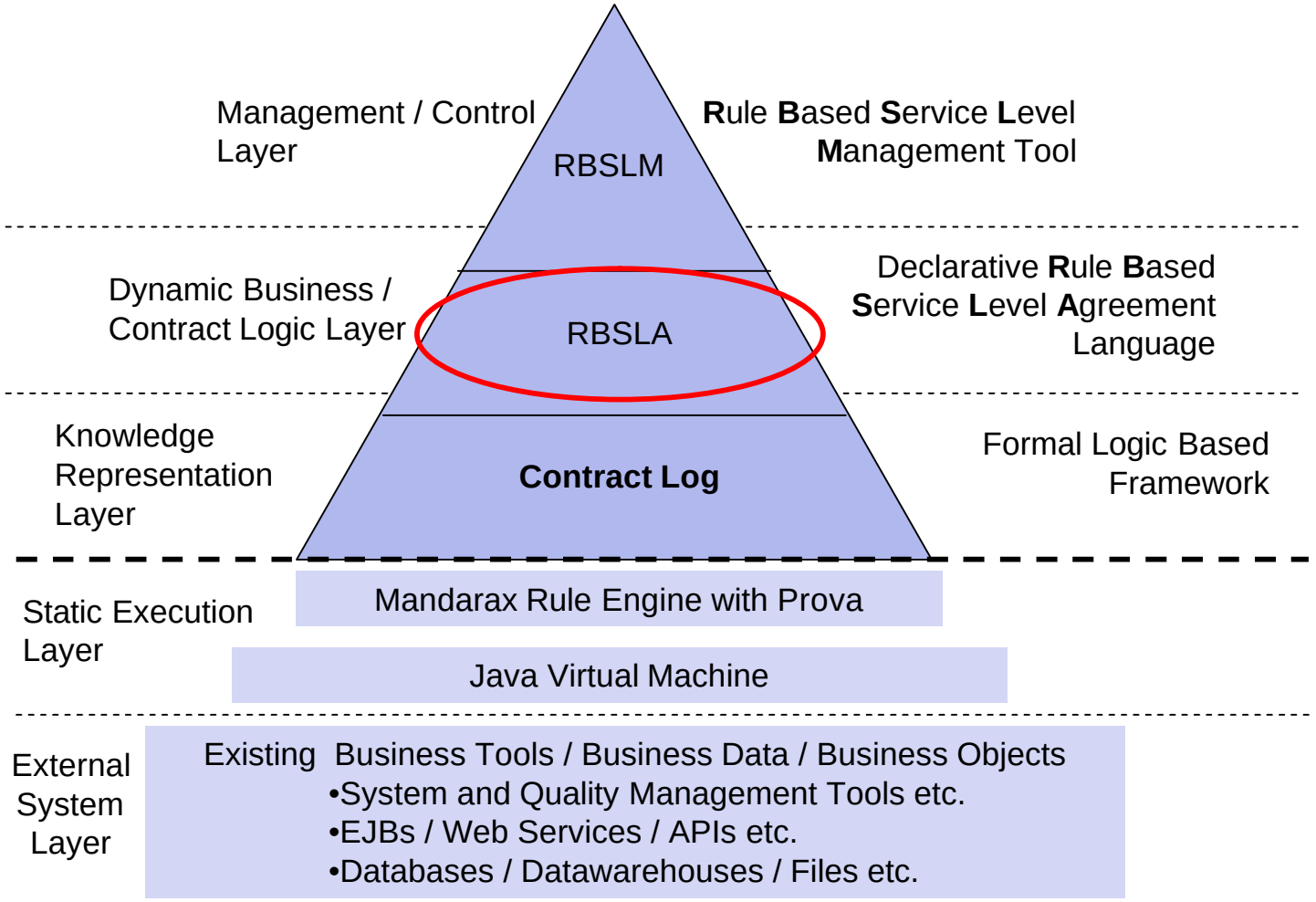
*If a **customer is a gold customer** then the customer qualifies for a **discount of 15%** on the service base price.*

Prerequesite	
Predicate	Variable
gold	Customer

→

Conclusion		
Predicate	Variable	Constant
discount	Customer	15%

Our Approach



Expressive KR: ContractLog

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

Rule Based Service Level Agreement Language

- Abtract 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 traget language, e.g. Prolog.

RuleML - Derivation Rules- Building Blocks

- **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: (Atom)
- **Queries** are derivation rules with empty heads: <*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 → compact syntax)**
- **Download:** RBSLA project site: http://ibis.in.tum.de/staff/paschke/rbsla/index.htm.
- Layered structure (unitized in modules):
  - KR Layers: ContractLog layers: ECA, EC, Defeasible, Deontic, Typed LP, domain-specific …
  - 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
  - *Narrowing:* $A_1,..,A_N.?\ B$ and $A_1,..,A_N.?\ C$ becomes $A_1,..,A_N.?\ A$; $A..?\ B$; $A ?\ C$ (eliminates redundancies)
  - *Removing Disjunctions*: $A_1 .. A_n , (B_1 \acute{U} B_2) ?\ C$ 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(+)* ← *p(-)*
- Validation
  - Dynamic via test cases
- Defeasible Compiler (handling incomplete, contradicting knowldege)
  - 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 seperated 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

- Paschke, A.: RBSLA - A declarative Rule-based Service Level Agreement Language based on RuleML, International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC 2005), Vienna, Austria, 2005.
- Paschke, A., Dietrich, J., Kuhla, K.: A Logic Based SLA Management Framework, Semantic Web and Policy Workshop (SWPW), 4th Semantic Web Conference (ISWC 2005), Galway, Ireland, 2005.
- Jens Dietrich and Adrian Paschke, On the Test-Driven Development and Validation of Business Rules, 4th International Conference on Information Systems Technology and its Applications (ISTA 2005), New Zealand, May 2005.
- Paschke, A., Schnappinger-Gerull, E.: A Categorization Scheme for SLA Metrics, MKWI 2006, Passau, Germany, 2006.
- 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.
- Paschke, A.: Rule Based Service Level Agreements, Project Site: http://ibis.in.tum.de/staff/paschke/rbsla/index.htm.

Thank you for attention !!!!

Questions?