ContractLog: An Approach to Rule Based Monitoring and Execution of Service Level Agreements

A Rule Based Approach using Logic Programming Techniques

Adrian Paschke, Martin Bichler, Jens Dietrich

RuleML Conference *12.11.2005*

Overview

- Service Level Agreements
- **ContractLog:** Rule Based SLA Management
- Discussion

Service Level Agreement

- *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.*

State of the Art

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

2. 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

3. 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

State of Art

- In most commercial SLA tools the contract/business logic is buried in the application code or database tiers
- Formal representation languages such as WSLA based on imperative interpreters supporting only simple boolean logic.

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

SLA Wish List

What changes could be made to your current SLA to meet your standards?

	Number of Responses
Better guaranteed availability	10
Better responsiveness to SLA violations	6
More flexibility to modify SLA terms	5
Better measurement metrics	5
Reduce the complexity of the document	4
Better remuneration of SLA violations	3
Other	3

Source: Zona Research Inc., Redwood City, Calif.

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

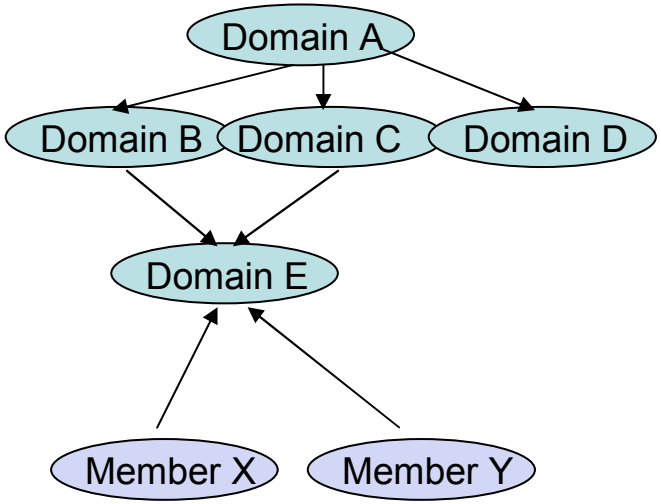
  "*The provider is **obliged** to repair an unavailable service in $t_{time\text{-}to\text{-}repair}$. If she fails to do so (**violation**) the customer is **permitted** to cancel the contract.*"

ContractLog Overview

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)

Typed Horn Logic with Naf

- Derivation Rules:
  - $P_1 \wedge \ldots \wedge P_n \wedge \sim P_{n+1} \wedge \ldots \wedge \sim P_m \vee P_k \rightarrow C_1 \wedge .. \wedge C_n$
  - „~“ : Negation as Failure + Disjunctions + Conjunctions in rule head
- Terms in rules are either un-typed / typed
  - Order-sorted Type Systems (directed acyclic graph)



We support:
- **Class hierarchies (Java Type System)**
- **Semantic Web Taxonomies (RDFS/res. OWL)**
- **External Databases (relational DB)**
- **XML Schema Datatypes**

Extended Unification: (greatest lower bound)
Un-Typed/Typed Variables/Constants

- Extensive Support for Procedural Attachments represented as complex terms or boolean-valued predicates in the rules
  - $O_m[p_1..p_n] \rightarrow [r_1..r_n]$ e.g., $\text{java.lang.Integer}_{\text{parseInt}}[1234] \rightarrow \text{Integer}(1234)$
  - Integration of external Databases / EJB´S / Web Services / Tools etc.

Typed Logic

- Input-Output Modes
  - "+" The term is intended to be *input* (constant or bound variable)
  - "-" The term is intended to be *output* (free variable)
  - "?" The term is *undefined/arbitrary* (input or output) (default value)
  - Example
    add(+,+,-)
    add(X_1,X_2,R) ← bound(X_1), bound(X_2), free(R) // add(+,+,-)

- Types / Modes
  - Instrumentation of a logic program, i.e. *approximation of the intended interpretation*
  - Static Type/Mode Checking
    - Type Failure: No Greatest Lower Bound of Types
    - Mode Failure: "+" and "-" conflicts
  - Dynamic Type/Mode Checking

→ Link Domain-Models / Semantic Web Vocabularies to domain-independent rules
→ Restrict search space (increase performance of reasoning process)
→ Safeguard authoring process

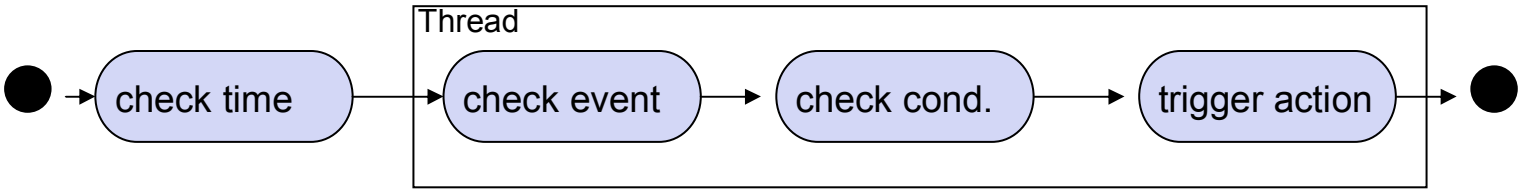
Event Condition Action

- ECA Rules: eca(time,event,condition,action)
  - Problem: Query-driven Backward Reasoning (simulate activity)

ECA rule: *eca (everyMinute, pingService,notMaintenance notify)*

Referenced Derivation Rules: (queried by daemon process)

Time:	*everyMinute()* ← …
Event:	*pingService()* ← …
Condition:	*notMaintenance()* ← …
Action:	*notify()* ← …



- Active Rules (via update primitives in ECA rules)
  - Assert "+" ; Retract/RetractAll "-"

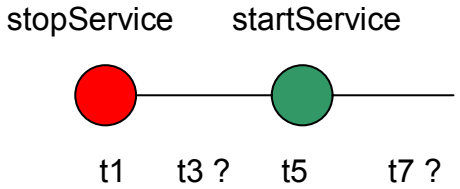
    Example: **-** *contact(L, P)* ← **-** *esc_lv(L), contact(L, P)*

Event Calculus

- Effects of Events/Actions on Fluents
  - Rules for state transitions / derive actual contract State ~ Context
  - Contract State Tracking
  - Time-based / Context based complex events
- ***EC Basic Axioms***:
  - *happens(E,T)* — *event E happens at time point T*
  - *initiates(E,F,T)* — *event E initiates fluent F for all time>T*
  - *terminates(E,F,T)* — *event E terminates fluent F for all time>T*
  - *holdsAt(F,T) fluent F holds at time point T*
- ***EC Extensions***:
  - *valueAt(P,T,X)* — *parameter P has change-able value X at time point T*
  - *planned(E,T)* — *event E is believed to happen at time point T*

Example:

initiates(*stopService,serviceUnavailable,T*)
terminates(*startService,serviceUnavailable,T*)
happens(*stopService,t1*); ***happens***(*startService,t5*)
holdsAt(*serviceUnavailable,t3*)? → ***true***
holdsAt(*serviceUnavailable,t7*)? → ***false***



Deontic Logic

- Logic of normative concepts: oblige, permit, forbid
  - Normative propositions with truth values
- Standard Deontic Logic (SDL)
  - OA , PA, FA …
  - Inference Rules
    OA → PA; FA → O¬A; PA → ¬FA …
    Consequential Closure: A → B / OA → OB
    Valid Scheme: ¬(OA ∧ O¬A)
- Shortcomings
  - Norms are not personalized / object oriented
  - Norms are time-less
  - Effects of events/actions are ignored
  - Exceptions and Violations lead to Paradoxes, i.e. SDL is inconsistent.

Event Calculus based Deontic Logic

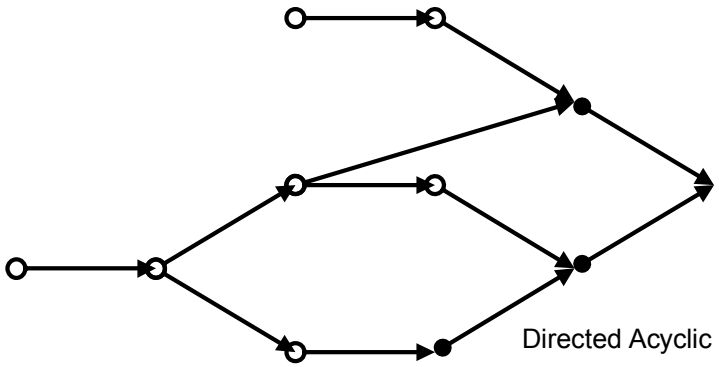
- Norms are personalized, time-varying Fluents
  $N_{S,O}A$ (N=norm, S=Subject, O=Object, A=Action)
- They are relative to their "context"
  - OA and O¬A might hold at the same time for different Subjects / Objects (or at different times)
- Embedded in Event Calculus
  - Initially(**oblige**(S,O,A)) — *Norm holds initially*
  - Initiates(E1,**oblige**(S,O,A),T) — *Event E1 initiates norm*
  - Terminates(E2,**oblige**(S,O,A),T) — *Event E2 terminates norm*
- Authorization Control
  - Positive Authorization: "*By default everything is forbidden*" - holdsAt (norm,T)?
- Deontic Rules with conditional norms
  - A1 → OA2 : *happens(A1,T)* *initiates(A1,oblige(S,O,A2),T)*
- Defeasible Obligations / Prima Facie Obligations (~ promised duties)
  - Subject to **Exceptions:** *OA* and *E* → *O¬A* (E=Exception Event)

    terminates(E,oblige(S,O,A),T) initiates(E,oblige(S,O,notA),T)
- Contrary to Duty Obligations (CTD)
  - Subject to **Violations**: (¬A1 ∧ OA1) → V → OA2 (V= Violation OA2=CTD)

    holdsAt(oblige(S,O,A1),T), happens(notA1,T) → happens(V,T)
    initiates(V, oblige(S,O,A2),T)

Prioritization

rootCause(X) :- notCauseExists(X), morePriority(X,Y), *hasCausesSequence (Y)*.
notCauseExists(X) :- causeExists(X), !, fail. notCauseExists(X).
hasCausesSequence(endElement).
hasCausesSequence(X):- exists(X), morePriority(X,Y), hasCausesSequence (Y).
morePriority(X, Y):- [*Define conditional priority relation*]
causeExists(X) :- [*Define events*]



Directed Acyclic Graph

Example

informPerson(Role) :- rootCause(Role),…
morePriority(Admin,Amunensis).
morePriority(Manager,Admin).
causeExists(X):- holdsAt(ill(X),T).

"Search for white nodes which are reached via the inverse priority relation of a sequence of black nodes." (nodes ~ prioritization ; branches ~ states)

Prioritisation via Defeasible Logic

- Reasoning with incomplete or inconsistent information
  - Defeasible rules: body ==> head
  - Superiority relations: overrides(r1,r2)

- Transformation into meta-program in LP (ambiguity blocking with control literals)

(A)definitely(p(x1,x2,…,xn)). // fact
(B) definitely(p) :- definitely(q1), …, definitely(qk). // strict rules
(C) defeasible(X) :- definitely(X). // defeasible inference
(D) translation defeasible rules
defeasible(p) :- defeasible(q1),…, defeasible(qk), not(definitely(neg(p))),
ok(“this_rule_ID”, x1,…,xn).
ok(“this_rule_ID”, x1,…,xn) :- ok_line(“this_rule_ID”, “s1”, x1,…,xn),…,
ok_line(“this_rule_ID”, “sn”, x1,…,xn). // rules with head neg(p)
ok_line(“this_rule_ID”, “s”, x1,…,xn) :- blocked(“s”, x1,…,xn).
ok_line(“this_rule_ID”, “s”, x1,…,xn) :- defeated(“s”, x1,…,xn).
blocked(“s”, x1,…,xn) :- not(defeasible(qi)). // for all qi of rule “s”
defeated(“s”, x1,…,xn) :- not(blocked(“si”, x1,…,xn)), overrides(“this_rule_ID”, “si”).

based on Antoniou, G. et.al.: Embedding Defeasible Logic into Logic Programs

Discussion on Deontic Logic

- Large number of Paradoxes
  - sets of sentences that derive sentences with a counterintuitive reading.
- Most norms in SLAs are time-based
  → terminate primary norm and initiate secondary norm
- Problem: time-less paradoxes (gentle murder)
  (1) The service provider must not violate an agreed service level.
  (2) But, if a service level is violated, the violation should be as small as possible.
- Possible Solutions:
  - Concepts from Dyadic Deontic Logic with norm contexts
  - We use concepts from Defeasible Logic
    - Define contradicting norm as defeasible
    - Secondary norm has higher priority than primary norm (overriden)

Discussion Defeasible Logic

- Pros:
  +Simple rule-based approach
  +low computational complexity compared to mainstream non-monotonic reasoning
- Cons:
  - needs large meta-programs

  Solution: User Interface + Abstract syntax (RBSLA) + automated transformation into Meta-program (compiler)

Further Remarks

- Active Rules
  - Knowledge Updates (Loops/Termination, Confluence)
  - Using techniques from active databases
- Description Logic Programs (Benjamin Grosof et. al.)
  - Inference Rules implemented as Derivation Rules, e.g.:
    a:C, i.e., the individual *a* is an instance of the class *C*: C(a)
    C⊆ D, i.e., class C is subclass of D: D(X) ← C(X)
    C≡ D, i.e., class C is equivalent to class D: D(X) ← C(X)
    C(X) ← D(X)
    - (Pre-)Compilation of the equivalent classes and equivalent individuals (Closed World Assumption)
    - Apply Updates to the compiled model

Other Works

- Rule Based SLA Language (RBSLA)
  - Abtract declarative syntax → Simplify authoring/writing of SLAs
  - Based on RuleML
  - Goals:
    - Machine-Readability and Execution (via Transformation)
    - Tool-Support
    - Interoperability with other languages
- Verification and Validation of SLA Specifications / Rule Sets
  - Static Testing
  - Dynamic Testing
- User Interface: Rule Based Service Level Management Tool (RBSLM)
  - Service Dashboard for Visualization of Monitoring / Contract Tracking Results
  - Contract Manager with a Repository:
    - Supports two roles (Experts which define Templates / Functions / Interfaces / Contract vocabularies etc. and Business Practitioners which reuse the predefined templates
    - Safeguards authoring process via Validation and Verification
    - Translation of abstract syntax into logical core

Key Findings in a Nutshell

- 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.
- Reuse existing functionalities from external implementations and databases
- Combine logic programming and OO Programming
- Complex Event Processing (Context / Time-based Derived / Composite Events)
- (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

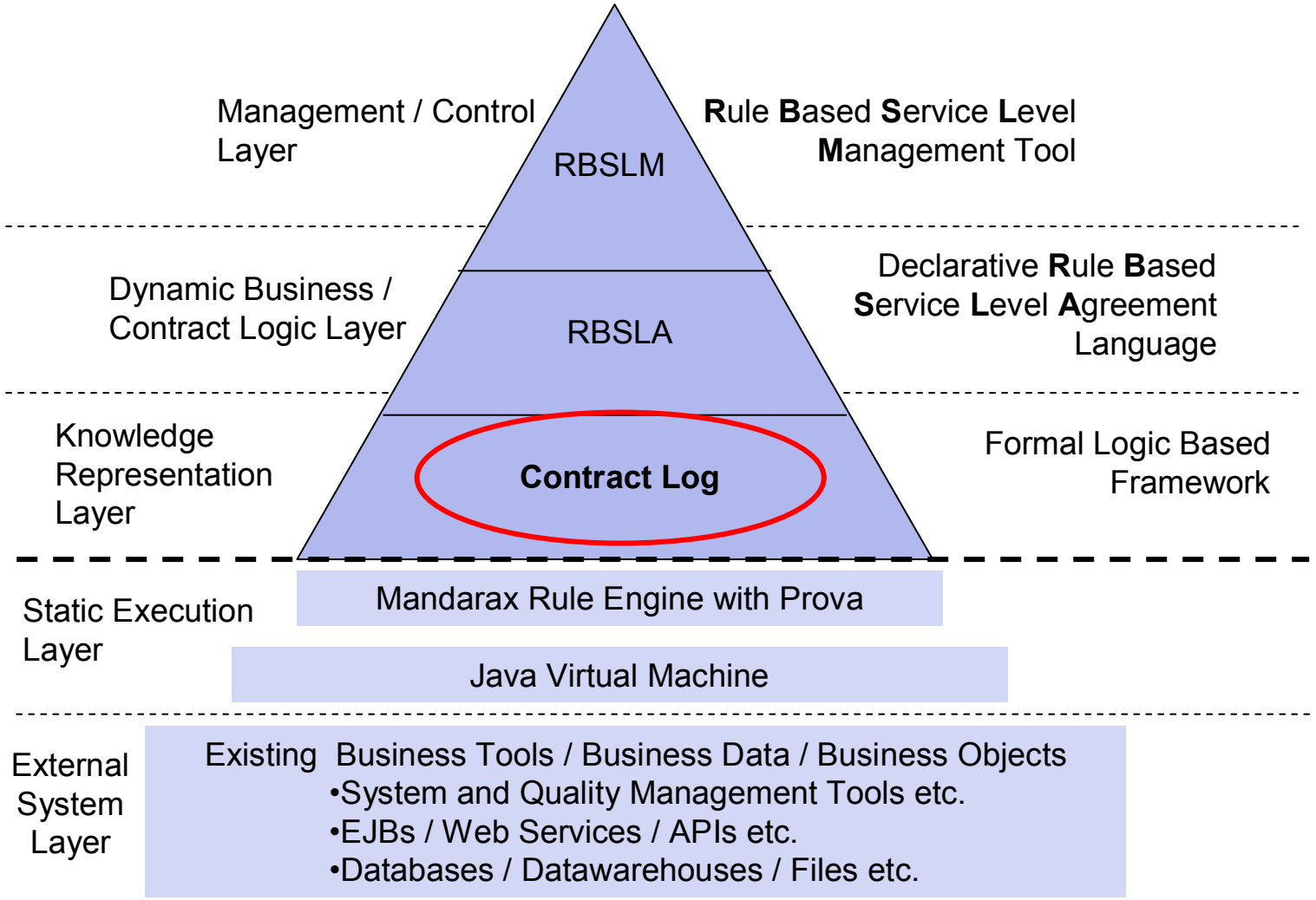
- Paschke, A.: RBSLA - A declarative Rule-based Service Level Agreement Language based on RuleML, International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC 2005), Vienna, Austria, 2005.
- Paschke, A., Dietrich, J., Kuhla, K.: A Logic Based SLA Management Framework, 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. submitted to MKWI 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?

Backup

Architecture

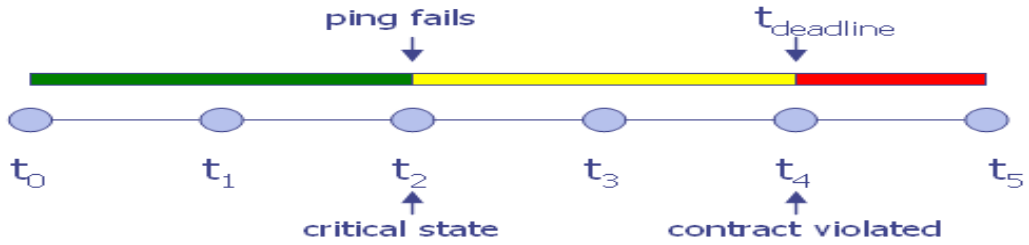


ContractLog Example

The service availability will be measured every t_{check} by a ping on the service.

*If the service is unavailable, the SP is **obliged** to restore it within $t_{deadline}$.*

*If SP fails to restore the service in $t_{deadline}$, SC is **permitted** to cancel the contract.*



Monitoring results:

t_0	service available
t_2	service unvailable → obligation(SP,Service,Start)
t_4	deadline exceeded → permission(SC,Contract,Cancel)

RBSLA

- Rule Based SLA Language (RBSLA)
  - Abtract declarative syntax → Simplify authoring/writing of SLAs
  - Based on RuleML
  - Goals:
    - Machine-Readability and Execution (via Transformation)
    - Tool-Support
    - Interoperability with other languages

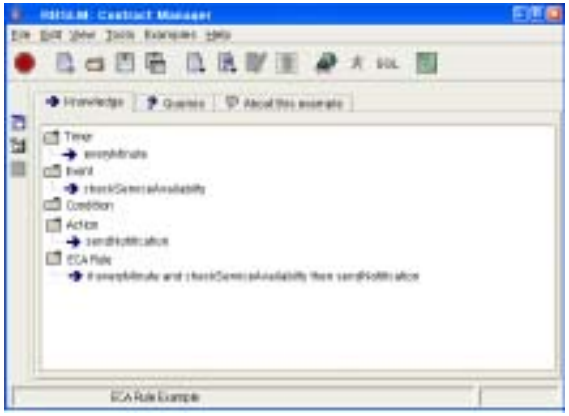
- Main extensions to RuleML:
  - *Typed Logic and Procedural Attachments*
  - *External Data Integration*
  - *Event Condition Action Rules with Sensing, Monitoring and Effecting*
  - *(Situated) Update Primitives*
  - *Complex Event Processing and State Changes (Fluents)*
  - *Deontic Norms and Norm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

RBSLA → ContractLog

- Refactoring of rules
  - *Narrowing:* $A_1,..,A_N. \rightarrow B$ and $A_1,..,A_N. \rightarrow C$ becomes $A_1,.., A_N. \rightarrow A$; $A.. \rightarrow B$; $A \rightarrow C$ (eliminates redundancies)
  - *Removing Disjunctions*: $A_1 .. A_n , (B_1 \vee B_2) \rightarrow C$ becomes $A_1 .. A_n , B' \rightarrow C$ and $B_1 \rightarrow B'$ and $B_2 \rightarrow B'$ (clausal normal form)
  - *Removing conjunctions from rule heads:* $B \rightarrow (H \wedge H')$ via Lloyd-Topor transformation into $B \rightarrow H$ and $B \rightarrow H`$)
  - Other examples are removing function symbols from rule heads etc.
- Type and Mode Checking
  - Static, e.g. $p(+) \leftarrow p(-)$
  - Dynamic via test cases
- Defeasible Compiler
  - Translation into Metaprogram

RBSLM

- Rule Based Service Level Management Tool
- Contract Manager (Contract Mgt. and Authoring)



- Service Dashboard (Monitoring and Contract Tracking)