Lifting Events in RDF from Interactions with Annotated Web Pages

Roland Stühmer[1], Darko Anicic[1], Sinan Sen[1], Jun Ma[1], Kay-Uwe Schmidt[2], and Nenad Stojanovic[1]

[1] FZI Research Center for Information Technology
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
{roland.stuehmer,darko.anicic,sinan.sen,jun.ma,nenad.stojanovic}@fzi.de
http://www.fzi.de/
[2] SAP AG, Research
Vincenz-Prießnitz-Straße 1, 76131 Karlsruhe
kay-uwe.schmidt@sap.com
http://www.sap.com

Abstract. In this paper we present a method and an implementation for creating and processing semantic events from interaction with Web pages which opens possibilities to build event-driven applications for the (Semantic) Web. Events, simple or complex, are models for things that happen e.g., when a user interacts with a Web page. Events are consumed in some meaningful way e.g., for monitoring reasons or to trigger actions such as responses. In order for receiving parties to understand events e.g., comprehend what has led to an event, we propose a general event schema using RDFS. In this schema we cover the composition of complex events and event-to-event relationships. These events can then be used to route semantic information about an occurrence to different recipients helping in making the Semantic Web active. Additionally, we present an architecture for detecting and composing events in Web clients. For the contents of events we show a way of how they are enriched with semantic information about the context in which they occurred. The paper is presented in conjunction with the use case of Semantic Advertising, which extends traditional clickstream analysis by introducing semantic short-term profiling, enabling discovery of the current interest of a Web user and therefore supporting advertisement providers in responding with more relevant advertisements.

Keywords: Complex Event Processing, Event Representation, Semantic Advertising, User Profiling, RDFa, event-condition-action, ECA.

1 Introduction

Currently the Semantic Web largely obeys a request/response style of communication. This is owed to the predominating way in which the traditional Web is used. However, the Active Web is making progress demonstrated by applications such as Twitter which are able to *push* information to recipients at the

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 893–908, 2009.

time when the information is created. Event-driven ways of disseminating data were identified as being efficient as well as intuitive for many business applications e.g., in [1]. The (Semantic) Web of the future will also benefit from active technologies which help deliver information on time and in a resource saving manner i.e., without having to poll for new or changed information.

In this paper we propose an architecture for capturing and processing complex events on the Web, based on the use of Semantic Web technologies, making it suitable for the future (Active) Semantic Web. As part of this we produce an event representation which facilitates mutual understanding of events on the Semantic Web. Additionally, we demonstrate the benefits of event processing and evaluate them in the field of Semantic Advertising as a use case scenario.

To achieve this, we assess Semantic Web technologies to enable reactivity and design and implement a client-side framework to generate and process events. Furthermore, we demonstrate this framework in the aforementioned advertising use case. The main contributions of this work will be an extensible representation for events on the (Semantic) Web, as well as an implementation of a client-side framework to create these events based on user interactions with Web documents as a source of events. These contributions serve to advance the state of reactivity on the Web and promote new ways of efficiently communicating Web-based information, which we see as a necessary factor for future Semantic Web applications.

This paper is structured as follows: In Section 2 we will describe the requirements for generating meaningful events from Web documents, followed by means of processing these events to detect complex situations. In subsequent sections we will name key parts and technologies for our architecture such as a client-side reactive rule language in Section 3, an RDFS ontology for representing complex events in Section 4 and a way of obtaining meaningful context for events from annotations in Section 5. We will then present implementation details of our architecture in Section 6. The whole approach will be evaluated for performance and usefulness in the advertising scenario in Section 7 and we will discuss related work and conclude the paper in the last remaining sections.

2 Event Generation and Processing from Semantic Web Pages

The main issue in making the Web active is to enable capturing of actions or changes in Web documents. These can be treated as events, which an event-driven system will react to. For our use case of advertising we will focus on events created from a user's interaction with Web documents. After having extracted events from a Web document, they must be processed in order to interpret them semantically, to be able to react on them appropriately. The following two subsections describe our approach for these two issues: generation and processing of Web events.

2.1 Event Generation

A simple event in Web clients is characterized by two dimensions; the type of event (e.g. click, mouseover) and the part of the Web page, where the event occurred (e.g. a node of the Document Object Model of the Web document). This node is, however, just a syntactical artifact of the document as it is presented in a Web browser. Adding this node or parts of it to the event body will not significantly add meaning to the event and not ease the understanding of the event for the recipient of the event.

We therefore propose to add semantic information to the event which pertains to the actual domain knowledge that the Web page is about. In order to enable this, the first step is to represent the content of a Web page in a form that can be used for generating meaningful events. To do so without having to manually annotate every Web document, we envision a mechanism, which ensures the relevance of the annotations. This can be done in many (semi-) automatic ways, e.g. by providing Web forms (page templates), which for a given user's input, automatically adds the proper semantic relationships between the form fields. In this way all user generated content will be annotated. The Web forms are created based on supported vocabularies for a particular Web site. Our particular focus is on widely spread vocabularies such as Dublin Core[1], Creative Commons [2], FOAF[3], GeoRSS[4] and OpenCalais[5]. Regarding the format of structured data, RDFa [2], eRDF[6] and Microformats[7] are all good candidates for this purpose. They support semantics embedded within actual Web page data and allow reusable semantic markup inside of Web pages. In our implementation we use RDFa, since in comparison to eRDF it is a more encouraged candidate by the W3C. Comparing it further to Microformats, RDFa is more flexible in mixing different existing vocabularies.

In the remaining part of this section we give an example demonstrating the generation of events in the context of a Semantic Advertising scenario. The ad space is a part of the Web page which can be dynamically filled by an ad provider as a response to an event the client sends. In our approach ad content is created based on a current user's attention. In order to accomplish this we need as much (meta-) information as possible about the content of the Web page. Therefore, we assume semantically enriched Web content such that context extraction is easier and more precise. Additionally, every page is split up in a number of *Semantic Web Widgets* (SWW). We introduce *Semantic Web Widgets* as self-contained components annotated with semantic data and displayed in a Web page. Semantic Web Widgets give a high-level description of the content, and provide the basic context of data contained in the widgets. For instance on a

[1] Dublin Core: `http://dublincore.org`
[2] Creative Commons: `http://creativecommons.org`
[3] FOAF: `http://foaf-project.org`
[4] GeoRSS: `http://georss.org`
[5] OpenCalais: `http://opencalais.com`
[6] eRDF: `http://research.talis.com/2005/erdf`
[7] Microformats: `http://microformats.org`

```
<div xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:dc = "http://purl.org/dc/elements/1.1/"
     xmlns:vCard = "http://www.w3.org/2001/vcard-rdf/3.0#"
     xmlns:iCal = "http://www.w3.org/2002/12/cal/ical#">
    <ul about="events/Mary_Poppins_Show">
      <li typeof="cal:Vevent">
        <a property="ical:categories">Classic, Comedy, Kid Friendly, Musical</a>
        <a property="cal:dtstart" content="20081008T180000Z">October 8th at 18am</a>
        <a property="cal:duration" content="PT2H">2 hour</a>
        <vCard:TEL rdf:parseType="Resource">
         <rdf:value>(212) 307-4100</rdf:value>
         <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#work"/>
        </vCard:TEL>
        <vCard:ADR rdf:parseType="Resource">
         <vCard:Street>   214 West 42nd Street </vCard:Street>
         <vCard:Locality> New York City </vCard:Locality>
         <vCard:Pcode>    NY 10036 </vCard:Pcode>
         <vCard:Country>  USA </vCard:Country>
        </vCard:ADR>
        <a property="cal:description">Mary Poppins takes up residence at
            magnificent New Amsterdam Theater.
        </a>
      </li>
    </ul>
</div>
```

Listing 1. An example for a musical listed in a Semantic Web Widget

news portal incorporating semantic advertising one widget could be used for listing all news belonging to one subcategory, e.g., politics, another one for arts, etc..

In Figure 1 we show an RDFa example of the semantic description for an arts event[8] listed in a widget related to musicals. The code snippet presents an event named "Mary_Poppins_Show" described using RDF Schemata for Dublin Core, vCard and iCal vocabularies. Information such as categories, start and duration of the musical are provided together with contact information, location and so on.

2.2 Complex Event Processing

Simple events extracted from Web documents can be combined in order to detect complex situations. This is the task of Complex Event Processing, that we describe in the context of the Semantic Advertising use case. Detecting the behavior of Web users according to our proposal is divided into design time and run time. The design time consists of (i) semantically enhancing the Web page and then (ii) recording average viewing statistics of the annotated elements, e.g. from log files. From the statistical data we generate client-side rules. Once these rules are created they are pulled by the next client request and loaded into the rule engine for the run time.

For the run time we have developed a client-side event-condition-action (ECA) rule engine. It uses a lightweight rule language which supports ECA rules

[8] An *event* in the sense of a gathering of people.

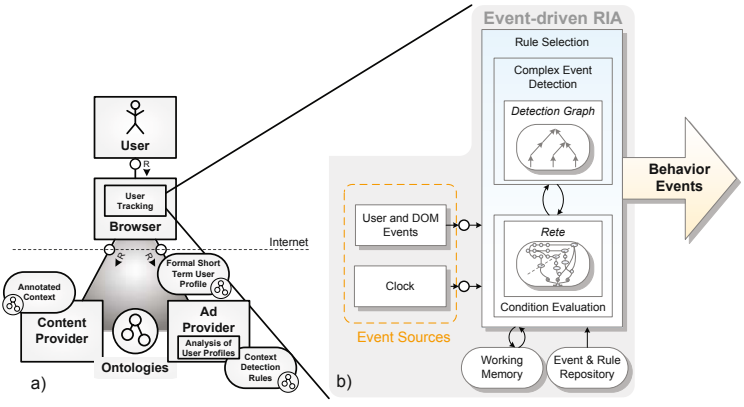

Fig. 1. Architecture: a) Logical Architecture b) Client-side User Behavior Analysis

described in more detail in Section 3. The rules on the client serve to detect users exhibiting interesting behavior as learned from the average usage patterns. The user causes events to occur by interacting with the Web page, detected by the event processor and rule engine. Rules are triggered which create intermediate events in a hierarchy of event abstraction. These events are subsequently accumulated until sufficient interest according to the ad provider is recorded (threshold achieved) and actions can be taken by further rules.

The distinction between run time and design time in this section is not a strict temporal distinction as the names would suggest. Rather, because new users will inevitably alter our knowledge of what is interesting there is a loop in the process, feeding back from the run time into the design time to evolve new rules for future users.

Figure 1 shows a rough architecture of our approach: Part b) on the right hand side of the figure depicts the components of our client-side rule engine. Multiple event sources provide input for the event detection, creating complex events. Also, a working memory submits its changes to a Rete network, evaluating rule conditions. The logic for both the event detection and condition evaluation is supplied by rules from a repository, generated from past user activities. Part a) on the left hand side places the client-side components above the protocol boundary dividing client and server. Below on the server or several distributed servers hold the Web content as well as the advertising content. The Web content is annotated, providing semantical relations to the advertisements. Short-term user models provide a temporal model of how a user interacts with the Web content. The ad provider analyses user models to provide up-to-date and personalized advertisements.

In the next three sections we present in detail three mechanisms which enable the realization of the approach we present in Section 6.

3 JSON-Rules: A Client-Side Rule Language

To facilitate client-side advertisement we use JSON-Rules, our client-side rule language. It resembles a lightweight reaction rule language tailored to the needs of Rich Internet Applications, specifically applications that profit from or require Complex Event Processing, condition evaluation on a working memory, and running rule actions written in JavaScript. As a representation for our rules we use JSON[9], because it is natively usable within JavaScript. JSON can specify objects, arrays and primitives. Rule objects in our JSON-Rules language contain the three attributes `event`, `condition` and `action`. The event part consists of patterns in the event pattern language Snoop [3]. The condition part consists of conjunctive predicates over variables from a working memory. The action part in turn contains one or more JavaScript code blocks to gain a maximum degree of versatility for the rule author. Alternatively for rule actions we offer to trigger certain desired events as well as manipulations of the working memory. The latter types of action offer greater declarativity while formulating rules. This increase is, however, bought at the price of some flexibility. Thus, we still offer all three kinds of rule actions which can be freely mixed.

For the event part of each rule the usual Snoop operators are available: $Or(E_1, E_2)$, $And(E_1, E_2)$, $Any(m, E_1, E_2, \ldots)$, $Seq(E_1, E_2)$, $A(E_1, E_2, E_3)$, $A^*(E_1, E_2, E_3)$, $P(E_1, TI[\text{:parameters}], E_3)$, $P^*(E_1, TI\text{:parameters}, E_3)$, $Not(E_1, E_2, E_3)$, and $Plus(E_1, TI)$. We only briefly list them here, their semantics are documented in [3]. Additionally we define further event operators $Mask(E_1, \text{condition})$ and $Thres(E_1, \text{threshold})$ as follows. The operator $Filter$ enforces a condition on each occurrence of events E_1. This allows e.g. for fine-grained content-based filtering/masking of events. The operator $Thres$ is another content-based operator which we need to extend the Snoop algebra with. $Thres(E_1, \text{threshold})$ accumulates the events of type E_1 until the boolean function `threshold` returns true, releasing all accumulated events as a complex event and starting accumulation anew.

A condition in our language may use comparison operators on facts from the working memory and literal values. The condition part is a conjunction of predicates. Comparison operators are $<$, $>$, $=$, $<=$ and $>=$. Variables specify items from the working memory. For the advertising use case conditions are not needed, the current rules react purely to events.

Rule actions are JavaScript code blocks or new events or modifications to the working memory to be triggered on rule execution. A code block has access to the set of events and facts that has led to the firing of the rule. Thus, rule authors may create applications that do calculations on the parameters of the collected events and matched condition variables.

Listing 2 shows an example rule. It can be automatically created from analyzing histories of interesting behavior. The only requirement is knowledge, that e.g. states that only two percent of users look at a politics item followed by a science item. The actual rule consists of an event part starting at line 5 and an

[9] JavaScript Object Notation: `http://www.json.org/`

```
1  {
2    "meta": {
3      "rule": "Politics->Science=>2%"
4    },
5    "event": {
6      "type": "SEQ",
7      "children": [
8        {
9          "type": "DOM",
10         "selector": "div[property=dc:keywords][content?=politics]",
11         "event": "click"
12       },
13       {
14         "type": "DOM",
15         "selector": "div[property=dc:keywords][content?=science]",
16         "event": "click"
17       }
18     ]
19   },
20   "action": [
21     {
22       "type": "EVENT",
23       "trigger": "unusual",
24       "parameters": {"probability": 0.02}
25     }
26   ]
27 }
```

Listing 2. Example of a single Rule

action part starting at line 20. The rule resembles an event-condition-action rule where the condition is left blank, i.e. is always true.

The event part in this example describes a sequence of two sub-events. Both sub-events are of type "DOM" which means they are adding handlers to the Web page. In this case each one listens to clicks on DIV elements in the document object model (DOM) where the keywords politics and science are annotated. The rule action is of type "EVENT" which means the rule raises another event. The event to be created is called "unusual" and carries a parameter containing a probability. This event can be subscribed to by further rules. In our case there is a rule aggregating all events of this type until enough unusualness (in terms of aggregated probability) is observed. This example rule is a small part of our whole architecture [4,5] which detects, aggregates and finally submits the user profile in form of one or more complex events.

4 RDFS

In this chapter we present an RDFS ontology for events. It covers a schema for simple events and for modeling derived events such as events composed of one or more simpler event occurrences. The basic classes of the ontology are shown in Figure 2.

An event in terms of our ontology is either a simple event or a complex event. All events have a type, some timestamps and may contain a body (also called payload). A simple event is an instantaneous occurrence where start and end times are equal. A complex event is composed of one or more events which means that

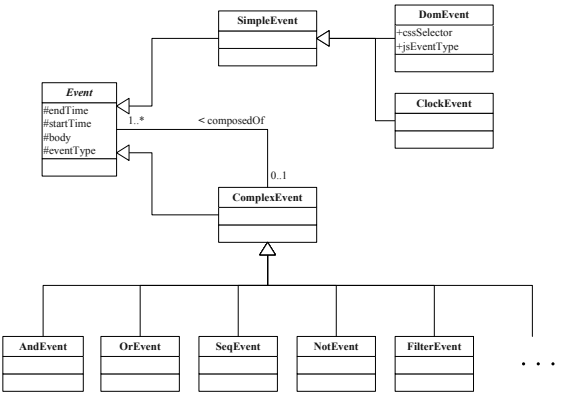

Fig. 2. RDFS ontology for events. The composition of complex events is conceptualized via classes with the names of the operators used in their detection. For simple events we currently only require the subclasses `DomEvent` for user interaction with a Web page and `ClockEvent` for purely temporal events such as timers, reminders, etc..

a complex event can occur over an interval of time, where start and end time are different. A complex event instance is usually detected by applying an event pattern to a stream of events. A complex event may contain its contributing events which led to the detection of the complex event. So, for example for a pattern of $And(E_1, E_2)$ an instance contains one event of each type E_1 and E_2. To distinguish such a complex event from other complex event which are incidentally composed of two contributing events, we create a type for the complex event pertaining to the event operator which was used in the detection.

Therefore, we obtain an ontology of subtypes of `ComplexEvent` which represent event operators. To come up with a list of operators we started with one of the oldest general purpose event languages named Snoop [3]. Snoop contains a fairly comprehensive list of boolean and temporal operators. They are modeled in our ontology. What is missing in Snoop are operators which inspect the contents of input events such as attributes other than timestamps and type. Therefore, we added a `FilterEvent` as an example of what is needed to filter events by their content.

For simple events we identify two significant subtypes which are relevant to the client-side of the Web. These are the actual events created in the browser after a user interaction is detected and events form the client's clock. They are represented as `DomEvent` and `ClockEvent` respectively. These two types of events make up all of the client-side event load in a Web browser.

5 RDFa

In the previous section we described our schema for events, simple or complex. In this section we focus on enriching simple events with semantics from the context of the Web page in which the event occurred.

A simple event in Web clients is characterized by two dimensions; the type of event (e.g. click, mouseover) and the part of the Web page, where the event occurred (e.g. a node in the Document Object Model of the Web document). Subscribing to simple events of these types therefore requires the specification of type and the specification of the node or nodes where the events may originate. Both dimensions are retained in an event instance by using the attributes `jsEventType` and `cssSelector` (cf. Figure 2).

In order to better understand these events and make sense of what happened we must enrich the content of events when they are produced. The `jsEventType` tells us what a user has done and the `cssSelector` tells us where on the Web page the user did it. However, the latter is a purely presentation-dependent measure. There is no semantics which has any meaning beyond the context of a specific Web page structure. We propose to extract presentation-independent semantic information from the Web page if present. Instead of creating events from interaction with purely syntactic items of a Web document, we create events about interaction with semantic concepts which the document stands for. As an example, an event should not represent e.g., a click on a certain headline element of a Web document but rather a user's interaction with an article talking about politics and certain persons mentioned within.

To annotate a Web page with semantic data such as the topics of an article, we use RDFa. Defined in [2] RDFa is a means of adding RDF data to existing Web pages by using inline XHTML attributes.

After detecting an event which happened in the context of a certain DOM node of a Web document, we collect all semantic information in the Web page about the thing that is reported in that given DOM node. We currently achieve this by employing the client-side RDFa library ubiquity[10]. The lifting of context is achieved in a two-phase process. In the first phase we collect the list of RDF subjects of possible triples. This is done close to where the event happened in the document to provide *accurate* context. In the second phase we collect every triple with these subjects from the overall document in order to provide a very *rich* context.

To find valid subjects the first phase traverses the node where the event happened and its complete subtree[11]. If the given main node does not contain a subject, the immediate dominator node containing a subject is added to the list. This serves two purposes, guaranteeing a single root subject for orphan properties and objects in the subtree and guaranteeing a non-empty result set.

In the second phase all triples with the given subjects are collected from the entire document tree[12]. The gathered triples are then reified and appended as a bag to the event payload.

[10] The Ubiquity RDFa parser project: `http://ubiquity-rdfa.googlecode.com/`

[11] In the use case example this could include a news article about a politician and all the contained paragraphs.

[12] In the use case example this includes all triples about the politician from the article as well as possible extra information such as scattered information areas on the remaining Web page.

Even if the event itself becomes part of more complex events during the process of correlating and aggregating events, this basic data is retained as part of the simple event.

6 Implementation: Client-Side Event-Enabled Rule Engine

For our implementation we chose JavaScript from the available Web programming languages, for reasons of widespread availability. The data structures and program logic we implemented are roughly divided into the following areas: adapters for the rule language and remote event sources, the working memory, condition representation and evaluation as well as complex event detection.

For Complex Event Processing we are using a graph based approach as proposed in [3]. Initially the graph is a tree with nested complex events being parents of their less deeply nested sub-events, down to the leaves being simple events. However, common subtrees may be shared by more than one parent. This saves space and time compared to detecting the same sub-events multiple times, and renders the former tree a directed acyclic graph.

When using the term *event*, the distinction must be drawn between event occurrences (i.e. instances) and event types, usually done implicitly. In the detection graph the nodes are event types, they exist before there are any instances. Event instances exist after simple instances arrive and are fed into the graph at the leaves. Complex instances are then formed at the parent nodes, which in turn propagate their results upwards. Every complex event occurrence carries pointers to the set of its constituent event occurrences, so that the events and their parameters can be accessed later. Once an occurrence is computed at a node which is attached to a rule, the state of the associated Rete node is started and actions are triggered.

7 Evaluation

In this section we first evaluate the performance of our client side complex event processor. After that we present a first discussion of a running demo application collecting and aggregating events from test users we invited to a demo "news portal".

7.1 Performance

In this section we show the results from a performance test demonstrating that CEP and a client-side rule engine for the Web are indeed feasible. It is a technical evaluation of the event-processing capabilities of our ECA-rule-based framework. We will show that an event rate of about 64 events per second is possible with a given rule set on our test machine. Our test machine is a 2.4 GHz Intel Core2 CPU with four cores. Since JavaScript execution is inherently single-threaded it

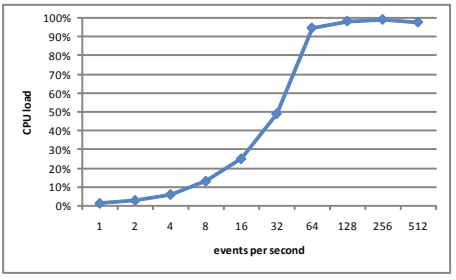

Fig. 3. CPU load by increasing event frequency

profits only from one CPU core. Having spare cores for other tasks combined with a generally low operating system load provides results which are uninfluenced by other running tasks. The chosen JavaScript engine is Mozilla Firefox 3.0.3 for Windows using the Firebug[13] profiler. The browser was installed freshly with no extra plug-ins.

We start out with the BEAST benchmark [6]. BEAST is an attempt at measuring CEP performance in early CEP applications, which use a similar event algebra than Snoop. We borrow some of the rules which are applicable to our CEP engine. Some of the event operators were not applicable to Snoop, like the count based window operator (cf. Figure 3 on page 8 of [6]). The event expressions from BEAST which are tested are described as follows, using event operators from the Snoop algebra:

$$\texttt{SEQ}(\mathrm{E}_1, \mathrm{E}_2) \tag{1}$$

$$\texttt{NOT}(\mathrm{E}_3, \mathrm{E}_4, \mathrm{E}_5) \tag{2}$$

$$\texttt{SEQ}(\mathrm{E}_6, \texttt{SEQ}(\texttt{OR}(\mathrm{E}_7, \mathrm{E}_8), \mathrm{E}_9)) \tag{3}$$

Event expression 1 represents a sequence of two events. Event expression 2 represents the non-occurrence of E_4 in the interval of two other specified events and finally: event expression 3 represents the sequence of one event followed by a disjunction and followed by another event.

The tested rules contain empty rule actions, so only the CPU load for Complex Event Processing is measured. We run each test for 30 seconds at various frequencies of simple events per second. The simple events in the mentioned patterns are entered into the detection system in a round robin manner. We then measure the load percentage caused by the detection system matching the incoming events and producing complex events.

The results are shown in Figure 3. The chart shows that our event detector can handle a maximum of about 64 events per second in real time. After that the JavaScript engine is used up to capacity and further incoming events are

[13] Firebug Web site: `http://getfirebug.com/`

queued up. Rule conditions, unlike event expressions, are not strictly needed for our approach to online advertising. Their evaluation can be found in [5].

The general use case for our framework is to monitor the user's event-based interaction with a Web page. This means reacting mainly to human actions. Since events from the human user are not occurring at millisecond rates, our framework should be fast enough to handle events at near real-time. Although expensive rule actions may lessen these results, upcoming new browsers promise a significant increase in general JavaScript performance due to newer compiling techniques. Currently, we expect the results to be sufficient for most client-side applications including our application of event-driven online advertising.

7.2 Ad Quality

To evaluate the return of targeted advertisements we created a demo Web page with some news articles. Each news article is contained in a separate part of the page, termed Semantic Web Widget (cf. Section 2.1). Each widget is annotated using RDFa using basic keywords and concepts pertaining to the article. For a user entering our demo, each widget is at first partially concealed. This is done to solicit an action from the user when "unfolding" the widget. Thereby the user expresses interest. This creates explicit events which can then be processed by our engine. Our initial evaluation of the ad quality was performed as follows:

1. We selected three different news domains (politics, culture, sports) in order to prove the domain-independence of the approach and pull into the demo Web page, as separate evaluation sessions.
2. We selected five users (PhD students from the Institute) with different cultural backgrounds.
3. The users should browse the demo Web page and judge about the relevance of generated ad-keywords in the case of a) the keywords generated statistically from the Web page (Google approach) and b) keywords generated by using the event-driven approach described in this paper. In order to ensure a fair comparison, the users did not know which list of ad-keywords was produced by which method.

We ask the users to rate the gathered keywords in terms of relevance to what they had been doing in the news portal and to compare this with a static list of keywords extracted from the overall page. The results are very encouraging: in the average 85% of keywords generated in our approach were described as "very relevant" and 98% as "relevant" (very similar results across all three domains). The traditional approach achieved 65% success for "very relevant" and 85% sucess for "relevant" ad-keywords. This result demonstrates the advantages of our approach for generating very relevant ads.

In comparison, Web Usage Mining (e.g., [7]) is used on log files which are analyzed on the server side at certain intervals or possibly in a continuous fashion. It is important, however, to stress that our approach detected all events on the client. Events occurred purely by folding and unfolding widgets as parts of

the page. No communication with the server took place and hence no artifacts are visible in server log files. Thus, our approach extends clickstream analysis to regions which were previously invisible to server-based mining techniques.

Moreover, our approach is a truly event-driven application, meaning that we detect events in real-time, as soon as they happen. In contrast, traditional mining techniques function in a query-driven manner where results are only created at intervals, such as daily analyses of the log files.

8 Related Work

In this section we discuss related work from several fields of research relevant for this paper, namely reactivity for the Web, online advertising related to our use case and Complex Event Processing in general and specifically for the Web.

There exist several approaches to reactivity for the Web. The approach from [8] describes a rule-based event processing language for XML events. The language design is described as focusing on aspects of data extraction, event composition (operators), temporal relationships and event accumulation. The approach is based on logic programming. Some drawbacks are inherited from this. The most striking fact is that events (simple and complex) are detected and reacted to in a query-driven fashion. This means that event patterns are only fulfilled when the query engine asks for the patterns. There is no data-driven way of fulfilling patterns in the moment each event arrives. This behavior is based on the fact that logic programming systems such as Prolog operate in a backward-chaining way, fulfilling queries only when they are posed. There is no first class notion of continuous queries. This means that the approach from [8] as well as others such as [9] are not truly event-driven, because events are not handled when they occur but are stored until the query is posed for the next time. Furthermore, it is unclear where the events come from and how they are entered into the logic programming system; There is no notion of subscribing to input streams or similar ways of accessing event sources. Consumption of events is also not defined; Events seem to have indefinite life-time and be reused in new patterns over and over. In comparison to our work there is no focus on client-side events which occur in a browser, e.g. from humans interacting with Semantic Web documents.

Another event processing language for the Web is presented in [10]. It is likewise an event-condition-action (ECA) rule-based approach, with pluggable language dialects for each of the E,C and A parts of a rule. An ontology of the compositional approach is presented. The question of connecting event sources is addressed in this work, but requires a degree of cooperation of nodes on the Web which is currently not practical. For example, a possible source of events is said to be the changes to XML data. However, such events are only created if change is monitored, e.g. with the help of an *active* XML database. As a workaround, so-called *ECA services* are proposed which provide active notifications from passive nodes. However, as this requires polling/querying, it is again not strictly event-driven. Our solution actively publishes events when they occur and as

such is fully event-driven. In a federated setup of the mentioned related work, our solution could possibly be used as a source of events.

In Web advertising there are essentially two main approaches, *contextual advertising* and *behavioral advertising*. Contextual advertising [11] is driven by the user's context, represented usually in the form of keywords that are extracted from the Web page content, are related to the user's geographical location, time and other contextual factors. An ad provider (ad serving service) utilizes these meta data to deliver relevant ads. Similarly, a users' search words can also be used to deliver related advertisement in search engine results page, Google's second pillar in online advertising. However, contextual advertising, although exploited today by major advertising players (e.g., GoogleAdsense[14], Yahoo! Publisher Network[15], Microsoft adCenter[16], Ad-in-Motion[17] etc.), shows serious weaknesses. Very often the automatically detected context is wrong, and hence ads delivered within that context are irrelevant[18]. For instance, a banner ad offering a travel deal to Florida can possibly be seen side-by-side to a story of a tornado tearing through Florida. This is happening because the context was determined using purely keywords such as "Florida, "shore" etc (i.e., without taking keyword semantics into account). While there are improvements in contextual advertising (e.g., language-independent proximity pattern matching algorithm [12]), this approach still often leads companies to investments that are wasting their advertising budgets, brand promotion and sentiment. In contrast, our approach utilizes semantics to cure major drawbacks of today's contextual advertising. Semantic Web technologies can be used to improve analysis of the meaning of a Web page, and accordingly to ensure that the Web page contains the most appropriate advertising.

The second approach to Web advertising is based on the user's behavior, collected through the user's Web browsing history (i.e., *behavioral targeted advertising*). The behavior model for each user is established by a persistent cookie. For example, Web sites for online shopping utilize cookies to record the user's past activities and thereby gain knowledge about the user or a cluster of users. There are several reasons why behavioral targeted advertisement via cookies is not a definitive answer to all advertisement problems. First, if a user, after browsing the information about an item purchases that item, he or she will not be interested in that particular good afterwards. Therefore, all ads and "special deals" offered to the user later while browsing that Web site are useless. Also, the short-term user interest should be detected more quickly (i.e., during the current user session). Displayed ads need to reflect current moods or transient user interest. For example, a user looking hastily to buy a gift of flowers is not interested in ads related to his/her long-term profile, created during previous

[14] GoogleAdsense: `http://google.com/adsense`
[15] Yahoo! Publisher Network: `http://publisher.yahoo.com`
[16] Microsoft adCenter: `http://adcenter.microsoft.com`
[17] Ad-in-Motion: `http://ad-in-motion.com`
[18] Adam Ostrow, When Contextual Advertising Goes Horribly Wrong - Mashable: `http://mashable.com/2008/06/19/contextual-advertising`

purchases unrelated good or services. Further on, there are problems with cookies. Computers are sometimes shared and users get to see ads governed by other user's cookies. Finally, given the European Union's Directive and US legislation concerned with restricted use of cookies, behavioral targeted advertisement based on cookies is not a promising direction for Web advertising.

We believe that *short-term profiling* (in contrast to long-term profiles created by cookies) is a valid and possibly augmenting approach in terms of personalization and identification of the user's interest. We realize a short-term profiling using client-side Complex Event Processing techniques (cf. Section 2.2), and background semantics (cf. Section 2). Such profiles are automatically detected, are always up-to-date and fully personalized.

The work from [13] describes event processing for Web clients. Events are observed on the client, however, complex events are not detected in the client. All simple events are propagated to the server for detection of patterns. This incurs latency and reduced locality for the processing of events, so the advantages of client-side event processing are lost.

9 Conclusion

In this paper we present a novel approach for generating and processing complex events form Web pages which opens possibilities to build event-driven applications for the (Semantic) Web. We envision the future of the (Semantic) Web as a huge, decentralized event repository (the so-called Event Cloud in CEP terminology), which will contain information about the real-time activities of different Web users. Such an event repository will enable different kinds of processing of the real-time information, making the Semantic Web really active, i.e. the environment can react and adapt itself on the signals sensed from the environment, connecting the Internet of Things with the Internet of Services, two basic elements of the Future Internet. As a result this paper makes contributions to the integration of the Semantic Web in the Future Internet. The presented Semantic Advertising use case shows clearly how efficient the contribution of active components and semantic technologies in future Internet applications can be.

Acknowledgments. We would like to thank Weiping Qu for his contribution to the RDF schema and to the implementation.

References

1. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
2. Adida, B., Birbeck, M., McCarron, S., Pemberton, S.: Rdfa in xhtml: Syntax and processing (October 2008), `http://www.w3.org/TR/rdfa-syntax/`

3. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K.: Composite events for active databases: Semantics, contexts and detection. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) 20th International Conference on Very Large Data Bases, Santiago, Chile proceedings, Los Altos, CA 94022, USA, September 12–15, pp. 606–617. Morgan Kaufmann Publishers, San Francisco (1994)
4. Schmidt, K.U., Stühmer, R., Stojanovic, L.: From business rules to application rules in rich internet applications. Scalable Computing: Practice and Experience 9(4), 329–340 (2008)
5. Schmidt, K.U., Stühmer, R., Stojanovic, L.: Gaining reactivity for rich internet applications by introducing client-side complex event processing and declarative rules. In: Stojanovic, N., Abecker, A., Etzion, O., Paschke, A. (eds.) The 2009 AAAI Spring Symposium on Intelligent Event Processing, Association for the Advancement of Artificial Intelligence, March 2009, pp. 67–72 (2009)
6. Geppert, A., Berndtsson, M., Lieuwen, D., Roncancio, C.: Performance evaluation of object-oriented active database systems using the beast benchmark. Theor. Pract. Object Syst. 4(3), 135–149 (1998)
7. Liu, B.: Web Data Mining. Data-Centric Systems and Applications. Springer, Heidelberg (2007)
8. Bry, F., Eckert, M.: Rule-based composite event queries: The language xchangeeq and its semantics. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 16–30. Springer, Heidelberg (2007)
9. Paschke, A., Kozlenkov, A., Boley, H.: A homogenous reaction rules language for complex event processing. In: International Workshop on Event Drive Architecture for Complex Event Process (2007)
10. May, W., Alferes, J.J., Amador, R.: An ontology- and resources-based approach to evolution and reactivity in the semantic web. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1553–1570. Springer, Heidelberg (2005)
11. Kenny, D., Marshall, J.: Contextual marketing–the real business of the Internet. Harvard Business Review 78(6), 119–125 (2000)
12. Schonfeld, E.: Proximic signs deals with yahoo and ebay to turn product listings into contextual ads; taking on adsense. Online Article (January 2008), `http://www.techcrunch.com/2008/01/15/proximic-signs-deals-with-yahoo-and-ebay-to-turn-product-listings-into-contextual-ads-taking-on-adsense/` (Last visited August 2009)
13. Carughi, G.T., Comai, S., Bozzon, A., Fraternali, P.: Modeling distributed events in data-intensive rich internet applications. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 593–602. Springer, Heidelberg (2007)