Keynote at 8th International Web Rule Symposium (RuleML 2014) @ ECAI 2014
Prague, Czech Republic, August 18-22, 2014

Semantic CEP
with
Reaction RuleML

Prof. Dr. Adrian Paschke

Corporate Semantic Web (AG-CSW)
Institute for Computer Science, Freie Universität Berlin
paschke@inf.fu-berlin
http://www.inf.fu-berlin/groups/ag-csw/
Agenda

• Introduction to Semantic Complex Event Processing
• Event Processing Reference Model and Reference Architecture
• Reaction RuleML
• Examples for Event Processing Functions
• Summary
Agenda

• Introduction to Semantic Complex Event Processing

• Event Processing Reference Model and Reference Architecture

• Reaction RuleML Standard

• Examples for Event Processing Functions

• Summary
Event Processing vs. Databases

Ex-Post Data Queries

Database Queries

Query Processor

Real Time Data Processing

Incoming Events

Event Subscriptions

Processing moment of events

Future

Time

past
Knowledge Value of Events

Value of Events

Predictive Analysis

Proactive actions

Real-Time

Complex Event Processing

Late reaction or Long term report

Historical Event

Before the event
At event
Some time after event
e.g. 1 hour

Time
CEP – Why do we need it?

Why do we need CEP?

- **RTE**
  - Quick decisions, and reactions to threats and opportunities according to events in business transactions

Example Application Domains

- **Monitoring, BAM, ITSM,**
  - Monitor and detect exceptional IT service and business behavior from occurred events

- **Information Dissemination**
  - Valuable Information at the Right Time to the Right Recipient

CEP Media

- Detect
- Decide
- Respond

- Expert Systems

- Enterprise Decision Management
Complex Events – What are they?

- Complex Events are aggregates, derivations, etc. of Simple Events

Complex Event Processing (CEP) will enable, e.g.

- Detection of state changes based on observations
- Prediction of future states based on past behaviors
Complex Event Processing – What is it?

- CEP is about complex event detection and reaction to complex events
  - Efficient (near real-time) **processing** of large numbers of events
  - Detection, **prediction** and **exploitation** of relevant complex events
  - Supports **situation awareness**, track & trace, sense & respond

```
INSERT INTO OrdersMatch_s
SELECT B.ID, S.ID, B.Price
FROM BuyOrders_s B, Trades_s T, SellOrders_s S,
MATCHING [30 SECONDS: B, !T, S]
on B.Price = S.Price = T.Price;
```
Complex Event Processing – What is it?

- Complex Event Processing (CEP) is a discipline that deals with event-driven behavior.
- Selection, aggregation, and event abstraction for generating higher level complex events of interest.
The Many Roots of CEP…

Complex Event Processing (CEP) is a discipline that deals with event-driven behavior.
Example Event Processing Languages

- **Inference Rules**
- **ECA / Reaction Rules**
- **Agent Oriented**
- **SQL extension**
- **State oriented**
- **Imperative/Script Based**

Languages:
- Reaction RuleML
- Starview
- EventZero
- Aleri
- Coral8
- Esper
- Netcool Impact
- WBE
- Prova

Tools:
- TIBCO
- XChangeEQ
- AMiT
- RuleCore
- Agent Logic
- Spade
- Oracle
- Streambase
- Apama

Related links:
## Commercial CEP Market

### CEP Market Players to June 2010

<table>
<thead>
<tr>
<th>Year</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>'99</td>
<td>iSpheres</td>
</tr>
<tr>
<td>'00</td>
<td>ePatterns</td>
</tr>
<tr>
<td>'01</td>
<td>Apama</td>
</tr>
<tr>
<td>'02</td>
<td>IBM AMIT</td>
</tr>
<tr>
<td>'03</td>
<td>Progress Apama</td>
</tr>
<tr>
<td>'04</td>
<td>Aptsoft</td>
</tr>
<tr>
<td>'05</td>
<td>IBM WSBSE</td>
</tr>
<tr>
<td>'06</td>
<td>Streambase</td>
</tr>
<tr>
<td>'07</td>
<td>IBM Infosphere Streams</td>
</tr>
<tr>
<td>'08</td>
<td>Rulecore</td>
</tr>
<tr>
<td>'09</td>
<td>TIBCO BusinessEvents</td>
</tr>
<tr>
<td>'10</td>
<td>Syndera</td>
</tr>
<tr>
<td></td>
<td>WestGlobal Vantifey</td>
</tr>
<tr>
<td></td>
<td>Rhysome Zoma</td>
</tr>
</tbody>
</table>

**Key**
- Query-based
- ECA Rule-based
- Inference Rule-based
- Other e.g. State-oriented
- Research project or not supported
- Left the market or embedded-only
- Left the market (permanently)

**Source:** Tibco Software Inc. 2010
Example Research Prototypes

- **Finite State Automata, e.g.:**
  - ADAM, ACOOD, ODE, SAMOS, COMPOSE, SNOOP (active database research in 80s & 90s)
  - SASE, Cayuga, Esper

- **Rule-Based CEP Engines, e.g.:**
  - Prolog, Prova, Drools, ETALIS

- **Data Stream Engines, e.g.:**
  - Stream CQL, PIPE, TelegraphCQ, Gigascope, Aurora Borealis, SPADE

- **Stream Reasoning, e.g.:**
  - C-SPARQL, EP-SPARQL, SCEPter, SPARQLStream, CQELS
Complex Event Definitions – How?

Example Event Algebra Operators:

- **Sequence Operator (;)**: \((E_1; E_2)\)
- **Disjunction Operator (\(\lor\))**: \((E_1 \lor E_2)\), at least one
- **Conjunction Operator (\(\land\))**: \((E_1 \land E_2)\)
- **Simultaneous Operator (=)**: \((E_1 = E_2)\)
- **Negation Operator (\(\neg\))**: \((E_1 \land \neg E_2)\)
- **Quantification (Any)**: \(\text{Any}(n)\) \(E_1\), when \(n\) events of type \(E_1\) occurs
- **Aperiodic Operator (Ap)**: \(Ap(E_2, E_1, E_3)\), \(E_2\) Within \(E_1 \& E_3\)
- **Periodic Operator (Per)**: \(Per(t, E_1, E_2)\), every \(t\) time-steps in between \(E_1\) and \(E_2\)

Core CEP Life Cycle

1. Event Production
2. Event Definition
3. Event Selection
4. Event Aggregation
5. Event Handling
6. Event Consumption
Basic Definitions* (1)

• **Atomic Event (also raw event or primitive event)**
  – An atomic event (also raw event or primitive event) is defined as an instantaneous (at a specific point in time), significant (relates to a context), indivisible (cannot be further decomposed and happens completely or not at all) occurrence of a happening.

• **Complex Event**
  – composed (*composite event*) or derived (*derived event*) from occurred atomic or other complex event instances, e.g. according to the operators of an event algebra or as a result of applying an algorithmic function or process to one or more other events
  – included events are called *components* while the resulting complex event is the *parent event*
  – first event instance contributing to the detection of a complex event is called *initiator*, where the last is the *terminator*; all others are called *interiors*

• **Event Pattern**
  – An event pattern (also event class, event definition, event schema, or event type) describes the structure and properties of an (atomic or complex) event
  – It describes on an abstract level the essential factors that uniquely identify the occurrence of an event of that type, i.e. its detection condition(s) and its properties with respect to instantiation, selection and consumption.
Basic Definitions* (2)

• Event Instance
  – A concrete instantiation of an event pattern is a specific event instance (also event object).

• Situation
  – A situation is initiated or terminated by one or more (complex) events, i.e. the effect of a complex event (complex event + conditional context)

• Situation Individuals
  – Situation individuals are discrete individuals of situations with a fixed context allocation of time, date, location, participant, etc. They are not repeatable and are temporally connected.

• Complex Action
  – Compound action from occurred atomic or other complex action instances
Classification of the Event Space* (1)

- **Event Processing**

  - **Atomic vs. complex event processing**: only support for atomic events or computation of complex events, i.e. combinations of several atomic (and complex) events
  
  - **Short term vs. long term**:
    - *Short term*- immediate reaction: transient, non-persistent, real-time selection and consumption of events (e.g. triggers, ECA rules)
    - *Long term*- retrospective, deferred, or prospective: Persistent events, typically processed in retrospective e.g. via KR event calculi reasoning or event algebra computations on a event instance history; but also prospective planning / proactive, e.g. KR abductive planning
  
  - **Separation of event definition, selection and consumption**:
    - Event definition: algebraic, temporal logic, event/action calculus, …
    - Event selection (from instance sequence): first, last, all, …
    - Event consumption: consume once, do not consume, …
  
  - **Deterministic vs. non-deterministic**: simultaneous occurred events trigger more than one rule and give rise to only one model (deterministic, e.g. by priority-based selection of rules) or two or more models (non-deterministic)
  
  - **Active vs. passive**: actively query / compute / reason / detect events (e.g. via monitoring, querying / sensing akin to periodic pull model or on-demand retrieve queries) vs. passively listen / wait for incoming events or internal changes (akin to push models e.g. publish-subscribe)
  
  - **Local vs. global**: events are processed locally within a context (e.g. a process, conversation, branch) or globally (apply global on all rules)
Classification of the Event Space* (2)

- **Event Type**
  - Flat vs. semi-structured compound data structure/type, e.g. simple flat representations (e.g. propositional) or complex objects with or without attributes, functions and variables
  - Primitive vs. complex, e.g. atomic, raw event types or complex event types
  - Homogenous vs. heterogeneous Subevents are different from the complex event

- **Temporal:**
  - absolute (e.g. calendar dates, clock times),
  - relative/delayed (e.g. 5 minutes after …),
  - durable (occurs within an interval),
  - durable with continuous, gradual change (e.g. clocks, countdowns, flows)

- **State or Situation:**
  - without explicit boundaries
    - State (e.g. “the fire alarm stopped“): selective or interval-oriented, homogenous (the state holds in all subintervals and points), without dynamic change
    - Process (e.g. “he runs“): interval-oriented, homogenous (the process is executed in sub-intervals), continuous dynamic change
    - Iterative event (e.g. “the alarm rings again and again“): selective or interval-oriented, homogenous
  - with explicit boundaries
    - dynamic change (e.g. the light changes the colour)
    - interval-based (within the next 5 minutes)
    - frequency (e.g. the telephone rung three times)

- **Spatio / Location:** durable with continuous, gradual change (approaching an object, e.g. 5 meters before wall, “bottle half empty”)

- **Knowledge Producing:** changes agents/systems knowledge/belief and not the state of the external world, e.g. look at the schedule → internal effect, e.g. **belief update and revision**
Classification of the Event Space* (3)

- **Event Source**
  - *Implicit* (changing conditions according to self-updates) vs. **explicit events** (e.g. production rules vs. ECA rules)
  - *By request* (query on database/knowledge base or call to external system) vs. **by trigger** (e.g. incoming event message, publish-subscribe, agent protocol / coordination)
  - *Internal database, KB update events* (e.g. add, remove, update, retrieve) or **external explicit events** (inbound event messages, events detected at external entities)
  - *Generated, produced* (e.g. phenomenon, derived action effects) vs. **occurred** (e.g. detected or received event)
Classification of the Condition Space*

- **Logical conditions vs. pattern-based test constraints:**
  - Logical conditions act as goals on complex conditional logic represented in terms of (derivation) rules as in e.g. backward-reasoning logic programming; might support e.g. variables (new free and bound event variables) + backtracking over different variable bindings, logical connectives (conjunction, disjunction and negations)
  - Ground pattern tests e.g. on changed conditions (~ implicit events in production rules) and pattern matching tests

- **Conditions on the state before/after/intermediate the action change**

- **Scoped conditions:** a scoped condition only applies on an explicitly defined scope (e.g. part of the knowledge base (e.g. a module) or working memory

- **External data integration:** conditions can define constructive views (queries) over external data sources and bind values/objects to variables
Classification of the Situation Space* (1)

• **Situation**
  - A situation is initiated or terminated by one or more (complex) events, i.e. complex event + conditional context

• **Situations as a narrow and closed whole without addressing the elapsed time**
  - heterogeneous: interior situations differ from the overall situation
  - Sub-types
    • Dynamic change
      e.g. the traffic light changes the color
    • With time frame
      e.g. within 5 minutes
    • Frequency
      e.g. it rings three times

• **Situation in the flow/process, without addressing the narrowness / closeness**
  - homogeneous: interior situations are similar to the general situation
  - Sub-types
    • State
      e.g. he lays on the floor
    • Process
      e.g. he runs
    • Iterative process
      e.g. he coughs (again and again)
    • Habitual process
      e.g. he smokes
Situation Individuals, Situation Types and Situation Descriptions (2)

- Situation are discrete individuals
  - Reification of individuals possible
- Situation descriptions assign content and temporal properties to situation individuals
  - Dating of situation individuals by mapping into the time
  - Different situations might have corresponding properties
- Situation descriptions connect situation types with validity intervals
  - Situation typen as relation with time argument
Situation Individuals, Situation Types and Situation Descriptions (3)

• **Situation Individuals/Token**
  – Have a fixed time allocation (Date), location, participant, …
  – Are not repeatable
  – Are temporally connected

• **Situation Types**
  – Might have several time allocations (validity intervals)
  – Can be repeated
  – Validity intervals might not be successional

• **Types of Situation Descriptions**
  – Based on differences between situation individuals (Sorting of Individuals)
  – Based on the differences between situation types with respect to their interaction with the time structure (Monotony, Homogeneity)
# Types of Situation Descriptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Point – Period</th>
<th>Change</th>
<th>Time Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Description</td>
<td>Relating to point or period</td>
<td>no: –dyn</td>
<td>homogeneous: The state holds for all interior interval and points</td>
</tr>
<tr>
<td>Process Description</td>
<td>Relating to a period, never relating to a point</td>
<td>continuous: +dyn</td>
<td>restricted homogeneous: In certain interior intervals the process also takes place</td>
</tr>
<tr>
<td>Event-based Description</td>
<td>Atomic point or complex interval</td>
<td>one: +dyn</td>
<td>heterogeneous: In interior intervals the event does not occur</td>
</tr>
</tbody>
</table>
Classification of the Action Space* (1)

• Action Processing
  – **Atomic vs. complex action processing**: only support for atomic action or execution of complex actions (e.g. complex workflows, execution paths)
  – **Transactional vs. non-transactional processing**: Transactional complex actions possibly safeguarded by post-conditional integrity constraints / test case tests might be rolled-back in case of failures or committed
  – **Hypothetical vs. real execution**: actions have a real effect on the internal (add new fact) or external world, or are performed hypothetically, i.e. the effect is derive or computed implicitly
  – **Concurrent vs. sequential execution**: actions can be processed concurrently (e.g. in parallel branches / threads)
  – **Local vs. global execution**: actions are executed in a global context or locally in a (workflow) process, scope or conversation station (e.g. outbound action messages in conversations).
  – **Variable iteration**: actions iterate over variable bindings of event and conditional variables including possible backtracking to binding variants
Classification of the Action Space* (2)

**Action Type**

- **Internal knowledge self-update actions** with internal effects on extensional KB or working memory (update facts / data) and intensional KB / rule base (update rules)

- **State actions** with effects on changeable properties / states / fluents, often actions directly relate to events and are treated similar (as e.g. in KR event and fluent calculi)

- **External actions** with side effects on external systems via calls (procedural attachments), outbound messages, triggering/effecting

- **Messaging actions**: outbound action messages, usually in a conversational context

- **Process actions**: process calculi and workflow pattern style actions such as join, merge, split, etc.

- **Complex actions** (e.g. delayed reactions, actions with duration, sequences of bulk updates, concurrent actions, sequences of actions) modelled by e.g. action algebras (~event algebras) or process calculi
Classification of the Event / Action / State Processing* (1)

• 1. Event/Action Definition Phase
  – Definition of event/action pattern by event algebra
  – Based on declarative formalization or procedural implementation
  – Defined over an atomic instant or an interval of time or situation context.

• 2. Event/Action Selection Phase
  – Defines selection function to select, e.g. “first”, “last”, event from several occurred events (stored in an event instance sequence e.g. in memory, database/KB, queue) of a particular type
  – Crucial for the outcome of a reaction rule, since the events may contain different (context) information, e.g. different message payloads or sensing information
  – KR view: Derivation over event/action history of happened or future planned events/actions
Classification of the Event / Action / State Processing* (2)

• **3. Event/Action Consumption / Execution Phase**
  - Defines which events are consumed after the detection of a complex event
  - An event may contribute to the detection of several complex events, if it is not consumed
  - Distinction in event messaging between “multiple receive” and “single receive”
  - Events which can no longer contribute, e.g. are outdated, should be removed
  - KR view: events/actions are not consumed but persist in the fact base

• **4. State / Transition Processing**
  - Actions might have an internal effect i.e. change the knowledge state leading to state transition from (pre)-condition state to post-condition state.
  - The effect might be hypothetical (e.g. a hypothetical state via a computation) or persistent (update of the knowledge base),
  - Actions might have an external side effect
  - Inactive (complex) actions should be removed
Semantic CEP: The Combination

(Complex) Event Processing: events, complex events, event patterns, ...

+ Semantic technologies: rules & ontologies
Motivation for Semantic CEP

• Today's environment:
  – High complex processes, events, ...
  – Existing domain background knowledge
  – Missing systems for detection of events based on available background knowledge.

• Many event processing use cases require:
  – High expressiveness
  – High flexibility
  – Simplicity
Semantic CEP

• The use of **Background Knowledge** in Complex Event Processing (**CEP**) offers **declarative and expressive description of complex events/situations and reactions.**

• It enhances CEP systems to more **agile** systems and improves **flexibility** to dynamic changes.
Benefits of Semantics in CEP

• What are **benefits** of using Semantic Technology in Complex Event Processing?

  • **Expressiveness:** It can precisely express complex events, patterns and reactions which can be directly translated into operations.

  • **Flexibility:** Changes can be integrated into CEP systems in a fraction of time.

  • Complex Event Patterns are **declaratively represented** and are defined based on abstract strategies.
Semantic Technologies for Declarative Knowledge Representation

1. Rules
- Describe derived conclusions and reactions from given information (inference)

\[ \text{takeoff}(X) \implies \text{fly}(X) \]
\[ \text{takeoff}(\text{flight123}) \]

2. Ontologies
- Ontologies described the conceptual knowledge of a domain (concept semantics):

\[ \text{disjoint with} \]
\[ \text{Homogenous Situation} \rightarrow \text{Heterogenous Situation} \]
\[ \text{is a} \]
\[ \text{State Situation} \]
Ontologies used for SCEP

• Top-Level Ontologies required for SCEP (core selection)
  – Spatial
  – Temporal
  – Event
  – Situation
  – Process (can be further specialized by domain ontologies such as OWL-S, WSMO, PSL)
  – Actor/Agent (can be further specialized, by special ontologies such as FIPA, ACL, ...)
  – Action

• Domain Ontologies for application verticals (samples domains of CEP applications)
  – Healthcare - e.g., Hospital Activity Monitoring
  – Finance - e.g., Fraud Detection
  – Logistics and Cargo
  – Supply Chain Management
  – Insurance
  – Mortgage
Examples of Ontologies which include Events

- CIDOC CRM: museums and libraries
- ABC Ontology: digital libraries
- Event Ontology: digital music
- DOLCE+DnS Ultralite: event aspects in social reality
- Event-Model-F: event-based systems
- VUevent Model: An extension of DOLCE and other event conceptualizations
- IPTC. EventML: structured event information
- GEM: geospatial events
- Event MultiMedia: multimedia
- LODE: events as Linked Data
- CultureSampo: Publication System of Cultural Heritage
- OpenCyC Ontology: human consensus reality, upper ontology with lots of terms and assertions
- Super BPEL: ontology for the business process execution language
- Semantic Sensor Net Ontology: ontology for sensor networks
- ...
Example: Semantic CEP - Filter Pattern

Filter Pattern:
Stocks of companies, which have production facilities in Europe and produce products out of metal and have more than 10,000 employees.

Event Stream – stock quotes
\{(Name, “OPEL”) (Price, 45) (Volume, 2000) (Time, 1) \}
\{(Name, “SAP”) (Price, 65) (Volume, 1000) (Time, 2)\}

Semantic Knowledge Base
\{(OPEL, is_a, car_manufacturer),
(car_manufacturer, build, Cars),
(Cars, are_build_from, Metall),
(OPEL, hat_production_facilities_in, Germany),
(Germany, is_in, Europe),
(OPEL, is_a, Major_corporation),
(Major_corporation, have, over_10,000_employees)\}
Knowledge-based Event Processing

Event Stream

Event Processing

Knowledge Base

A-Box Update Stream

T-Box A-Box

Complex Events

Event Query
Example of Semantic Event Queries

StockOf Company_X3
StockOf Company_X1
Company_Y1 has Knowledge

Time

E1: Stock_ID1
Price Volume

E2: Stock_ID2
Price Volume

E3: Stock_ID3
Price Volume

Events

StockOf

Knowledge

Company_X3 has finance Company_Y2

has

M1

Company_Y1

Company_X1

has

produce

uses
Semantic Technologies for Declarative Knowledge Representation

1. Rules
   - Describe derived conclusions and reactions from given information (inference)

2. Ontologies
   - Ontologies described the conceptual knowledge of a domain (concept semantics):

\[
\text{takeoff}(X) \Rightarrow \text{fly}(X) \\
\text{takeoff(flight123)}
\]
Usage of Rules

1. **Rules** that influence the operational processing & decision processes of an event processing agent:
   - *Derivation rules* (deduction rules): establish / derive new information that is used, e.g. in a decision process (e.g. routing).
   - **Reaction rules** that establish when certain activities should take place, e.g.:
     - *Condition-Action rules* (production rules)
     - *Event-Condition-Action (ECA) rules* + variants (e.g. ECAP).
     - *Messaging Reaction Rules* (event message reaction rules)

2. **Constraints** on event processing agent’s structure, behavior and information, e.g.:
   - *Structural constraints* (e.g. deontic assignments).
   - *Integrity constraints and state constraints*
   - *Process and flow constraints*
1. **(Temporal) KR event/action logics**
   - Members e.g. Event Calculus, Situation Calculus, Fluent Calculus, TAL
   - Actions with effects on changeable properties / states / fluents, i.e. actions ~ events
   - Focus: reasoning on effects of events/actions on knowledge states and properties

2. **KR evolutionary transaction, update, state transition logics**
   - Members e.g. transaction logics, dynamic LPs, LP update logics, transition logics,
   - Knowledge self-updates of extensional KB (facts / data) and intensional KB (rules)
   - Transactional updates possibly safeguarded by post-conditional integrity constraints / tests
   - Complex actions (sequences of actions)
   - Focus: declarative semantics for internal transactional knowledge self-update sequences (dynamic programs)

3. **Condition-Action / Production rules**
   - Members, e.g. OPS5, Clips, Jess, JBoss Rules/Drools, Fair Isaac Blaze Advisor, ILog Rules, CA Aion, Haley, ESI Logist, Reaction RuleML
   - Mostly forward-directed non-deterministic operational semantics for Condition-Action rules
   - Focus: primitive update actions (assert, retract); update actions (interpreted as implicit events) lead to changing conditions which trigger further actions, leading to sequences of triggering production rules
4. Active Database ECA rules

- Members, e.g. ACCOOD, Chimera, ADL, COMPOSE, NAOS, HiPac, Reaction RuleML, Prova, XChange
- ECA paradigm: “on Event when Condition do Action”; mostly operational semantics
- Instantaneous, transient events and actions detected according to their detection time
- Focus: Complex events: event algebra (e.g. Snoop, SAMOS, COMPOSE) and active rules (sequences of self-triggering ECA rules)

5. Process Calculi, Event Messaging and distributed rule-based Complex Event Processing

- Members, e.g. process calculi (CSP, CSS, pi-calculus, join-calculus), event/action messaging reaction rules (inbound / outbound messages), rule-based intelligent CEP with rule-based Event Processing Languages (EPLs, e.g. Prova, Reaction RuleML, AMIT, Rule Core)
- Focus: process calculi focus on the actions, event messaging and CEP on the detection of complex events; often follow some workflow pattern, protocol (negotiation and coordination protocols) or CEP pattern
Temporal KR Event / Action Logics

• **Members e.g.**
  - event calculus and variants,
  - the situation calculus,
  - feature and fluent calculi,
  - various (temporal) action languages (TAL),
  - versatile event logics.

• **The formalisms differ in:**
  - How to describe the qualification of events / actions (Qualification)
  - How to describe the effects of actions / events (Transitions)
  - How to describe indirect effects and interdependencies between fluents (Ramification)
  - How to describe which fluents remain unchanged over a transition (Frame Problem)
Summary Semantic CEP: Selected Benefits

- Event data becomes **declarative knowledge** while conforming to an underlying **formal semantics**
  - e.g., supports automated semantic enrichment and mediation between different heterogeneous domains and abstraction levels

- Reasoning over **situations and states** by event processing agents
  - e.g., *a process is executing when it has been started and not ended*
  - e.g. *a plane begins flying when it takes off and it is no longer flying after it lands*

- Better understanding of the **relationships between events**
  - e.g., temporal, spatial, causal, .., relations between events, states, activities, processes
  - e.g., *a service is unavailable when the service response time is longer than X seconds and the service is not in maintenance state*
  - e.g. *a landing starts when a plane approaches. During landing mobile phones must be switched off*

- **Declarative rule-based processing** of events and reactions to situations
  - Semantically grounded reaction rules
Agenda

• Introduction to Semantic Complex Event Processing

• Event Processing Reference Model and Reference Architecture

• Reaction RuleML Standard

• Examples for Event Processing Functions

• Summary
Reference Architecture and Reference Model

• Reference Architecture
A reference architecture models the abstract architectural elements in the domain independent of the technologies, protocols, and products that are used to implement the domain.

• Reference Model
A reference model describes the important concepts and relationships in the domain focusing on what distinguishes the elements of the domain.
Motivation and Benefits

• Motivation
  – Event Processing is evolving and exploiting many technologies
  – Potential adopters (stakeholders) need a reference to understand suppliers’ architectures and solutions.

• Benefits
  – a Reference Architecture predefines customizable abstract frames of reference for specific stakeholder concerns and application domains.
    • Aids reuse of successful EP architectures for frequently occurring EP design problems
  – Underlying Architecture is a Reference Model that defines the terminology and components in Event Processing architectures
Standards Classification (1)

• Common Standards, Laws

  Common standards and laws provide conditions and regulations which have to be considered in the business, e.g. laws for stock trading. Development of a software application has to follow these laws.

• Domain Reference Model

  Domain Reference Models provide a subset of best practices, reference business processes, etc. for a specific business domain. These models help adaptors in defining a proper application based on well proven best practices, e.g. a well proven reference business process for the manufacturing domain.

• Domain Use Case

  The use case describes how a problem can be solved, e.g. how to detect fraud in stock trading. It is often derived out of the business strategy and the reference model.
Standards Classification (2)

• Strategy

  The strategy defines the individual business strategy of a company which has to be considered in developing new applications. The strategy consists of business (business motivations, goals, policies, rules) as well of technical (applications infrastructure, allowed frameworks) conditions.

• Functional Model

  The functional model provides domain independent best practices and guidelines helping to design and develop a proper application, e.g. "Design Patterns - Elements of Reusable Object-Oriented Software"
Standards Classification - CIM

• Computer Independent Model

• The Computer Independent Model (CIM) describes the business functionality or system without reference to IT specifics. CIM should provide a common understanding between business users and software developers. In the CEP model it consists of following components:

  • Standard Vocabulary, Ontology The ontology provides a set of definitions and terms enabling a common understanding for all stakeholders.

  • Process description: Description of activities and its flow to produce a specific product or service enhanced with events occurring in it or influencing it.

  • Reaction Definition of the activities which have to be initiated based on the previous taken decision.

  • Decision Definition of rules, what has to be done if a relevant situation was detected by a pattern matching

  • Complex Events Definition of event correlations for detection of relevant situations defined by patterns representing knowledge from source events, e.g. for detection of fraud.

  • Simple Events Definition of attributes and types consisting of a simple event, e.g. the event "Stock Price" consists of the fields WKN, amount, etc.
Standards Classification - PIM

• Platform Independent Model Standards (PIM)

• The Platform Independent Model layer represents behavior, data, design, and messaging, independent from a particular EP platform.

• Event-Driven Behavior Effects of events lead to some (state) changes in the properties of the world which can be abstracted into situations.

• Event Data Platform-independent representation of events and their data is crucial for the interoperation between EP tools and between domain boundaries.

• Event Processing Design Platform-independent (reference) architecture and design models addressing different views for different stakeholders.

• Messaging PIM Messaging is addressing transport protocols and routing, coordination / negotiation mechanisms.
## COMMON FRAMEWORK AND GUIDELINES

General Frameworks and Guidelines for Developing Systems

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common Standards, Laws</strong></td>
<td>Industry standards, laws, etc.</td>
<td>Several laws (often country specific), e.g. BörsG for stock trading in Germany</td>
<td>A lot of existing standards are available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CEP Specific standards not necessary</td>
</tr>
<tr>
<td><strong>Strategy UseCase</strong></td>
<td>Overlying business motivations, goals, strategies, policies and business rules</td>
<td>Reduction of fraud by 10 percent, ...</td>
<td>OMG BMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>May need more emphasis on events and decision rules</td>
</tr>
<tr>
<td><strong>Reference Model</strong></td>
<td>Collection of best practices and reference business processes for a specific domain</td>
<td>Reference End-to-End process for logistics, e.g. <em>Pickup to Delivery</em></td>
<td>Various per domain for data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rarely handles events and rules</td>
</tr>
<tr>
<td><strong>Use Case</strong></td>
<td>Description of how to solve a problem</td>
<td>Fraud detection for stock trading</td>
<td>UML (not CEP or rules specific)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPTS UseCase Template</td>
</tr>
<tr>
<td><strong>Functional Model, Design Patterns ..</strong></td>
<td>Domain independent description of best practices, guidelines, etc.</td>
<td>Design Patterns for software engineering (Gamma)</td>
<td>Lots of different ones!</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPTS Functional Ref Architecture</td>
</tr>
</tbody>
</table>
### COMPUTER INDEPENDENT MODEL 1/2

**Computer Independent Model / Business Model for designing CEP Systems and Processes**

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Vocabulary, Ontology</strong></td>
<td>- Common definitions of words enabling a common understanding for all stakeholders</td>
<td>- Stock Order: Transaction for buying a specific amount of stocks for price x</td>
<td>- Glossary, ontologies for events, time …</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- OMG SBVR, W3C OWL</td>
</tr>
<tr>
<td><strong>Process Description</strong></td>
<td>- Description of activities and its flow to produce a specific product or service enhanced with events occurring in it</td>
<td>- Buy Stocks requires “estimate risk”, “check price”, “check deposit”, “check availability”, etc.</td>
<td>- Text descriptions of “events”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- BPMN, EPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Insufficient details on events and rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Create a common understanding between business user and software developer on a “big picture”</td>
</tr>
<tr>
<td><strong>Decision</strong></td>
<td>- Definition of rules, what has to be done if a relevant situation was detected by a pattern matching</td>
<td>- If a highly risk for the order was detected by a pattern the reaction “dropStockOrder” has to be done</td>
<td>- Decision Tree</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Decision Table (~DMN)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Decision Rules standard missing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Common understanding for all stakeholders involved</td>
</tr>
<tr>
<td><strong>Reaction</strong></td>
<td>- Definition of the activities which have to be done, based on the previous taken decision</td>
<td>- The stock order has to be dropped and the trader has to be informed</td>
<td>- None (Text based description)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CIM Standard for Reaction Rules is missing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Common understanding of possible outcomes all stakeholders involved</td>
</tr>
</tbody>
</table>
## COMPUTER INDEPENDENT MODEL 2/2

Computer Independent Model / Business Model for designing CEP Systems and Processes

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardization</th>
</tr>
</thead>
</table>
| **Simple Events** | Definition of attributes and types consisting an simple Event | StockPrice:  
- WKN (String 10)  
- ISIN (String 15)  
- Name (String 50)  
- Date/Time: (Time)  
- Price: (Decimal)  
- Currency: (String 3)  
... | UML (Gaps in specifics needed for EP, -EMP)  
- Improve modelling languages eg NEAR | Create a common understanding across business users / event sources, and developers |
| **Complex Events** | Definition of event correlations for detection of relevant situations defined by patterns representing knowledge from source events, e.g. for detection of fraud. | Stocks of companies, where the stock price increase 3% in the last three days and which have production facilities in Europe and produce products out of metal and have more than 10,000 employees | None but - OMG EMP proposed  
- New standard required | Better understanding of relation between involved events and roles |
### Description

- Event-Driven Behavior and Decisions:
  - Event occurrences lead to state changes in patterns, processes (= situations)
  - Decisions represent the choices a system can after patterns detected or process states change
  - Actions are triggered / performed as event reactions as the output of decisions and state/situation changes

### Example

- Event E is used in Pattern P = “E then F”
- Event E causes process R to start
- After Pattern P is detected we must decide how to respond to the situation S
- Process R must be invoked

### Existing Standards (+Gaps)

- Rules: W3C RIF + Reaction RuleML;
  - focused on rule-based behaviour
- OMG PRR
  - lacks explicit events
  - specific to production rules
- OMG Decision Modeling Notation
  - no explicit event
- OMG UML2 Behavioral View Diagrams
  - Limited expressiveness of events and events = actions/activities
- OMG BPEL
  - specific to business process execution with WS
- W3C WS Choreography
  - Specific to WS
- and further EDA standards

### Benefits of Standardization

- Declarative, explicit representation
- Publication and interchange of decisions and reactive behavior
- ...
### Data
- Information carried by events and used in event patterns and processes
- May be processed as a part of event processing in the context of CEP/EP
- Archived and analysed for machine learning, BI, etc

### Description
- Event Pattern P uses properties of event E1, E2
- Processing of event E1 requires operations on its properties
- Reaction to event E1 requires sending data D to system S

### Example
- Basic concepts related to data and event processing

### Existing Standards (+Gaps)
- **Software Engineering:**
  - UML Structural View diagrams
  - Not specific to events
- **Knowledge Representation:**
  - W3C RDFS/OWL
  - ISO CL, ISO Topic Maps, OMG ODM UML/OWL Profiles
  - Many existing event ontologies
- **Rules:**
  - W3C RIF + Reaction RuleML
  - Specific to rule-based EP
- **OASIS WS Topics**
  - Topic-based pubsub only
- **OMG Event Meta Model**
  - Not yet an RFP in OMG
- **OASIS Common Base Event**
  - For business enterprise apps
- **OASIS WS Notification and W3C WS Eventing**
  - Specific to WS
- **Rules:**
  - Further standardization in W3CRIF/RuleML

### Benefits of Standardization
- Declarative representation, translation and interchange of events
- Interchange and interoperation between different EP tools preserving the semantic interpretation of the event data
- Interchange events over domain boundaries which have different domain semantics / models
- Prerequisites Semantics:
  - Explicit domain semantics
  - Expressiveness
  - Extensibility
  - Reusability, specialisation and mapping

### Some of ontologies which include events
- **CIDOC CRM:** museums and libraries
- **ABC Ontology:** digital libraries
- **Event Ontology:** digital music
- **DOLCE+DnS Ultralite:** event aspects in social reality
- **Event-Model-F:** event-based systems
- **VUevent Model:** An extension of DOLCE and other event conceptualizations
- **IPTC. EventML:** structured event information
- **GEM:** geospatial events
- **Event MultiMedia:** multimedia
- **LODE:** events as Linked Data
- **CultureSampo:** Publication System of Cultural Heritage
- **OpenCyC Ontology:** human consensus reality, upper ontology with lots of terms and assertions
- **Semantic Complex Event Processing:** top level meta model / ontology for semantic CEP
Platform Independent Model / IT Model for design of CEP Systems

**Description**
- Platform-Independent (Reference) Architecture and Model Descriptions addressing different views for different stakeholders

**Example**
- Reference Architecture: models the abstract architectural design elements
- Reference Model: describes the important concepts and relationships in the design space
- EPN: Distributed Network Architecture
- Agents: abstract components given a specific abstract role and behavior/responsibilities

**Existing Standards (+Gaps)**
- ISO/IEC 42010:2007 Recommended Practice for Architectural Description of Software-intensive Systems
  - Needs to be tailored to EP architecture descriptions
- UML 2 Implementation View Diagrams
  - Limited expressiveness
  - Not specialised for EP design; only general component and composite structure design
- Agents: OMG Agent Metamodel; FIPA Agent Model; ...
  - Agent specific
- Workflow Management Coalition Reference Model
  - Workflow model

**Benefits of Standardization**
- Abstraction from the PSM design
- Increases understandability and reusability
- Agent model is an abstraction from technical IT components to role-based agents on a more abstract level
### Messaging

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport protocols and routing coordination / negotiation mechanisms</td>
<td>synchronous vs. asynchronous transport</td>
<td>Many transport protocols: JMS, JDBC, TCP, UDP, multicast, http, servlet, SMTP, POP3, file, XMPP</td>
<td>well understood and standardized transport protocols</td>
</tr>
<tr>
<td></td>
<td>Push vs. pull</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Coordination: Publish Subscribe, ContractNet, …</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Platform Specific Model (PSM) describes the concrete Implementation of a CEP Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPL CQL</td>
<td>SELECT x, y from REUTERS WHERE x.id == y.id AND x.price &gt; y.price * 1.1 IN WINDOW(600,&quot;secs&quot;)</td>
<td>ANSI SQL extended for continuous policy, ODBMS OQL extended for continuous policy</td>
<td>Interchange queries between vendor platforms (design, deployment) [IT department]</td>
</tr>
<tr>
<td></td>
<td>Vendor specific versions include Oracle CQL, Streambase CQL, Sybase CQL, TIBCO BQL</td>
<td></td>
<td>Encourage training / education on CQL (cf SQL takeup) [university, student]</td>
</tr>
<tr>
<td></td>
<td>INTERCHANGE QUERIES AMONG VENDOR PLATFORMS (DESIGN, DEPLOYMENT) [IT DEPARTMENT]</td>
<td></td>
<td>Dependancies: common data descriptors</td>
</tr>
</tbody>
</table>

| EPL ECA Rules | IF event. SigMvDet AND SigMvDet.stock = IBM AND SigMvDet.change = up THEN reqBuy(SigMvDet.stock) | W3C RIF PRD extended for ECA (RIF ECA), Reaction RuleML, JSR-94 rule execution API | Interchange rules between vendor platforms (design, deployment) [IT department] |
|              | Vendor examples include TIBCO BE, IBM WSBE, Progress Apama, Open Source, e.g. Prova, Drools |                          | Encourage training / education on ECA rules (cf Java takeup) [university, student] |
|              |                                                        |                          | Dependancies: common data descriptors |
# Platform Specific Model (PSM)

Platform Specific Model (PSM) describes the concrete implementation of a CEP Application.

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardisation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EPL</strong></td>
<td>Temporal Regular Expressions defining complex events in terms of event / non-event sequences over time</td>
<td>WHEN event. A, event.B WHERE event.A (*2) FOLLOWED BY no event.B IN time(200,ms)</td>
<td>Perl Regex with temporal extensions</td>
</tr>
<tr>
<td><strong>TRE</strong></td>
<td></td>
<td>Vendor examples include Oracle CQL (TRE embedded in SQL), Progress Apama (TRE), TIBCO PML</td>
<td></td>
</tr>
<tr>
<td><strong>Application Architecture</strong></td>
<td>Application component layouts / configurations, specifying various processes / engines / agents and how they are connected. Architecture may also be considered platform independent in terms of general requirements, but platform specific in terms of associating specific logical layouts, interfaces, and agent / engine numbers to achieve specified SLAs</td>
<td>Agent 1 type inference contains rulesets R1, R2 with destinations D1, D2 Deployment C1 is of 4 x Agent 1 role active-active</td>
<td>OMG AMP Agent Metamodel and Profile (draft)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vendor examples include TIBCO CDD Cluster Deployment Descriptor</td>
<td></td>
</tr>
</tbody>
</table>
# Platform Specific Model (PSM)

Platform Specific Model (PSM) describes the concrete Implementation of a CEP Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
<th>Existing Standards (+Gaps)</th>
<th>Benefits of Standardisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messaging System</td>
<td>Middleware type and configuration</td>
<td>JMS includes header information and payload information, the latter which includes XML document per some XSD. JMS extensions could be for guaranteed delivery, timeout etc.</td>
<td>JMS (Java spec)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OMG DDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AMQP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Gap: some abstract definition of a messaging software per the above</td>
</tr>
</tbody>
</table>
Reference Architecture and Reference Model

• Reference Architecture

A reference architecture models the abstract architectural elements in the domain independent of the technologies, protocols, and products that are used to implement the domain.

• Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Advanced design patterns in event processing. ACM DEBS 2012: 324-334; http://www.slideshare.net/isvana/eppts-debs2012-event-processing-reference-architecture-design-patterns-v204b


EPTS Reference Architecture
Architectural Descriptions

Event Processing, Complex Event Processing, Event Stream Processing
Event Abstraction, Event Pattern Detection, Event Composition etc. etc.

- Event Type Definitions, Event Processing Rules, etc.
- Event Management
- Design time
- Run time
- Administration

Event Producer
(Event Source, Event Emitter)

Derived Events

Event Consumer
(Event sink, event handler, event listener)
Recommended Practice for Architectural Description of Software-intensive Systems

Now an ISO/IEC 42010:2007 standard

Includes 6 elements

1. Architectural description
2. System stakeholders and their concerns
3. One or more architectural views
4. Viewpoints
5. A record of all known inconsistencies among the architectural description’s required constituents
6. A rationale for selection of the architecture
## Reference Architecture Viewpoints

<table>
<thead>
<tr>
<th>Viewpoint</th>
<th>Element</th>
<th>Viewpoint</th>
<th>Viewpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concepts</td>
<td>How to implement?</td>
<td>How to apply?</td>
<td>How to utilize / sell / own?</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>Architects / Engineers</td>
<td>Project Manager</td>
<td>Decision Maker, Customer, Provider</td>
</tr>
<tr>
<td>Concerns</td>
<td>Effective construction and deployment</td>
<td>Operational Management</td>
<td>Strategic and tactical management</td>
</tr>
<tr>
<td>Techniques / Languages</td>
<td>Modeling, Engineering</td>
<td>IT (service/appl) management, project management</td>
<td>Monitoring, Enterprise Decision Management, Governance</td>
</tr>
</tbody>
</table>
## Example Stakeholders and Viewpoints

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Component</th>
<th>Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Maker / End User</td>
<td>[Not applicable]</td>
<td>Inputs, Outputs and Processing Requirements View</td>
</tr>
<tr>
<td>Architect / Designer</td>
<td>Solution Components View</td>
<td>Functions carried out in CEP View</td>
</tr>
</tbody>
</table>
Reference Architecture: Functional View

**Event Production**
- Publication, Retrieval

**Event Consumption**
- Dashboard, Apps, External Reaction

**Event Preparation**
- Identification, Selection, Filtering, Monitoring, Enrichment

**Complex Event Detection**
- Consolidation, Composition, Aggregation

**Event Analysis**
- Analytics, Transforms, Tracking, Scoring, Rating, Classification

**Event Reaction**
- Assessment, Routing, Prediction, Discovery, Learning

**Event Process Monitoring, Control**
- Event Actions, Event Correlations and patterns, Event Computations, Event Selections

**Event Process Monitoring, Control**
- Process Updates, Resource Utilization, High Availability, Security, Start/Stop

**State Management**
- Event Definitions, Event Process Models, Event Driven Reaction Rules

**Design time**
- Event and Complex Event Definition, Modeling, (continuous) Improvement

**Run time**
- Event and Complex Event (Pattern, Control, Rule, Query, RegEx, etc)

**Administration**
- see: Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Advanced design patterns in event processing. ACM DEBS 2012: 324-334;
Reference Architecture: Functional View / Runtime

- **Event Reaction**: Assessment, Routing, Prediction, Discovery, Learning
- **Complex Event Detection**: Consolidation, Composition, Aggregation
- **Event Analysis**: Analytics, Transforms, Tracking, Scoring, Rating, Classification
- **Event Preparation**: Identification, Selection, Filtering, Monitoring, Enrichment
- **Event Production**: Publication, Retrieval
- **Event Consumption**: Dashboard, Apps, External Reaction

State Management

Event Actions
Event Correlations and Patterns
Event Computations
Event Selections
Event Production/Consumption
Run time
**Event Production**: the source of events for event processing.

- **Event Publication**: As a part of event production, events may be published onto a communication mechanism (e.g., event bus) for use by event consumers (including participants in event processing). This is analogous to a "push" system for obtaining events.

- **Event Retrieval**: As a part of event production, events may be explicitly retrieved from some detection system. This is analogous to a "pull" system for obtaining events.
Event Consumption: the process of using events from event publication and processing. Event processing itself can be an event consumer, although for the purposes of the reference architecture, event consumers are meant to indicate downstream consumers of events generated in event processing.

- Dashboard: a type of event consumer that displays events as they occur to some user community.
- Applications: a type of event consumer if it consumes events for its own processes.
- External Reaction: caused through some event consumption, as the result of some hardware or software process.
Event Preparation: the process of preparing the event and associated payload and metadata for further stages of event processing.

- **Entity Identification**: incoming events will need to be identified relative to prior events, such as associating events with particular sources or sensors.

- **Event Selection**: particular events may be selected for further analysis. Different parts of event processing may require different selections of events. See also event filtering.

- **Event Filtering**: a stream or list of events may be filtered on some payload or metadata information such that some subset is selected for further processing.

- **Event Monitoring**: particular types of events may be monitored for selection for further processing. This may utilise specific mechanisms external to the event processing such as exploiting event production features.

- **Event Enrichment**: events may be "enriched" through knowledge gained through previous events or data.
Reference Architecture: Functional View / Runtime

**Event Analysis**: the process of analysing suitably prepared events and their payloads and metadata for useful information.

- **Event Analytics**: the use of statistical methods to derive additional information about an event or set of events.
- **Event Transforms**: processes carried out on event payloads or data, either related to event preparation, analysis or processing.
- **Event Tracking**: where events related to some entity are used to identify state changes in that entity.
- **Event Scoring**: the process by which events are ranked using a score, usually as a part of a statistical analysis of a set of events. See also *Event Analytics*
- **Event Rating**: where events are compared to others to associate some importance or other, possibly relative, measurement to the event.
- **Event Classification**: where events are associated with some classification scheme for use in downstream processing.
Complex Event Detection: the process by which event analysis results in the creation of new event information, or the update of existing complex events.

- Event Consolidation: combining disparate events together into a "main" or "primary" event. See also event aggregation.
- Event Composition: composing new, complex events from existing, possibly source, events.
- Event Aggregation: combining events to provide new or useful information, such as trend information and event statistics. Similar to event consolidation.
Event Reaction: the process subsequent to event analysis and complex event detection to handle the results of analysis and detection.

- Event Assessment: the process by which an event is assessed for inclusion in some process, incorporation in some other event, etc.

- Event Routing: the process by which an event is redirected to some process, computation element, or other event sink.

- Event Prediction: where the reaction to some event processing is that some new event is predicted to occur.

- Event Discovery: where the reaction to some event processing is the disclosure of a new, typically complex, event type.
  
  - Note that event prediction is predicting some future event, usually of a known type, whereas event discovery is the uncovering of a new event type. See also event-based learning.

- Event-based Learning: the reaction to some event processing that uses new event information to add to some, typically statistical-based, understanding of events.
  
  - Note that event-based learning is a specialisation of general machine learning and predictive analytics.
Reference Architecture: Functional View / Design time

Covers the definition, modeling, improvement / maintenance of the artefacts used in event processing:

- event definitions, including event metadata and payloads,
- event and event object organisations and structures,
- event processing transformations / queries / rules / procedures / flows / states / decisions / expressions (although these can sometimes be considered as administrative updates in some situations)
Administrative concepts of monitoring and control. This may involve

- starting and stopping the application and event processing elements, including application monitors
- providing and updating security levels to event inputs and outputs (also can design-time)
- management of high availability and reliability resources, such as hot standby processes
- resource utilisation monitoring of the event processing components
- process updates, such as how-swapping of event processing definitions to newer versions.
Categorization of Event Processing Patterns

Source: Paschke, A.: Design Patterns for Complex Event Processing, DEBS'08, Rome, Italy, 2008
http://arxiv.org/abs/0806.1100v1
Categorization of Patterns

• Categorization according to Good and Bad Solutions
  – *CEP Patterns*
  – *CEP Anti-Patterns*

• Categorization according to the Abstraction Level
  – *Guidelines and Best Practices*
  – *Management patterns*
  – *Architecture patterns*
  – *Design patterns*
  – *Mapping patterns*
  – *Idioms / Realization patterns*
  – *Smells / Refactoring patterns*

• Categorization according to the Intended Goal
  – *Adoption patterns*
  – *Business patterns*
  – *Integration patterns*
  – *Composite patterns:*
  – …

• Categorization according to the Management Level
  – *Strategic patterns*
  – *Tactical patterns*
  – *Operational patterns*

Agenda

• Introduction to Event Processing
• Event Processing Reference Model and Reference Architecture
• Reaction RuleML Standard
• Examples for Event Processing Functions
• Summary
RuleML Enables ...
RuleML Rule interchange (on the PIM level)

Application A

Rule system 1

Rules

Data

serialize

Application B

Rule system 2

Rules

Data

de-serial

RuleML doc

Ontologies (ODM, OWL, RDF-S, CL)

XML doc (or others)

data
The Reaction RuleML Family

Reaction Rules: Sub-branches

1. **PR** Production Rules (Condition-Action rules)

2. **ECA** Event-Condition-Action (ECA) rules

3. **CEP** Rule-based Complex Event Processing (complex event processing reaction rules, (distributed) event messaging reaction rules, query reaction rules, etc.)

4. **DR and KR** Knowledge Representation
   **AI Reasoning** with Temporal/Event/Action/Situation/Transition/Process Logics and Calculi
Quick Overview: Reaction RuleML Dialects

• Derivation Reaction RuleML (if-then)
  - Time, Spatial, Interval

• KR Reaction RuleML (if-then or on-if-do)
  - Situation, Happens\texttt{(@type)}, Initiates, Terminates, Holds, fluent

• Production Reaction RuleML (if-do)
  - Assert, Retract, Update, Action

• ECA Reaction RuleML (on-if-do)
  - Event, Action, + (event / action algebra operators)

• CEP Reaction RuleML (arbitrary combination of on, if, do)
  - Receive, Send, Message

* + variants and alternatives
Reaction Rules: Specializable Syntax

```
<Rule @key @keyref @style @type ...+>
  <meta> <!-- descriptive metadata of the rule --></meta>
  <scope> <!-- scope of the rule e.g. a rule module --></scope>
  <guard> <!-- guard constraint --></guard>
  <evaluation> <!-- intended semantics --></evaluation>
  <signature> <!-- signature --></signature>
  <qualification> <!-- e.g. qualifying rule metadata, e.g. priorities, validity, strategy --></qualification>
  <quantification> <!-- quantifying rule declarations, e.g. variable bindings --></quantification>
  <oid> <!-- object identifier --></oid>
  <on> <!-- event part --></on>
  <if> <!-- condition part --></if>
  <then> <!-- (logical) conclusion part --></then>
  <do> <!-- action part --></do>
  <after> <!-- postcondition part after action, e.g. to check effects of execution --></after>
  <else> <!-- (logical) else conclusion --></else>
  <elsedo> <!-- alternative/else action, e.g. for default handling --></elsedo>
</Rule>
```
Reaction RuleML – Example Rule Types+

- **Derivation Rule:**
  
  \[
  \text{<Rule style="reasoning">}
  \text{<if>...</if>}
  \text{<then>...</then>}
  \text{</Rule>}
  \]

- **Production Rule:**
  
  \[
  \text{<Rule style="active">}
  \text{<if>...</if>}
  \text{<do>...</do>}
  \text{</Rule>}
  \]

- **Trigger Rule:**
  
  \[
  \text{<Rule style="active"> >}
  \text{<on>...</on>}
  \text{<do>...</do>}
  \text{</Rule>}
  \]

- **ECA Rule:**
  
  \[
  \text{<Rule style="active">}
  \text{<on>...</on>}
  \text{<if>...</if>}
  \text{<do>...</do>}
  \text{</Rule>}
  \]

+ variants, e.g. EA, ECAP, CAP … and combinations, e.g. if-then-do, if-then-elsedo, …
Derivation Reaction RuleML
Expressiveness Layering

Extended Disjunctive LP

Normal Disjunctive LP

Stratified Disjunctive LP

Definite Disjunctive LP

Extended LP

Normal LP

Stratified LP

Definite LP

see http://ruleml.org/reaction/0.2/
Production Reaction RuleML Layering

- Normal Disjunctive PR with (non-stratified) iNeg
  - Stratified Disjunctive PR with iNeg
  - Definite Disjunctive PR
    - Stratified PR with iNeg
    - Definitive PR
  - Normal PR with (non-stratified) iNeg
    - Stratified PR with iNeg

See [http://ruleml.org/reaction/0.2/](http://ruleml.org/reaction/0.2/)
ECA and CEP Reaction RuleML
Expressiveness Layering

Unrestricted CEP Reaction Rules
(arbitrary combination of all rule parts)

Serial Derivation Rules with local nested Reaction Rules

Active Reaction Rules with Alternative Actions (If-do-elsedo style)

Messaging RRs (serial reaction rules for workflow-style execution in branching logics)

Active ECAP RRs (with post condition)

Active ECA RRs

Trigger Reaction Rules (EA RRs)

see http://ruleml.org/reaction/0.2/
Example - The Generic Syntax Pattern of An Event

```xml
<Event @key @keyref @iri @type>

<!-- event info and life cycle management, modularization -->
<meta>  <!-- E: (semantic) metadata of the event --> </meta>
$scope>  <!-- E: scope of the event --> </scope>
<guard>  <!-- E: scope of the event --> </guard>

<!-- event pattern description -->
<evaluation>  <!-- E: semantics, e.g. selection, consumption-->
</evaluation>

<signature>  <!-- E: event pattern declaration --> </signature>

<!-- event instance -->
<qualification>  <!-- E: e.g. qualifying event declarations, e.g. priorities, validity, strategy --> </qualification>
<quantification>  <!-- E: quantifying rule declarations --> </quantification>
<oid>  <!-- E: object identifier --> </oid>
<arg>  <!-- E: event instance argument or content --> </arg>

</Event>
```

references to external ontologies
Reaction RuleML Signature Definitions

- **Signature Definitions** determine which terms and formulas are well-formed
  - knowledge and meta knowledge signatures

- Dialects‘ are defined by signature definitions
  - e.g., signatures defining sorts of rules such as derivation rules (if-then), production rules (if-do), ECA rules (on-if-then), sorts of events, actions, times, sorts of predicates, frames, positional and slotted terms, etc.

- Profiles can introduce sub-signature definitions to a dialect

- User-defined signature definitions in the **knowledge interface** (<signature>) can further specialize dialects / profile signatures
  - e.g., restrict the scope, mode, arity, etc.
Reaction RuleML Signature Definitions

• Sorted Signature Definition
  
  \[ S = \langle T, C, SC, M, \text{arity}, \text{sort}, \text{scope}, \text{mode} \rangle \]

  – \( S \) and \( T_i \in T = \{T_1, \ldots, T_n\} \) are symbols denoting signature names called **sorts**, with \( T \) being the **signature pattern / declaration** of \( S \).

  – \( C \) set of constant symbols called the **universe** which might be further partitioned into

    1. non-overlapping* (scoped) **domain of discourse** called **scopes**, where \( SC_j \in SC = \{SC_1, \ldots, SC_m\} \) being symbols denoting the scope name

    2. pairwise disjoint sets of input-output symbols, called **modes**, where \( M_k \in M = \{M_1, \ldots, M_m\} \) being symbols denoting the mode name

  – The function

    • **sort** associates with each symbol \( c \in C \) its sort \( t \in T \)
    
    • **scope** associates with each symbol \( c \in C \) its scope \( sc \in SC \)
    
    • **mode** associates with each symbol \( c \in C \) its mode \( m \in SC \)

  * after flattening composite scopes
Multi-Sorted Signature Definitions

• Multi-Sorted Base Signature

\[ S_b = \langle T, P, F, C, SC, M, \text{arity}, \text{sort}, \text{scope}, \text{mode} \rangle \]

– is used as the basis to define Reaction RuleML dialects
– introduces predicates \((P)\), functions \((F)\) and terms \((C)\)
– the function \(\text{sort}\) associates functions, predicates and constants to their sort.
– the function \(\text{arity}\) gives the arity of predicates, functions and terms
Labelled Logic with Meta Knowledge Annotations

- **Combined Signature with Meta Knowledge**

  \[ S_{b\text{meta}} = \langle S_b \cup S_{\text{meta}} , @ \rangle \]

  - \( S_{\text{meta}} \) meta knowledge signature defining **labels** (meta knowledge formulas)
  - the function \( @ \) is a labelling function that assigns a set of meta knowledge formulas \( L_i \in S_{\text{meta}} \) to a knowledge formula \( \Phi \in S_b : \)

    \[ @(L_1), \ldots, @(L_n) \Phi \]

- Reaction RuleML distinguishes between
  - descriptive metadata \(<\text{meta}>\)
  - qualifying metadata \(<\text{qualification}>\)
Example - Descriptive and Qualifying Metadata

Descriptive Metadata

Qualifying Metadata

<Assert key="#module1">
  <meta> <!-- descriptive metadata -->
    <Atom><Rel iri="dc:creator"/><Ind>Adrian Paschke</Ind></arg><Atom>
  </meta>
  <meta> <!-- time metadata -->
    <Time type="&ruleml;TimeInstant"><Data xsi:type="xs:date">2011-01-01</Data></Time>
  </meta>
  <meta> <!-- source metadata -->
    <Atom><Rel>src</Rel><Ind>./module1.prova</Ind></Atom>
  </meta>
  <qualification> <!-- qualifying metadata -->
    <Atom> <!-- the module is valid for one year from 2011 to 2012 -->
      <Rel>valid</Rel>
      <Interval type="&ruleml;TimeInstant">
        <Time type="&ruleml;TimeInstant"><Data xsi:type="xs:date">2011-01-01</Data></Time>
        <Time type="&ruleml;TimeInstant"><Data xsi:type="xs:date">2012-01-01</Data></Time>
      </Interval>
    </Atom>
  </qualification>
</Assert>

<Rulebase key="#innermodule1.1">
  <meta> ... </meta> <qualification> ... </qualification>
  <Rule key="#rule1">
    <meta> ... </meta> <qualification> ... </qualification>
  </Rule>
  <Rule key="#rule2">
    <meta> ... </meta> <qualification> ... </qualification>
  </Rule>
</Rulebase>
Signature Pattern for Sort Definition

• A sort \( T \) is declared by a signature pattern \( T = \{ T_1, \ldots, T_n \} \) in the sort’s signature definition

• Example

\[
\begin{align*}
\text{Event} &= \langle \text{Term}={} \rangle \\
\text{Time} &= \langle \text{Term}={} \rangle \\
\text{PositionalBinaryPredicate} &= \langle \text{PositionalPredicate}={}\langle \text{Term},\text{Term} \rangle, \text{arity}=2 \rangle \\
\text{Happens} &= \langle \text{PositionalBinaryPredicate}={}\langle \text{Event},\text{Time} \rangle \rangle
\end{align*}
\]

– Non-Polymorphic Classical Dialects
  • Unique signature name per sort definition

– Dialects with Polymorphism
  • multiple (to countable infinite) different signatures with the same signature name
Example User-defined Signature Definition

```xml
<signature>
  <Time>
    <oid><Ind>Datetime</Ind></oid>
    <slot><Ind>date</Ind><Var type="xs:date"/></slot>
    <slot><Ind>time</Ind><Var type="xs:time"/></slot>
  </Time>
</signature>

<signature>
  <Event>
    <oid><Ind>TimeEvent</Ind></oid>
    <slot><Ind>event</Ind><Var type="xs:string"/></Var></slot>
    <slot><Ind>time</Ind><Var type="Datetime"/></slot>
  </Event>
</signature>
```

Frame type definitions:
- **Datetime** sort
- **TimeEvent** sort
Multi-Sorted Interpretation

- **Multi-Sorted Structure** interprets predicates, functions and constants in accordance with their sorts ($T$)

  $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I}, \text{Val})$, where

  - $\Delta^\mathcal{I}$ universe (the union* of its sorts)
  - $\cdot^\mathcal{I}$ interpretation function

  - $c^\mathcal{I} \subseteq T^\mathcal{I}$
  - $p^\mathcal{I} \subseteq T_1^\mathcal{I} \times \ldots \times T_n^\mathcal{I}$
  - $f^\mathcal{I} \subseteq T_1^\mathcal{I} \times \ldots \times T_n^\mathcal{I} \rightarrow T_{n+1}^\mathcal{I}$

  - $\text{Val}(\varphi, \sigma) \in TV$ truth valuation function, where

  - $TV$ is a set of partially or totally ordered truth values*
Example – Truth Degree Valuation

<Answer>
  <degree><Data>1</Data></degree>
  ...
</Answer>

truth degree 1 = true

<Answer>
  <degree><Data>0</Data></degree>
  ...
</Answer>

truth degree 0 = false

<Answer>
  <degree><Data>0.5</Data></degree>
  ...
</Answer>

truth degree 0.5 = undefined
RuleML Types (Sorted Logic)

• External and internal types (sorts) can be assigned by using the @type attribute

• External vocabularies / ontologies define types, e.g.,

  `<Var type="&vo;DatetimeEvent">E</Var>`

  `<Ind iri="&vo;e1" type="&vo;DatetimeEvent"/>`

• Semantics defined by **Semantic Profiles**
  – e.g. multi-sorted or order-sorted logic for type interpretation
  – e.g., import semantics (not just union of imported sorts)
  – e.g., mapping semantic with isomorphic structures interpreting composite structures as flat FOL structures, etc.
Reaction RuleML as „Webized“ Language
Prefix and Vocabulary Mapping

- Reaction RuleML has attributes which act as local or external identifiers and references, e.g., type, iri, node, key, keyref, ...
  - including references to the (Reaction) RuleML vocabulary

- There values can be terms, Curies and relative IRIs which need to be mapped to absolute IRIs
  - to enable Web-based imports, interchange and modularization

- @prefix
  - attribute defining a list of prefixes for the mapping of Compact URIs (Curies) to Internationalized Resource Identifiers (IRIs) as a white space separated list of prefix-name IRI pairs

- @vocab
  - an Internationalized Resource Identifier (IRI) that defines the vocabulary to be used when a term is referenced

```xml
<Var vocab="http://www.w3.org/2001/XMLSchema/" type="date"/>
```
Example - Typed Complex Event **Pattern** Definition

```xml
<Event key="#ce2" type="&ruleml;ComplexEvent">
  <signature> <!-- pattern signature definition -->
    <Sequence>
      <signature>
        <Event type="&ruleml;SimpleEvent">
          <signature><Event>...event_A...</Event></signature>
        </Event>
        <signature>
          <Event type="&ruleml;ComplexEvent" keyref="ce1"/>
        </signature>
        <signature>
          <Event type="cbe:CommonBaseEvent" iri="cbe.xml#xpointer(//CommonBaseEvent)"/>
        </signature>
      </signature>
    </Sequence>
    <signature>
      <Event key="#ce1" keyref="#ce2">
        <arg>...</arg>
      </Event>
    </signature>
  </signature>
</Event>

<Event key="#e1" keyref="#ce2">
  <arg>...</arg>
</Event>
```
Selected Reaction RuleML Algebra Operators

- **Action Algebra**

- **Event Algebra**
  - *Sequence* (Ordered), *Disjunction* (Or), *Xor* (Mutual Exclusive), *Conjunction* (And), *Concurrent*, *Not*, *Any*, *Aperiodic*, *Periodic*, *AtLeast*, *ATMost*, *Operator* (generic Operator)

- **Interval Algebra** (Time/Spatio/Event/Action/… Intervals)
  - *During*, *Overlaps*, *Starts*, *Precedes*, *Meets*, *Equals*, *Finishes*, *Operator* (generic Operator)

- **Counting Algebra**
  - *Counter*, *AtLeast*, *AtMost*, *Nth*, *Operator* (generic Operator)

- **Temporal operators**
  - *Timer*, *Every*, *After*, *Any*, *Operator* (generic Operator)

- **Negation operators**
  - *Naf*, *Neg*, *Negation* (generic Operator)
Reaction RuleML Examples with Types from RuleML Metamodel and External Ontologies

<Quantifier type="&ruleml;Forall"> == <Forall>

<Operator type="&ruleml;And"> == <And>

<Operator type="&ruleml;Conjunction"> == <Conjunction>

<Negation type="&ruleml;InflationaryNegation"> == <Naf>

<Action type="&ruleml;Assert"> == <Assert>

<Action type="&ruleml;Retract"> == <Retract>

<Event type="&ruleml;SimpleEvent"> == <Atom> …</Atom>

<Event type="ibm:CommonBaseEvent"> == IBM CBE
Reaction RuleML Metamodel

Top Level Reaction RuleML Ontologies

General concepts such as space, time, event, action and their properties and relations

Spatio Ontology  Action Ontology  Situation Ontology  Process Ontology  Agent Ontology

Temporal Ontology  Event Ontology

Domain Ontologies  Task Activities Ontologies

Vocabularies related to specific domains by specializing the concepts introduced in the top-level ontology

Vocabularies related to generic tasks or activities by specializing the concepts introduced in the top-level ontology

Specific user/application ontologies

Application Ontologies

E.g. ontologies describing roles played by domain entities while performing application activities
Example - Event Metamodel
(for defining Event Types of the Metamodel Event Class)

Integration of existing domain ontologies by defining their properties and values in an event classes in the Metamodel
Extended Event Metamodel (with computational properties and complex value definitions)
Top-Level Situation Types defined in Reaction RuleML Ontology

- Heterogeneous Situation
- Homogenous Situation
Example: Situation Types

- Metadata Properties
- Fluent Properties

**Situation Properties**
(time, location, participants, ...)

**Situation Content**

**Situation Types**
- Heterogeneous Situation
- Homogenous Situation

**Domain ontologies**
- Dynamic Change Situation
- Time Frame Situation
- Frequency Situation
- State Situation
- Process Situation
- Iterative Situation
- Habitual Situation

**Situation Descriptions**
- TrafficLight Changes
- Within5 Minutes
- Rings3 Times
- LaysOn TheFloor
- He Runs
- He Coughs
- He Smokes

use the other top ontologies
Reaction RuleML Event Types

- Event
  - Simple Event
  - Complex Event
Reaction RuleML Time Types

- **Time Types**
  - Time Instant
  - Time Interval
    - Time Point

- continuous time model
- discrete time model
Interval Top Ontology Metamodel

- Empty Interval
- Degenerated Interval
- Left Closed Right Open Interval
- Left Open Right Closed Interval
- Left Open Interval
- Left Closed Interval
- Right Closed Interval
- Right Open Interval
Action Top Ontology Metamodel

- Action
  - Atomic Action
  - Complex Action

Action Types
Semantic Profiles

• Define *intended semantics* for knowledge interpretation, reasoning, execution, …

• A semantic profile (SP) can define

  – profile (sub-)signature $S_{SP}$
  – profile language $\Sigma_{SP}$
  – intended interpretation semantics $I_{SP}$
  – axioms $\Phi_{SP}$
  – a semantics preserving translation function into Reaction RuleML $\tau(.)$
Expanded Profile Semantic Multi-Structure

\( I = \langle I_R, I_D, I_{SP} \rangle \) with

- \( I_R \) basic multi-sorted interpretation
  - basic structure of Reaction RuleML
- \( I_D \) default dialect interpretation
  - if no profile is specified \( I_D \) is used for default interpretation
- \( I_{SP} \) semantic profiles with expansion structures
  - multiple alternative semantic profiles can be specified (with possible preference ordering)
Semantic Profiles for Evaluation

- A semantics $SEM'$ extends $SEM$ ($SEM(P) \subseteq SEM'(P)$) for a rule program $P$ iff all formula $F$ which are true in $SEM(P)$ are also true in $SEM'(P)$, but in $SEM'(P)$ more formula can be true or false than with $SEM(P)$.

- $SEM'$ is defined for a class of programs that strictly includes the class of programs with the semantics $SEM$.

- $SEM'$ coincides with $SEM$ for all programs of the class of programs for which $SEM$ is defined.

A semantic profile $SEM'$ ($=\text{intended evaluation semantics of the inference/execution engine}$) can be used for all programs (or scoped modules) with $SEM(P) \subseteq SEM'(P)$.
# Example - Semantic LP Profiles

<table>
<thead>
<tr>
<th>Class</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definite LPs</td>
<td>Least Herbrand model: $M_p$</td>
</tr>
<tr>
<td></td>
<td>Supported Herbrand model: $M_p^{supp}$</td>
</tr>
<tr>
<td></td>
<td>Clark’s Completion: $COMP$</td>
</tr>
<tr>
<td></td>
<td>3-valued Completion: $COMP_3$</td>
</tr>
<tr>
<td></td>
<td>Well-founded Semantics: $WFS$</td>
</tr>
<tr>
<td></td>
<td>$WFS^+$ and $WFS'$</td>
</tr>
<tr>
<td></td>
<td>$WFS_C$</td>
</tr>
<tr>
<td>Stratified LPs</td>
<td>Strong Well-founded Semantics: $WFS_E$</td>
</tr>
<tr>
<td>Normal LPs</td>
<td>Extended Well-founded Semantics: $WFS_S$</td>
</tr>
<tr>
<td></td>
<td>Stable Model Semantics: $STABLE$</td>
</tr>
<tr>
<td></td>
<td>Generalized WFS: $GWFS$</td>
</tr>
<tr>
<td></td>
<td>$STABLE^+$</td>
</tr>
<tr>
<td></td>
<td>$STABLE_C$</td>
</tr>
<tr>
<td></td>
<td>$STABLE^{rel}$</td>
</tr>
<tr>
<td>General Disjunctive</td>
<td>Pereira’s $O – SEM$</td>
</tr>
<tr>
<td></td>
<td>Partial Model Semantics: $PARTIAL$</td>
</tr>
<tr>
<td></td>
<td>Regular Semantics: $REG – SEM$</td>
</tr>
<tr>
<td></td>
<td>Preferred Semantics: $PREFERRED$</td>
</tr>
<tr>
<td></td>
<td>Disjunctive WFS: $DWFS$</td>
</tr>
<tr>
<td></td>
<td>Generalized Disjunctive WFS: $GDWFS$</td>
</tr>
<tr>
<td>Stratified Disjunctive</td>
<td>Disjunctive Stable: $DSTABLE$</td>
</tr>
<tr>
<td></td>
<td>Perfect model $PERFECT$</td>
</tr>
<tr>
<td></td>
<td>Weakly Perfect: $WPERFECT$</td>
</tr>
<tr>
<td>Positive Disjunctive</td>
<td>Generalized Closed World Assumption: $GCWA$</td>
</tr>
<tr>
<td></td>
<td>Weak generalization closed world assumption: $WGCWA$</td>
</tr>
</tbody>
</table>
A semantic $\text{SEM}'$ extends a semantic $\text{SEM}$:

$\text{SEM}' \succeq \text{SEM}$ iff $\forall (P, F) \; \text{SEM}(P) \models F \Rightarrow \text{SEM}'(P) \models F$
Example: Use of Semantic LP Profiles for Interpretation

<-- rule interface with two alternative interpretation semantics and a signature.  
The interface references the implementation identified by the corresponding key -->

<Rule key="#r1">
  <evaluation index="1">
    <!-- WFS semantic profile -->
    <Profile type="&ruleml;Well-Founded-Semantics" />
  </evaluation>

  <evaluation index="2">
    <!-- alternative answer set stable model semantic profile -->
    <Profile type="&ruleml;Answer-Set-Semantics" />
  </evaluation>

  <!-- the signature defines the queryable head of the backward-reasoning rule -->
  <signature>
    <Atom><Rel>likes</Rel><Var mode="+"/><Var mode="-"/></Atom>
  </signature>
</Rule>

<-- implementation of rule 1 which is interpreted either by WFS or by ASS semantics and only allows queries according to it's signature definition. -->

<Rule keyref="#r1" style="reasoning">
  <if>...</if>
  <then>
    <Atom><Rel>likes</Rel><Var>X</Var><Var>Y</Var></Atom>
  </then>
</Rule>
Example - Integration of Semantic Profiles

1. Include/Import external Semantic Profile (includes profile axioms to knowledge base)

```xml
<xi:include href="../../profiles/SituationCalculusProfile.rrml" xpointer="xpointer(/RuleML/*)="/>
<evaluation>
  <Profile keyref="&ruleml;ReifiedClassicalSituationCalculus" />
</evaluation>
```

2. Reference pre-defined Semantic Profile as profile type

```xml
<evaluation>
  <Profile type="&rif;RDFS" iri="http://www.w3.org/ns/entailment/RDFS"/>
</evaluation>
```

3. Locally defined Semantic Profile

```xml
<Assert>
  <evaluation>
    <Profile key="&ruleml;ReifiedClassicalSituationCalculus">
      <formula><Rulebase> ... RuleML definition... </Rulebase></formula>
      <content> ... xs:any XML content, e.g. RIF, Common Logic 2 XML... </content>
    </Profile>
  </evaluation>
  <Rulebase>
    <Rule> ... </Rule>
    <Rule> ... </Rule>
  </Rulebase>
</Assert>
```

Reference published external semantic profiles, e.g. RIF, OWL, profiles ...

Note: also other non RuleML content models are supported
Example – Reified Situation Calculus
Axioms

(1) \( \forall (A1, A2, S1, S2) \ (do(A1,S1) = do(A2,S2)) \Rightarrow (A1 = A2) \)

(2) \( \forall (S1, S2, A) \ (S1 < do(A,S2)) \land (S1 \leq S2) \)

(3) \( \forall (S1, S2) \ (S1 \leq S2) \land (S1 < S2) \lor (S1 = S2) \)

(4) \( \forall (S1, S2) \ (S1 < S2) \Rightarrow \neg (S2 < S1) \)

(5) \( \forall (S, F) \ [holds(F,s0) \land (\forall (A) \ (holds(F,S) \Rightarrow holds(F,do(A,S))))] \Rightarrow holds(F,S) \)

(6) \( \neg S < s0 \)

...
Example: Axiomatization of Situation Calculus Profile in RuleML

<Assert>
<evaluation>
<Profile key="&ruleml;ReifiedSituationCalculus" >
<Rulebase key="&ruleml;ReifiedSituationCalculusBasicAxioms" >
<!-- Forall(A1, A2, S1, S2) (do(A1,S1) = do(A2,S2)) => (A1 = A2) -->
<Rule key="&ruleml;SituationCalculusBasicAxiom1" >
<quantification> <Forall>
  <declare><Var>A1</Var></declare>  <declare><Var>A2</Var></declaration>
  <declare><Var>S1</Var></declare>  <declare><Var>S2</Var></declaration>
</Forall> </quantification>
<if>
<Equal>
<Situation>
  <Do> <Action><Var>A1</Var></Action><Situation><Var>S1</Var></Situation>  </Do>
</Situation>
<Situation>
  <Do> <Action><Var>A2</Var></Action><Situation><Var>S2</Var></Situation></Do>
</Situation>
</Equal>
</if>
<then>
<Equal> <Action><Var>A1</Var></Action> <Action><Var>A2</Var></Action>  </Equal>
</then>
</Rule>
....
</Profile>
</evaluation>
</Assert>
# Example – RIF Semantic Web Compatibility Profiles

<table>
<thead>
<tr>
<th>Profile</th>
<th>IRI of the Profile</th>
<th>Model</th>
<th>Satisfiability</th>
<th>Entailment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td><a href="http://www.w3.org/ns/entailment/Simple">http://www.w3.org/ns/entailment/Simple</a></td>
<td>RIF-Simple-model</td>
<td>satisfiability</td>
<td>RIF-Simple-entailment</td>
</tr>
<tr>
<td>RDF</td>
<td><a href="http://www.w3.org/ns/entailment/RDF">http://www.w3.org/ns/entailment/RDF</a></td>
<td>RIF-RDF-model</td>
<td>RIF-RDF-satisfiability</td>
<td>RIF-RDF-entailment</td>
</tr>
<tr>
<td>RDFS</td>
<td><a href="http://www.w3.org/ns/entailment/RDFS">http://www.w3.org/ns/entailment/RDFS</a></td>
<td>RIF-RDFS-model</td>
<td>RIF-RDFS-satisfiability</td>
<td>RIF-RDFS-entailment</td>
</tr>
<tr>
<td>D</td>
<td><a href="http://www.w3.org/ns/entailment/D">http://www.w3.org/ns/entailment/D</a></td>
<td>RIF-D-model</td>
<td>RIF-D-satisfiability</td>
<td>RIF-D-entailment</td>
</tr>
<tr>
<td>OWL Direct</td>
<td><a href="http://www.w3.org/ns/entailment/OWL-Direct">http://www.w3.org/ns/entailment/OWL-Direct</a></td>
<td>RIF-OWL Direct-model</td>
<td>RIF-OWL Direct-satisfiability</td>
<td>RIF-OWL Direct-entailment</td>
</tr>
</tbody>
</table>

Simple < RDF < RDFS < D < OWL RDF-Based

Example:  
```xml
<evaluation>
  <Profile type="&rif;RDFS" iri="http://www.w3.org/ns/entailment/RDFS"/>
</evaluation>
```
Complex Event Processing – Semantic Profiles
(defined in <evaluation> semantic profiles)

1. **Definition**
   - Definition of event/action pattern e.g. by event algebra
   - Based on declarative formalization or procedural implementation
   - Defined over an atomic instant or an interval of time, events/actions, situation, transition etc.

2. **Selection**
   - Defines selection function to select one event from several occurred events (stored in an event instance sequence e.g. in memory, database/KB) of a particular type, e.g. “first”, “last”
   - Crucial for the outcome of a reaction rule, since the events may contain different (context) information, e.g. different message payloads or sensing information

3. **Consumption**
   - Defines which events are consumed after the detection of a complex event
   - An event may contribute to the detection of several complex events, if it is not consumed
   - Distinction in event messaging between “multiple receive” and “single receive”
   - Events which can no longer contribute, e.g. are outdated, should be removed

4. **Execution**
   - Actions might have an internal effect i.e. change the knowledge state leading to state transition from (pre-)condition state to post-condition state
   - The effect might be hypothetical (e.g. a hypothetical state via a computation) or persistent (update of the knowledge base),
   - Actions might have an external side effect
Example – Event Processing with Semantic Profiles

```xml
<Rule style="active">
  <evaluation>
    <Profile>e.g. definition, selection and consumptions policies</Profile>
  </evaluation>

  <signature>
    <Event key="#ce1">
      ... event pattern definition (for event detection)
    </Event>
  </signature>

  <on>
    <Event keyref="#ce1"/>
    <!-- use defined event pattern for detecting events -->
  </on>

  <if>
    ...
  </if>

  <do>
    <Assert safety="transactional">
      <!-- transactional update -->
      <formula>
        <Atom>
          ....
        </Atom>
      </formula>
    </Assert>
  </do>

</Rule>
```

**Interface**
(specific semantic profile + event pattern signature for event processing / detection)

**Implementation**
(reaction rule triggered by event detection)
Example – Event Definition Profile with Interval-based Global Time Model

[Lamport]

- **Event signature** $S_\mathcal{E} = \langle \mathcal{E}, \text{sort}, < \rangle$
  - $\mathcal{E}$ events, sort gives sort of event, $<$ precedence relation

- **Interval-based Event Model:** $\mathcal{I}_\mathcal{E}$ is a structure for $S_\mathcal{E}$ iff
  - *finite predecessor property:* $\forall e \in \mathcal{E}$ the set $\{ e' \in \mathcal{E} \mid e' < e \}$ is finite
  - *termination property:* if $e$ is of sort $\text{NonTerminatingEvent}$ then there is no $e' \in \mathcal{E}$ such that $e < e'$
  - *finitness property:* if $e$ is of sort $\text{TerminatingEvent}$ then $\{ e' \in \mathcal{E} \mid e' \geq e \}$ is finite.
  - *interval ordering property:* $\forall e_1, e_2, e_3, e_4 (e_1 < e_2 \land e_3 < e_4 \rightarrow e_1 < e_4)$.

- **Event Representations** $\tau(e), e \in \mathcal{E}$,
  - is a representation of $\mathcal{I}_\mathcal{E}$ in a linear interval-ordering $(\mathcal{T}, <_\mathcal{T})$ iff $\tau(e)$ is an interval in $\mathcal{T}$, and $e <_{\mathcal{I}_\mathcal{E}} e'$ iff $\tau(e) <_{\mathcal{T}} \tau(e')$.
  - **Simple Events:** $\tau(e)$ of form $[t, t] = [t]$
  - **Terminating Events:** $\tau(e)$ of form $[t, t')$
  - **Non-terminating Event:** $\tau(e)$ of form $[t, \infty)$

*e.g. left-closed, right open time intervals in $\mathcal{T}$

linear ordering of instantaneous events

events with duration and partial ordering of time intervals
Simple Events and Complex Events

- **Simple Events**: $e_1, e_2, \ldots$ of sort $SimpleEvent$
- **Complex Events**: $E_1, E_2, \ldots$ of sort $ComplexEvent$
  - $e_i \in E$: simple event $e_i$ is part of complex event $E$

**Precedence ordering**: $E \prec E'$ iff $\forall e \in E \forall e' \in E' (e \prec e')$
  - $E \prec e'$ iff $\forall e \in E (e \prec e')$ and $e' \prec E$ iff $\forall e \in E (e' \prec e)$

**Time ordering** (with explicit representation of temporal domain):
  - $\text{initiates}(e,E,T) / \text{terminates}(e',E,T)$: $e$ and $e'$ initiates/terminates $E$
    - transitive and irreflexive: $\text{initiates}(e,E,T) \prec \text{initiates}(e',E',T') \land \text{initiates}(e',E',T') \prec \text{initiates}(e'',E'',T'') \rightarrow \text{initiates}(e,E,T) \prec \text{initiates}(e'',E'',T'')$.  
      and never $\text{initiates}(e,E,T) \prec \text{initiates}(e,E,T)$
    - precedence: $E \prec E'$ iff $\text{initiates}(e,E,T) \prec \text{initiates}(e',E',T')$

**Representation**
  - $\tau(E) = [\text{initiates}(e,E,t), \text{terminates}(e',E,t')] = [t,t']$
Example – Complex Event Detection
Time-point vs. Interval-based Semantics

– Time-point semantics
  • Event definition: B; (A; C) (Sequence)
    event instance sequence EIS = b, a, c => detect event
    event instance sequence EIS = a, b, c => detect event
      – because the detection timepoint of (a; c) is c which is after the
        timepoint of b, i.e. the timepoint of (a; c) follows the timepoint of b.

– Interval-based semantics
  • Event definition B;(A; C) (Sequence)
    event instance sequence EIS = b, a, c => detect event
    event instance sequence EIS = a, b, c => do not detect event
      – because the interval [a, c] does not sequentially follow [b, b] but
        overlaps
Example: Interval-based Event Calculus Profile for Event Algebra Operators

- Event initiate and terminate Situations (Fluents) which hold at an time interval
- Interval-based Event Calculus semantics (model-theory + proof theory) models (complex) events as fluents ($Fl$) based on interval-based global time model ($Ti$)

$$I: Ti \times Fl \rightarrow TV$$

- Example: $B;(A;C)$ (Sequence)

$$ (B;(A;C)) [T1,T3] \leq holdsInterval([b,b] [T1,T1]) \land holdsInterval([a,c] [T2,T3]) \land [T1,T1] \leq [T2,T3]$$

- Rule-based implementation of EC event algebra, e.g. as meta logic program as part of (importable) semantic profile
- Rule-based arithmetic involving times and durations, e.g. Allen’s interval logic
Example Complex Event Algebra Operator - Sequence

- **Event / Action Algebra based on Interval-based Event Calculus**

  \[(A;B;C) \equiv \text{detect}(e,[T1,T3]) \iff \text{holdsInterval}([a,b],[T1,T2],[a,b,c]),
  \text{holdsInterval}([b,c],[T2,T3],[a,b,c]),
  [T1,T2] \leq [T2,T3].\]

- **Meta Program for CEP Algebra Operators**
  - Sequence, conjunction, or, xor, concurrent, neg, any, aperiodic

  \[\text{detect}(e,T) :- \text{event}([a,b],T), \quad \% \text{detection condition for the event e} \\
  \text{update}(\text{eis}(e),"\text{occurs}(e,\_0).",",[T]), \quad \% \text{add e with key eis}(e) \\
  \text{consume}(\text{eis}(a)), \text{consume}(\text{eis}(b)). \% \text{consume all a and b events}\]

  - Transient events: \text{occurs}(E,T)
  - Non-Transient Events: \text{happens}(E,T)
  - Consumption policy: Remove transient events from event instance sequence eis (managed by ID)

Testing Semantic Properties

• **Entailment Tests for Classical Logics**
  – *right weakening, reflexivity, and, or, left logical equivalence, cautious monotony, cut, rationality, negation rat., disjunction rat.*, …

• **Entailment Tests for Skeptical Logics**
  – *cumulativity, rationality, …*

• **Weak Semantic Properties Tests**
  – *elimination of tautologies, generalized principle of partial evaluation, positive/negative reduction, elimination of non-minimal rules, independence, relevance, …*
Example – Cautious Montony Test

\[
P: \quad a \leftarrow \text{not } b \\
\quad b \leftarrow \text{not } a \\
\quad c \leftarrow \text{not } c \\
\quad c \leftarrow a
\]

\[
P': \quad a \leftarrow \text{not } b \\
\quad b \leftarrow \text{not } a \\
\quad c \leftarrow \text{not } c \\
\quad c \leftarrow a
\]

\[
T: \{a=>\text{true}, c=>\text{true}\}
\]

- \text{STABLE}(P) \models_{TC} \{a, \text{not } b, c\}, \text{ i.e. } T \text{ succeeds.}
- \text{STABLE}(P') \models_{TC} \{\text{not } a, b, c\}, \text{ i.e. } T \text{ fails.}

→ Test Case: STABLE does not fulfil not cautious monotony
Test Cases for Self-validating Rule Bases

- **Test Cases** constrain the possible models and approximate the intended models of the rule base
  - **Queries** are used to test the rule base

- A test case is defined by $T := \{X, A, N\}$, where
  - $X \subseteq L$ assertion base (input data, e.g. facts)
  - $A \in L$ a formula denoting a test query
  - $N := +, -$ a positive or negative label

- **Semantics**

  $$M_0 \models_{TC} (X, A, +) \iff \forall m \in M_0 : m \in \sum (\text{Mod}(X), R) \Rightarrow m \in \text{Mod}(A)$$

  $$M_0 \models_{TC} (X, A, -) \iff \exists m \in M_0 : m \in \sum (\text{Mod}(X), R) \Rightarrow m \notin \text{Mod}(A)$$

  - $\models_{TC}$ compatibility relation
  - $\text{Mod}$ association function between sets of formulas and sets of models
  - $\Sigma$ model selection function

- $A \notin C_R(X)$ for $T := \{X, A, +\}$ and $A \in C_R(X)$ for $T := \{X, A, -\}$
- $C_R(X)$ deductive closure of $X$. Decidable inference operator based on formal proofs
Rule Markup for Test Cases / Test Suites

```xml
<TestSuite ruleBase="SampleBase.xml">
  <Test id="ID001" purpose="...">
    <Assert><formula>
      <And>
        <Atom>
          <Rel>parent</Rel>
          <Ind>John</Ind>
          <Ind>Mary</Ind>
        </Atom>
      </And>
    </formula></Assert>
    <TestItem expectedAnswer="yes">
      <Query><formula>
        <Atom closure="universal">
          <Rel>uncle</Rel>
          <Ind>Mary</Ind>
          <Var>Nephew</Var>
        </Atom>
      </formula></Query>
      <expectedResult>
        <VariableValuePair>
          <Var>Nephew</Var>
          <Ind>Tom</Ind>
        </VariableValuePair>
        <VariableValuePair>
          <Var>Nephew</Var>
          <Ind>Irene</Ind>
        </VariableValuePair>
      </expectedResult>
    </TestItem>
    <InferenceEngineSemantics><Profile>&ruleML;MinimalHerbrandSemantics</Profile></InferenceEngineSemantics>
  </Test>
</TestSuite>
```
Knowledge Modularization

- (Local) knowledge distribution
  - **key-keyref** attributes

- Knowledge imports
  - **Xinclude** (syntactic) and **Consult** (semantic) import action

- Knowledge modules
  - **Modes** partition the universe into subsets with input (+), output (-), open mode (?)
  - **Scopes** define subsets of the universe as modules with scoped interpretation: global, local, private, dynamic scopes
Separation of Interface and Implementation
(Distributed Knowledge)

- **Interface 1** (@key=ruleinterface1)
  - <Rule key="ruleinterface1">
    - <evaluation><Profile> ...p1… </Profile><evaluation>
    - <evaluation><Profile> …p2 …</Profile></evaluation>
    - <signature>…s1…</signature>
  </Rule>

- **Interface 2** (@key=ruleinterface2)
  - <Rule key="ruleinterface2">
    - <evaluation><Profile> ...p3… </Profile><evaluation>
    - <signature>…s2…</signature>
  </Rule>

- **Implementation 1** (@keyref=ruleinterface1)
  - <Rule keyref="ruleinterface1" key="ruleimpl1">
    - <if> … </if>
    - <do></do>
  </Rule>

- **Implementation 2** (@keyref=ruleinterface2)
  - <Rule keyref="ruleinterface2" key="ruleimpl2">
    - <if> … </if>
    - <do></do>
  </Rule>
Distributed Complex Event **Pattern** Definition

```xml
<Event key="#ce2" type="&ruleml;ComplexEvent">
  <signature> <!-- pattern signature definition -->
    <Sequence>
      <signature>
        <Event type="&ruleml;SimpleEvent">
          <signature><Event>...event_A...</Event></signature>
        </Event>
      </signature>
      <signature>
        <Event type="&ruleml;ComplexEvent" keyref="ce1"/>
      </signature>
      <signature>
        <Event type="cbe:CommonBaseEvent" iri="cbe.xml#xpointer(//CommonBaseEvent)"/>
      </signature>
    </Sequence>
  </signature>
</Event>

<Event key="#ce1" keyref="#ce2">
  <signature> <!-- event pattern signature -->
    <Concurrent>
      <Event><meta><Time>...t3</Time></meta><signature>...event_B</signature></Event>
      <Event><meta><Time>...t3</Time></meta><signature>...event_C</signature></Event>
    </Concurrent>
  </signature>
</Event>

<Event key="#e1" keyref="#ce2"><content>…</content></Event>
```
1. (Locally) Include external knowledge base

```
<xi:include href="../kb1.rrml"/>
...
```

2. Consult (import action) importing external knowledge base

```
<do>
  <Consult iri="../kb1.rrml"/>
</do>
```

3. Consult enclosed knowledge base

```
<Consult>
  <payload>
    <RuleML>
      ...
    </RuleML>
  </payload>
  <enclosed>
    <Message>
      ...
    </Message>
  </enclosed>
</Consult>
```
Module

- Imports become Modules ($\mathcal{M}$)
  - module interface $\mathcal{M}$ (meta knowledge)
  - module base $\mathcal{M}$

$\mathcal{M}$ - $\langle \mathcal{F}, \mathcal{I}, \mathcal{O}, \mathcal{G}, \mathcal{L}, \mathcal{P} \rangle$ with
  - $\mathcal{F}$ all knowledge formulas in $\mathcal{M}$
  - $\mathcal{I}, \mathcal{O}$ sets of input or output formulas in $\mathcal{M}$ with input (+) or output mode (-)
  - $\mathcal{G}, \mathcal{L}, \mathcal{P}$ sets of formulas with global, local, private scope in $\mathcal{M}$
Example – Modes and Scopes

<signature>
  <Atom type="ruleml:Happens" arity="2" mode="+" scope="global">
    <Rel>planned</Rel>
    <Var type="ruleml:Event" mode="+"/>
    <Var type="ruleml:Time" mode="+"/>
  </Atom>
</signature>

<signature>
  <Event key="#ce1" scope="private">
    <Concurrent> ... </Concurrent>
  </Event>
</signature>

<scope>
  <Atom>oid</Ind>m1</Ind>oid><Rel>source</Rel><Ind>module1</Ind></Atom>
</scope>

<signature>
  <Atom type="ruleml:Happens" arity="2" mode="+" scope="m1">
  ...
</signature>
Import Semantics - Module Composition

- **Semantic Profiles for Module Composition** need to address
  - composite modules with nested scopes (flattening)
  - interfaces
    - global, local, and private scope
    - input and output modes
  - cyclic dependencies

1. **Syntactic Operation**
   - combines the modules (e.g. pure syntactic *union* vs. transformations guaranteeing compositionality)

2. **Semantic Operation**
   - joins the models of the modules (natural *join* of semantic models vs. other semantic operations)
(Default) Module Composition

\[ M \oplus M' = \langle F_M \cup F_{M'}, (I_M \setminus O_{M'} \cup I_{M'} \setminus O_M), (O_M \cup O_{M'}) , (G_M \cup G_{M'}) , (L_M \cup L_{M'}) , (P_M \cup P_{M'}) \rangle \]

- remove input by output formulas from imports
  - \( I_M \setminus O_{M'} \) and \( I_{M'} \setminus O_M \)

- respect private signatures
  - \( \text{signature}(M) \cap P_M = \emptyset \) and \( \text{signature}(M') \cap P_M = \emptyset \)

- output signatures are disjoint *
  - \( O_{M'} \cap O_M = \emptyset \)

- modules are mutual independent *

  - no (positive) cyclic dependencies through input – output signatures between \( M \) and \( M' \)

* semantic profiles might relax these conditions, but must guarantee compositionality
Modular Semantic Multi-Structure

- **Semantic Multi-Structure** \( \mathcal{I} - \langle \mathcal{I}_{\mathcal{M}_0}, \mathcal{I}_{\mathcal{M}_1}, \mathcal{I}_{\mathcal{M}_2}, \ldots \rangle \)
  
  set of semantic structures of the importing KB \((\mathcal{M}_0)\) and the imported modules \((\mathcal{M}_1, \mathcal{M}_2, \ldots)\)

  - Semantic structures of modules coincide (by default) in the interpretation of global symbols, but might differ in the interpretation of scoped symbols (local, private,..) in each modules.

  - **Natural Outer Join** as *default semantic operation* joining the sets of interpretations \(\mathcal{I}_{\mathcal{M}}\) and \(\mathcal{I}_{\mathcal{M}'}\), of two modules \(\mathcal{M}\) and \(\mathcal{M}'\):

\[
\mathcal{I}_{\mathcal{M}} \bowtie \mathcal{I}_{\mathcal{M}'} = \{ \mathcal{M}^\mathcal{I} \cup \mathcal{M}'^\mathcal{I} \mid \mathcal{M}^\mathcal{I} \in \mathcal{I}_{\mathcal{M}} \text{ and } \mathcal{M}'^\mathcal{I} \in \mathcal{I}_{\mathcal{M}'} \text{, and } \mathcal{M}^\mathcal{I} \cap (\mathcal{F}_{\mathcal{M}'} \setminus \mathcal{P}_{\mathcal{M}'}) = \mathcal{M}'^\mathcal{I} \cap (\mathcal{F}_{\mathcal{M}} \setminus \mathcal{P}_{\mathcal{M}}) \}
\]

  - Semantic Profiles can define other semantics join operations
Semantic Profiles in Nested Scopes

\[
\text{SEM}''(\#\text{module1}) \geq \text{SEM}'(\#\text{innermodule1.1}) \geq \text{SEM}(\#\text{rule1}) \\
\text{SEM}(\#\text{rule2}) = \text{SEM}''(\#\text{module1}) \\
\text{SEM}'(\#\text{innermodule1.2}) = \text{SEM}''(\#\text{module1})
\]
Scopes – Constructive Views on the Knowledge Base

• A **scope definition** defines a static or dynamic **constructive view** on the knowledge base by meta constraints on the meta knowledge \((@L_1), \ldots, @(L_n) \Phi)\) of the formulas in the KB

  – example scopes e.g. quantification scopes, qualification scopes, module scopes, conversation scopes, execution scopes (including transactions), user-define metadata scopes

• **default scopes**, e.g., quantification scope is "universal", default module visibility scope of relations and functions is "global", local conversation scopes, etc.

• **user-defined scopes**
  
  – `<scope>` user defined metadata scope definition
  
  – `scope=" "` reference to scope definitions by scope name
Scoped Interpretation and Reasoning

- **Scoped Interpretation**: $I_{sc}$ is the reduct $I_{KB} \upharpoonright S_{sc}$ of $I_{KB}$ to $S_{sc} \subseteq S_{KB}$
- **Scoped Reasoning** $I_{sc} \models \Phi$ iff $I_{sc} \models \Phi$
- **Scoped Literals** (query/goal)
  - $\{ @(L_1), \ldots, @(L_n) \} \Phi$, where $L_1 \ldots L_n$ are (possibly quantified) meta knowledge constraints defining a scope
    - e.g., $@author(A)@valid(T)src("module1")p(X)$
  - **scope definition** literal
- **Scope Closure, Boundedness, Monotonicity,**...
  - Semantic Profiles can define the scope semantics, e.g.,
    - *local closure*: only set of all formula which meta knowledge satisfies the constraints of the scope definition
    - *transitive closure*: local closure plus all formula which are in the scopes of (scoped) literals in the scope
    - *scope inheritance*: outer scopes are inherited to nested inner scopes
    - *scope boundedness*: scope is recursively only depending on scoped literals (i.e. closed)
    - *scope monotonicity*: consequences from scope are monotonic wrt addition of scopes
Example: Constructive Views with Scopes and Guards

<Assert key="#module1">
  <meta> <!-- descriptive metadata -->
    <Atom><Rel iri="dc:creator"/><Ind>Adrian Paschke</Ind></arg></Atom>
  </meta>
  <qualification> <!-- qualifying metadata -->
    <Atom> <!-- the module is valid for one year from 2011 to 2012 -->
      <Rel>valid</Rel>
      <Interval type="&ruleml;TimeInstant">
        <Time type="&ruleml;TimeInstant"><Data xsi:type="xs:date">2011-01-01</Data></Time>
        <Time type="&ruleml;TimeInstant"><Data xsi:type="xs:date">2012-01-01</Data></Time>
      </Interval>
    </Atom>
  </qualification>
  <Rulebase key="#innermodule1.1">
    <scope> <!-- scope: only knowledge authored by "Adrian Paschke" -->
      <Atom><Rel iri="dc:creator"/><Ind>Adrian Paschke</Ind></arg></Atom>
    </scope>
    <scope> <!-- scope: only valid knowledge; validity value will be bound to variable V -->
      <Atom><Rel>valid</Rel><Var> V </Var></Atom>
    </scope>
    <Rule key="#rule1"> <!-- scope: only knowledge from source "module1" -->
      <scope><Atom><Rel>source</Rel><Ind>module1</Ind></Atom></scope>
      <guard> <!-- guard on the validity: "current date during validity of module1" -->
        <Operator type="&ruleml;During"><Expr iri="…getDateTime()"/><Var> V </Var></Operator>
      </guard>
      <if>…</if> <then>  </then>
    </Rule>
  </Rulebase>
</Assert>
Example - Semantic Profile for Event Based System Executions with Scoped Interpretation

• Event Based System Execution
  – multi-sorted interpretation \( I_\mathcal{E} \) of a multi-sorted signature \( \mathcal{S}_{KB} \) which contains a (precedence based or time based) event signature
  \( \mathcal{S}_\mathcal{E} = \langle \mathcal{E}, \text{sort}, \prec \rangle (\mathcal{S}_\mathcal{E} \subseteq \mathcal{S}_{KB}) \), where the scoped interpretation of \( I_\mathcal{E} \) in the scope \( SC(\mathcal{S}_\mathcal{E}) \) is a global time model (see slide 129).

• Event Based System Model
  • collection of system executions \( I_\mathcal{M} \) in a fixed modular signature*

• Composition of System Models
  \( \mathcal{M} \) and \( \mathcal{M}' \) are scoped modules with \( \mathcal{M} \oplus \mathcal{M}' \) being their composition
  – Serial: \( I_\mathcal{M} \otimes I_\mathcal{M}' = I_\mathcal{M} \odot I_\mathcal{M}' \) (join union)
  – Parallel: \( I_\mathcal{M} \parallel I_\mathcal{M}' = \{ I_{\mathcal{M} \oplus \mathcal{M}'} | I_{\mathcal{M} \oplus \mathcal{M}'} \mathcal{M} \in I_\mathcal{M} \text{ and } I_{\mathcal{M} \oplus \mathcal{M}'} \mathcal{M}' \in I_\mathcal{M}' \} = I_\mathcal{M} \cap I_\mathcal{M}' \) (intersection)

* imports and updates are added as scoped modules \( \mathcal{M}_i \)
Knowledge Update Actions

• Imported knowledge (<Consult> action) and knowledge actions (<Assert>,<Retract>,<Update>,<Action>) are added as modules which are automatically labelled with meta knowledge about
  • source: @source([locator])
  • label/name: @label([oid])

• **Labelled transition logic**: transition of states by module updates
  • **Assert**: $KB_i = KB_{i-1} \cup M_{\text{assert}}^{oid}$ and Retract: $KB_i = KB_{i-1} \setminus M_{\text{retract}}^{oid}$
  
  – **local states** can be managed using **module scopes** and scoped reasoning using the module scopes is possible

Examples

```xml
<Retract scope="module1"/>  Retract module with scope “module1”
<Query scope="module1">… Query module with scope "module1"
<on><Event scope="module1">… Detect event within scope „module1“
```
Example – Transaction Logic Semantic Profile

[Bonner et al., 1993]

• **Execution path** ≡ sequence of knowledge base states
  • Truth values are over execution paths, not over states
    – a path being true means *execution* of that path

• **Elementary state transitions**: update actions that cause a priori defined state transitions
  
  **assert(Φ)**: \( D' = D \cup \Phi \) – add \( \Phi \) to state \( D \) leading to state \( D' \)
  
  **retract(Φ)**: \( D' = D \setminus \Phi \) – delete \( \Phi \) from state \( D \) leading to state \( D' \)
  
  **update(Φ, Φ’)**: \((\text{retract}(\Phi) \otimes \text{assert}(\Phi'))\) – atomic update

• **Model-theoretic Semantic**
  
  – A *path structure* maps execution paths to the ordinary semantic structures used in classical predicate logic:
    \( I(\pi) = M \), where \( \pi \) - path, \( M \) – classical semantic structure, which says which transactions can execute along the path \( \pi \)

  *In addition:*
  
  • If \( \pi = <D> \) is a path that consists of only one database state then \( I(\pi) \) must make every fact in \( D \) true.
  • If \( \pi = <D, D \cup \Phi> \) then \( I(\pi) \) should make \( \text{assert}(\Phi) \) true
  • If \( \pi = <D, D \setminus \Phi> \) then \( I(\pi) \) should make \( \text{retract}\{\Phi}\) true
Transaction Logic Semantics

• Transactional Satisfaction

\[ I(<D_0, \ldots, D_n>) \models a \otimes b \iff \exists D_k \text{ such that } I(<D_0, \ldots, D_k>) \models a \text{ and } I(<D_k, \ldots, D_n>) \models b \]

\[ I(<D_0, \ldots, D_n>) \models a \land b \iff I(<D_0, \ldots, D_n>) \models a \text{ and } I(<D_0, \ldots, D_n>) \models b \]

\[ I(<D_0, \ldots, D_n>) \models a \rightarrow b \iff I(<D_0, \ldots, D_n>) \models a \text{ then } I(<D_0, \ldots, D_n>) \models b \]

• Transactional Actions

<Action safety="transactional"> … transactional execution of action

Example:

if (condition \land action) is true, but post-condition false, then (condition \land action) \otimes postcondition is false on \( \pi \).

if (condition \land action) \otimes postcondition is false on \( \pi \), then else-action is true on \( \pi \).

\rightarrow rollback (e.g., by reversing the actions on updated modules using their module scope)

\rightarrow else action (e.g., by executing compensation action)
Extension - Concurrent Transaction Logic

- **CTR (Concurrent Transaction Logic)** [Bonner et al., 1996]
  - A general logic for **state-changing actions**
  - Supports both **serial** and **concurrent** execution

- Every transaction is a **sequence of state changes** (i.e., a execution **path**)

- **Database Operation Semantics**
  - **Data oracle** $\mathcal{O}^d(D)$: specifies queries to a particular state $D$
  - **Transition oracle** $\mathcal{O}^t(D_1, D_2)$: specifies state transitions from $D_1$ to $D_2$

- **Complex Transactions**
  - Serial transactions $\phi \otimes \phi$; **concurrent transactions** $\phi \mid \phi$; nondeterministic transactions $\phi \lor \phi$;
Distributed Reaction RuleML Bases - Coupling Approaches

- **Strong coupling**
  - Interaction through a stable interface
  - API call is hard coded

- **Loose coupling**
  - Resilient relationship between two or more systems or organizations with some kind of exchange relationship
  - Each end of the transaction makes its requirements explicit, e.g. as an *interface description*, and makes few assumptions about the other end

- **Decoupled**
  - de-coupled in time using *(event)* messages (e.g. via Message-oriented Middleware (MoM))
  - Often asynchronous stateless communication (e.g. publish-subscribe or CEP event detection)
• **Send a message:**  \(<\text{Send}>\) \(<\text{Message}>\)

• **Receive a message:**  \(<\text{Receive}>\) \(<\text{Message}>\)

\(<\text{Message}>\)

\(<\text{cid}>\)  \(<!-\text{ conversation ID--}->\)  \(<\text{cid}>\)

\(<\text{protocol}>\)  \(<!-\text{ transport protocol -->}\)  \(<\text{protocol}>\)

\(<\text{directive}>\)  \(<!-\text{ pragmatic directive -->}\)  \(<\text{directive}>\)

\(<\text{sender}>\)  \(<!-\text{ sender agent/service -->}\)  \(<\text{sender}>\)

\(<\text{receiver}>\)  \(<!-\text{ receiver agent/service -->}\)  \(<\text{receiver}>\)

\(<\text{payload}>\)  \(<!-\text{ message payload -->}\)  \(<\text{payload}>\)

\(</\text{Message}>\)

- **cid** is the *conversation identifier* (enabling also subconversations)
- **protocol**: *protocol* definition (high-level protocols and transport prot.)
- **agent** (*send*, **receiver**): denotes the target or sender **agent** of the message
- **directive**: pragmatic context **directive**, e.g. FIPA ACL primitives
- **payload**: Message **payload** (*<content>* for arbitrary XML **content**)
Reaction RuleML Messaging

- `<cid>` local conversation scope in which system executions of send and receive events/actions take place

- `<protocol>` protocol language, e.g. with semantics defined by system executions and local states

- `<receiver/sender>` agent A with (external) knowledge base \( KB_A \) and public signatures \( S_{A_{public}} \subseteq S_A \) and external background structure \( \mathcal{I}_A \)

  - the universe of \( \mathcal{I}_A \) contains the sorts of the public signatures* of A  

- `<directive>` pragmatic context used for interpretation of the payload/content formulas (e.g., interpretation as query)

- `<payload>` or `<content>` RuleML formulas or XML content

- `<Send>` action and `<Receive>` event

* Note, agent's signature \( S_A \) might contain further private signatures
Example: Loosley-Coupled Communication via Messages to Agent Interface

<Message>
  <cid> <Ind>conversation1</Ind> </cid>
  <protocol> <Ind>esb</Ind> </protocol>
  <directive><Ind iri="acl:query-ref"/></directive>
  <sender> <Ind>Agent1</Ind> </sender>
  <receiver> <Ind>Agent2</Ind> </receiver>
  <payload>
    <Atom>
      <Rel>likes</Rel>
      <Ind>John</Ind>
      <Ind>Mary</Ind>
    </Atom>
  </payload>
</Message>

- Event Message is local to the conversation scope (cid) and pragmatic context (directive)

FIPA ACL directive

Interpreted by Agent 2 as query according to ACL:query-ref

Note: the receiver „Agent2“ needs to specify an appropriate <signature> for „likes“
<Rule>
...
<do><Send><Message> ...query1  </Message></Send></do>
<do><Send><Message> ...query2  </Message></Send></do>
<on><Receive><Message> ...response2</Message>  </Receive></on>
<if> prove some conditions, e.g. make decisions on the received answers </if>
<on><Receive><Message> ...response1 </Message></Receive></on>
....
</Rule>

Note: The „on“, „do“, „if“ parts can be in arbitrary combinations, e.g. to allow for a flexible workflow-style logic with subconversations and parallel branching logic
Example – Serial Horn Rules with Agent Messaging

(Implemented in Prova – http://www.prova.ws and Rule Responder http://responder.ruleml.org)

• Send a message
  $sendMsg(CID, Protocol, Agent, Performative, [Predicate|Args]|Context)$

• Receive a message
  $rcvMsg(CID, Protocol, Agent, Performative, [Predicate|Args]|Context)$

• Receive multiple messages
  $rcvMult(CID, Protocol, Agent, Performative, [Predicate|Args]|Context)$

send and receive event/action built-ins can occur in the head and body of **serial horn rules**, e.g.,

$$rcvMsg(...) :- p(...), sendMsg(...), not(q(...)), rcvMsg(...).$$

$$q(...) :- rcvMult(...), r(...), ... .$$

• **serial forward execution**: precondition $\otimes$ $sendMsg/rcvMsg/rcvMult \otimes$ postcondition

• **backward reasoning**: $rcvMsg/rcvMult$ predicate is interpreted as a goal literal

• **proof computations**: resolvent clauses are locally managed within a conversation scope (defined by CID) waiting for incoming „messages“ with a predicate unifying with the postponed $rcvMsg/rcvMult$ goal literal

* concurrent transaction logic semantics
Agenda

• Introduction to Event Processing
• Event Processing Reference Model and Reference Architecture
• Reaction RuleML Standard
• Examples for Event Processing Functions
• Summary
Patterns Coverage

Event Reaction
Assessment, Routing, Prediction, Discovery, Learning

Complex Event Detection
Consolidation, Composition, Aggregation

Event Preparation
Identification, Selection, Filtering, Monitoring, Enrichment

Event Analysis
Analytics, Transforms, Tracking, Scoring, Rating, Classification

Event Production
Application Time Publication, Retrieval

Event Consumption
Dashboard, Apps, External Reaction

State Management

Event Actions
Event Correlations and Patterns
Event Computations
Event Selections
Event Production/Consumption

see: Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Tutorial on advanced design patterns in event processing. DEBS 2012: 324-334; www.slideshare.net/isvana/eps-debs2012-event-processing-reference-architecture-design-patterns-v204b
Example: Semantic CEP - Filter Pattern

Filter Pattern:
Stocks of companies, which have production facilities in Europe and produce products out of metal and have more than 10,000 employees.

Event Stream – stock quotes
{(Name, “OPEL”) (Price, 45) (Volume, 2000) (Time, 1) }
{(Name, “SAP”) (Price, 65) (Volume, 1000) (Time, 2)}

Semantic Knowledge Base
{(OPEL, is_a, car_manufacturer),
(car_manufacturer, build, Cars),
(Cars, are_build_from, Metall),
(OPEL, hat_production_facilities_in, Germany),
(Germany, is_in, Europe),
(OPEL, is_a, Major_corporation),
(Major_corporation, have, over_10,000_employees)}
Knowledge-based Semantic Complex Event Processing

http://corporate-semantic-web.de/semantic-complex-event-processing.html
Example: Semantic Enrichment of Event Stream

Event Stream: e1, e2, e1

Raw Events: e1, e2, e1

Semantic Enrichment

Derived Events: e3, e4, e3

Knowledge Base

Final Processing on EPN

Complex Events: e1, e2, e1

Example: Semantic Event Query Pre-Processing

Event Stream: e1 e2 e1

Raw Events

Event Processing Network

Complex Events

Final Processing distributed on a network of processing Agents

Rewrite Simple Queries

q1 q2 q3

Complex Query Pre-Processing

Event Query Pre-Processing

Knowledge Base

Semantic Query Filer:
Stocks of companies, which have production facilities in Europe and produce products out of metal and have more than 10,000 employees.

Event Stream
{(Name, “OPEL”) (Price, 45) (Volume, 2000) (Time, 1)}
{(Name, “SAP”) (Price, 65) (Volume, 1000) (Time, 2)}

Semantic Knowledge Base
{(OPEL, is_a, car_manufacturer),
(car_manufacturer, build, Cars),
(Cars, are_build_from, Metall),
(OPEL, hat_production_facilities_in, Germany),
(Germany, is_in, Europe),
(OPEL, is_a, Major_corporation),
(Major_corporation, have, over_10,000_employees)}

rcvMult(SID, stream, “S&P500“, inform,
tick(Name^^car:Major_corporation, P^^currency:Dollar,
T^^time:Timepoint)) :- ...
<semantic filter inference> .
% Filter for car manufacturer stocks and enrich the stock tick
% event with data from Wikipedia (DBPedia) about the manufacturer
% and the luxury cars
rcvMult(SID,stream,"S&P500", inform, tick(S,P,T)) :-
carManufacturer(S,Man), % filter car manufacturers
findall([luxuryCar|Data],luxuryCar(Man,Name,Car),Data), % query
EnrichedData = [S,Data], % enrich with additional data
sendMsg(SID2,esb,"epal", inform, happens(tick(EnrichedData,P),T)).

% rule implementing the query on DBPedia using SPARQL query
luxuryCar(Manufacturer,Name,Car) :-
Query="SELECT ?manufacturer ?name ?car % SPARQL RDF Query
    WHERE { ?car <http://purl.org/dc/terms/subject>
    ?car foaf:name ?name .
    } ORDER by ?manufacturer ?name",
sparql_select(Query,manufacturer(Manufacturer),name(Name),car(Car)).
Preparation: Enrichment: Implementations: Prova: External Data and Object Integration

- **Java**
  ```java
  ... L=java.util.ArrayList(), ... java.lang.String(“Hello”)
  ```

- **File Input / Output**
  ```
  ..., fopen(File,Reader), ...
  ```

- **XML (DOM)**
  ```
  document(DomTree,DocumentReader) :-
  XML(DocumenReader),
  ```

- **SQL**
  ```
  ...,-sql_select(DB,cla,[pdb_id,”1alx“],px,Domain]).
  ```

- **RDF**
  ```
  ...,-rdf(http://...,"rdfs",Subject,"rdf_type","gene1_Gene"),
  ```

- **XQuery**
  ```
  ...,XQuery = ' for $name in StatisticsURL//Author[0]/@name/text() return $name',
  xquery_select(XQuery,name(ExpertName)),
  ```

- **SPARQL**
  ```
  ...,sparql_select(SparqlQuery,name(Name),class(Class),
  definition(Def)),
  ```

**Event Preparation**
Identification, Selection, Filtering, Monitoring, **Enrichment**
**Patterns Coverage**

- **Event Reaction**
  - Assessment, Routing, Prediction, Discovery, Learning

- **Complex Event Detection**
  - Consolidation, Composition, Aggregation

- **Event Analysis**
  - Analytics, Transforms, Tracking, Scoring, Rating, Classification

- **Event Preparation**
  - Identification, Selection, Filtering, Monitoring, Enrichment

- **Event Production**
  - Application Time Publication, Retrieval

- **Event Consumption**
  - Dashboard, Apps, External Reaction

- **State Management**

---

see: Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Tutorial on advanced design patterns in event processing. DEBS 2012: 324-334; [www.slideshare.net/isvana/eps-debs2012-event-processing-reference-architecture-design-patterns-v204b](http://www.slideshare.net/isvana/eps-debs2012-event-processing-reference-architecture-design-patterns-v204b)
% stream1 is trusted but stream2 is not, so one solution is found: X=e1

@src(stream1) event(e1).
@src(stream2) event(e2).

%note, for simplicity this is just a simple fact, but more complicated rating, trust, reputation policies could be defined
trusted(stream1). %only event from „stream1“ are trusted

ratedEvent(X):-
    @src(Source) %scoped reasoning on @src
    event(X) [trusted(Source)]. %guard on trusted sources
:-solve(ratedEvent(X)). % => X=e1 (but not e2)
Example - Scoring Implementation in Prova: Metadata Scoped Reasoning

% event instances with metadata @key(value, [ , value ])*

@score(1) @src(stream1) @label(e1) tick("Opel",10,t1).
@score(3) @src(stream2) @label(e2) tick("Opel",15,t1).

happens(tick(S,P),T):-
% scope defined over @score metadata bound to "Value"
@score(Value) tick(S,P,T) [Value>2]. %guard Value > 2

@score(1) @src(stream1) @label(e1) tick("Opel",10,t1).
@score(3) @src(stream2) @label(e2) tick("Opel",15,t1).

happens(tick(S,P),T):-
% select ticks from "stream1"
@score(Value) @src(stream1) @label(E) tick(S,P,T),
NewValue = Value + 3, %add 3 to score "Value"
%update selected events from "stream1" with "NewValue"
update(@label(E), @score(NewValue) tick(S,P,T)),
@score(S) tick(S,P,T) [S>3]. %guard Value > 3
Prova Dynamic Java Programming

- Ability to embed Java calls
  \[ ... : x = \texttt{java.lang.Math.abs}(2.99), x+1, ... \]
- Java constructors, object instance + methods and static methods, and public field access; (+ Java exception handling)

Prova Functional Programming

- single- and multi-valued functions,
- functional composition with the extended \textit{derive} built-in;
- partial evaluation;
- lambda functions;
- monadic functions;
- monadic \textit{bind} using a composition of \textit{map} and \textit{join};
- \textit{maybe}, \textit{list}, \textit{state}, \textit{tree} and \textit{fact} monads
Analysis:Classification:Implementations: Prova: Rules - Example

Event Stream
{((Name, "OPEL") (Price, 45) (Volume, 2000) (Time, 1))}

rcvMult(SID, stream, "S&P500", inform, tick(Name^^Car:Manufacturer, P, T)) :- size(Name, Employees), Employees > 10000, %type discovery

sendMsg(SID, self, 0, inform, tick(Name^^Major_Cooperation, P, T))

Classification according to Type System

Classification by Type Discovery (may also include Type Learning)

Semantic Knowledge Base
(T-Box Model is used as Type System)

{(OPEL, is_a, Car_Manufacturer),
 (Car_Manufacturer, sameAs, Motor_Vehicel_Manufacturer),
 (Car_Manufacturer, subClassOf, Automotive_Company),
 (Automotive_Company, sameAs, Automotive_Corporation),
 (Automotive_Company, subClassOf, Automotive_Industry),
 (Automotive_Corporation, subClassOf, Corporation),
 (Major_Corporation, have, over_10,000_employees),
 (Major_Corporation, subClassOf, Corporation)}
Patterns Coverage

Event Reaction
- Assessment, Routing, Prediction, Discovery, Learning

Complex Event Detection
- Consolidation, Composition, Aggregation

Event Analysis
- Analytics, Transforms, Tracking, Scoring, Rating, Classification

Event Preparation
- Identification, Selection, Filtering, Monitoring, Enrichment

Event Production
- Application Time Publication, Retrieval

Event Consumption
- Dashboard, Apps, External Reaction

State Management
- Event Actions
- Event Correlations and Patterns
- Event Computations
- Event Selections
- Event Production/Consumption

see: Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Tutorial on advanced design patterns in event processing. DEBS 2012: 324-334; www.slideshare.net/isvana/epst-debs2012-event-processing-reference-architecture-design-patterns-v204b
Detection:Consolidation:Implementations: Prova: Consolidation Example

- Interval-based Event Calculus semantics (model-theory + proof theory) based on *time intervals* modeled as *fluents*

\[ I: T_{interval} \times Fl \to \{\text{true, false}\} \]

- Example: \( D = A; (B; C) \) (consolidation: derive event \( D \) from sequence composition)

\[
\begin{align*}
T1 & \quad T2 & \quad T3 & \quad T4 \\
A & \quad B & \quad C \\
[A,A] & \quad [B,C] & \quad \downarrow \\
[[A,A],[B,C]] & \quad \text{Consolidation} \\
D = [A,C] & \quad \downarrow \\
\end{align*}
\]

- Example: derived situation from complex event detection (consolidation: initiate situation1 by event \( D \))

\[
\begin{align*}
\text{initiates}(D, \text{situation1}, T). \\
\text{holdsAt}(\text{situation1}, t5) \quad \Rightarrow \quad \text{yes}
\end{align*}
\]
Detection:Aggregation:Implementations: Prova: Example with Time Counter

% This reaction operates indefinitely. When the timer elapses (after 25 ms), the groupby map Counter is sent as part of the aggregation event and consumed in or group, and the timer is reset back to the second argument of @timer.

`groupby_rate() :-
Counter = ws.prova.eventing.MapCounter(), % Aggr. Obj.
@group(g1) @timer(25,25,Counter) % timer every 25 ms
rcvMsg(XID,stream,From,inform,tick(S,P,T)) % event
[IM=T,Counter.incrementAt(IM)]. % aggr. operation

`groupby_rate() :-
% receive the aggregation counter in the or reaction
@or(g1) rcvMsg(XID,self,From,or,[Counter]),
... <consume the Counter aggregation object>.
Patterns Coverage

Event Reaction
- Assessment, Routing, Prediction, Discovery, Learning

Complex Event Detection
- Consolidation, Composition, Aggregation

Event Analysis
- Analytics, Transforms, Tracking, Scoring, Rating, Classification

Event Preparation
- Identification, Selection, Filtering, Monitoring, Enrichment

Event Production
- Application Time
- Publication, Retrieval

Event Consumption
- Dashboard, Apps, External Reaction

State Management

see: Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Tutorial on advanced design patterns in event processing. DEBS 2012: 324-334;
www.slideshare.net/isvana/epst-debs2012-event-processing-reference-architecture-design-patterns-v204b
% detect suspicious logins by assessing the IP numbers of the login events from the same user login

\texttt{rcvMsg(XID,Protocol,From,request,login(User,IP))} :-

% if the next follow up event (@\texttt{count(1)}) that follows the previous login in the same reaction group \texttt{g1} is send from another IP (IP2!\texttt{=}IP)

@\texttt{group(g1) @count(1)}

\texttt{rcvMsg(XID,Protocol,From,request,login(User,IP2)) [IP2\texttt{=}IP],

println(['"Suspicious login",User,IP,IP2"," "]).}
rcvMsg(XID, esb, From, query-ref, buy(Product)) :-
    routeTo(Agent, Product), % derive processing agent
    % send order to Agent in new subconversation SID2
    sendMsg(SID2, esb, Agent, query-ref, order(From, Product)),
    % receive confirmation from Agent for Product order
    rcvMsg(SID2, esb, Agent, inform-ref, order(From, Product)).

% route to event processing agent 1 if Product is luxury
routeTo(epa1, Product) :- luxury(Product).
% route to epa 2 if Product is regular
routeTo(epa2, Product) :- regular(Product).

% a Product is luxury if the Product has a value over ...
luxury(Product) :- price(Product, Value), Value >= 10000.
% a Product is regular if the Product has a value below ...
regular(Product) :- price(Product, Value), Value < 10000.
Message Driven Routing
Prova : Event Routing in Event-Driven Workflows

```prolog
rcvMsg(XID, Process, From, event, ["A"]):-
    fork_b_c(XID, Process).

fork_b_c(XID, Process):-
    @group(p1) rcvMsg(XID, Process, From, event, ["B"]),
    execute(Task1), sendMsg(XID, self, 0, event, ["D"]).

fork_b_c(XID, Process):-
    @group(p1) rcvMsg(XID, Process, From, event, ["C"]),
    execute(Task2), sendMsg(XID, self, 0, event, ["E"]).

fork_b_c(XID, Process):-
    % OR reaction group "p1" waits for either of the two
    % event message handlers "B" or "C" and terminates the
    % alternative reaction if one arrives
    @or(p1) rcvMsg(XID, Process, From, or, _).
```

A diagram showing the workflow with the following tasks:
- Task1
- Task2
- D
- E
- Groups p1 and p2
Distributed Rule Base Interchange
Prova: Mobile Code

% Manager

upload_mobile_code(Remote, File) :
    Writer = java.io.StringWriter(), % Opening a file fopen(File, Reader),
    copy(Reader, Writer),
    Text = Writer.toString(),
    SB = StringBuffer(Text),
    sendMsg(XID, esb, Remote, eval, consult(SB)).

% Service (Contractor)

rcvMsg(XID, esb, Sender, eval, [Predicate|Args]):- derive([Predicate|Args]).
Adrian Paschke and Harold Boley: Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web, in Special Issue on Intelligent Distributed Computing in International Journal on Artificial Intelligence Tools (IJAIT), Vol. 20,6, 2011

Rule Responder: [http://responder.ruleml.org](http://responder.ruleml.org)
Summary of Selected Reaction RuleML Features

- Different Rule Families and Types (DR, KR, PR, ECA, CEP)
- Support for Event/Action/Situation... **Reasoning and Processing**
- Different **algebras** (Event, Action, Temporal,...)
- Specialized Language Constructs
  - Intervals (Time, Event), situations (States, Fluents)
- **Rule Interface and Implementation for Distributed Knowledge**
- Knowledge Life Cycle Management, Modularization and Dynamic Views (metadata, scopes, guards, situations/fluents, update actions)
- Decoupled **event messaging and loosley-coupled send/receive interaction against rule (KB) interface within conversations, coordination/negotiation protocols and pragmatic directives**
- Different detection, selection and consumption semantics (**profiles**)
- Top Level **Meta Model** for semantic Type Definitions and predefined Types in **Top-Level Ontologies**
- External (event) query languages, external data and ontology models
- ...
Questions?

Acknowledgement to the members of the Reaction RuleML technical group

Acknowledgment to the Event Processing Technical Society Reference Architecture working group members

Acknowledgement to the members of the Corporate Semantic Web group at FU Berlin
RuleML Online Community

• RuleML MediaWiki (http://wiki.ruleml.org)

• Mailing lists (http://ruleml.org/mailman/listinfo)

• Technical Groups
  (http://wiki.ruleml.org/index.php/Organizational_Structure#Technical_Groups)
  – Uncertainty Reasoning
  – Defeasible Logic
  – Reaction Rules
  – Multi-Agent Systems
  – …

• RuleML sources are hosted on Github
  (https://github.com/RuleML)
Further Reading – Surveys and Tutorials

  http://www.igi-global.com/book/handbook-research-emerging-rule-based/465

• Adrian Paschke, Alexander Kozlenkov: Rule-Based Event Processing and Reaction Rules. RuleML 2009: 53-66
  http://link.springer.com/chapter/10.1007%2F978-3-642-04985-9_8

• Adrian Paschke, Paul Vincent, Florian Springer: Standards for Complex Event Processing and Reaction Rules. RuleML America 2011: 128-139
  http://link.springer.com/chapter/10.1007%2F978-3-642-24908-2_17


• Jon Riecke, Opher Etzion, François Bry, Michael Eckert, Adrian Paschke, Event Processing Languages, Tutorial at 3rd ACM International Conference on Distributed Event-Based Systems. July 6-9, 2009 - Nashville, TN
  http://www.slideshare.net/opher.etzion/debs2009-event-processing-languages-tutorial

  http://www.igi-global.com/chapter/rule-markup-languages-semantic-web/35852
Further Reading – RuleML and Reaction RuleML

- Adrian Paschke: Reaction RuleML 1.0 for Rules, Events and Actions in Semantic Complex Event Processing, Proceedings of the 8th International Web Rule Symposium (RuleML 2014), Springer LNCS, Prague, Czech Republic, August, 18-20, 2014

  http://dx.doi.org/10.1007/978-3-642-16289-3_15

- Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian and Tara Athan: Reaction RuleML 1.0: Standardized Semantic Reaction Rules, 6th International Conference on Rules (RuleML 2012), Montpellier, France, August 27-31, 2012
  http://link.springer.com/chapter/10.1007%2F978-3-642-32689-9_9
  http://www.slideshare.net/swadpasc/reaction-ruleml-ruleml2012paschketutorial

  http://www.igi-global.com/chapter/rule-markup-languages-semantic-web/35852

  http://www.igi-global.com/book/handbook-research-emerging-rule-based/465

- Adrian Paschke and Harold Boley: Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web, in Special Issue on Intelligent Distributed Computing in International Journal on Artificial Intelligence Tools (IJAIT), Vol. 20,6, 2011
Further Reading – EPTS Event Processing Reference Architecture and Event Processing Patterns

• Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Advanced design patterns in event processing. DEBS 2012: 324-334; http://www.slideshare.net/isvana/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b


Further Reading – Rule-Based Semantic CEP

- Corporate Semantic Web – Semantic Complex Event Processing


- Adrian Paschke and Harold Boley: Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web, in Special Issue on Intelligent Distributed Computing in International Journal on Artificial Intelligence Tools (IJAIT), V0l. 20,6, 2011

- Kia Teymourian, Adrian Paschke: Towards semantic event processing. DEBS 2009
Further Reading – RuleML Test Cases

• Adrian Paschke: Verification, Validation, Integrity of Rule Based Policies and Contracts in the Semantic Web, 2nd International Semantic Web Policy Workshop (SWPW’06), Nov. 5-9, 2006, Athens, GA, USA

  http://km.aifb.kit.edu/ws/swese2006/final/paschke_full.pdf

• Adrian Paschke and Jens Dietrich: On the Test-Driven Development and Validation of Business Rules. ISTA 2005: 31-48
  http://subs.emis.de/LNI/Proceedings/Proceedings63/article3604.html

• Adrian Paschke: The ContractLog Approach Towards Test-driven Verification and Validation of Rule Bases - A Homogeneous Integration of Test Cases and Integrity Constraints into Evolving Logic Programs and Rule Markup Languages (RuleML), in IBIS 10/2005
  http://rbsla.ruleml.org/docs/ContractLog_VVI.pdf
Further Reading – Rules and Logic Programming, Prova

- **Adrian Paschke: Rules and Logic Programming for the Web.** A. Polleres et al. (Eds.): Reasoning Web 2011, LNCS 6848, pp. 326–381, 2011. [http://dx.doi.org/10.1007/978-3-642-23032-5_6](http://dx.doi.org/10.1007/978-3-642-23032-5_6)

- **Prova Rule Engine [http://www.prova.ws/](http://www.prova.ws/)**

- **Prova 3 Semantic Web Branch**
  - Prova 3 version with Semantic Web support on GitHub ([https://github.com/prova/prova/tree/prova3-sw](https://github.com/prova/prova/tree/prova3-sw))


- **Prova CEP examples:** [http://www.slideshare.net/isvana/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b](http://www.slideshare.net/isvana/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b)