A Framework and Components for ECA Rules in the Web (Demo)

Erik Behrends, Oliver Fritzen, Wolfgang May, Franz Schenk, Daniel Schubert
Institut für Informatik, Universität Göttingen, Germany
{behrends,fritzen,may,schenk,schubert}@informatik.uni-goettingen.de

The ECA Framework

Event-Condition-Action (ECA) rules are a popular paradigm for specifying behavior: "ON event IF condition DO action" has a clear declarative semantics and induces an immediate operational realization. The core of our approach for ECA rules in the Web and the Semantic Web is a model and architecture for ECA rules that uses *heterogeneous* event, query, and action languages. The condition component is divided into queries (that can be expressed in different languages) and a test component (cf. Figure 1):

ON *event* AND *additional knowledge*, IF *condition*
THEN DO *something*.

For dealing with heterogeneous languages, the approach is parametric in the used component languages. Users register rules in the ECA-ML language [3] at an ECA service in the Web that provides the infrastructure and global rule semantics:

```
<eca:rule xmlns:eca="http://.../eca/2006/eca-ml">
  <eca:event>...</eca:event>
  <eca:query>...</eca:query>
  <eca:test>...</eca:test>
  <eca:action>...</eca:action>
</eca:rule>
```

For the rule components, the users may use component languages of their choice. The markup of the rules indicates the "language borders" between the ECA level and the nested components by their namespaces. The components are specified as nested subexpressions of the form

```
<eca:component xmlns:lang="embedded-lang-ns-uri">
  embedded fragment in embedded language's
  markup and namespace
</eca:component>
```

in suitable event, query, or action formalisms or languages. The approach does only minimally constrain the component languages: Information flow between the ECA engine and the event, query, test, and action components is provided by *logical variables* in the style of deductive rules, production rules etc. Thus, languages following a functional style (such as XPath/XQuery), a logic style (such as SPARQL, or both (F-Logic) can be used as query languages. The semantics of the event part (that is actually a "query" against an event stream that is evaluated incrementally, usually specified by some event algebra) is –from that point of view– very similar. The action component, specified e.g. by a process algebra, takes variable bindings as input.

For processing the components, the ECA engine determines a *language processor node* for the indicated specification language, and submits the task to that node. A detailed description of the ECA engine can be found in [2]. In the meantime, component services for handling atomic events, composite events based on the SNOOP event algebra, *opaque* language services, i.e., data sources that are based on XPath, XQuery or SPARQL, and atomic actions have been integrated. A component for processing CCS-based action specifications [1] is under implementation. Domain nodes can use a node architecture based on Jena extended with triggers [5]; a domain broker is under implementation. The service identification based on the language's namespace URIs is done by a *Language&Service Registry*. An online prototype can be found at `http://www.semwebtech.org/eca/frontend`.

Figure 2 illustrates the overall communication between different kinds of language services and domain services: A client registers a rule (e.g., in the travel domain) at the ECA engine (Step 1.1). It submits the event component to the appropriate *composite event detection* service (1.2), here, a SNOOP service. The SNOOP engine registers all atomic event patterns at the appropriate *atomic event matcher* (AEM) (1.3). The AEM looks at the namespaces of the atomic events and sees that the travel ontology is relevant. It contacts a travel domain broker (1.4) that keeps it informed (2.2) about atomic events (e.g., happening at Lufthansa (2.1a) and SNCF (2.1b)). The AEM matches the events against the registered patterns, and in case of a success, reports the matched event and the extracted variable bindings to the SNOOP service (3). Only after detection of a registered composite event, SNOOP submits the result to the ECA engine (4) that then evaluates the query and test components, and submits the action to the CCS service (5.1) that in turn submits the atomic actions to the domain broker and other services (5.2 and 5.3).

The global architecture and the general markup principles are described in [4] and [3].

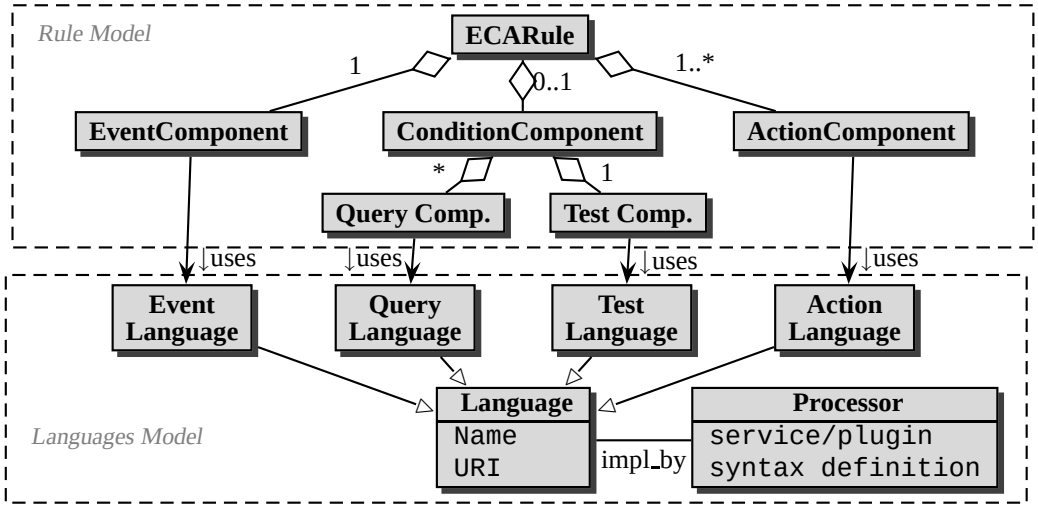

Figure 1. ECA Rule Components and Corresponding Languages (from [3])

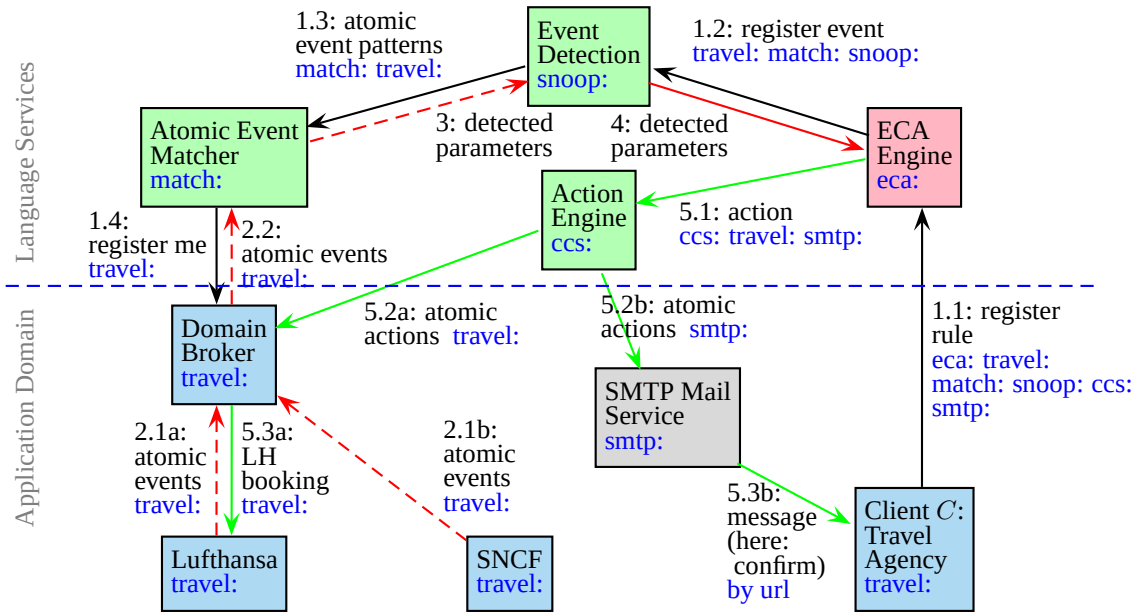

Figure 2. Language-Centered Communication

Acknowledgements. This research has been partially funded by the European Commission within the 6th Framework Programme project REWERSE, no. 506779.

References

[1] E. Behrends, O. Fritzen, W. May, and F. Schenk. Combining ECA Rules with Process Algebras for the Semantic Web. In *Rule Markup Languages (RuleML)*, to appear with IEEE, 2006.

[2] E. Behrends, O. Fritzen, W. May, and D. Schubert. An ECA Engine for Deploying Heterogeneous Component Languages in the Semantic Web. In *Web Reactivity (EDBT Workshop)*, Springer LNCS 4254, 2006.

[3] W. May, J. J. Alferes, and R. Amador. Active rules in the Semantic Web: Dealing with language heterogeneity. *Rule Markup Languages (RuleML)*, Springer LNCS 3791, 2005.

[4] W. May, J. J. Alferes, and R. Amador. An ontology- and resources-based approach to evolution and reactivity in the Semantic Web. In *Ontologies, Databases and Semantics (ODBASE)*, Springer LNCS 3761, 2005.

[5] W. May, F. Schenk, and E. v. Lienen. Extending an OWL Web node with reactive behavior. In *Principles and Practice of Semantic Web Reasoning (PPSWR)*, Springer LNCS 4187, 2006.