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/

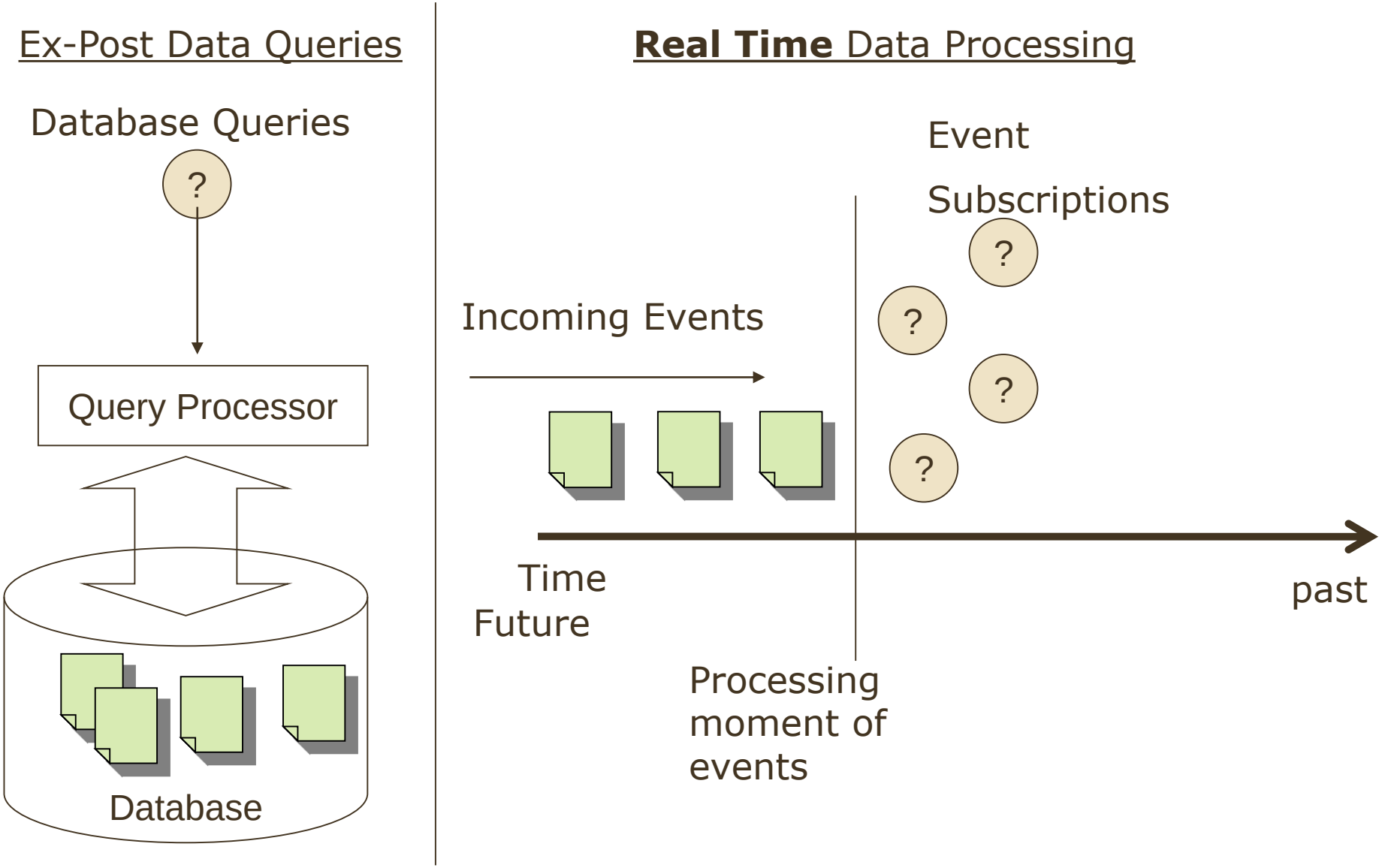
Agenda

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

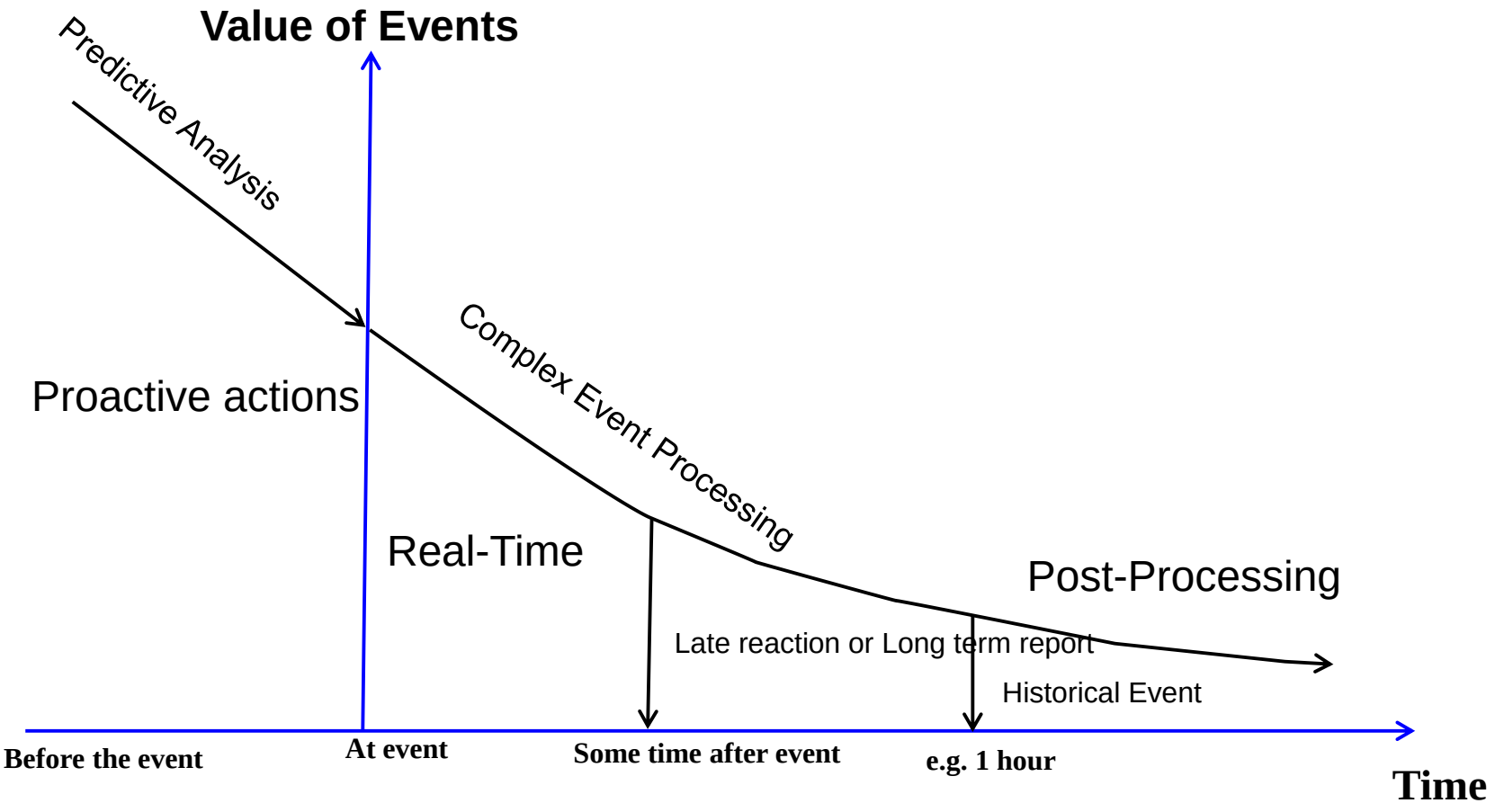
Agenda

- **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



Knowledge Value of Events

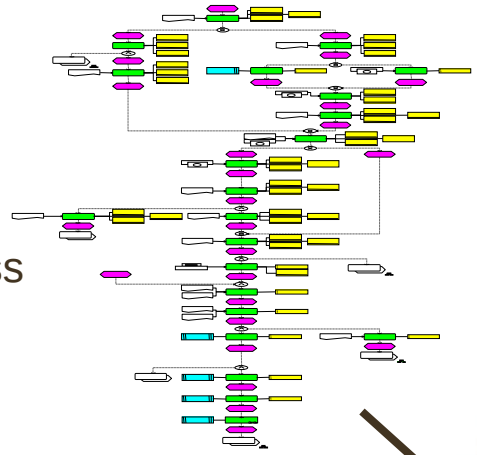


CEP – Why do we need it?

Example Application Domains

RTE

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



Monitoring, BAM, ITSM,

Monitor and detect exceptional IT service and business behavior from occurred events

Valuable Information at the Right Time to the Right Recipient

Information Dissemination

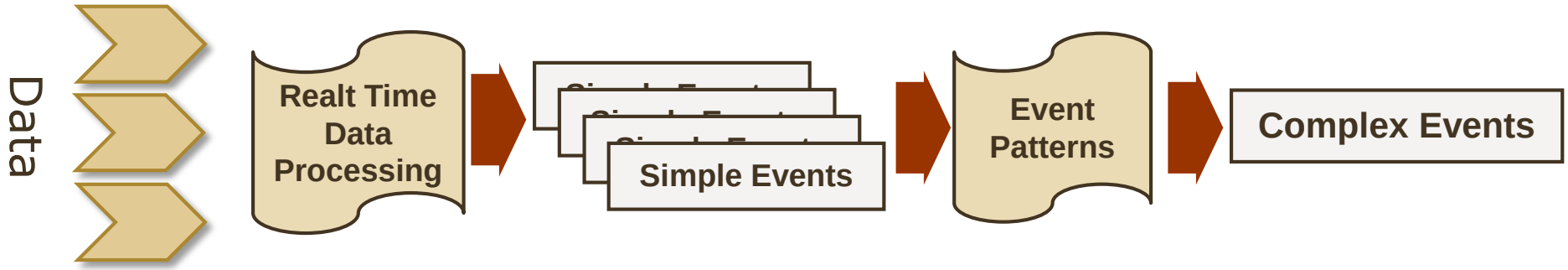
Enterprise Decision Management

Expert Systems

Expert Decision Management

Complex Events – What are they?

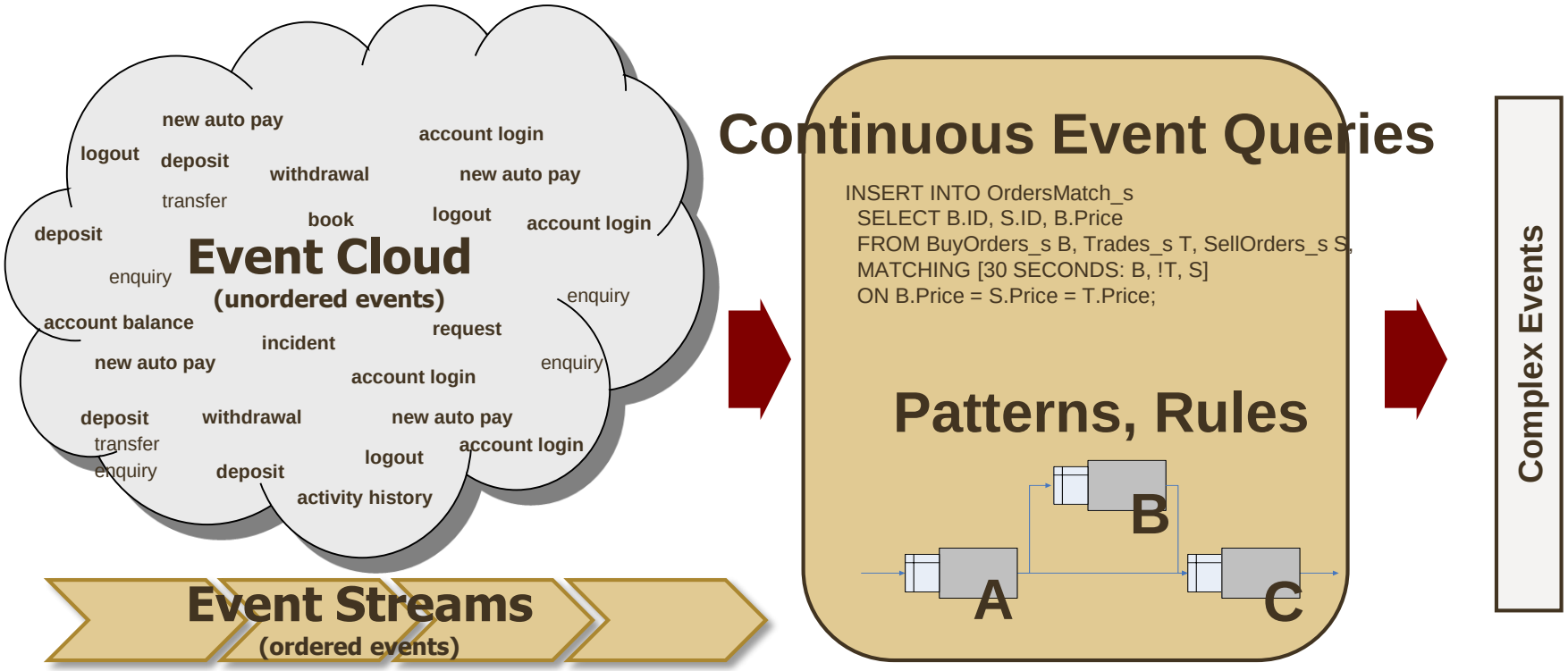
- **Complex Events are aggregates, derivations, etc. of Simple Events**



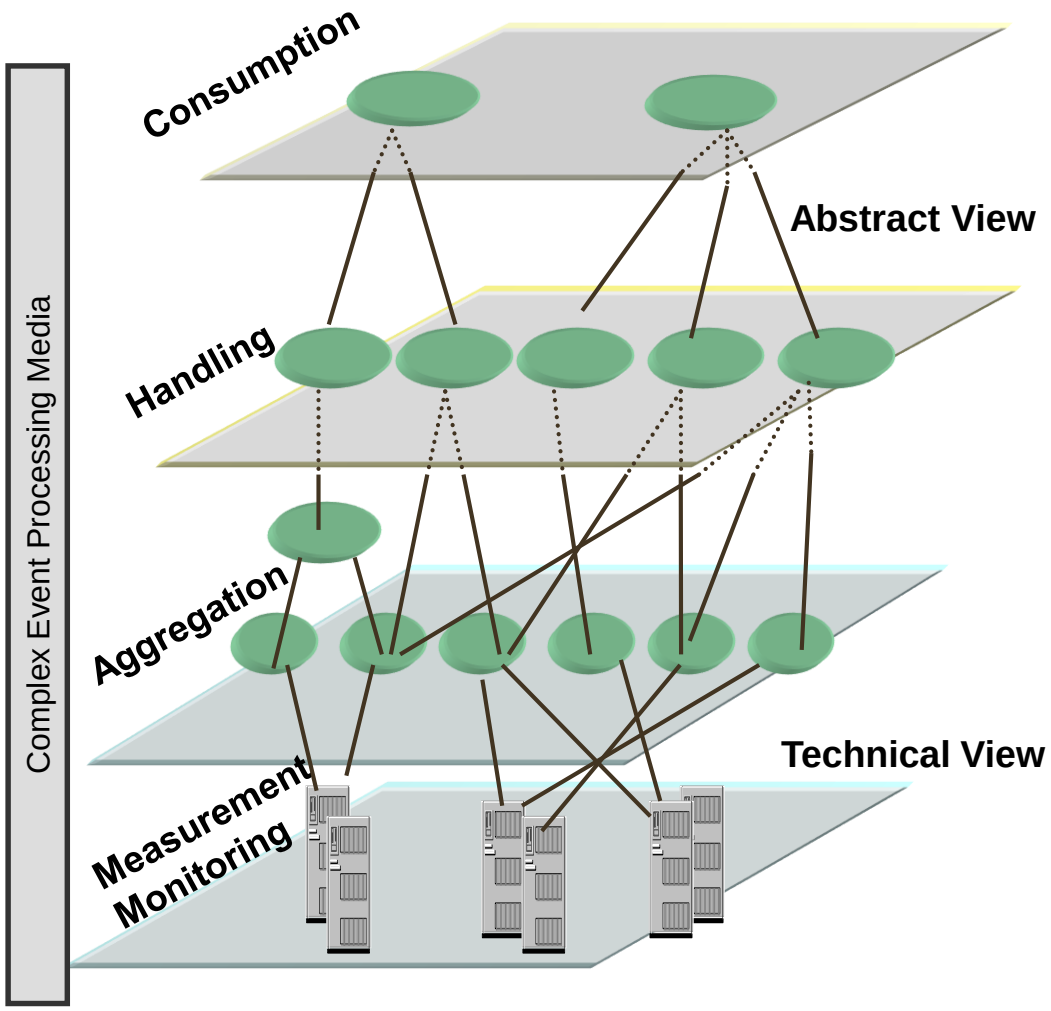
- Complex Event Processing (CEP) will enable, e.g.
  - **Detection** of state changes based on observations
  - **Prediction** of future states based on past behaviors

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**



Complex Event Processing – What is it?

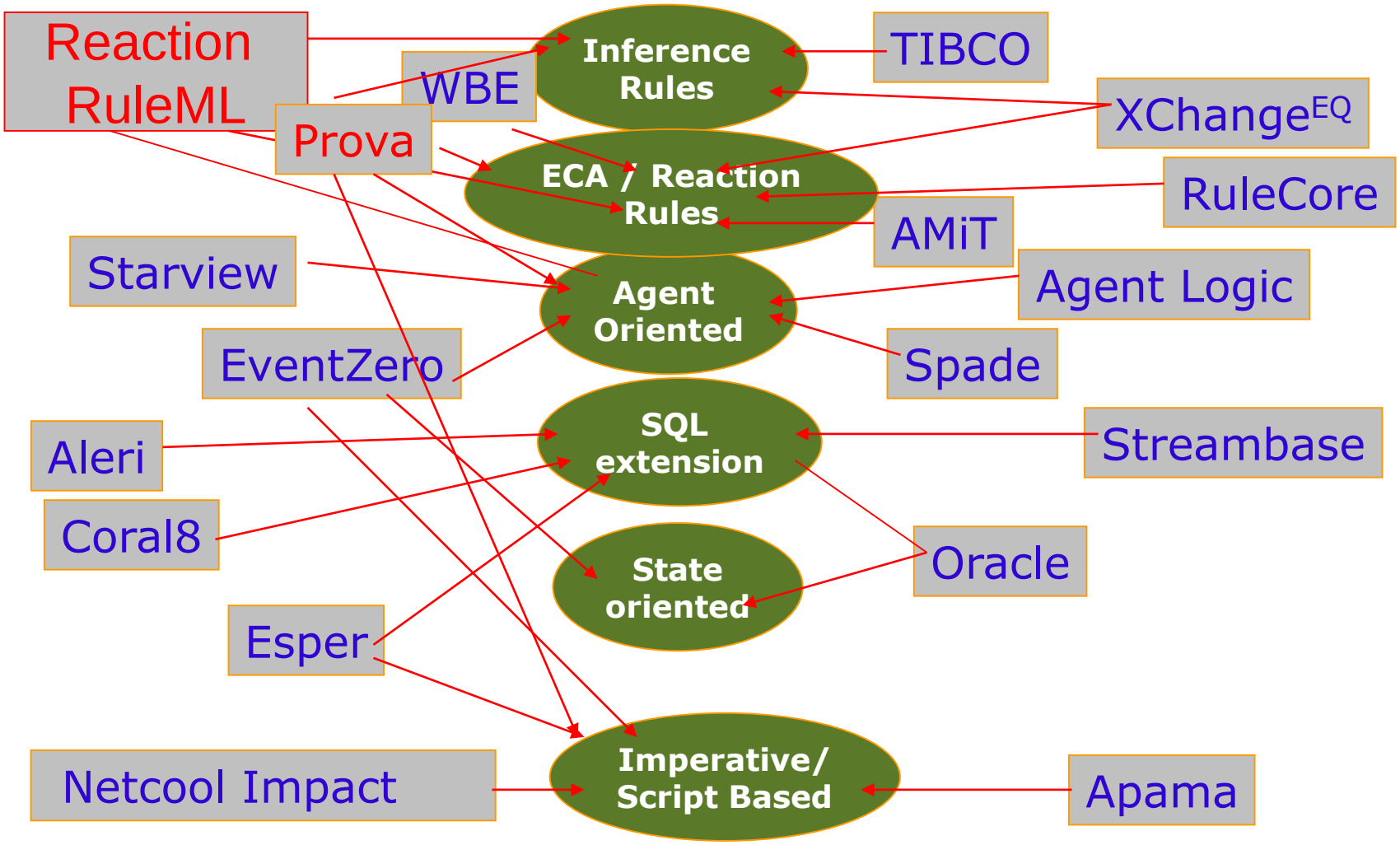


- Complex Event Processing (CEP) is a **discipline** that deals with event-driven behavior
- Selection, aggregation, and event abstraction for generating higher level complex events of interest

The Many Roots of CEP…

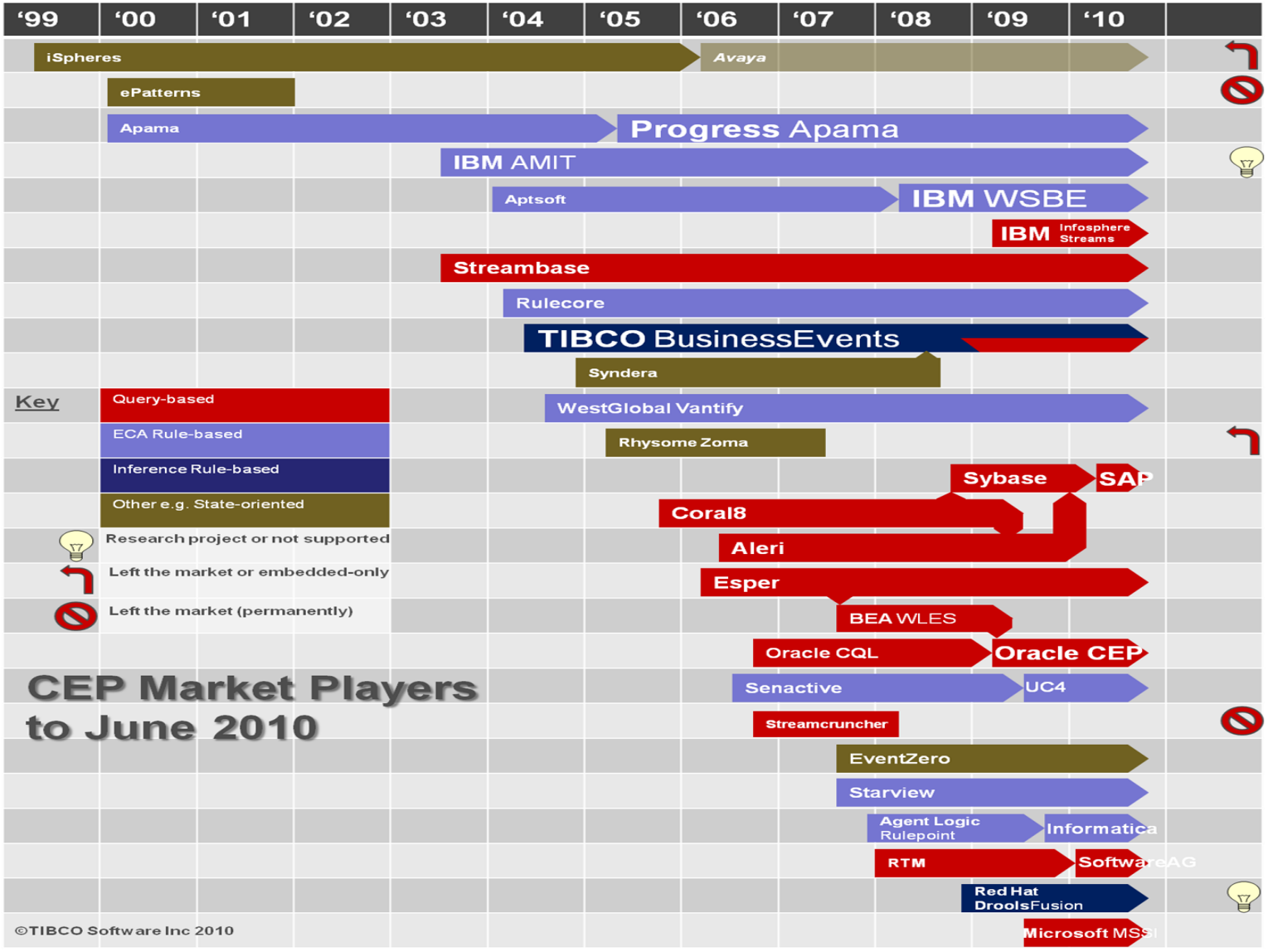
Complex Event Processing (CEP) is a **discipline** that deals with event-driven behavior

Example Event Processing Languages



see DEBS 2009 EPTS Language Tutorial - http://www.slideshare.net/opher.etzion/debs2009-event-processing-languages-tutorial

Commercial CEP Market



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 (;):** *(E1;E2)*
- **Disjunction Operator (∨):** *(E1 ∨ E2)* , at least one
- **Conjunction Operator (∧):** (E1 ∧ E2)
- **Simultaneous Operator (=):** *(E1 = E2)*
- **Negation Operator (¬):** *(E1 ∧ ¬ E2)*
- **Quantification (Any):** Any(n) E1, when n events of type E1 occurs
- **Aperiodic Operator (Ap):** *Ap(E2, E1, E3)* , E2 Within E1 & E3
- **Periodic Operator (Per):** Per(t, E1, E2) , every t time-steps in between E1 and E2

Adrian Paschke: ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics CoRR abs/cs/0610167: (2006); http://arxiv.org/ftp/cs/papers/0610/0610167.pdf

Core CEP Life Cycle

Basic Definitions* (1)

* Paschke, A., Boley, H.: Rules Capturing Event and Reactivity, in Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, ISBN:1-60566-402-2, 2009
http://www.igi-global.com/book/handbook-research-emerging-rule-based/465
http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=16435934&article=0&fd=pdf

- **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)

* Reaction RuleML Classification of the Event/Action/State Porcessing and Reasoning Space http://arxiv.org/ftp/cs/papers/0611/0611047.pdf

- **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. in 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

Type	Point – Period	Change	Time Relation
State Description	Relating to point or period	no: –dyn	homogeneous: The state holds for all interior interval and points
Process Description	Relating to a period, never relating to a point	continuous: +dyn	restricted homogeneous: In certain interior intervals the process also takes place
Event-based Description	Atomic point or complex interval	one: +dyn	heterogeneous: In interior intervals the event doe not occur

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 preformed 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

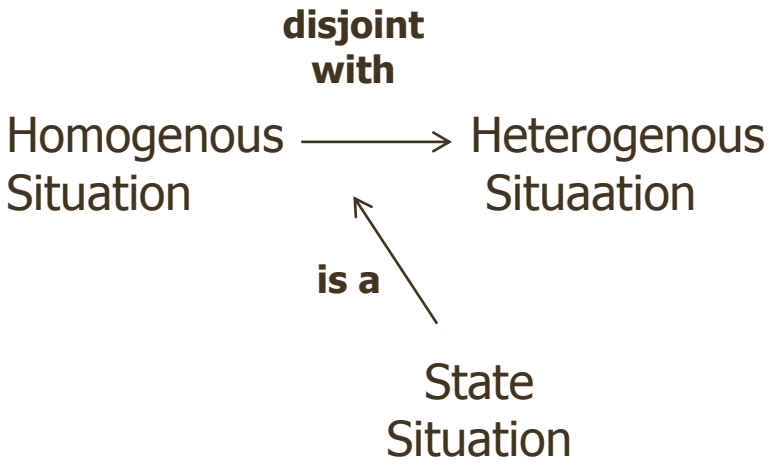
1. Rules

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

```
takeoff(X) ➔ fly(X)
takoff(flight123)
```

2. Ontologies

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



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**

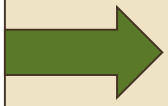
Examples of Ontologies which include Events

- **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**
- **…**

Kia Teymourian, Gökhan Coskun, Adrian Paschke: Modular Upper-Level Ontologies for Semantic Complex Event Processing. WoMO 2010: 81-93

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.

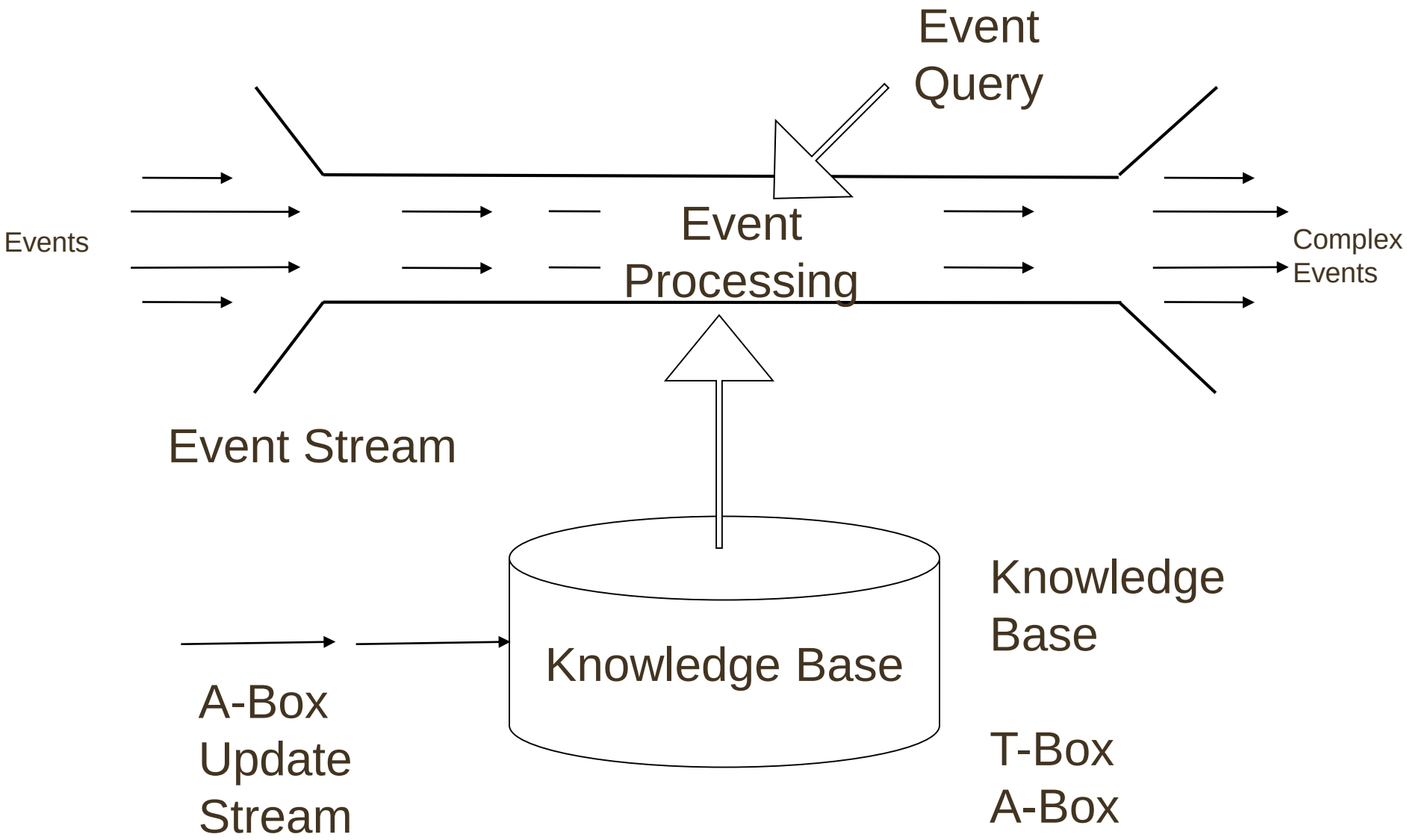


Event Stream – stock quotes
{(Name, "*OPEL*")(Price, 45)(Volume, 2000)(Time, 1) }
{(Name, "SAP")(Price, 65)(Volume, 1000) (Time, 2)}

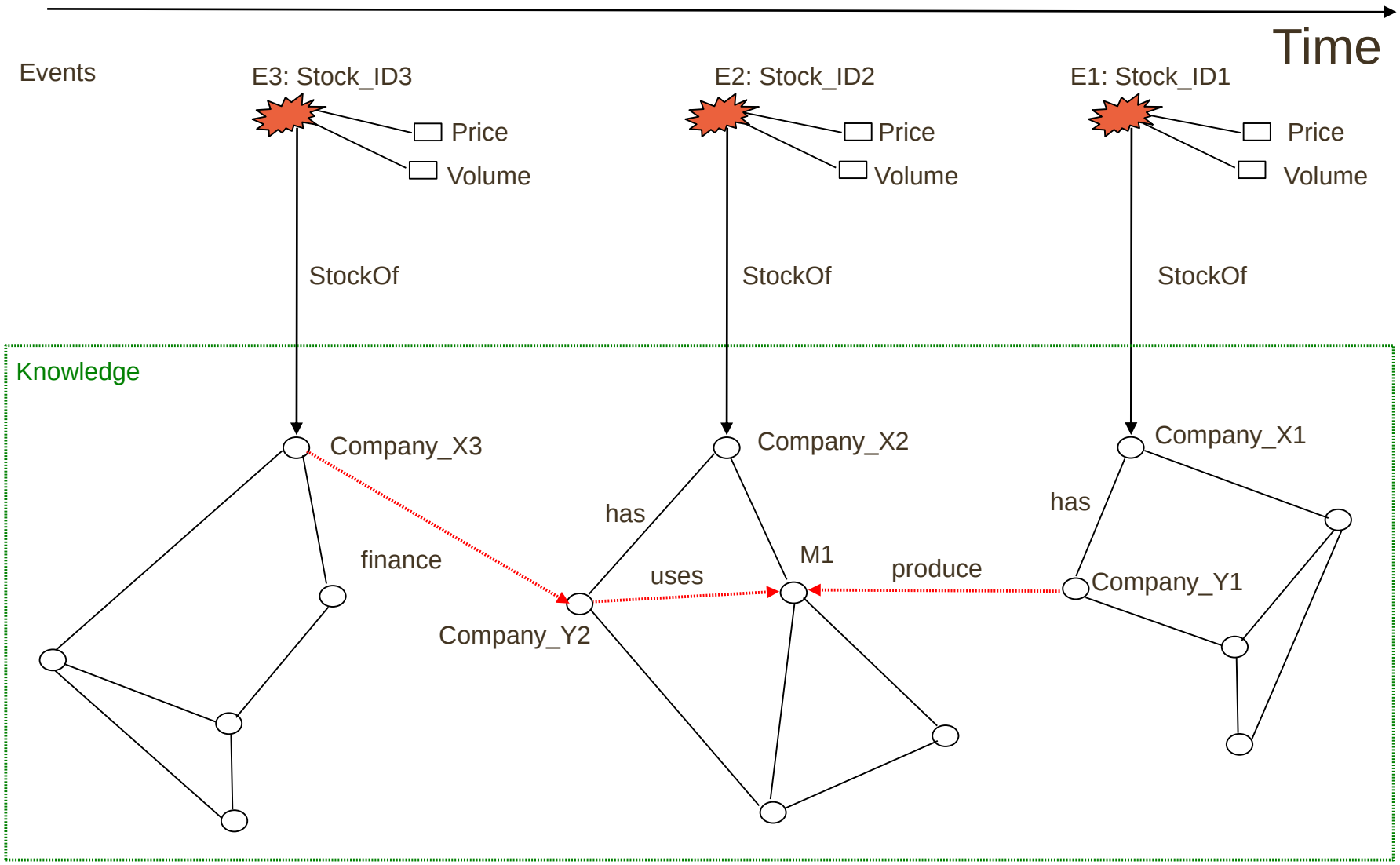
Semantic Knowledge Base

{(*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



Example of Semantic Event Queries



Semantic Technologies for Declarative Knowledge Representation

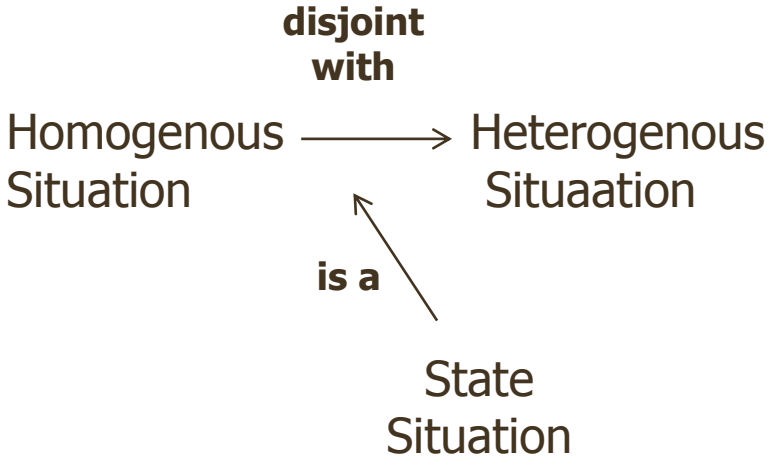
1. Rules

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

```
takeoff(X) ➔ fly(X)
takoff(flight123)
```

2. Ontologies

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



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).
   - **Re*action 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*

Reaction Rules* (1)

* Adrian Paschke, Alexander Kozlenkov: **Rule-Based Event Processing and Reaction Rules**. RuleML 2009: 53-66

- **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

Reaction Rules* (2)

- **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.

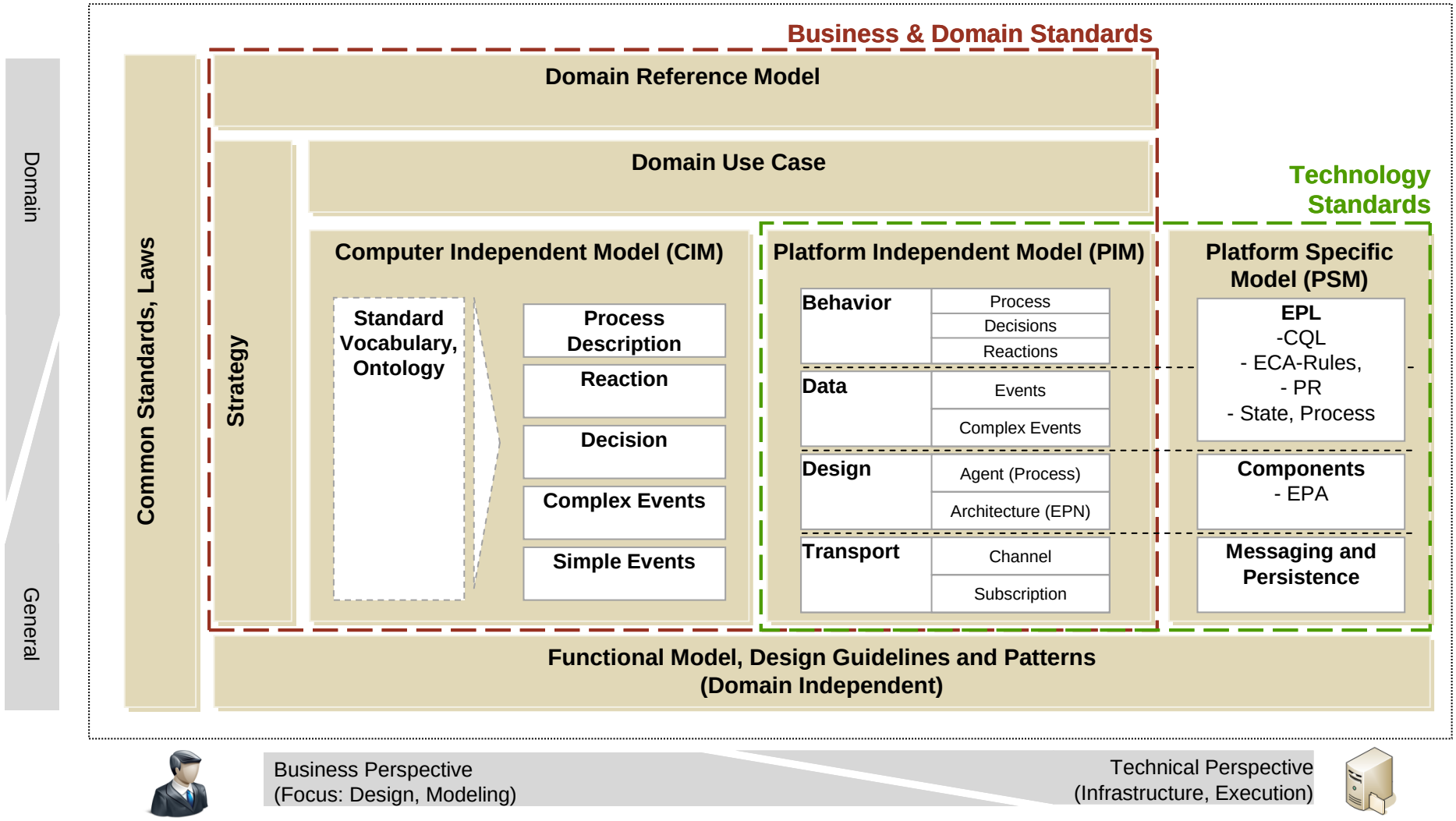
- **Reference Model**

  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*

EPTS CEP /Reaction RuleML Standards Reference Model



see.: Adrian Paschke, Paul Vincent, Florian Springer: Standards for Complex Event Processing and Reaction Rules. RuleML America 2011: 128-139; http://www.slideshare.net/isvana/ruleml2011-cep-standards-reference-model

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

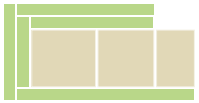
- **Platform Independent Model Standards (PIM)**

- **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

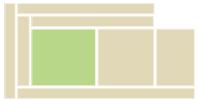
General Frameworks and Guidelines for Developing Systems



	Description	Example	Existing Standards (+Gaps)	Benefits of Standardization
Common Standards, Laws	■ Industry standards, laws, etc.	■ Several laws (often country specific), e.g. BörsG for stock trading in Germany	■ A lot of existing standards are available ■ CEP Specific standards not necessary	■ Regulation of Use Cases ■ No Benefit for CEP
Strategy UseCase	■ Overlying business motivations, goals, strategies, policies and business rules	■ Reduction of fraud by 10 percent, …	■ OMG BMM ■ May need more emphasis on events and decision rules	■ Clarify the Business strategy
Reference Model	■ Collection of best practices and reference business processes for a specific domain	■ Reference End-to-End process for logistics, e.g. *Pickup to Delivery*	■ Various per domain for data ■ Rarely handles events and rules	■ Helps in developing an efficient application
Use Case	■ Description of how to solve a problem	■ Fraud detection for stock trading	■ UML (not CEP or rules specific) ■ EPTS UseCase Template	■ Create a common understanding between business user and software developer
Functional Model, Design Patterns ..	■ Domain independent description of best practices, guidelines, etc.	■ Design Patterns for software engineering (Gamma)	■ Lots of different ones! ■ EPTS Functional Ref Architecture	■ Helps in implementing a proper application

COMPUTER INDEPENDENT MODEL 1/2

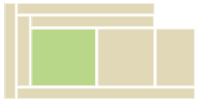
Computer Independent Model / Business Model for designing CEP Systems and Processes



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

COMPUTER INDEPENDENT MODEL 2/2

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



	Description	Example	Existing Standards (+Gaps)	Benefits of Standardization
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
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

PLATFORM INDEPENDENT MODEL 1/4

Platform Independent Model / IT Model for design of CEP Systems

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardization
Behavior	■ Event-Driven Behavior and Decisions: ■ Event occurences 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	■ 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	■ 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 expressivness of events and events = actions/activities ■ OMG BPEL – specific to business process execution with WS ■ W3C WS Choreoagraphy – Specific to WS ■ and further EDA standards	■ Declarative, explicit representation ■ Publication and interchange of decisions and reactive behavior ■ ...

PLATFORM INDEPENDENT MODEL 2/4

Platform Independent Model / IT Model for design of CEP Systems

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardization
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 +analysed for machine learning, BI, etc	■ 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	■ 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 ■ Standards for other domains needed, e.g. stream processing	■ 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 ■ Prerequisits Semantics: Explicit domain semantics Expressiveness Extensibility Reusability, specialisation and mapping

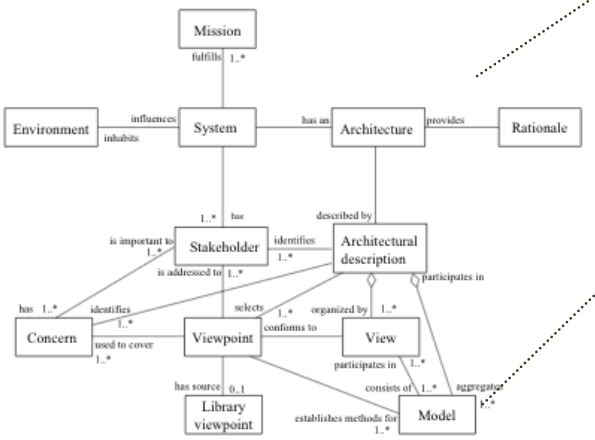
Some of ontologies which include events

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 3/4

Platform Independent Model / IT Model for design of CEP Systems

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardization
Design	■ Platfrom-Independent (Reference) Architecture and Model Descriptions addressing different views for different stakeholders	■ 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	■ 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 expressivness – 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	■ 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



PLATFORM INDEPENDENT MODEL 4/4

Platform Independent Model / IT Model for design of CEP Systems

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardization
Messaging	■ Transport protocols and routing, coordination / negotiation mechanisms	■ synchronous vs. asynchronous transport ■ Push vs. pul ■ Coordination: Publish Subscribe, ContractNet, … ■ …	■ Many transport protocols: JMS, JDBC, TCP, UDP, multicast, http, servlet, SMTP, POP3, file, XMPP – just needed for transport ■ Message Routing Patterns – message routing ■ Coordination and Negotiation Mechanisms/Patterns	■ well understood and standardized transport protocols

PLATFORM SPECIFIC MODEL 1/3

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

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardisation
EPL CQL	■ Continuous query language running queries against policies such as time windows, numbers of events, etc	■ SELECT x, y from REUTERS WHERE x.id == y.id AND x.price > y.price * 1.1 IN WINDOW(600,“secs“) ■ Vendor specific versions include Oracle CQL, Streambase CQL, Sybase CQL, TIBCO BQL	■ ANSI SQL extended for continuous policy ■ ODBMS OQL extended for continuous policy	■ Interchange queries between vendor platforms (design, deployment) [IT department] ■ Encourage training / education on CQL (cf SQL takeup) [university, student] ■ Dependancies: common data descriptors
EPL ECA Rules	■ Event-Condition-Action rules triggered by creation of complex events representing event patterns (eg stream outputs), determing reactions based on certain conditions	■ IF event. SigMvDet AND SigMvDet.stock = IBM AND SigMvDet.change = up THEN reqBuy(SigMvDet.stock) ■ Vendor examples include TIBCO BE, IBM WSBE, Progress Apama ■ Open Source, e.g. Prova, Drools	■ W3C RIF PRD extended for ECA (RIF ECA) ■ Reaction RuleML ■ JSR-94 rule execution API	■ Interchange rules between vendor platforms (design, deployment) [IT department] ■ Encourage training / education on ECA rules (cf Java takeup) [university, student] ■ Dependancies: common data descriptors

PLATFORM SPECIFIC MODEL 2/3

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

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardisation
EPL TRE	■ Temporal Regular Expressions defining complex events in terms of event / non-event sequences over time	■ WHEN event. A, event.B WHERE event.A (*2) FOLLOWED BY no event.B IN time(200,ms) ■ Vendor examples include Oracle CQL (TRE embedded in SQL), Progress Apama (TRE), TIBCO PML	■ Perl Regex with temporal extensions	■ Interchange rules between vendor platforms (design, deployment) [IT department] ■ Encourage training / education on TRE patterns (cf Java takeup) [university, student] ■ Dependancies: common data descriptors
Application Architecture	■ 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	■ Agent 1 type inference contains rulesets R1, R2 with destinations D1, D2 Deployment C1 is of 4 x Agent 1 role active-active ■ Vendor examples include TIBCO CDD Cluster Deployment Descriptor	■ OMG AMP Agent Metamodel and Profile (draft)	■ Compare application designs across vendors [IT operations, Enterprise Architects] ■ Dependancies: common agent and interface descriptors

PLATFORM SPECIFIC MODEL 3/3

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

	Description	Example	Existing Standards (+Gaps)	Benefits of Standardisation
Messaging System	■ Middleware type and configuration	■ 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.	■ JMS (Java spec) ■ OMG DDS ■ AMQP ■ Gap: some abstract definition of a messaging software per the above	■ Late selection of messaging type [IT, IT operations] ■ Dependancies: common payload descriptors

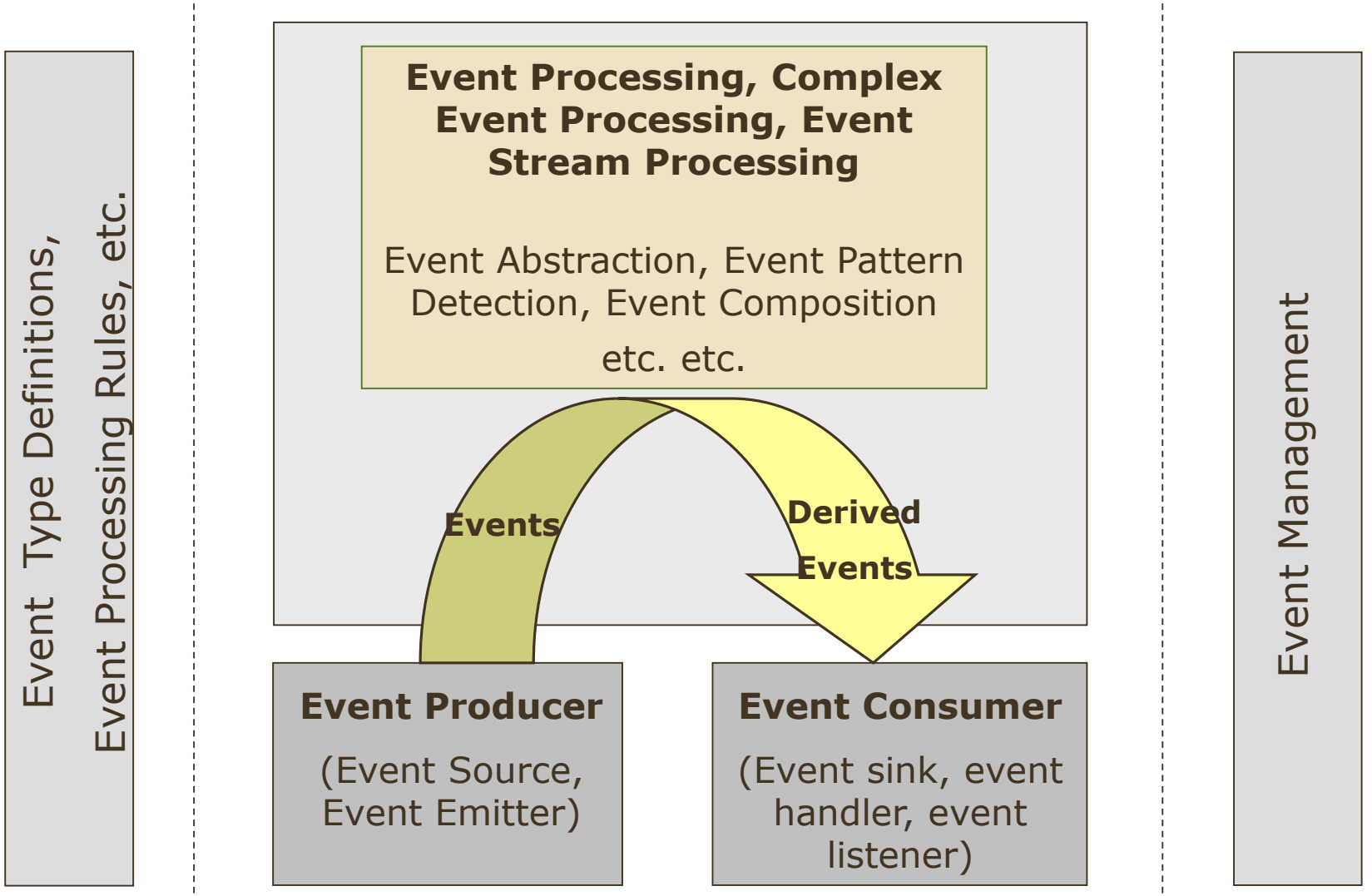
Reference Architecture and Reference Model

- **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/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b
- **Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Architectural and functional design patterns for event processing. DEBS 2011: 363-364**
  http://www.slideshare.net/isvana/epts-debs2011-event-processing-reference-architecture-and-patterns-tutorial-v1-2
- **Paschke, A., Vincent, P., and Catherine Moxey: Event Processing Architectures, *Fourth ACM International Conference on Distributed Event-Based Systems* (DEBS '10). ACM, Cambridge, UK, 2010.**
  http://www.slideshare.net/isvana/debs2010-tutorial-on-epts-reference-architecture-v11c
- **Paschke, A. and Vincent, P.: A reference architecture for Event Processing. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (DEBS '09). ACM, New York, NY, USA, 2009.**
- **Francois Bry, Michael Eckert, Opher Etzion, Adrian Paschke, Jon Ricke: Event Processing Language Tutorial, DEBS 2009, ACM, Nashville, 2009**
  http://www.slideshare.net/opher.etzion/debs2009-event-processing-languages-tutorial
- **Adrian Paschke. A Semantic Design Pattern Language for Complex Event Processing, AAAI Spring Symposium "Intelligent Event Processing", March 2009, Stanford, USA**
  http://www.aaai.org/Papers/Symposia/Spring/2009/SS-09-05/SS09-05-010.pdf
- **Paschke, A.: Design Patterns for Complex Event Processing, 2nd International Conference on Distributed Event-Based Systems (DEBS'08), Rome, Italy, 2008.**
  http://arxiv.org/abs/0806.1100v1

EPTS Reference Architecture
Architectural Descriptions



ANSI/IEEE Std 1471 :: ISO/IEC 42010 Methodology

- ***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

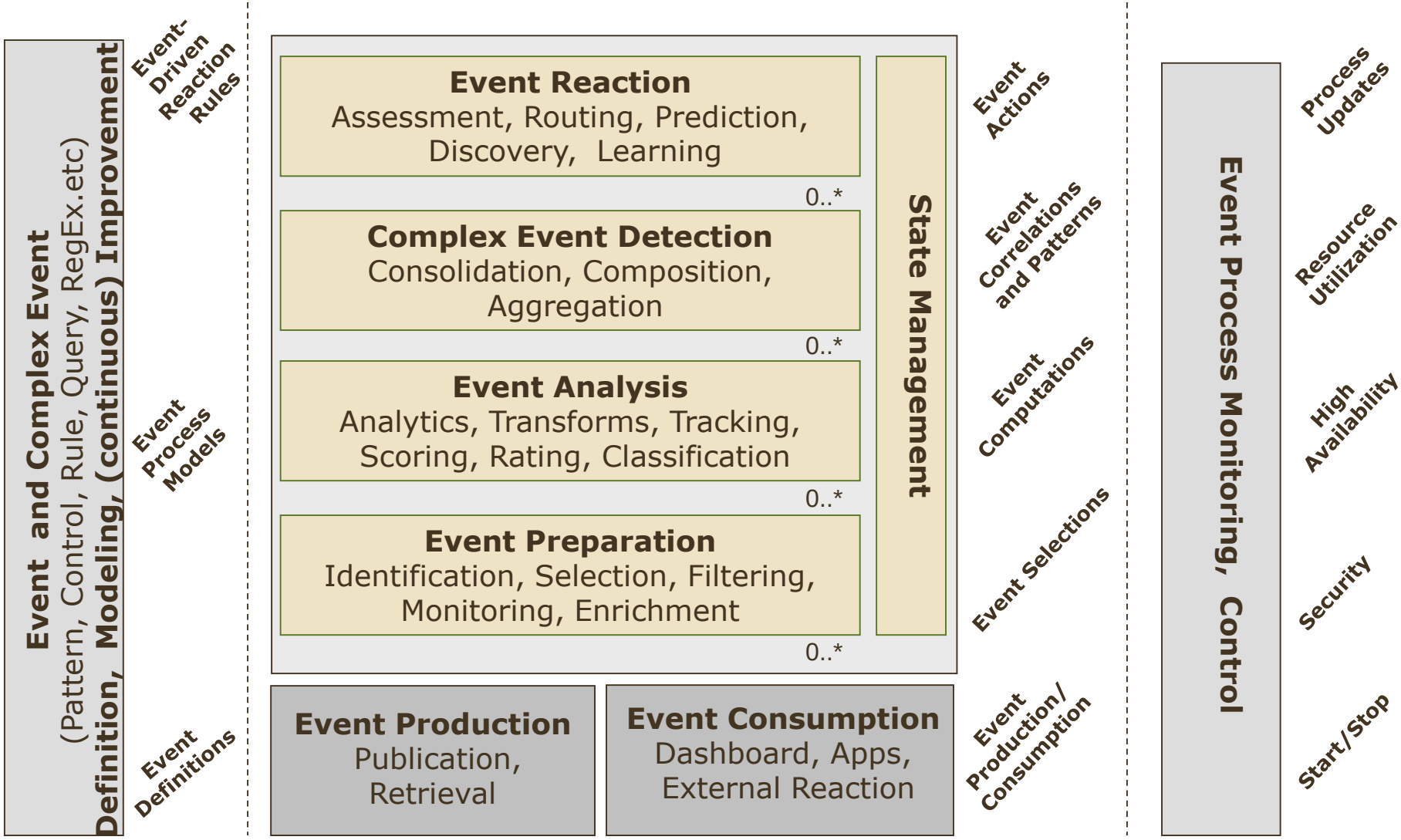
Reference Architecture Viewpoints

Viewpoint Element	**Viewpoint**		
	Engineering EP Architecture	*Managing EP Architecture*	*Business with EP Architecture*
Concepts	How to implement?	How to apply?	How to utilize / sell / own?
Stakeholders	Architects / Engineers	Project Manager	Decision Maker, Customer, Provider
Concerns	Effective construction and deployment	Operational Management	Strategic and tactical management
Techniques / Languages	Modeling, Engineering	IT (service/appl) management, project management	Monitoring, Enterprise Decision Management, Governance

Example Stakeholders and Viewpoints

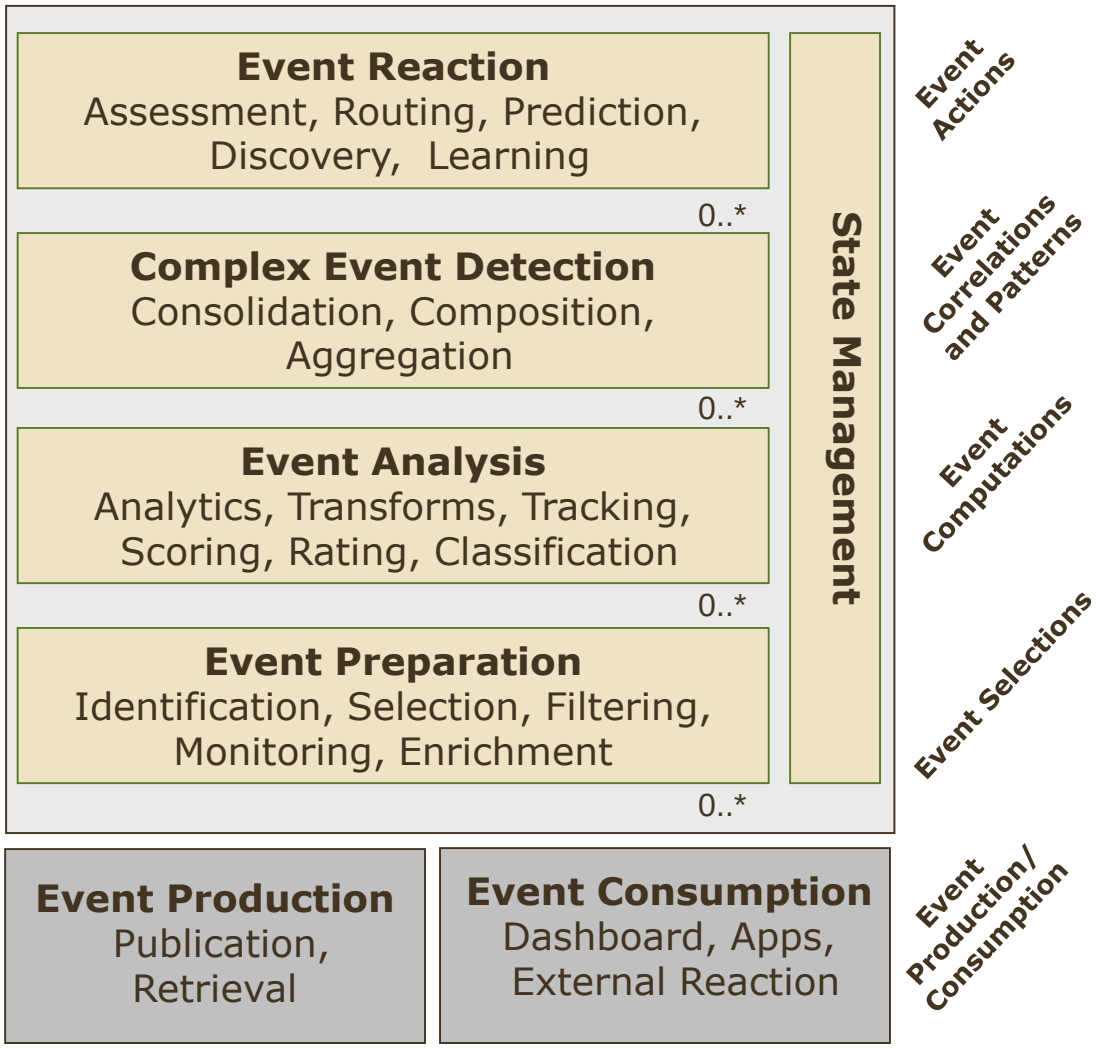
	Viewpoints (for CEP / stateful EP)	
Stakeholder	***Component***	***Functional***
Decision Maker / End User	[Not applicable]	Inputs, Outputs and Processing Requirements View
Architect / Designer	Solution Components View	Functions carried out in CEP View

Reference Architecture: Functional View



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



Run time

Reference Architecture: Functional View / Runtime

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 (eg 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.

Reference Architecture: Functional View / Runtime

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.

Reference Architecture: Functional View / Runtime

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.

Event Preparation
Identification, Selection, Filtering, Monitoring, Enrichment

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.

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

Reference Architecture: Functional View / Runtime

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.

Complex Event Detection
Consolidation, Composition, Aggregation

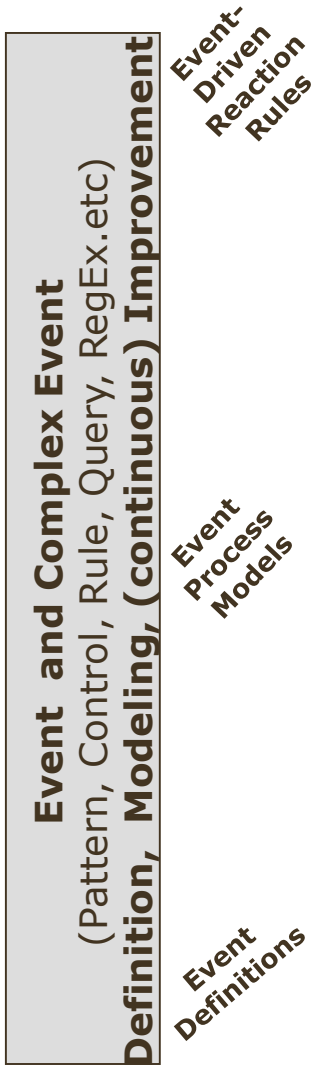
Reference Architecture: Functional View / Runtime

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.

Event Reaction
Assessment, Routing, Prediction, Discovery, Learning

Reference Architecture: Functional View / Design time



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

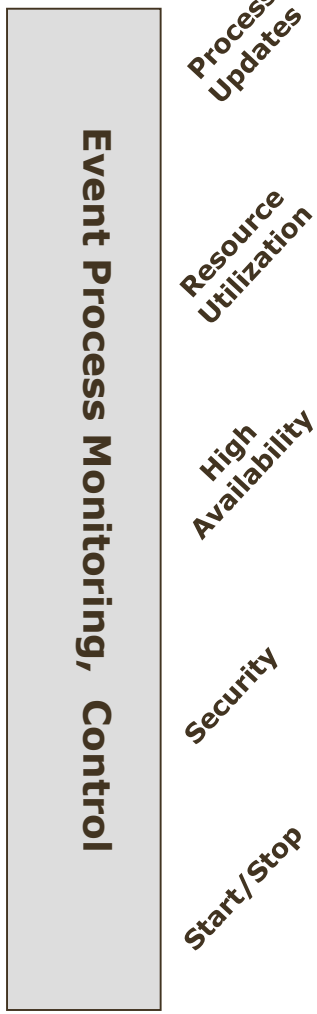
- event definitions, including event metadata and payloads,
- event and event object organisations and structures,
- event processing transformations / queries / rules / procedures / flows / states / decisions / expressions (although these can sometimes be considered as administrative updates in some situations)

Design time

Reference Architecture: Functional View / Administration

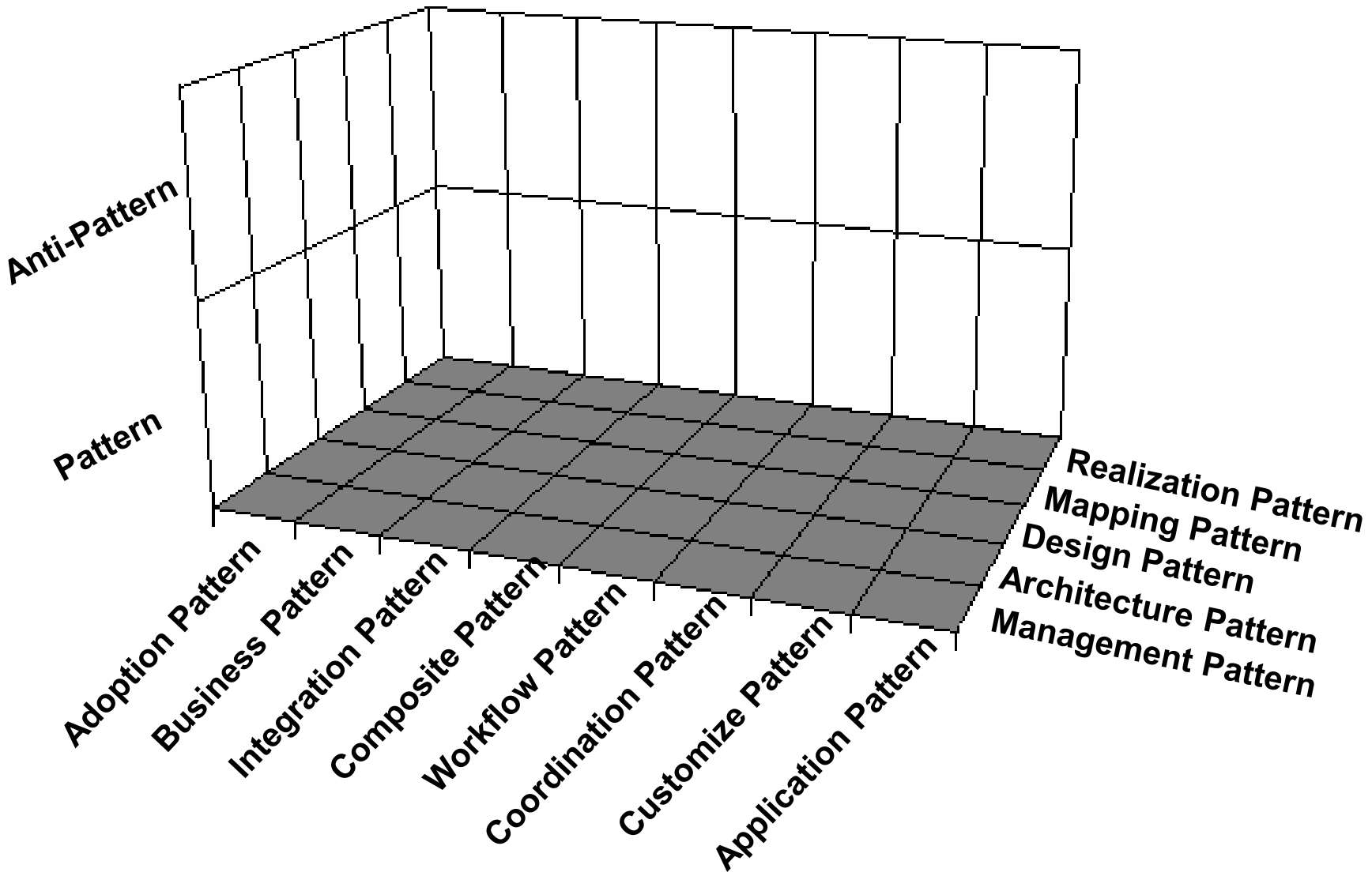
Administrative concepts of
monitoring and control. This may involve

- 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.



Administration

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

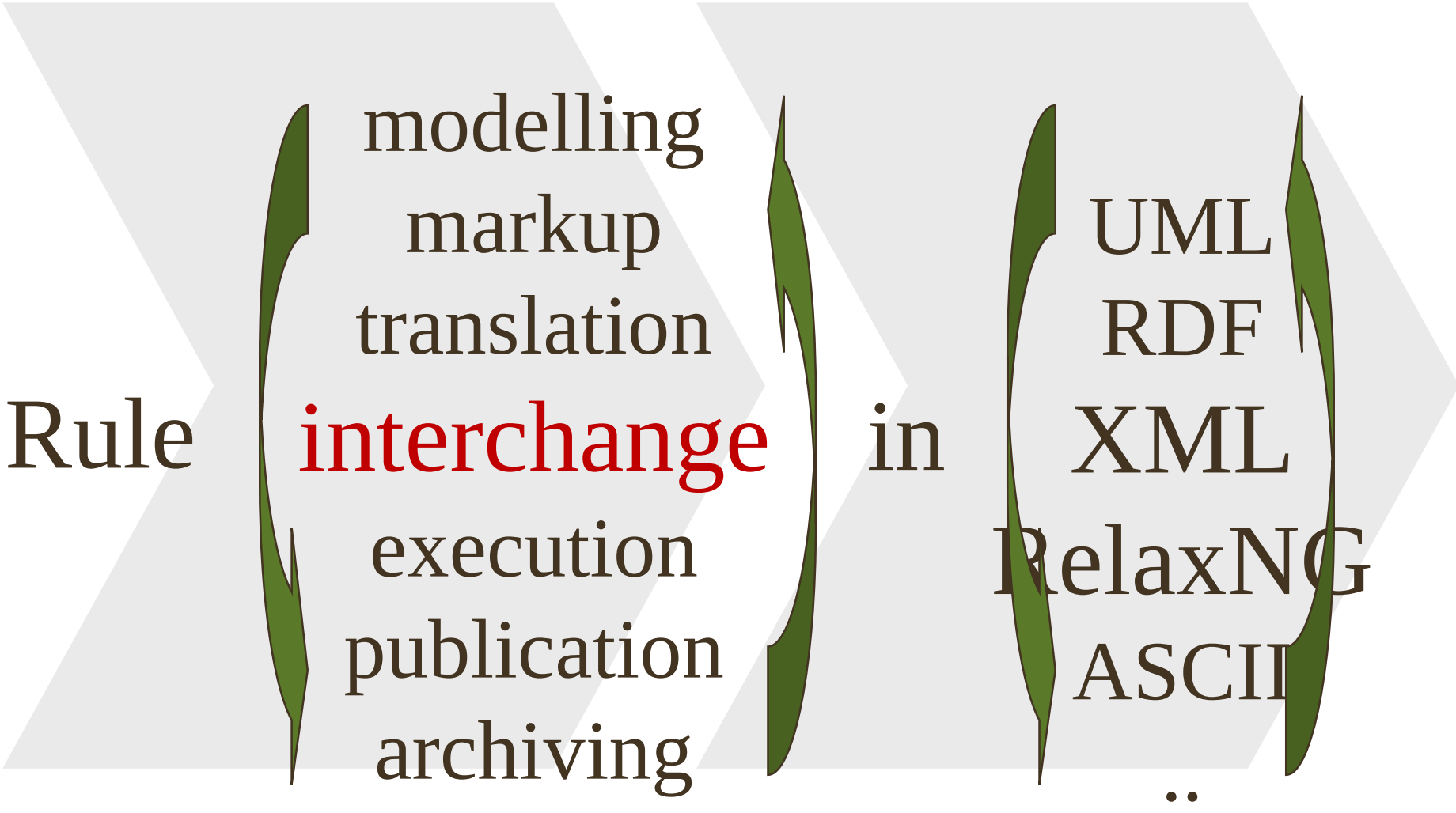
- Categorization according to Good and Bad Solutions
  - *CEP Patterns*
  - *CEP Anti-Patterns*
- Categorization according to the Abstraction Level
  - *Guidelines and Best Practices*
  - *Management patterns*
  - *Architecture patterns*
  - *Design patterns*
  - *Mapping patterns*
  - *Idioms / Realization 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*

Source:Paschke, A.: **A Semantic Design Pattern Language for Complex Event Processing, *Intelligent Event Processing, Papers from the 2009 AAAI Spring Symposium*** **(2009) , p. 54-60.**
http://www.aaai.org/Papers/Symposia/Spring/2009/SS-09-05/SS09-05-010.pdf

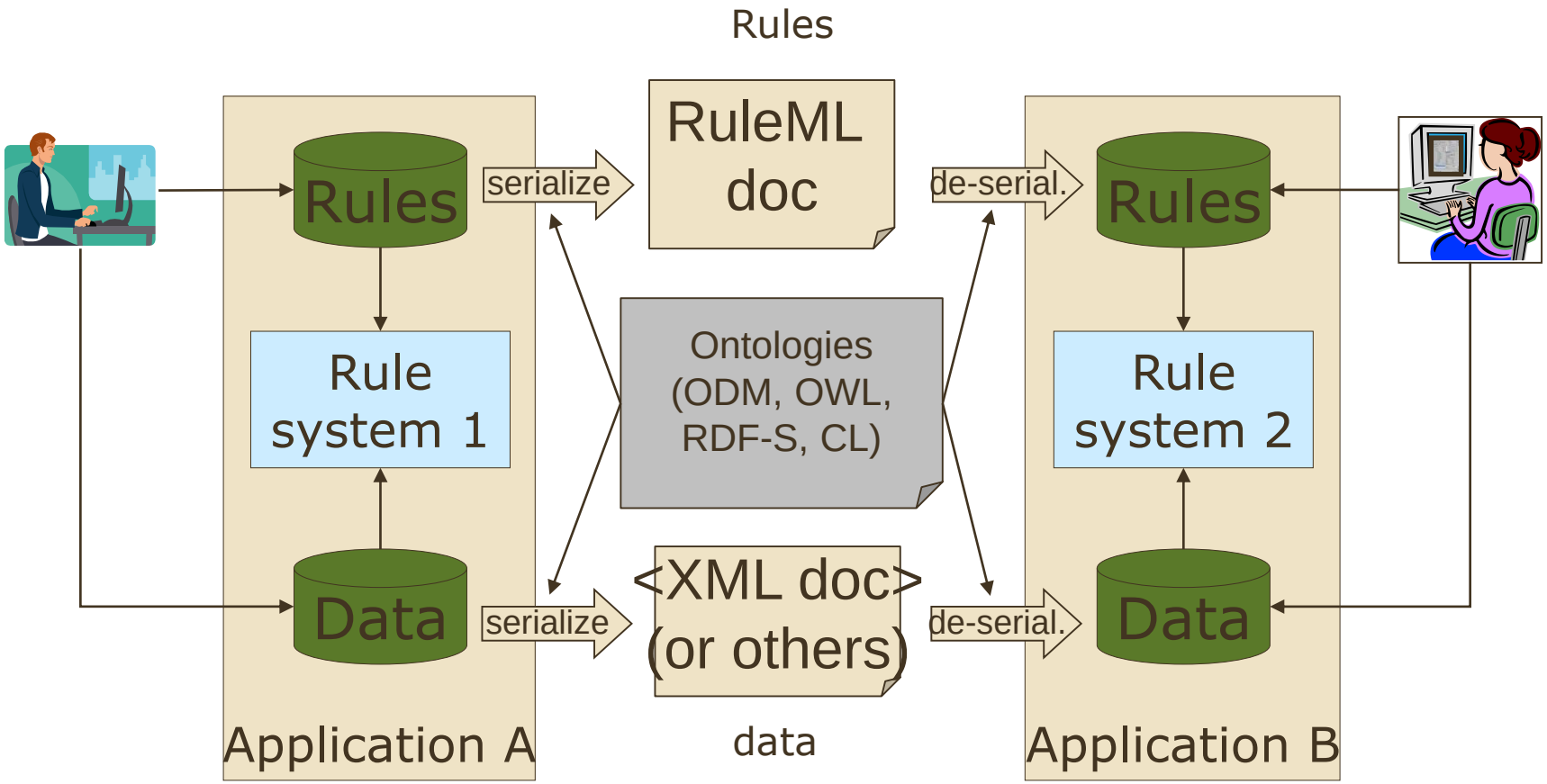
Agenda

- Introduction to Event Processing
- Event Processing Reference Model and Reference Architecture
- **Reaction RuleML Standard**
- Examples for Event Processing Functions
- Summary

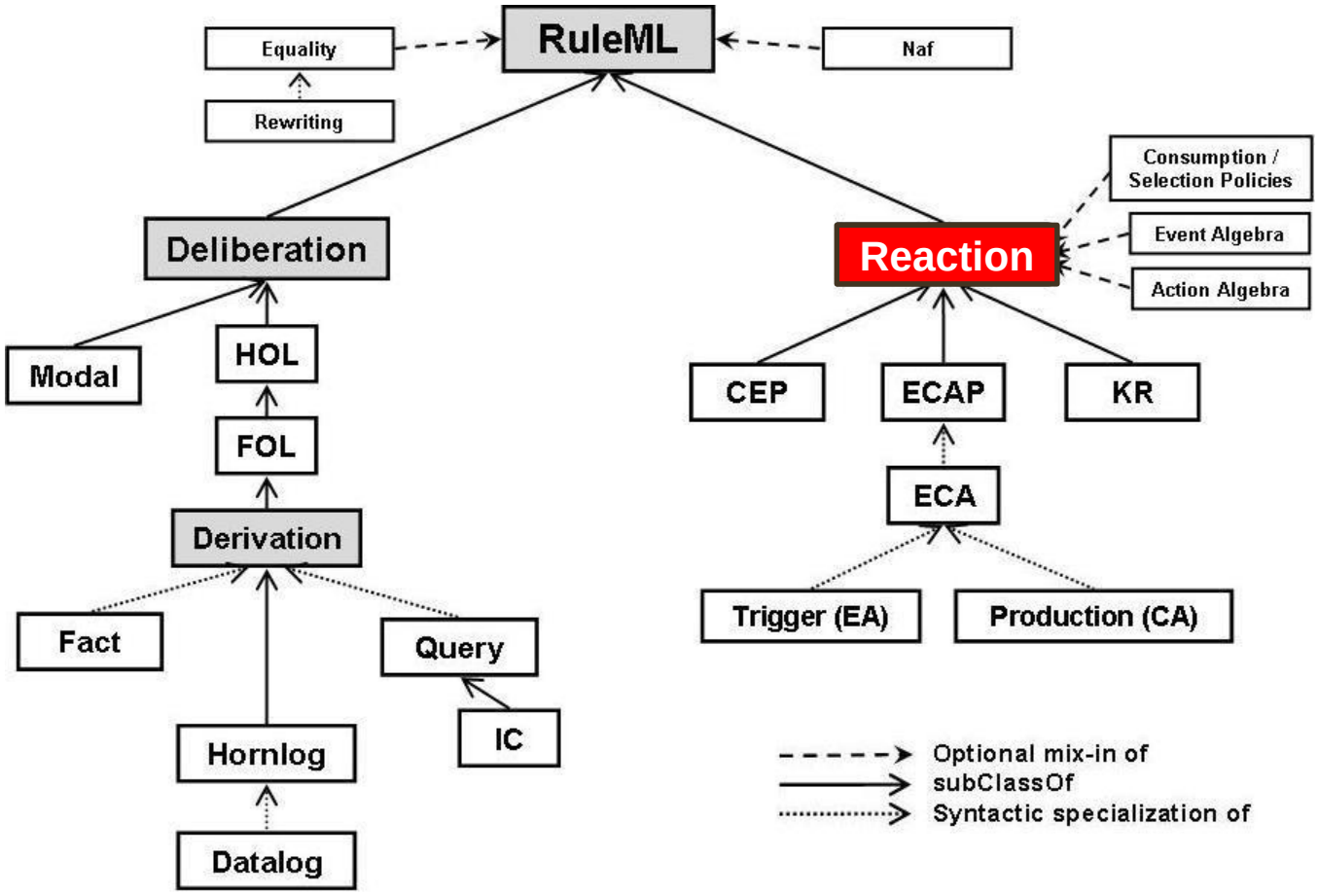
RuleML Enables ...



RuleML Rule interchange (on the PIM level)



The Reaction RuleML Family



Harold Boley, Adrian Paschke and Omair Shafiq: **RuleML 1.0: The Overarching Specification of Web Rules** , in Semantic Web Rules, Proceedings of RuleML-2010, Lecture Notes in Computer Science, 2010, Volume 6403/2010, 162-178.

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

* + variants and alternatives

- **Derivation Reaction RuleML (*if-then*)***
  - Time, Spatial, Interval
- **KR Reaction RuleML (*if-then* or *on-if-do*) ***
  - Situation, Happens$_{(@type)}$, 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

Reaction Rules: Specializable Syntax

+ further attributes

```
<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>
```

Info, Life Cycle Mgt.

Interface

Imple-mentation

Reaction RuleML – Example Rule Types+

- **Derivation Rule:**
  (temporal/event/action/situation reasoning)

```
<Rule style="reasoning">
  <if>...</if>
  <then>...</then>
</Rule>
```

- **Production Rule:**

```
<Rule style="active">
  <if>...</if>
  <do>...</do>
</Rule>
```

- **Trigger Rule:**

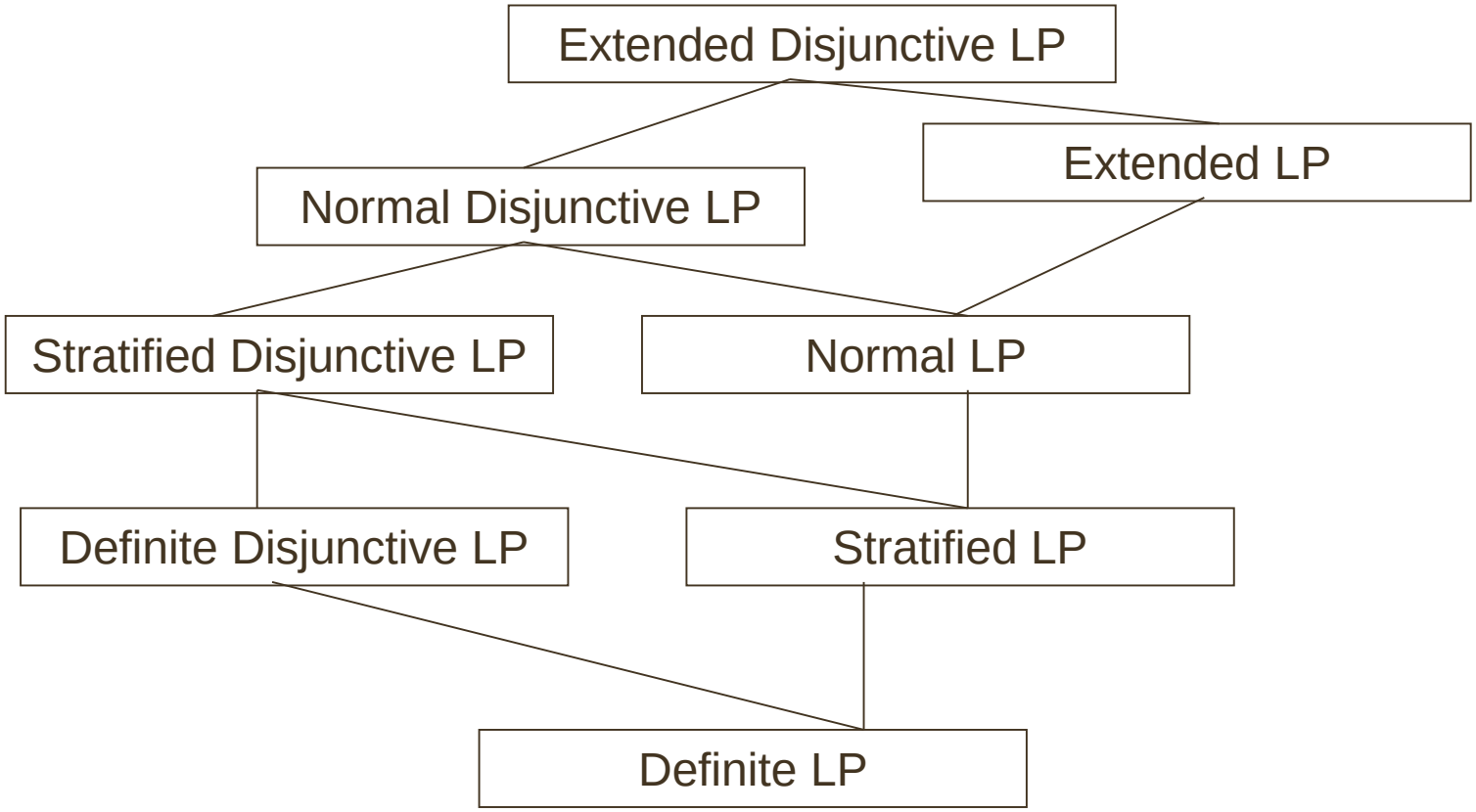
```
<Rule style="active"> >
  <on>...</on>
  <do>...</do>
</Rule>
```

- **ECA Rule:**

```
<Rule style="active">
  <on>...</on>
  <if>...</if>
  <do>...</do>
</Rule>
```

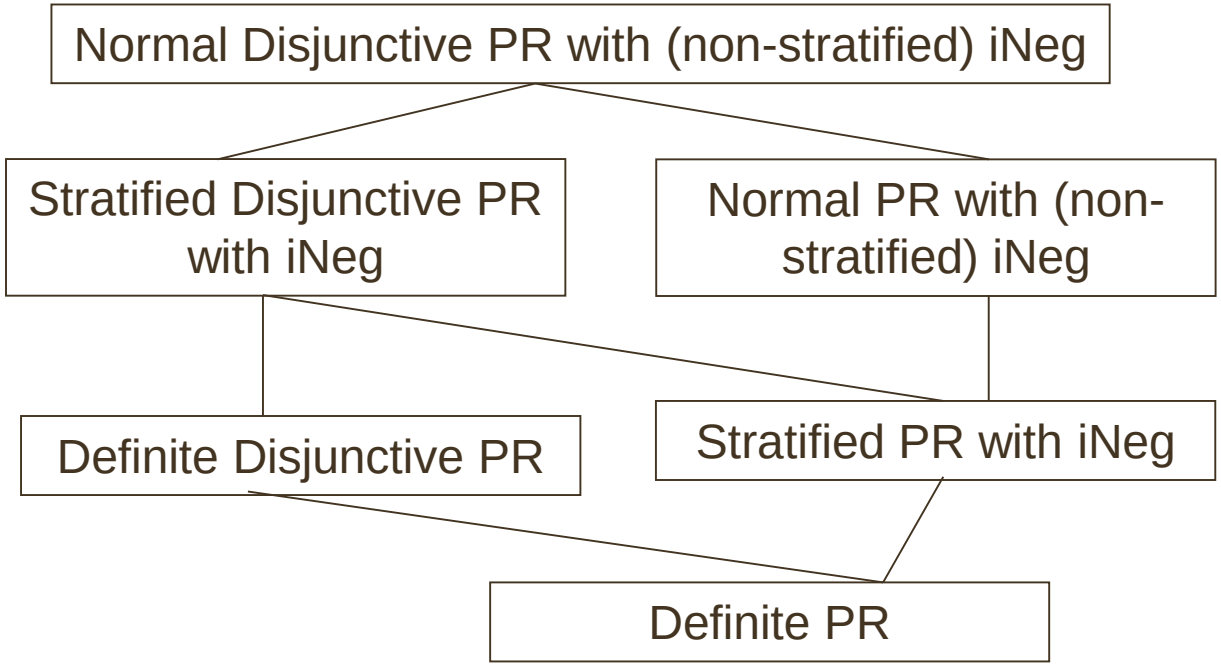
+ variants, e.g. EA, ECAP, CAP … and combinations, e.g. if-then-do, if-then-elsedo, …

Derivation Reaction RuleML Expressiveness Layering



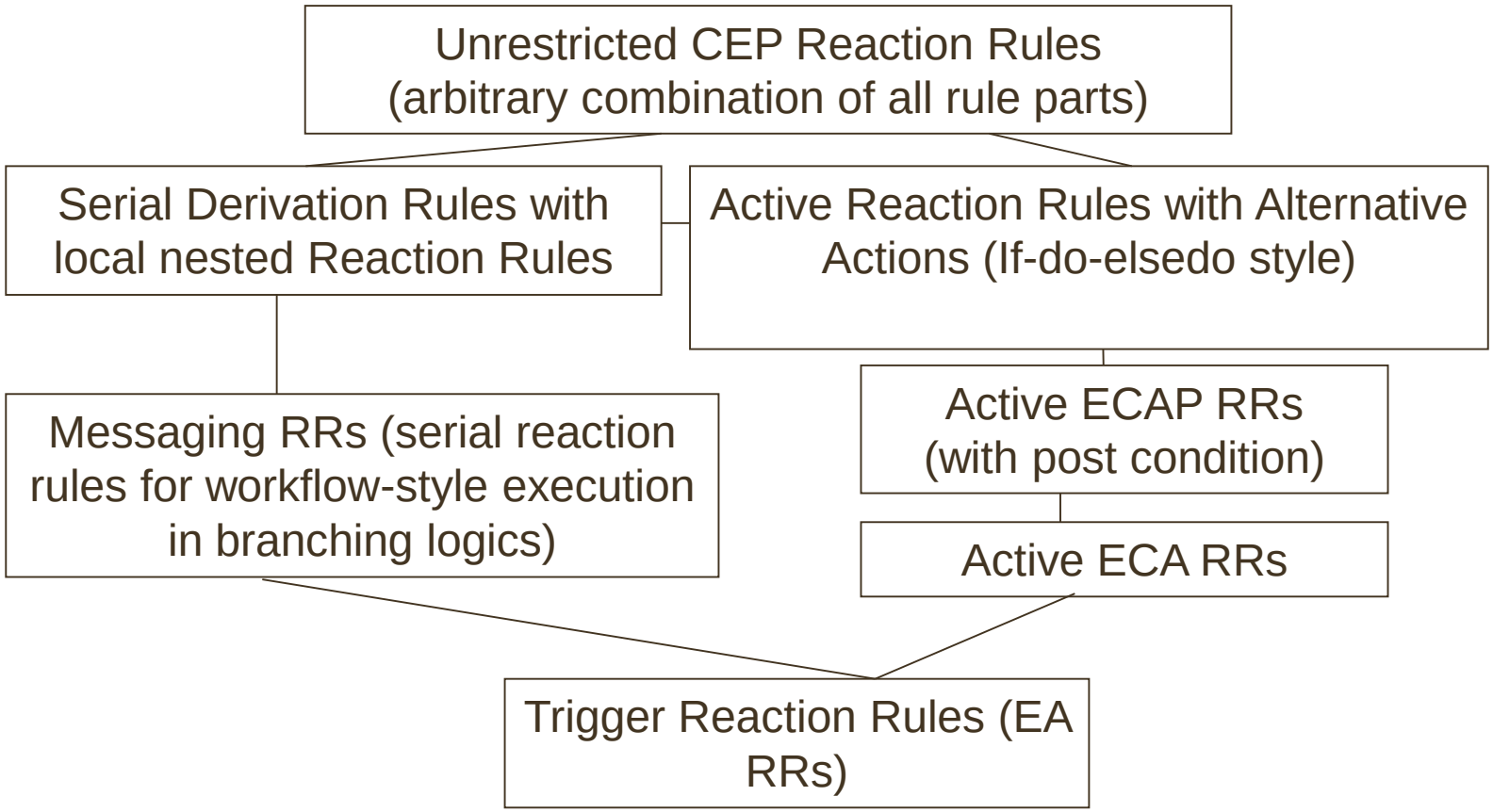
see http://ruleml.org/reaction/0.2/

Production Reaction RuleML Layering



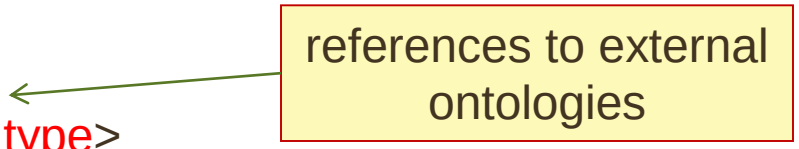
see http://ruleml.org/reaction/0.2/

ECA and CEP Reaction RuleML Expressiveness Layering



see http://ruleml.org/reaction/0.2/

Example - The Generic Syntax Pattern of An Event




```
<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 -->                           </scope>

<!-- 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>
```

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.

Example - Interface and Implementation

```
<Rule style="active">

<signature>
   <Atom>
            <Rel>likes</Rel>
            <arg><Var mode="+"/></arg>  <!-- mode=+ i.e. input argument -->
            <arg><Var mode="?"/> </arg> <!-- input or output argument -->
   </Atom>
 </signature>

 <if>
   <Atom>
    <op><Rel>likes</Rel></op>
    <arg><Var>X</Var></arg>
    <arg><Ind>wine</Ind></arg>
   </Atom>
 </if>
 <do>
   <Assert>  <formula>
     <Atom>
        <op><Rel>likes</Rel></op>
         <arg><Ind>John</Ind></arg>
        <arg><Var>X</Var></arg>
     </Atom>
 </formula>  </Assert>  </do>
</Rule>
```

Interface defining rule signature declaration

Implementation

Reaction RuleML Signature Definitions

- **Sorted Signature Definition**

  ***S* = ‹ *T*, *C*, *SC*, *M*, *arity*, *sort*, *scope*, *mode* ›**

  – ***S*** and $T_i \in \boldsymbol{T} = \{T_1, \ldots, T_n\}$ 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 $SC_j \in \boldsymbol{SC} = \{SC_1, \ldots, SC_m\}$ being symbols denoting the scope name
  2. pairwise disjoint sets of input-output symbols, called **modes**, where $M_k \in \boldsymbol{M} = \{M_1, \ldots, M_m\}$ being symbols denoting the mode name

  * after flattening composite scopes

  – The function

  - ***sort*** associates with each symbol c ∈ *C* its sort t ∈ *T*
  - ***scope*** associates with each symbol c ∈ *C* its scope sc ∈ *SC*
  - ***mode*** associates with each symbol c ∈ *C* its mode m ∈ *SC*

Multi-Sorted Signature Definitions

- **Multi-Sorted Base Signature**

  S_b = ‹ *T*, *P*, *F*, *C*, *SC*, *M*, *arity*, sort, scope, mode ›

  – is used as the basis to define Reaction RuleML dialects

  – introduces predicates (*P*), functions (*F*) and terms (C)

  – the function *sort* associates functions, predicates and constants to their sort.

  – the function *arity* gives the arity of predicates, functions and terms

Labelled Logic with Meta Knowledge Annotations

- **Combined Signature with Meta Knowledge**

$$S_{bmeta} = \langle S_b \cup S_{meta} , @ \rangle$$

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

$$@(L_1), \ldots, @(L_n)\ \Phi$$

  - Reaction RuleML distinguishes between
    - descriptive metadata <meta>
    - qualifying metadata <qualification>

Example- Descriptive and Qualifying Metadata

```
<Assert key="#module1”>
 <meta> <!-- descriptive metadata -->
   <Atom><Rel iri="dc:creator"/><Ind>Adrian Paschke</Ind></arg></Atom>
 </meta>
 <meta>
   <Time type="&ruleml;TimeInstant"><Data xsi:type="xs:date">2011-01-01</Data></Time>
 </meta>
 <meta>
   <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>
<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>
 </Assert>
```

Descriptive Metadata

Qualifying Metadata

Signature Pattern for Sort Definition

- A sort ***T*** is declared by a **signature pattern** ***T*** = {T_1, …, T_n} in the sort‘s signature definition
- Example

Event = ‹ *Term={}* › ***Time*** = ‹ *Term={}* ›

PositionalBinaryPredicate = ‹ *PositionalPredicate={Term,Term}, arity=2* ›

Happens = ‹ *PositionalBinaryPredicate={Event,Time}* ›

  - 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

```
<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>
```

Frame type
signature defining

Datetime sort

```
<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
signature defining

TimeEvent sort

Multi-Sorted Interpretation

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

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

  - $\Delta^{\mathcal{I}}$ universe (the union* of its sorts)
  - $\cdot^{\mathcal{I}}$ interpretation function
    - $c^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
    - $p^{\mathcal{I}} \subseteq T_1^{\mathcal{I}} \times \ldots \times T_n^{\mathcal{I}}$
    - $f^{\mathcal{I}} \subseteq T_1^{\mathcal{I}} \times \ldots \times T_n^{\mathcal{I}} \rightarrow T_{n+1}^{\mathcal{I}}$
  - $\mathit{Val}(\varphi, \sigma) \in TV$ truth valuation function, where
    - TV is a set of partially or totally ordered truth values*

* Note, for scoped domains it is more than just the union of its sorts, e.g., local and private symbols

(* being a complete lattice and reverse truth ordering for negation operators (anti-monotonic))

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>
```

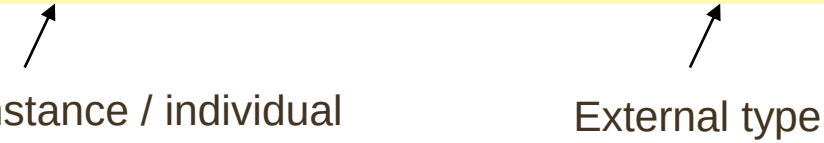
truth degree

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 <u>Semantic Profiles</u>**
  - 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

```
<Var vocab="http://www.w3.org/2001/XMLSchema/" type="date"/>
```

*Example - Typed Complex Event **Pattern** Definition*

```
<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">
  <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 Pattern defining **Event Templates** signature

```
<Event key="#e1" keyref="#ce2"><arg>…</arg></Event>
```

Event Instance

Selected Reaction RuleML Algebra Operators

- **Action Algebra**
  Succession **(Ordered Succession of Actions),** ***Choice*** **(Non-Determenistic Choice),** ***Flow*** **(Parallel Flow),** ***Loop*** **(Loops),** ***Operator*** **(generic Operator)**

- **Event Algebra**
  Sequence **(Ordered),** ***Disjunction*** **(Or) ,** ***Xor*** **(Mutal 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>

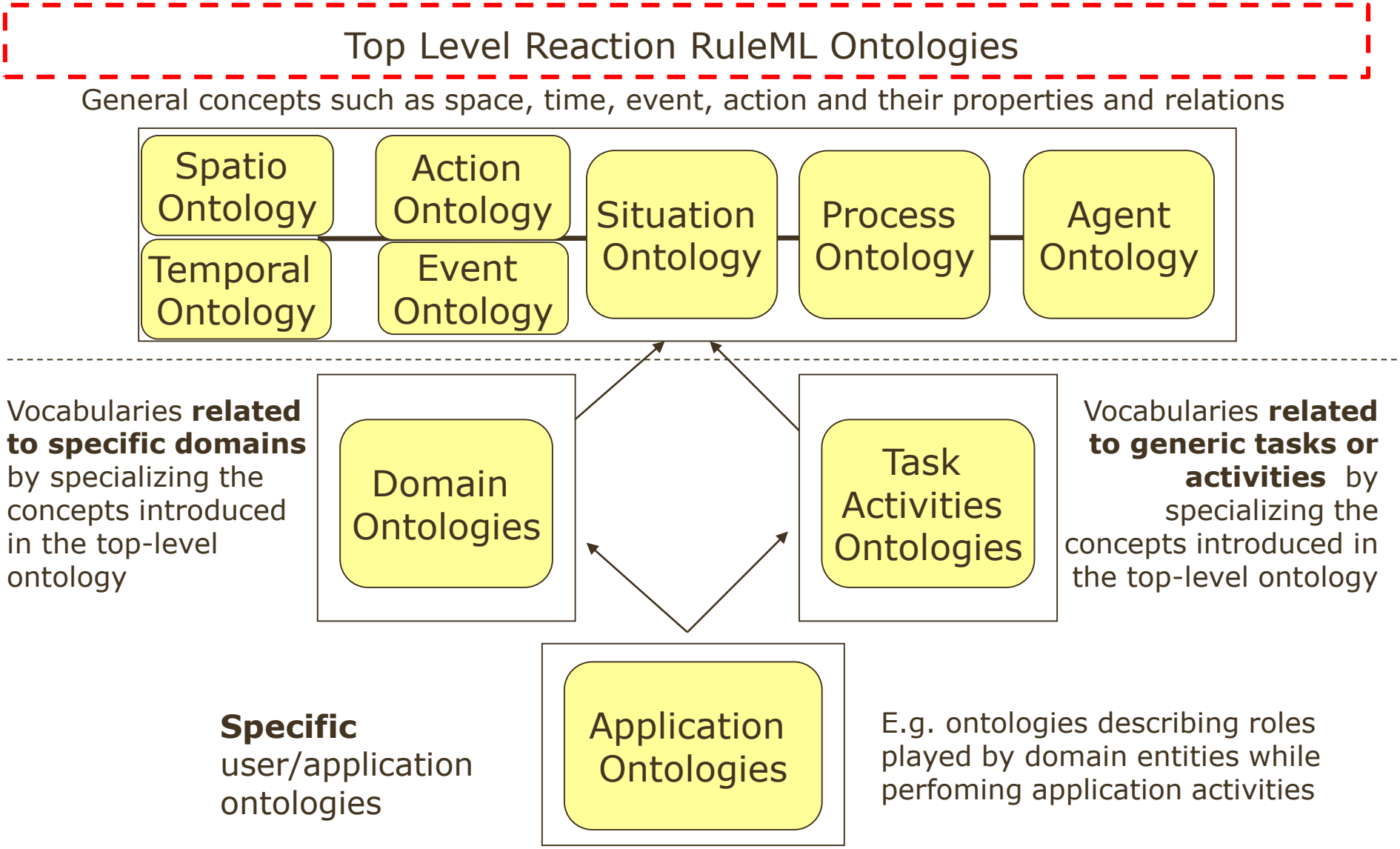
<Action type="&ruleml;Assert"> == <Assert>

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

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

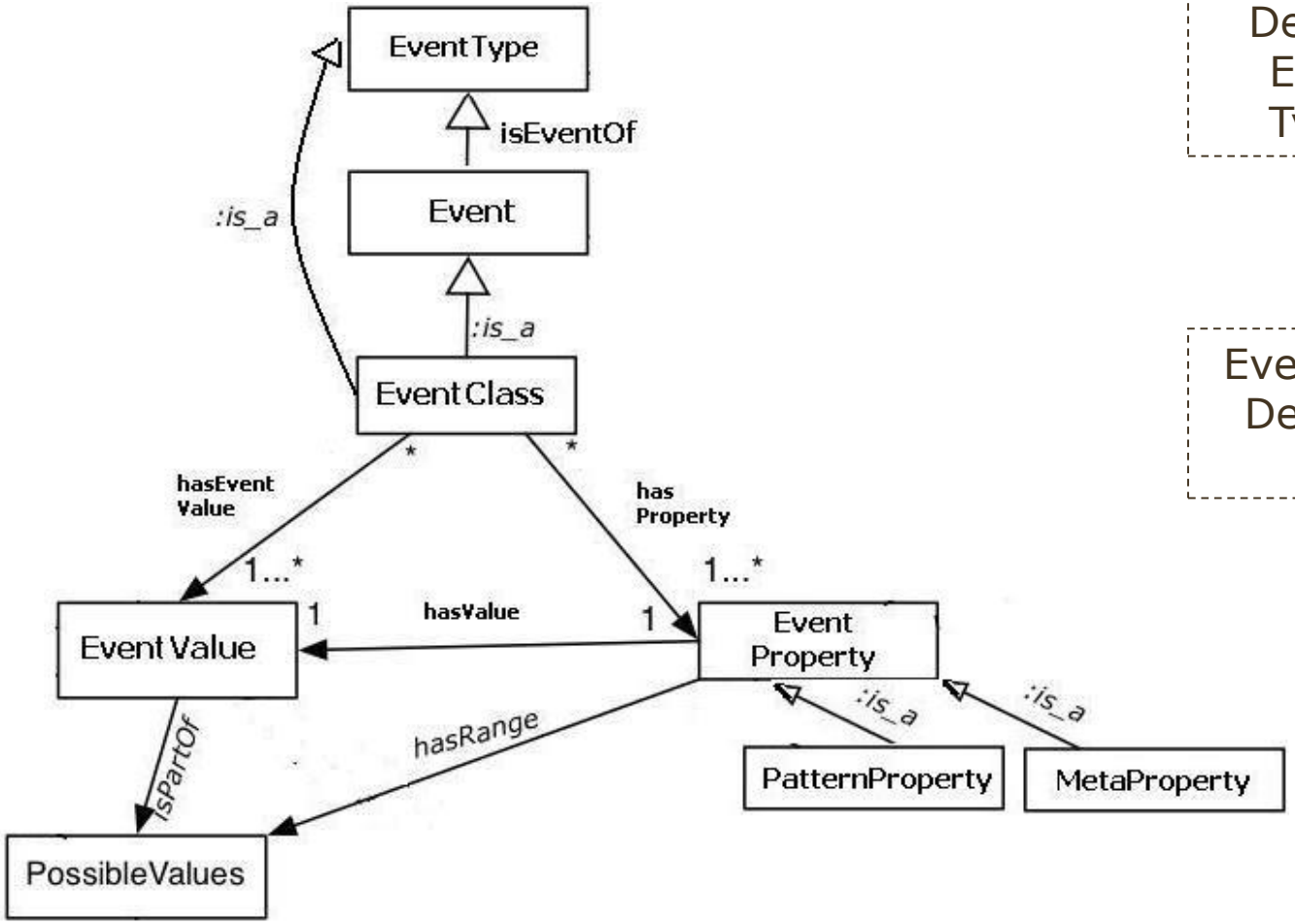
<Event type="ibm:CommonBaseEvent"> == IBM CBE

Reaction RuleML Metamodel



Example - Event Metamodel

(for defining Event Types of the Metamodel Event Class)

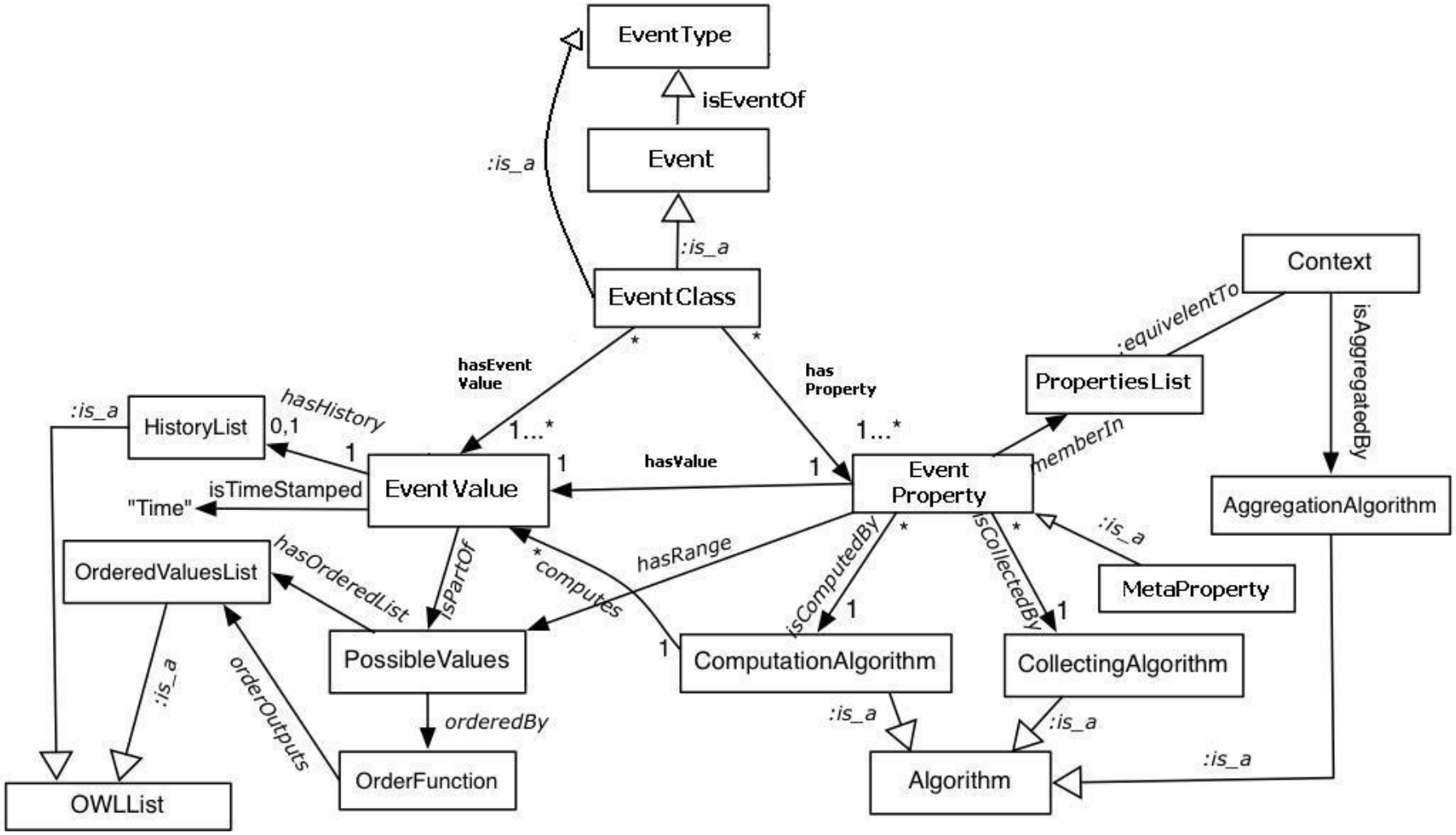


Defined Event Types

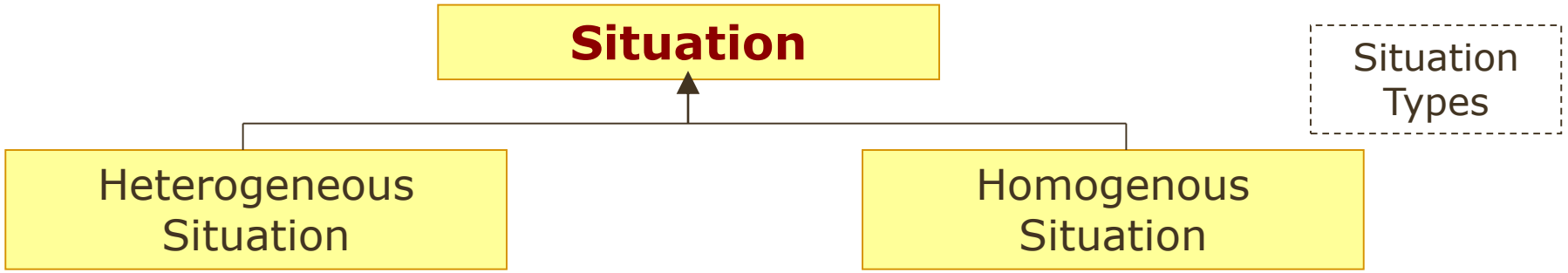
Event Class Definition

Integration of existing domain ontologies by defining their properties and values in an event classes in the Metamodel

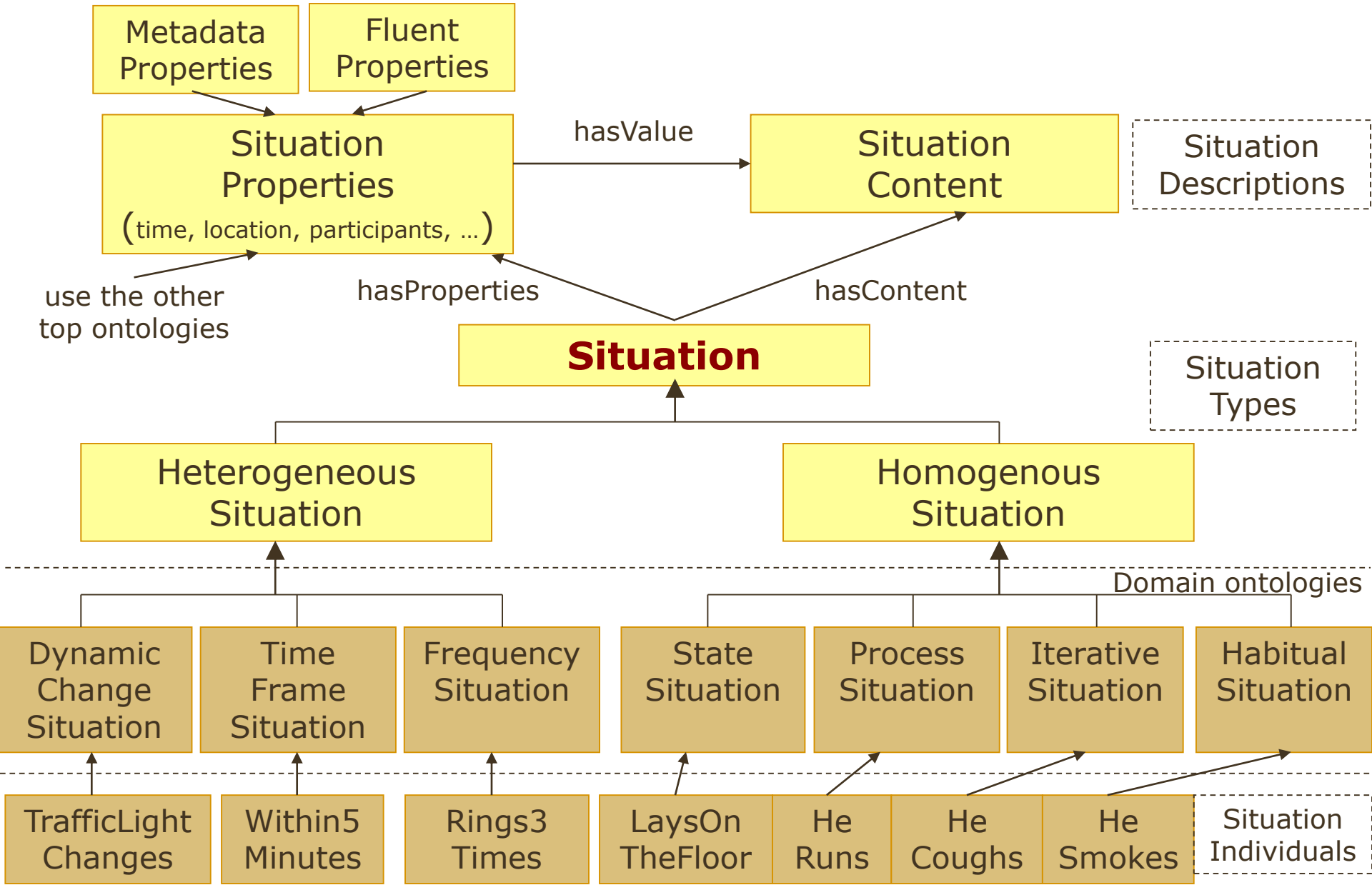
Extended Event Metamodel
(with computational properties and complex value definitions)



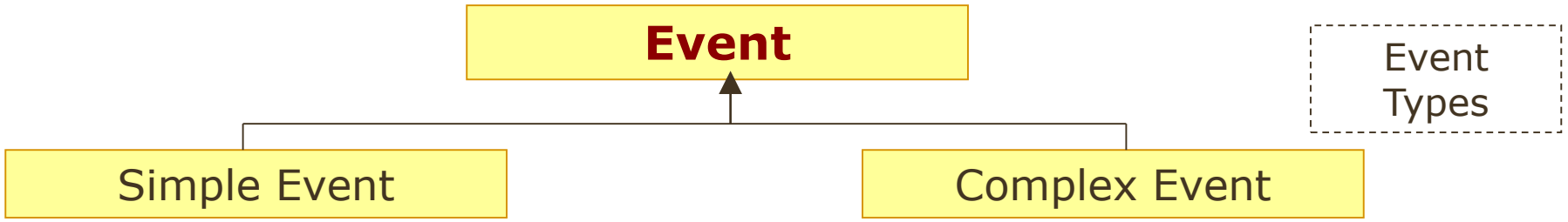
Top-Level Situation Types defined in Reaction RuleML Ontology



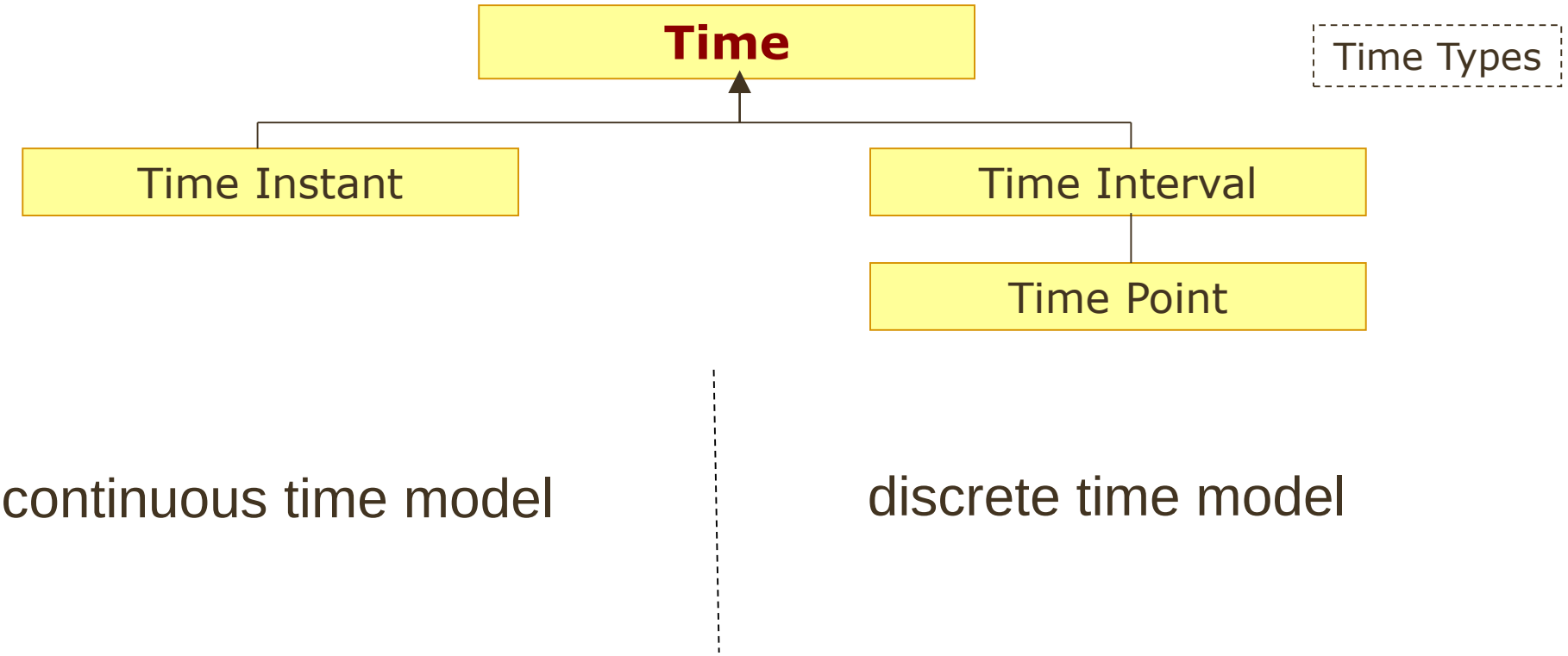
Example: Situation Types



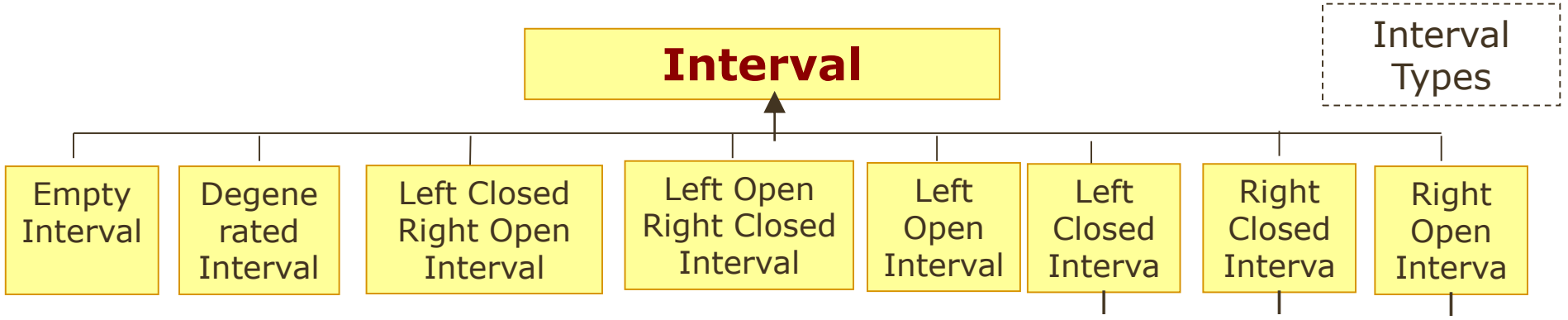
Reaction RuleML Event Types



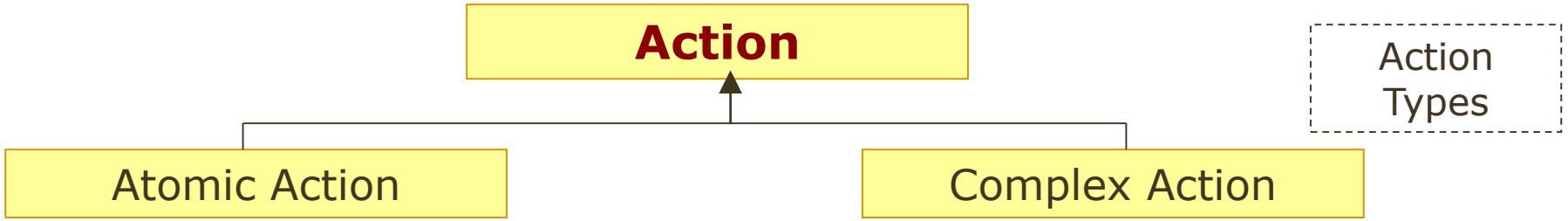
Reaction RuleML Time Types



Interval Top Ontology Metamodel



Action Top Ontology Metamodel



Semantic Profiles

- **Define *intended semantics* for knowledge interpretation, reasoning, execution, …**
- **A semantic profile (SP) can define**
  - profile (sub-)signature S_{SP}
  - profile language $\sum_{SP}$
  - intended interpretation semantics I_{SP}
  - axioms Φ_{SP}
  - a semantics preserving translation function into Reaction RuleML $\tau(.)$

Expanded Profile Semantic Multi-Structure

- $\mathcal{I} = \langle \mathcal{I}_{\mathcal{R}}, \mathcal{I}_{\mathcal{D}}, \mathcal{I}_{\mathcal{SP}} \rangle$ with
  - $\mathcal{I}_{\mathcal{R}}$ basic multi-sorted interpretation
    - basic structure of Reaction RuleML
  - $\mathcal{I}_{\mathcal{D}}$ default dialect interpretation
    - if no profile is specified $\mathcal{I}_{\mathcal{D}}$ is used for default interpretation
  - $\mathcal{I}_{\mathcal{SP}}$ semantic profiles with expansion structures
    - multiple alternative semantic profiles can be specified (with possible preference ordering)

Semantic Profiles for Evaluation

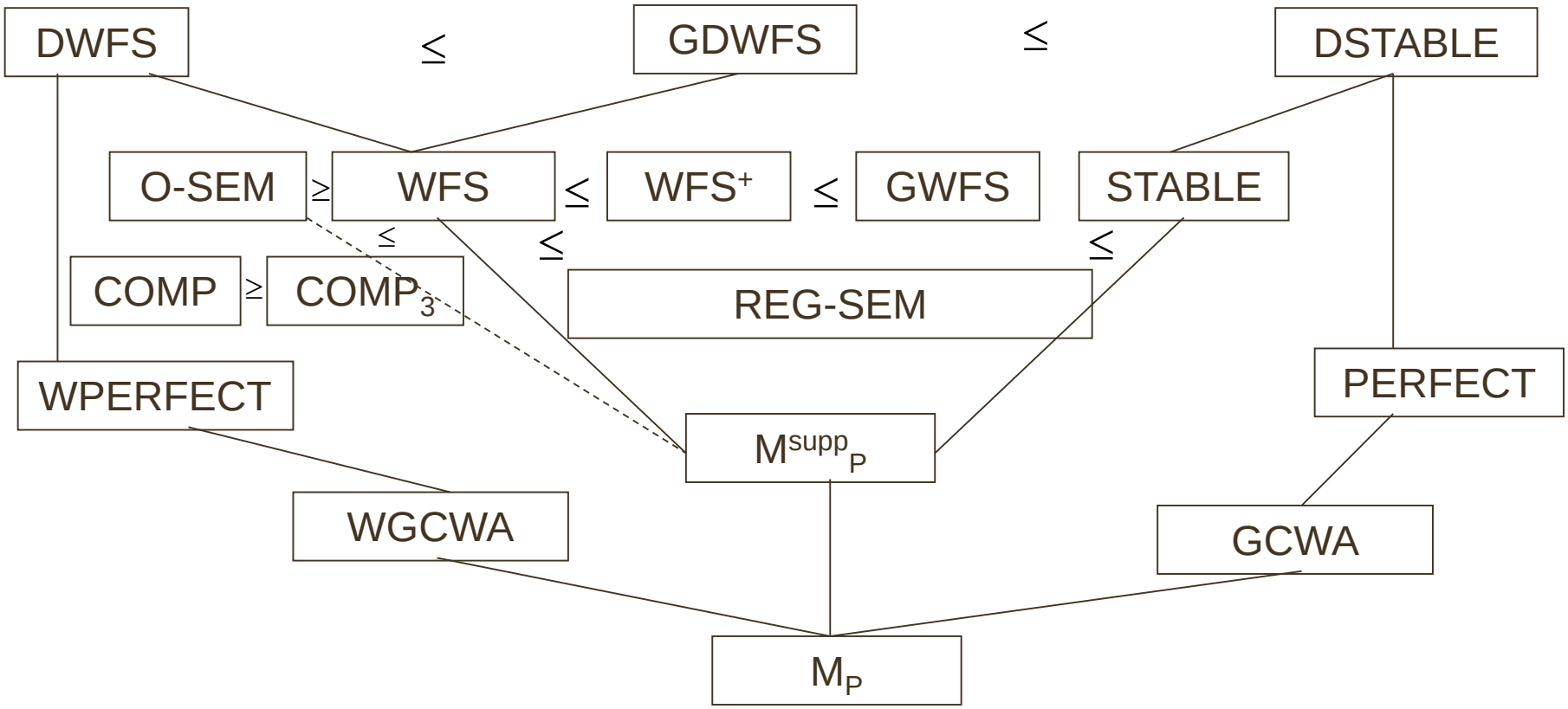
- **A semantics *SEM‘* extends *SEM* (SEM(P) ⩽ SEM’(P)) for a rule program *P* iff all formula *F* which are true in *SEM(P)* are also true in *SEM‘(P),* but in SEM‘(P) more formula can be true or false than with SEM(P).**
- ***SEM'* is defined for a class of programs that strictly includes the class 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’* (= intended evaluation semantics of the inference/execution engine) can be used for all programs (or scoped modules) with SEM(P) ⩽ SEM’(P)**

Example - Semantic LP Profiles

Class	Semantics
Definite LPs	Least Herbrand model: M_p
Stratified LPs	Supported Herbrand model: M_p^{supp}
Normal LPs	Clark's Completion: $COMP$
	3-valued Completion: $COMP_3$
	Well-founded Semantics: WFS
	WFS^+ and WFS'
	WFS_C
	Strong Well-founded Semantics: WFS_E
	Extended Well-founded Semantics: WFS_S
	Stable Model Semantics: $STABLE$
	Generalized WFS: $GWFS$
	$STABLE^+$
	$STABLE_C$
	$STABLE^{rel}$
	Pereira's $O-SEM$
	Partial Model Semantics: $PARTIAL$
	Regular Semantics: $REG-SEM$
	Preferred Semantics: $PREFERRED$
General Disjunctive	Disjunctive WFS: $DWFS$
	Generalized Disjunctive WFS: $GDWFS$
	Disjunctive Stable: $DSTABLE$
Stratified Disjunctive	Perfect model $PERFECT$
	Weakly Perfect: $WPERFECT$
	Generalized Closed World Assumption: $GCWA$
Positive Disjunctive	Weak generalized closed world assumption: $WGCWA$

Semantic Layering - Example LP Semantic Profiles



A semantic SEM‘ extends a semantic SEM:

SEM‘ ⩾ SEM iff ∀ (P, F) SEM(P) |= F ➔ SEM‘(P) |= 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>
```

Interface with **evaluation semantics** and public rule **signature**

Implementation

Example - Integration of Semantic Profiles

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

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

2. **Reference pre-defined Semantic Profile as profile type**

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

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

3. **Locally defined Semantic Profile**

```
<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>
```

Note: also other non RuleML content models are supported

Example – Reified Situation Calculus Axioms

(1) ∀(A1, A2, S1, S2) (do(A1,S1) = do(A2,S2)) ➔ (A1 = A2)

(2) ∀(S1, S2, A) (S1 < do(A,S2)) ´ (S1 <= S2)

(3) ∀(S1, S2) (S1 ⩽ S2) ´ (S1 < S2) ∨ (S1 = S2)

(4) ∀(S1, S2) (S1 < S2) ➔ ¬ (S2 < S1)

(5) ∀(S, F) [holds(F,s0) ∧ (∀(A) (holds(F,S) ➔ holds(F,do(A,S)))] ➔ holds(F,S)

(6) ¬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></declare>
             <declare><Var>S1</Var></declare>   <declare><Var>S2</Var></declare>
           </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>
```

Profile key can be referenced by keyref

Example – RIF Semantic Web Compatibility Profiles

Profile	IRI of the Profile	Model	Satisfiability	Entailment
Simple	http://www.w3.org/ns/entailment/Simple	RIF-Simple-model	satisfiability	RIF-Simple-entailment
RDF	http://www.w3.org/ns/entailment/RDF	RIF-RDF-model	RIF-RDF-satisfiability	RIF-RDF-entailment
RDFS	http://www.w3.org/ns/entailment/RDFS	RIF-RDFS-model	RIF-RDFS-satisfiability	RIF-RDFS-entailment
D	http://www.w3.org/ns/entailment/D	RIF-D-model	RIF-D-satisfiability	RIF-D-entailment
OWL Direct	http://www.w3.org/ns/entailment/OWL-Direct	RIF-OWL Direct-model	RIF-OWL Direct-satisfiability	RIF-OWL Direct-entailment

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

Example:

```
<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

```
<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

(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 $\mathcal{S}_\mathcal{E} = \langle\, \mathcal{E}, \mathrm{sort}, \prec \,\rangle$**
  - $\mathcal{E}$ events, sort gives sort of event, $\prec$ precedence relation
- **Interval-based Event Model: $\mathcal{I}_\mathcal{E}$ is a structure for $\mathcal{S}_\mathcal{E}$ iff**
  - *finite predecessor property*: $\forall e \in \mathcal{E}$ the set $\{e' \in \mathcal{E} \mid e' \prec e\}$ is finite
  - *termination property*: if e is of sort *NonTerminatingEvent* then there is no $e' \in \mathcal{E}$ such that $e \prec e'$
  - *finitness property*: if e is of sort *TerminatingEvent* then $\{e' \in \mathcal{E} \mid e' \nvdash e\}$ is finite.
  - *interval ordering property*: $\forall e_1,e_2,e_3,e_4\, (e_1 \prec e_2 \nvdash e_3 \prec e_4 \rightarrow e_1 \prec e_4)$.
- **Event Representations *τ(e)*, $e \in \mathcal{E}$,**
  - is a representation of $\mathcal{I}_\mathcal{E}$ in a linear interval-ordering $(\mathcal{T}, <_\mathcal{T})$ * iff *τ(e)* is an interval in $\mathcal{T}$, and $e <_{\mathcal{IE}} e'$ iff $\tau(e) \prec_\mathcal{T} \tau(e')$.
  - **Simple Events**: *τ(e)* of form [t,t]=[t]
  - **Terminating Events**: *τ(e)* of form [t,t')
  - **Non-terminating Event**: *τ(e)* of form [t,∞)

*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):**
  - *initiates(e,E,T)* / *terminates(e',E,T)*: e and e' initiates/terminates E
    - transitive and irreflexive: *initiates(e,E,T)* $\prec$ *initiates(e',E',T')* $\wedge$ *initiates(e',E',T')* $\prec$ *initiates(e'',E'',T'')* $\rightarrow$ *initiates(e,E,T)* $\prec$ *initiates(e'',E'',T'')*. and never *initiates(e,E,T)* $\prec$ *initiates(e,E,T)*
    - precedence: $E \prec E'$ iff *initiates(e,E,T)* $\prec$ *initiates(e',E',T')*
  - Representation
    - $\tau(E)$ = [*initiates(e,E,t)*, *terminates(e',E,t')*] = [t,t']

Example – Complex Event Detection
Time-point vs. Interval-based Semantics

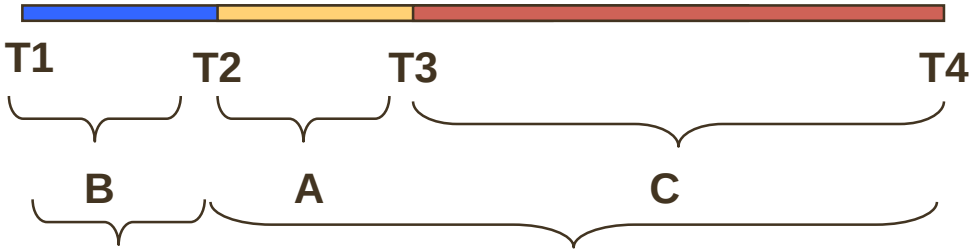
- Time-point semantics
  - Event definition: B; (A;C) (Sequence)
    event instance sequence EIS = b, a, c => detect event
    event instance sequence EIS = a, b, c => detect event 
    - because the detection timpoint of (a;c) is c which is after the timepoint of b, i.e. the timepoint of (a;c) follows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

holdsInterval([b,b] [T1,T1]) Λ holdsInterval([a,c] [T2,T3]) Λ [T1,T1] ≤ [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) ≡ detect(e,[T1,T3]) :-  holdsInterval([a,b],[T1,T2],[a,b,c]),
                                holdsInterval([b,c],[T2,T3],[a,b,c]),
                                [T1,T2] ≤[T2,T3].
```

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

```
detect(e,T):-  event(sequence(a,b),T),                  % detection condition for the event e
               update(eis(e), "occurs(e,_0).“, [T]),    % add e with key eis(e)
               consume(eis(a)), consume(eis(b)).% consume all a and b events
```

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

Adrian Paschke: ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics CoRR abs/cs/0610167: (2006); http://arxiv.org/ftp/cs/papers/0610/0610167.pdf

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:      a <- not b          P':     a <- not b
        b <- not a                  b <- not a
        c <- not c                  c <- not c
        c <- a                              c <- a
                                            c
```

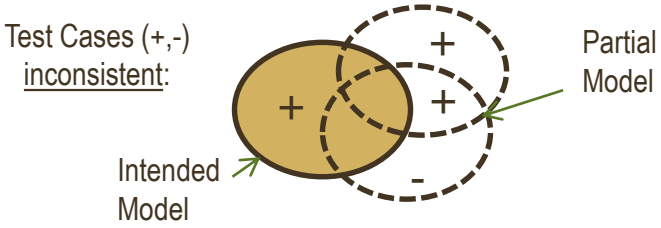
```
T:{a=>true,c=>true}
```

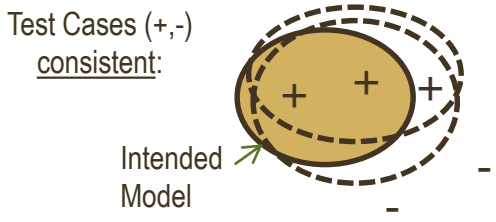
- **STABLE(P) $|=_{TC}$ {a, not b, c}, i.e. T succeeds.**
- **STABLE(P') $|=_{TC}$ {not a, b, c}, i.e. T fails.**

➔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**

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

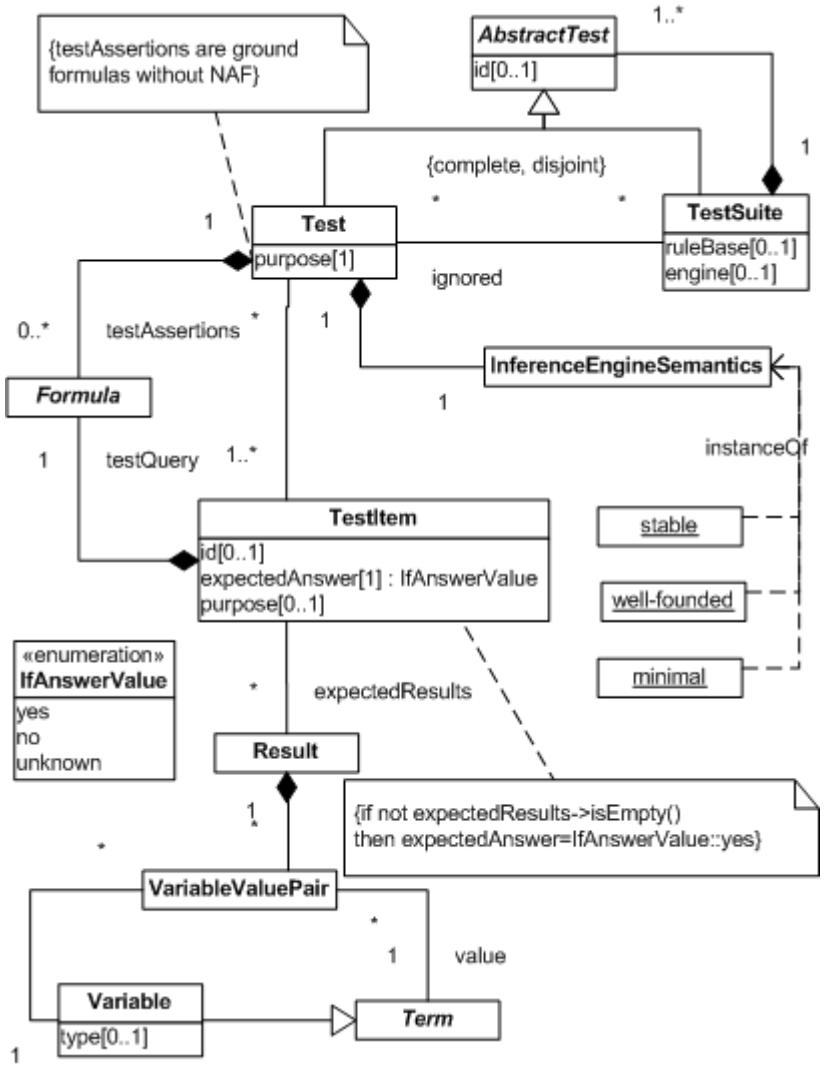
$$M_0 \models_{TC} (X, A, -) \text{ iff } \exists m \in M_0 : m \in \sum(Mod(X), R) \Rightarrow m \notin Mod(A)$$

  - $\models_{TC}$ compatibility relation
  - Mod association function between sets of formulas and sets of models
  - Σ 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




```
<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”/></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)

```
<Rule key="ruleinterface1">
  <evaluation><Profile> ...p1… </Profile><evaluation>
  <evaluation><Profile> …p2 …</Profile></evaluation>
 <signature>…s1…</signature>
</Rule>
```

Interface 1
(@key=ruleinterface1)

```
<Rule key="ruleinterface2">
  <evaluation><Profile> ...p3… </Profile><evaluation>
  <signature>…s2…</signature>
</Rule>
```

Interface 2
(@key=ruleinterface2)

…

```
<Rule keyref="ruleinterface2"  key=" ruleimpl2 ">
  <if>  …  </if>
  <do> </do>
 </Rule>
```

Implementation 2
(@keyref=ruleinterface2)

```
<Rule keyref="rulreinterface1"  key="ruleimpl1 ">
  <if>  …  </if>
  <do> </do>
 </Rule>
```

Implementation 1
(@keyref=ruleinterface1)

*Distributed Complex Event **Pattern** Definition*

```
<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">
  <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 Pattern defining **Event Templates** signature

```
<Event key="#e1" keyref="#ce2"><content>…</content></Event>
```

Event Instance

Example - Imports

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 **i**nput or **o**utput formulas in $\mathcal{M}$ with input (+) or output mode (-)
  - $\mathcal{G}$, $\mathcal{L}$, $\mathcal{P}$ sets of formulas with **g**lobal, **l**ocal, **p**rivate 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>
```

global scope

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

private scope

```
 <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">
…
```

user-defined scope

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

- $\mathcal{M} \oplus \mathcal{M}' = \langle \mathcal{F}_{\mathcal{M}} \cup \mathcal{F}_{\mathcal{M}'}, (\mathcal{I}_{\mathcal{M}} \setminus \mathcal{O}_{\mathcal{M}'} \cup \mathcal{I}_{\mathcal{M}'} \setminus \mathcal{O}_{\mathcal{M}}), (\mathcal{O}_{\mathcal{M}} \cup \mathcal{O}_{\mathcal{M}'}), (\mathcal{G}_{\mathcal{M}} \cup \mathcal{G}_{\mathcal{M}'}), (\mathcal{L}_{\mathcal{M}} \cup \mathcal{L}_{\mathcal{M}'}), (\mathcal{P}_{\mathcal{M}} \cup \mathcal{P}_{\mathcal{M}'}) \rangle$
  - remove input by output formulas from imports
    - $\mathcal{I}_{\mathcal{M}} \setminus \mathcal{O}_{\mathcal{M}'}$ and $\mathcal{I}_{\mathcal{M}'} \setminus \mathcal{O}_{\mathcal{M}}$
  - respect private signatures
    - *signature*($\mathcal{M}$) $\cap \mathcal{P}_{\mathcal{M}'} = \emptyset$ and *signature*($\mathcal{M}'$) $\cap \mathcal{P}_{\mathcal{M}} = \emptyset$
  - output signatures are disjoint *
    - $\mathcal{O}_{\mathcal{M}'} \cap \mathcal{O}_{\mathcal{M}'} = \emptyset$
  - modules are mutual independent *
    - *no (positive) cyclic dependencies through input – output signatures between $\mathcal{M}$ and $\mathcal{M}'$*

* semantic profiles might relax these conditions, but must guarantee compositionality

Modular Semantic Multi-Structure

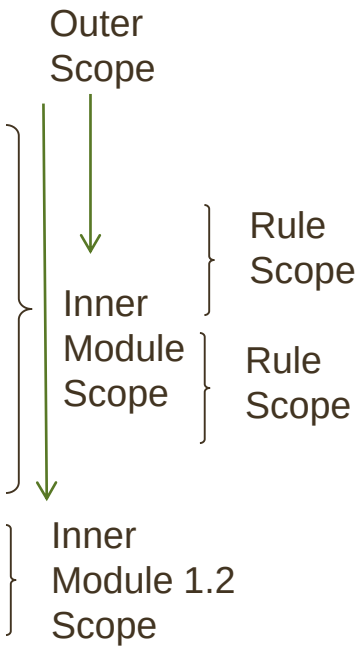
- **Semantic Multi-Structure** $\mathcal{I} = \langle \mathcal{I}_{\mathcal{M}_0}, \mathcal{I}_{\mathcal{M}_1}, \mathcal{I}_{\mathcal{M}_2}, \ldots \rangle$ set of semantic structures of the importing KB ($\mathcal{M}_0$) and the imported modules ($\mathcal{M}_1, \mathcal{M}_2, \ldots$)
  - Semantic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

    $\mathcal{I}_{\mathcal{M}} \bowtie \mathcal{I}_{\mathcal{M}}$, = $\{\mathcal{M}^{\mathcal{I}} \cup \mathcal{M}'^{\mathcal{I}} \mid \mathcal{M}^{\mathcal{I}} \in \mathcal{I}_{\mathcal{M}}$ *and* $\mathcal{M}'^{\mathcal{I}} \in \mathcal{I}_{\mathcal{M}}$, *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

```
<Assert key="#module1”>
   <evaluation><Profile> ...SEM’’… </Profile><evaluation>

<Rulebase key="#innermodule1.1">
      <evaluation><Profile> ...SEM’… </Profile><evaluation>
   <Rule key=“#rule1”>
      <evaluation><Profile> ...SEM… </Profile><evaluation>
   </Rule>
   <Rule key=“#rule2”>
      …
   </Rule>
 </Rulebase>

<Rulebase key="#innermodule1.2"> … </Rulebase>

 </Assert>
```



SEM‘‘(#module1) ⩾ SEM‘(#innermodule1.1) ⩾ SEM(#rule1)
SEM(#rule2) = SEM‘‘(#module1)
SEM‘(#innermodule1.2) = SEM‘‘(#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), …, @(L_n) Φ**) 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:** $\mathcal{I}_{\mathcal{SC}}$ is the reduct $\mathcal{I}_{\mathcal{KB}}{\restriction}\mathcal{S}_{\mathcal{SC}}$ of $\mathcal{I}_{\mathcal{KB}}$ to $\mathcal{S}_{\mathcal{SC}} \subseteq \mathcal{S}_{\mathcal{KB}}$
- **Scoped Reasoning** $\mathcal{I}_{\mathcal{SC}} \vdash \boldsymbol{\Phi}$ iff $\mathcal{I}_{\mathcal{SC}} \models \boldsymbol{\Phi}$
- **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="…getDate()“/><Var>V</Var></Operator>
      </guard>
      <if> … </if>  <then>   </then>
   </Rule>
 </Rulebase>
</Assert>
```

Outer Scope

Inner Scope + Guard

Example - Semantic Profile for Event Based System Executions with Scoped Interpretation

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

- ***Event Based System Model***
  - collection of system executions $\mathcal{I}_{\mathcal{M}}$ in a fixed modular signature*

* imports and updates are added as scoped modules $\mathcal{M}_i$

- ***Composition of System Models***

  $\mathcal{M}$ and $\mathcal{M}$' are scoped modules with $\mathcal{M} \oplus \mathcal{M}$' being their composition

  - ***Serial:*** $\mathcal{I}_{\mathcal{M}} \otimes \mathcal{I}_{\mathcal{M}'} = \mathcal{I}_{\mathcal{M}} \bowtie \mathcal{I}_{\mathcal{M}'}$ (join union)
  - ***Parallel:*** $\mathcal{I}_{\mathcal{M}} \,||\, \mathcal{I}_{\mathcal{M}'} = \{\mathcal{I}_{\mathcal{M}\oplus\mathcal{M}'} \mid \mathcal{I}_{\mathcal{M}\oplus\mathcal{M}'}\restriction\mathcal{M} \in \mathcal{I}_{\mathcal{M}}$ and $\mathcal{I}_{\mathcal{M}\oplus\mathcal{M}'}\restriction\mathcal{M}' \in \mathcal{I}_{\mathcal{M}'}\} = \mathcal{I}_{\mathcal{M}} \cap \mathcal{I}_{\mathcal{M}'}$ (intersection)

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:* $KB_i = KB_{i-1} \cup M^{assert}_{oid}$ and Retract: $KB_i = KB_{i-1} \setminus M^{retract}_{oid}$
  - **local states** can be managed using module scopes and scoped reasoning using the module scopes is possible

Examples

<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** $\cup$ Φ – add Φ to state **D** leading to state **D'**

  retract(Φ): **D'** = **D** $\setminus$ Φ – delete Φ from state **D** leading to state **D'**

  update(Φ, Φ'): (retract(Φ) $\otimes$ assert(Φ')) – atomic update
- **Model-theoretic Semantic**
  - A *path structure* maps execution paths to the ordinary semantic structures used in classical predicate logic:

I(π) = M , where π - path, M – classical semantic structure, which says which transactions can execute along the path π

In addition:

- If π = <**D**> is a path that consists of only one database state then **I**(π) must make every fact in **D** true.
- If π = <**D**, **D**$\cup\Phi$> then **I**(π) should make assert(Φ) true
- If π = <**D**, **D**$\setminus\Phi$> then **I**(π) should make retract{Φ} true

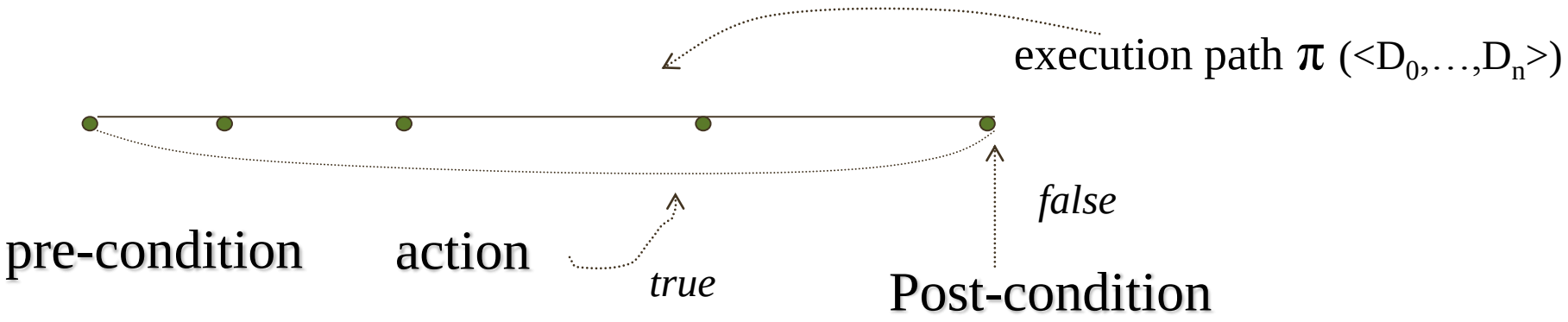
Transaction Logic Semantics

- Transactional Satisfaction

$I(<D_0,\ldots,D_n>) \models a \otimes b$ iff $\exists D_k$ such that $I(<D_0,\ldots,D_k>) \models a$ and $I(<D_k,\ldots,D_n>) \models b$
$I(<D_0,\ldots,D_n>) \models a \wedge b$ iff $I(<D_0,\ldots,D_n>) \models a$ and $I(<D_0,\ldots,D_n>) \models b$
$I(<D_0,\ldots,D_n>) \models a \rightarrow b$ iff $I(<D_0,\ldots,D_n>) \models a$ then $I(<D_0,\ldots,D_n>) \models b$

- Transactional Actions

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



Example:

if (condition ∧ action) is true, but post-condition false,
then (condition ∧ action) ⊗ postcondition is false on π.

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

if (condition ∧ action) ⊗ postcondition is false on π,
then else-action is true on π.

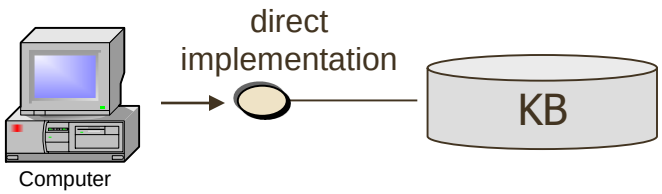
➔ else action (e.g., by executing compensation action)

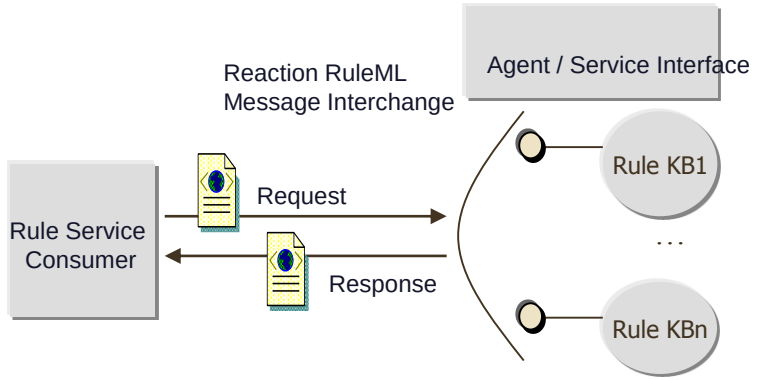
Extension - Concurrent Transaction Logic

- **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 $\phi \otimes \varphi$; **concurrent transactions $\phi \mid \varphi$**; nondeterministic transactions $\phi \vee \varphi$;

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)





CEP Reaction RuleML - Messaging

- **Send a message:** <*Send*> <*Message*>
- **Receive a message:** <*Receive*> <*Message*>

```
<Message>
  <cid>   <!-- conversation ID-->    </cid>
  <protocol>   <!-- transport protocol -->   </protocol>
  <directive>   <!– pragmatic directive -->   </directive>
  <sender>  <!-- sender agent/service -->   </sender>
  <receiver>  <!-- receiver agent/service --> </receiver>
  <payload>  <!-- message payload -->   </payload>
</Message>
```

- *cid is the conversation identifier (enabling also subconverstations)*
- *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 $\mathcal{A}$ with (external) knowledge base $\mathcal{KB}_{\mathcal{A}}$ and public signatures $\mathcal{S}_{\mathcal{A}public} \subseteq \mathcal{S}_{\mathcal{A}}$ and external background structure $\mathcal{I}_{\mathcal{A}}$
  - the universe of $\mathcal{I}_{\mathcal{A}}$ contains the sorts of the **public signatures*** of $\mathcal{A}$ * Note, agent's signature $\mathcal{S}_{\mathcal{A}}$ might contain further private signatures
- **<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

Example: Loosley-Coupled Communication via Messages to Agent Interface

FIPA ACL directive

```
<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>
```

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

Note: the receiver „Agent2“ needs to specify an appropriate <signature> for „likes“

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

Messaging Reaction Rules

```
<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(…) :- rcvMul(…), r(…), … .
```

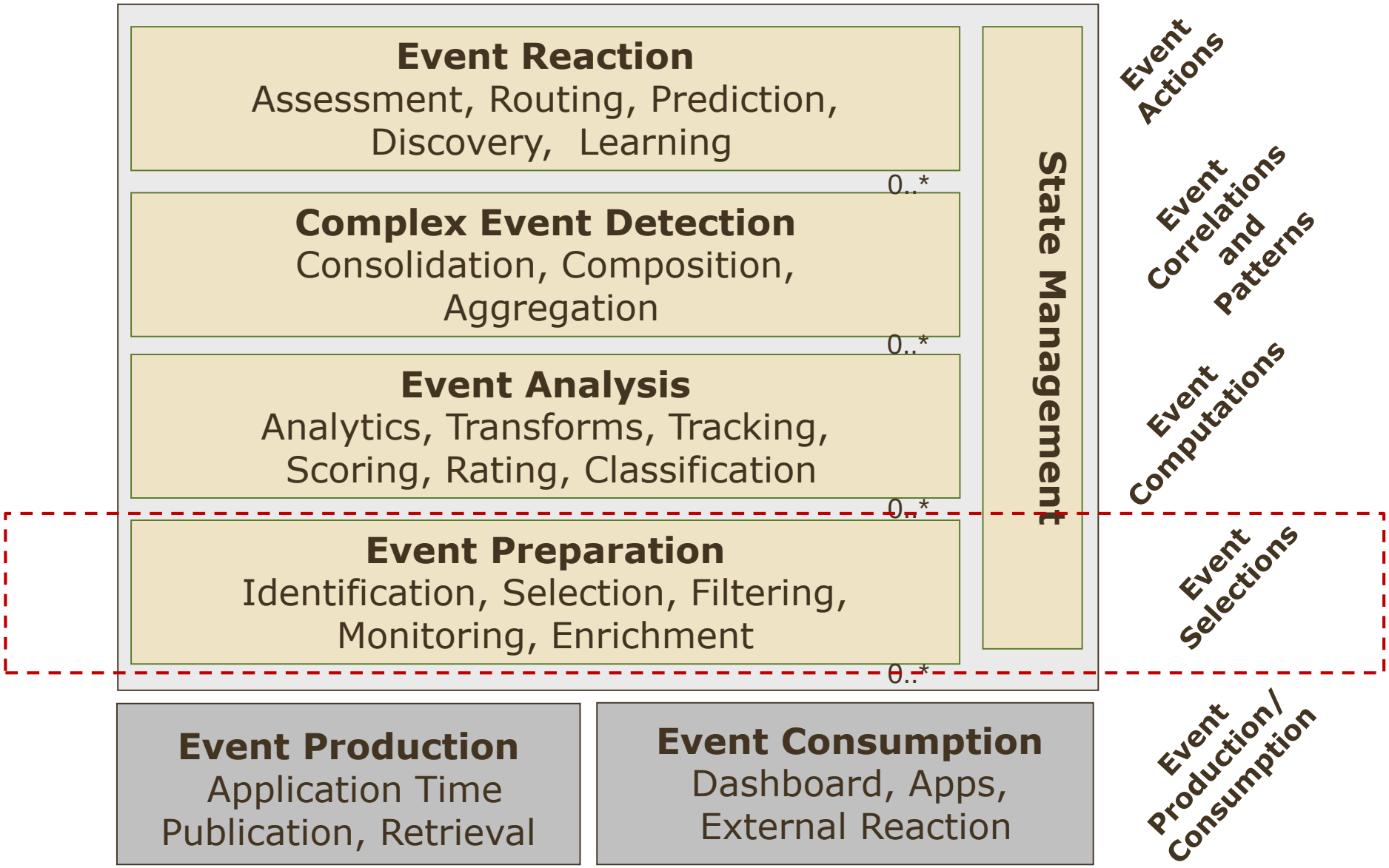
* concurrent transaction logic semantics

- **serial forward execution***: precondition ⊗ sendMsg/rcvMsg/rcvMult ⊗ postcondition
- **backward reasoning**: rcvMsg/rcvMult predicate is interpreted as a **goal literal**
- **proof computations**: **resolvent** clauses are locally managed within a converstation **scope** (defined by CID) waiting for incoming „messages“ with a predicate **unifying** with the postponed rcvMsg/rcvMult goal literal

Agenda

- Introduction to Event Processing
- Event Processing Reference Model and Reference Architecture
- Reaction RuleML Standard
- **Examples for Event Processing Functions**
- Summary

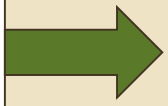
Patterns Coverage



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/epts-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.

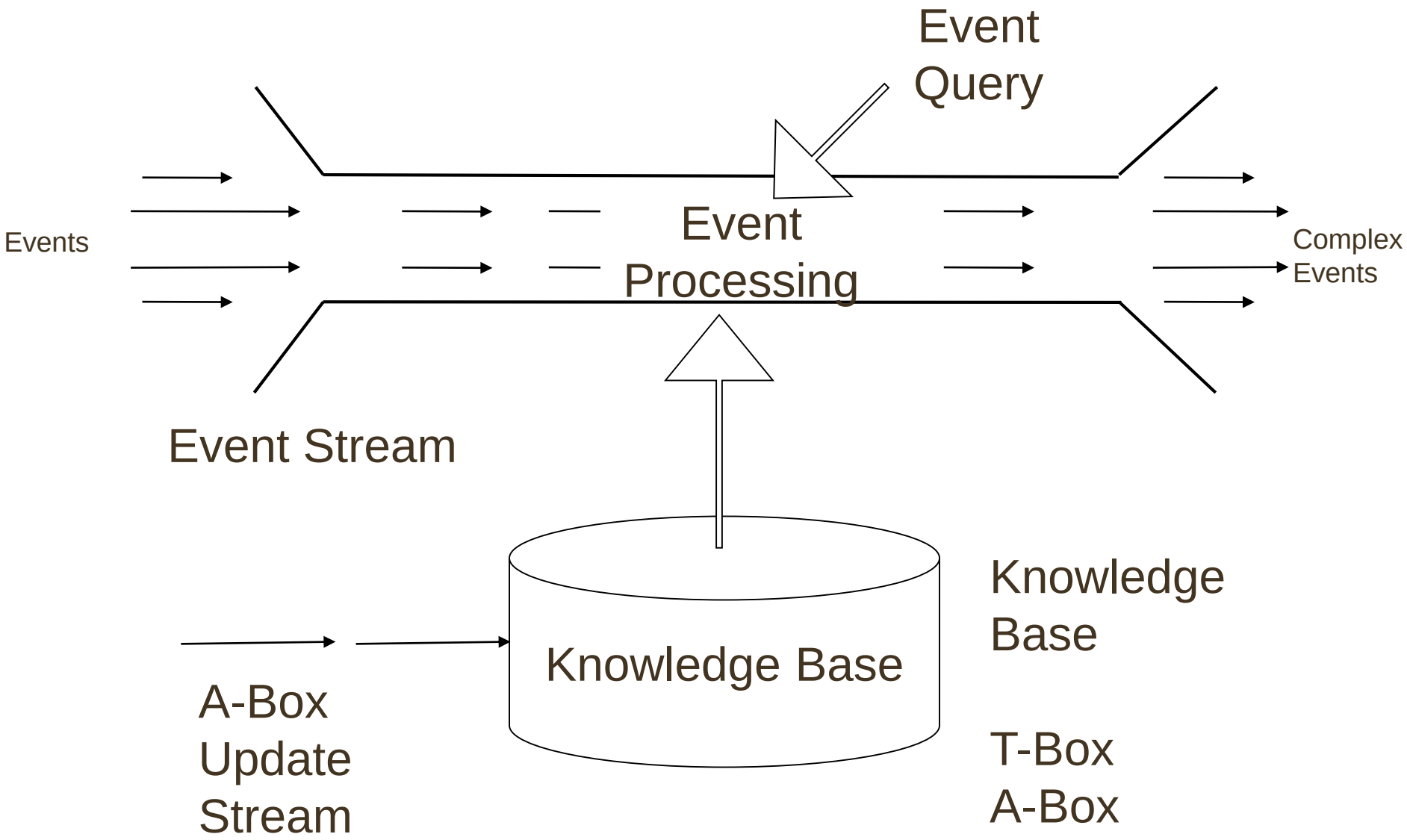


Event Stream – stock quotes
{(Name, "*OPEL*")(Price, 45)(Volume, 2000)(Time, 1) }
{(Name, "SAP")(Price, 65)(Volume, 1000) (Time, 2)}

Semantic Knowledge Base

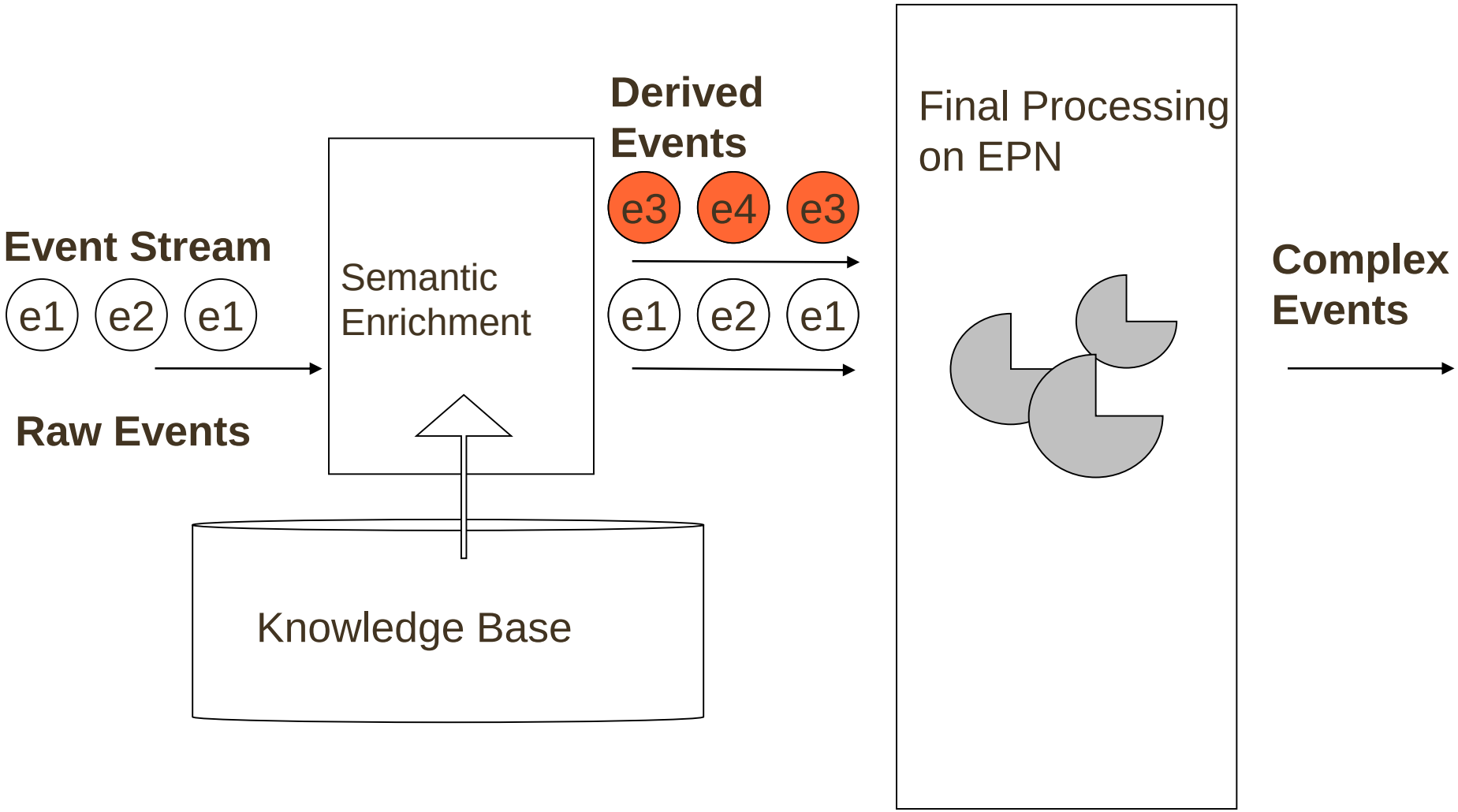
{(*OPEL*, is_a, car_manufacturer),
(car_manufacturer, build, Cars),
(Cars, are_build_from, Metall),
(*OPEL*, hat_production_facilities_in, Germany),
(Germany, is_in, Europe)
(OPEL, is_a, Major_corporation),
(Major_corporation, have, over_10,000_employees)}

Knowledge-based Semantic Complex Event Processing



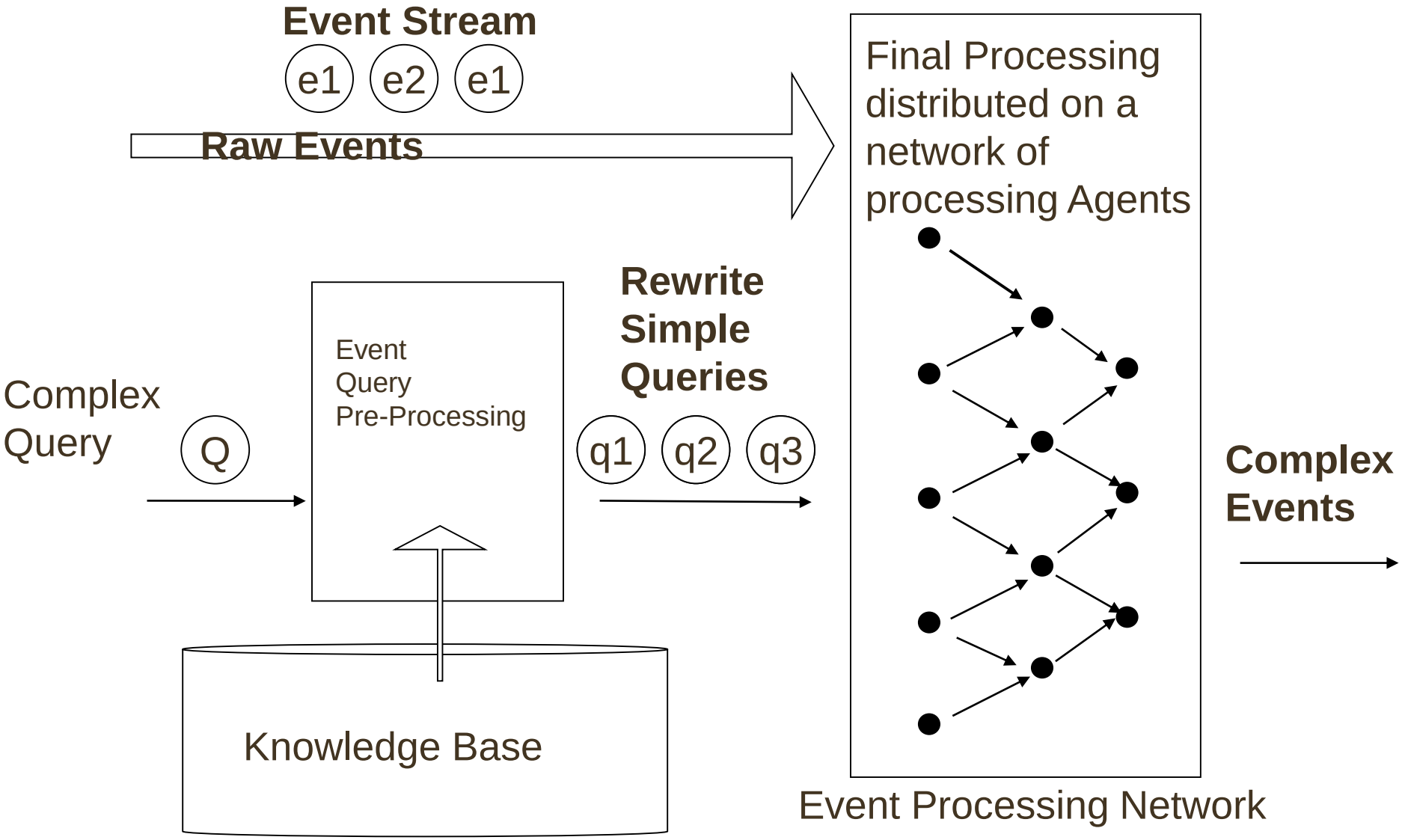
http://corporate-semantic-web.de/semantic-complex-event-processing.html

Example: Semantic Enrichment of Event Stream



Kia Teymourian, Adrian Paschke: **Plan-based Semantic Enrichment of Event Streams**, 11th Extended Semantic Web Conference (ESWC 2014), 25-29 May 2014, Crete, Greece

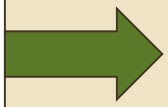
Example: Semantic Event Query Pre-Processing



Preparation:Filter:Implementations: Prova: Semantic Event Processing Example

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> .
```

Preparation:Enrichment:Implementations: Prova: Example with SPARQL Query

```
% 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,"epa1", 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>
        <http://dbpedia.org/resource/Category:Luxury_vehicles> .
                ?car foaf:name ?name .
                ?car dbo:manufacturer ?man .
                ?man foaf:name ?manufacturer.
        } ORDER by ?manufacturer ?name",
sparql_select(Query,manufacturer(Manufacturer),name(Name),car(Car)).
```

Preparation:Enrichment:Implementations: Prova: External Data and Object Integration

- **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"),
```

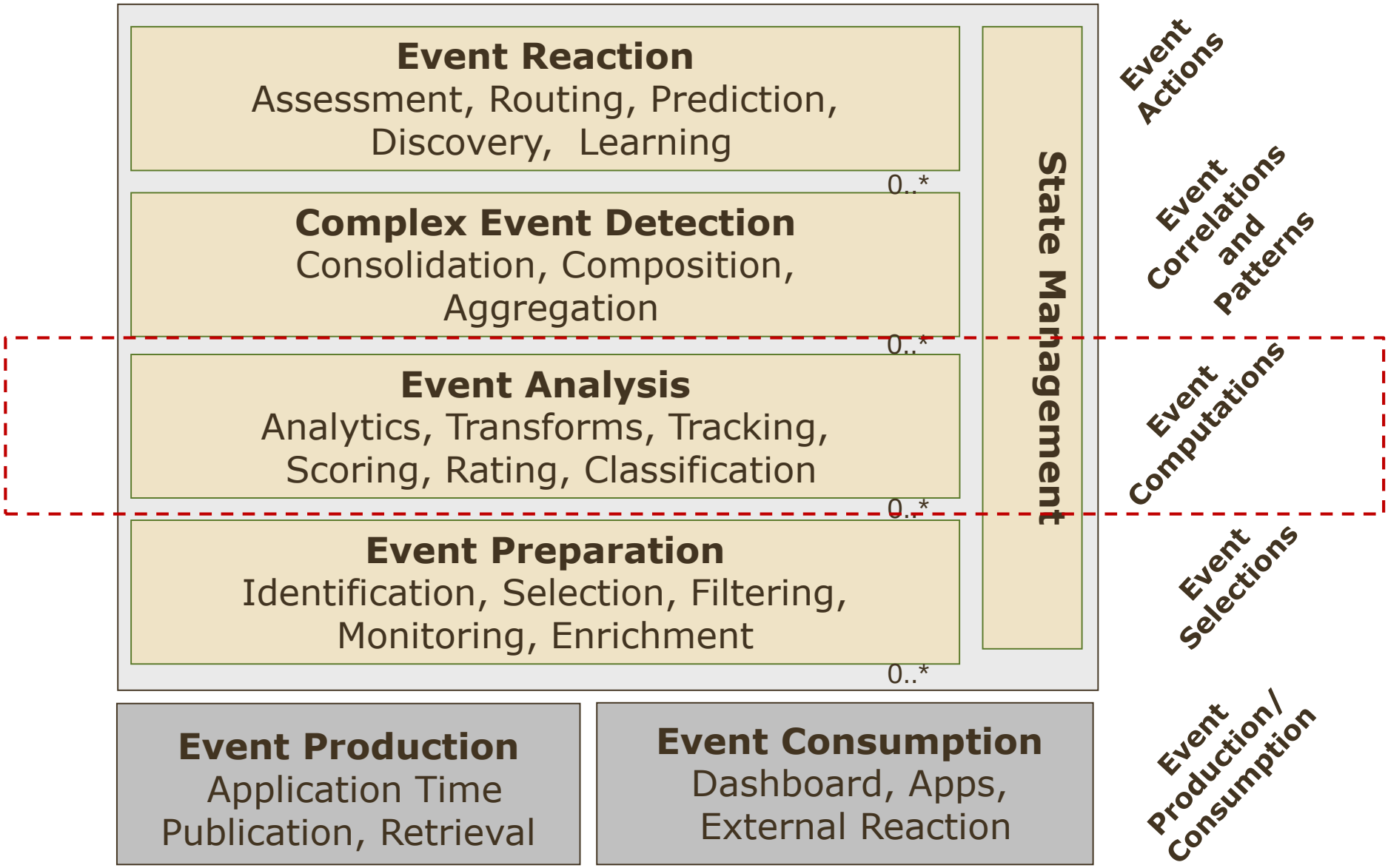
- **XQuery**

```
..., 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)),
```

Patterns Coverage



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/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b

Example - Rating Implementation in Prova: Metadata Scoped Reasoning and Guards

```
% 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
```

Analysis:Analytics:Implementations: Prova: Java Programming and Functional Programming

Prova Dynamic Java Programming

- Ability to embed Java calls
  … `:-` `X = 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 *derive* built-in;
- partial evaluation;
- lambda functions;
- monadic functions;
- monadic *bind* using a composition of *map* and *join*;
- *maybe*, *list*, *state*, *tree* and *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)
```

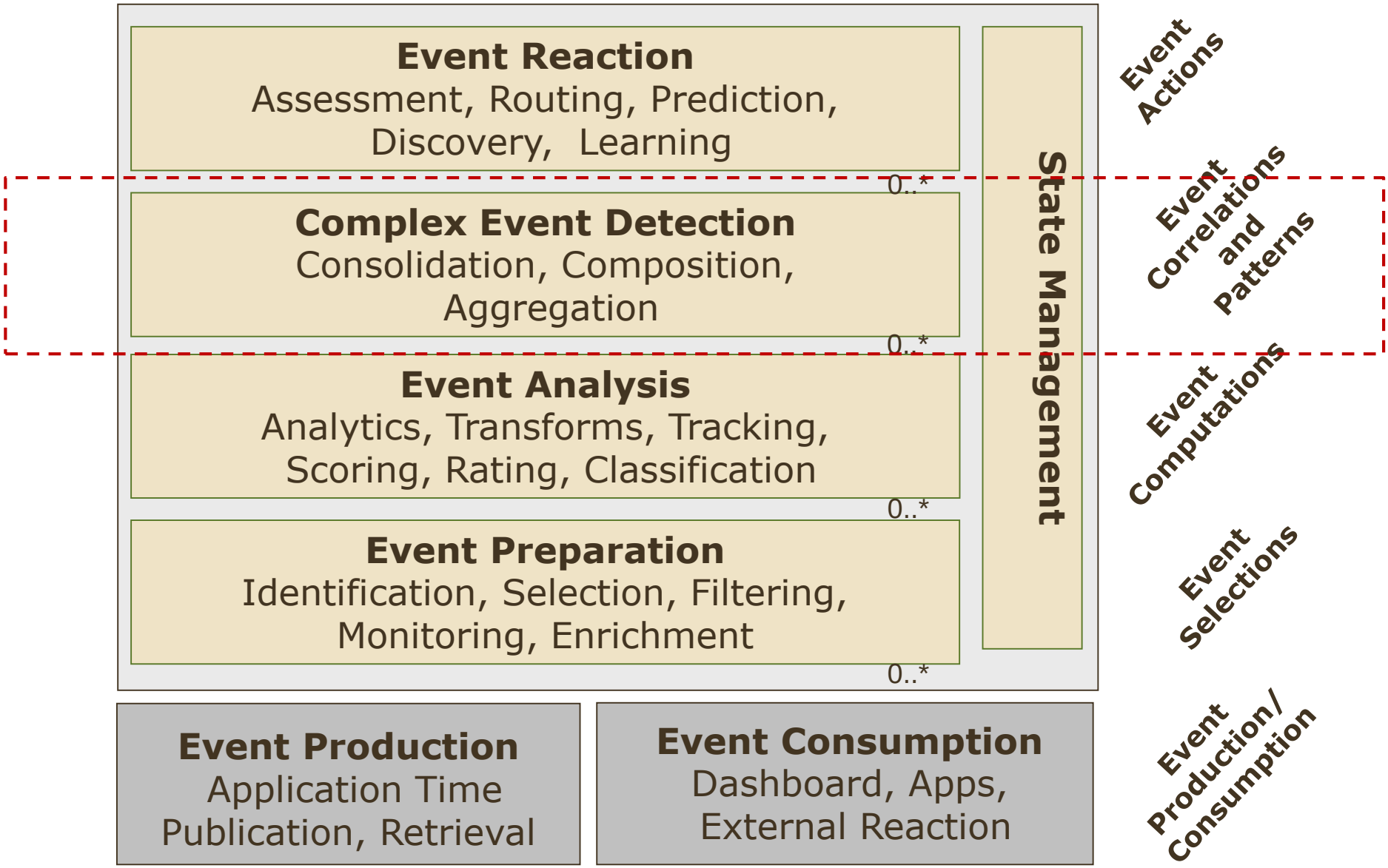
Classification according to Type System

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



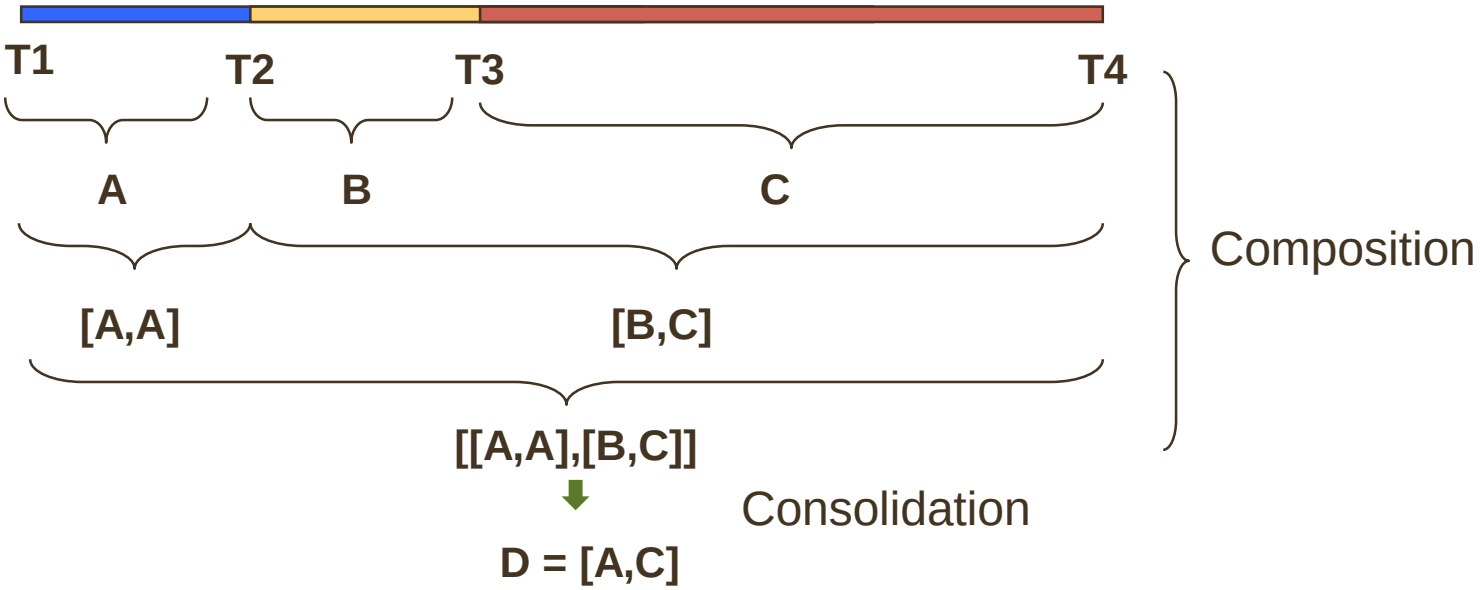
see: Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Tutorial on advanced design patterns in event processing. DEBS 2012: 324-334;
www.slideshare.net/isvana/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b

Detection:Consolidation:Implementations: Prova: Consolidation Example

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

$$I: T_{interval} \times Fl \rightarrow \{true, false\}$$

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



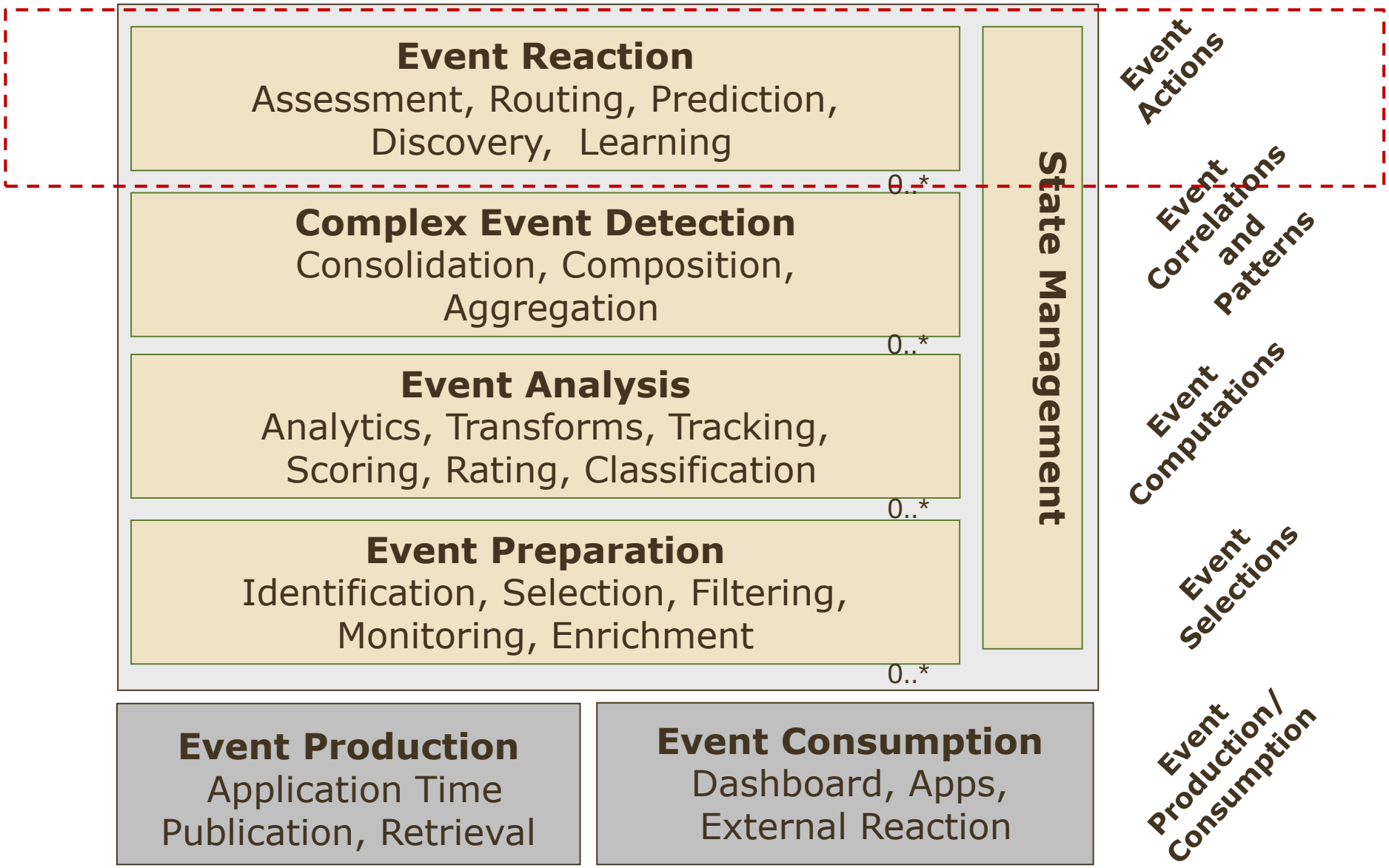
- Example: derived situation from complex event detection (consolidation: initiate situation1 by event D)

```
initiates(D,situation1,T).
holdsAt(situation1, t5)?  => yes
```

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 aggreation object>.
```

Patterns Coverage



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/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b

Reaction:Assesment:Implementations: Prova: Example – Detect Suspicious Logins

```
% detect suspicious logins by assessing the IP numbers
of the login events from the same user login

rcvMsg(XID,Protocol,From,request,login(User,IP)) :-

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

 @group(g1) @count(1)
 rcvMsg(XID,Protocol,From,request,login(User,IP2))
[IP2!=IP],

 println(["Suspicious login",User,IP,IP2]," ").
```

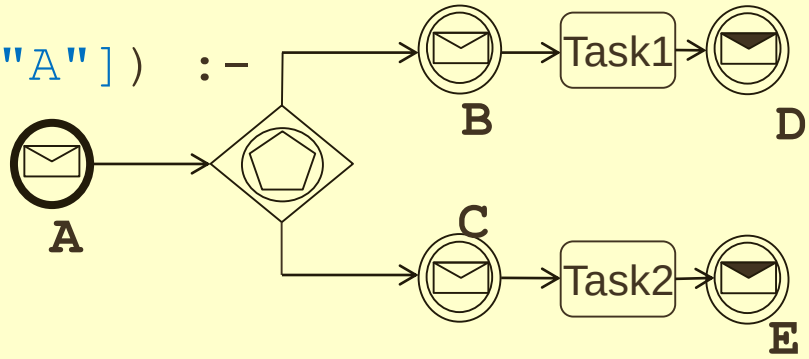
Reaction - Routing
Prova: Example with Agent (Sub-) Conversations

```
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,oder(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 ha a value below …
regular(Product) :- price(Product,Value), Value < 10000.
```

corresponding serialization with Reaction RuleML <Send> and <Receive>

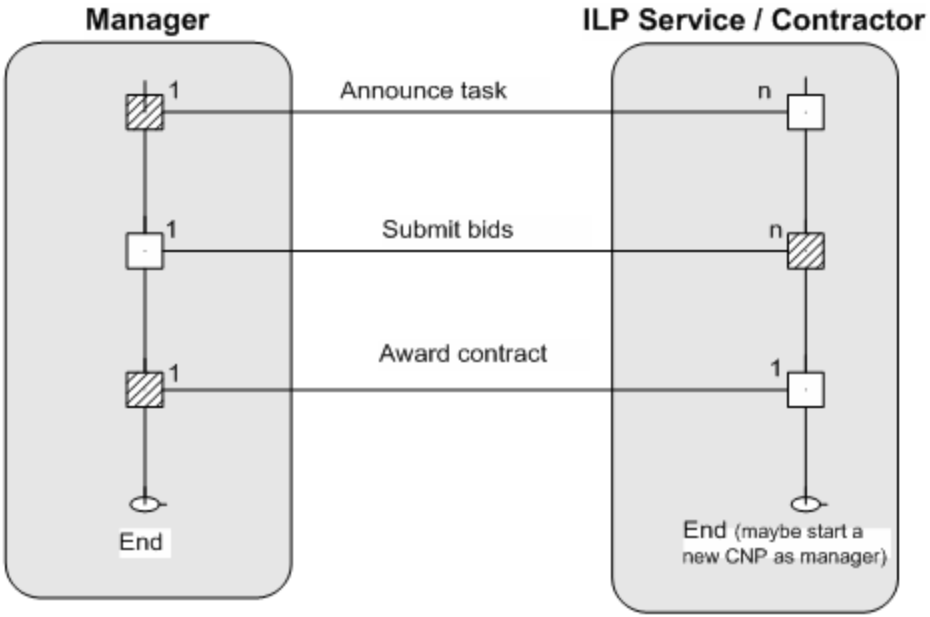
Message Driven Routing
Prova : Event Routing in Event-Driven Workflows

```
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,_).
```

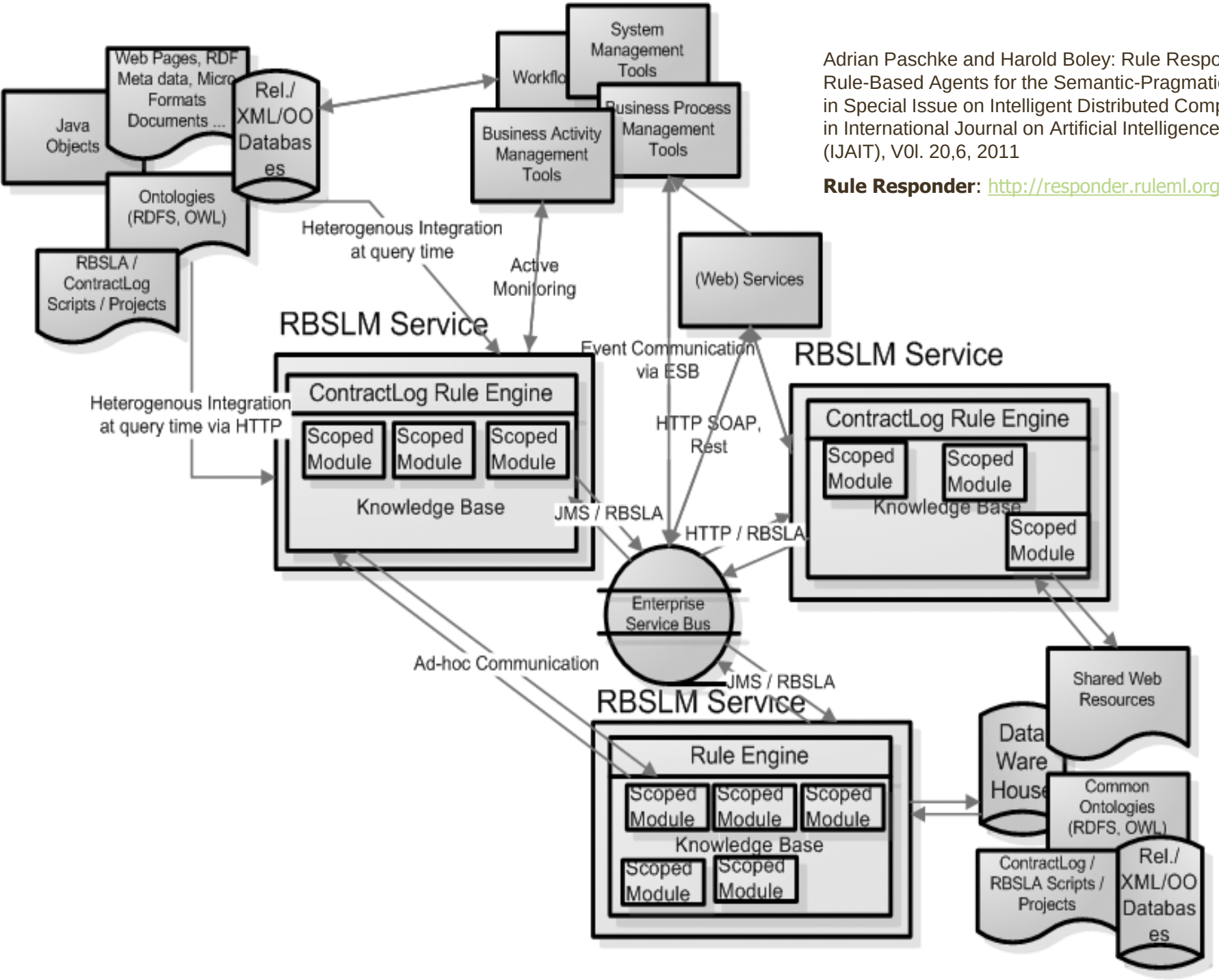


Distributed Rule Base Interchange
Prova: Mobile Code

```
% 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]).
```



Contract Net Protocol



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

Rule Responder: 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**
- **Knowledege 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

- **Paschke, A., Boley, H.: Rules Capturing Event and Reactivity, in Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, ISBN:1-60566-402-2, 2009**
  http://www.igi-global.com/book/handbook-research-emerging-rule-based/465
  Sample Chapter: http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=16435934&article=0&fd=pdf
- **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
  http://www.slideshare.net/isvana/ruleml2011-cep-standards-reference-model
- **Adrian Paschke: The Reaction RuleML Classification of the Event / Action / State Processing and Reasoning Space. CoRR abs/cs/0611047 (2006)**
  http://arxiv.org/ftp/cs/papers/0611/0611047.pdf
- **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
- **Paschke, A., Boley, H.: Rule Markup Languages and Semantic Web Rule Languages, in Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, ISBN:1-60566-402-2, 2009**
  http://www.igi-global.com/chapter/rule-markup-languages-semantic-web/35852
  Sample: Chapter: http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=18533385

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**
- **Harold Boley, Adrian Paschke, Omair Shafiq: RuleML 1.0: The Overarching Specification of Web Rules. RuleML 2010: 162-178**
  http://dx.doi.org/10.1007/978-3-642-16289-3_15
  http://www.cs.unb.ca/~boley/talks/RuleML-Overarching-Talk.pdf
- **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
- **Paschke, A., Boley, H.: Rule Markup Languages and Semantic Web Rule Languages, in Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, ISBN:1-60566-402-2, 2009**
  http://www.igi-global.com/chapter/rule-markup-languages-semantic-web/35852
  Sample chapter: http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=18533385
- **Paschke, A., Boley, H.: Rules Capturing Event and Reactivity, in Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, ISBN:1-60566-402-2, 2009**
  http://www.igi-global.com/book/handbook-research-emerging-rule-based/465
  Sample Chapter: http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=16435934&article=0&fd=pdf
- **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**

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
- **Adrian Paschke, Paul Vincent, Alexandre Alves, Catherine Moxey: Architectural and functional design patterns for event processing. DEBS 2011: 363-364** http://www.slideshare.net/isvana/epts-debs2011-event-processing-reference-architecture-and-patterns-tutorial-v1-2
- **Paschke, A., Vincent, P., and Catherine Moxey: Event Processing Architectures, *Fourth ACM International Conference on Distributed Event-Based Systems* (DEBS '10). ACM, Cambridge, UK, 2010.** http://www.slideshare.net/isvana/debs2010-tutorial-on-epts-reference-architecture-v11c
- **Paschke, A. and Vincent, P.: A reference architecture for Event Processing. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (DEBS '09). ACM, New York, NY, USA, 2009.**
- **Francois Bry, Michael Eckert, Opher Etzion, Adrian Paschke, Jon Ricke: Event Processing Language Tutorial, DEBS 2009, ACM, Nashville, 2009** http://www.slideshare.net/opher.etzion/debs2009-event-processing-languages-tutorial
- **Adrian Paschke. A Semantic Design Pattern Language for Complex Event Processing, AAAI Spring Symposium "Intelligent Event Processing", March 2009, Stanford, USA** http://www.aaai.org/Papers/Symposia/Spring/2009/SS-09-05/SS09-05-010.pdf
- **Paschke, A.: Design Patterns for Complex Event Processing, 2nd International Conference on Distributed Event-Based Systems (DEBS'08), Rome, Italy, 2008.** http://arxiv.org/abs/0806.1100v1

Further Reading – Rule-Based Semantic CEP

- **Corporate Semantic Web – Semantic Complex Event Processing**
  - http://www.corporate-semantic-web.de/semantic-complex-event-processing.html
- **Adrian Paschke: ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics CoRR abs/cs/0610167: (2006); http://arxiv.org/ftp/cs/papers/0610/0610167.pdf**
- **Adrian Paschke: Semantic Rule-Based Complex Event Processing. RuleML 2009: 82-92 http://link.springer.com/chapter/10.1007%2F978-3-642-04985-9_10**
- **Adrian Paschke, Alexander Kozlenkov, Harold Boley: A Homogeneous Reaction Rule Language for Complex Event Processing, In Proc. 2nd International Workshop on Event Drive Architecture and Event Processing Systems (EDA-PS 2007) at VLDB 2007; http://arxiv.org/pdf/1008.0823v1.pdf**
- **Adrian Paschke: ECA-LP / ECA-RuleML: A Homogeneous Event-Condition-Action Logic Programming Language CoRR abs/cs/0609143: (2006); http://arxiv.org/ftp/cs/papers/0609/0609143.pdf**
- **Teymourian, K., Coskun, G., and Paschke, A. 2010. Modular Upper-Level Ontologies for Semantic Complex Event Processing. In Proceeding of the 2010 Conference on Modular ontologies: Proceedings of the Fourth international Workshop (WoMO 2010) O. Kutz, J. Hois, J. Bao, and B. Cuenca Grau, Eds. Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, The Netherlands, 81-93. http://dl.acm.org/citation.cfm?id=1804769**
- **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://ceur-ws.org/Vol-207/paper01.pdf

- **Adrian Paschke, Jens Dietrich, Adrian Giurca, Gerd Wagner, Sergey Lukichev: *On Self-Validating Rule Bases,* Proceedings of 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), Athens, Georgia, USA (6th November 2006), November 2006**
  http://www.rewerse.net/publications/rewerse-description/REWERSE-RP-2006-166.html
  http://km.aifb.kit.edu/ws/swese2006/final/paschke_full.pdf
  http://www.researchgate.net/publication/228359327_On_self-validating_rule_bases

- **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

- **Prova Rule Engine** http://www.prova.ws/
  - Prova 3 documentation http://www.prova.ws/index.html?page=documentation.php
    - Journal: 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
  - **Prova 3 Semantic Web Branch**
    - Paper: Paschke, A.: A Typed Hybrid Description Logic Programming Language with Polymorphic Order-Sorted DL-Typed Unification for Semantic Web Type Systems, OWL-2006 (OWLED'06), Athens, Georgia, USA.
    - Prova 3 version with Semantic Web support on GitHub (https://github.com/prova/prova/tree/prova3-sw)
  - Prova ContractLog KR: http://www.rbsla.ruleml.org/ContractLog.html
    - Paschke, A.: Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management, Book ISBN 978-3-88793-221-3, Idea Verlag GmbH, Munich, Germany, 2007. http://rbsla.ruleml.org
  - Prova CEP examples: http://www.slideshare.net/isvana/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b