Client-Side Event Processing for Personalized Web Advertisement

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. The market for Web advertisement is continuously growing and correspondingly, the number of approaches that can be used for realizing Web advertisement are increasing. However, current approaches fail to generate very personalized ads for a current Web user that is visiting a particular Web content. They mainly try to develop a profile based on the content of that Web page or on a long-term user's profile, by not taking into account current user's preferences. We argue that by discovering a user's interest from his current Web behavior we can support the process of ad generation, especially the relevance of an ad for the user. In this paper we present the conceptual architecture and implementation of such an approach. The approach is based on the extraction of simple events from the user interaction with a Web page and their combination in order to discover the user's interests. We use semantic technologies in order to build such an interpretation out of many simple events. We present results from preliminary evaluation studies. The main contribution of the paper is a very efficient, semantic-based client-side architecture for generating and combining Web events. The architecture ensures the agility of the whole advertisement system, by complexly processing events on the client. In general, this work contributes to the realization of new, event-driven applications for the (Semantic) Web.

Keywords: Complex Event Processing, User Profiling, Rich Internet Applications, Semantic Web Technologies, RDF, RDFa, event-condition-action, ECA, Web Advertisement.

1 Introduction

The market for Web advertisement is continuously growing[1]. Correspondingly, the number of approaches that can be used for realizing Web advertisement are

[1] IAB/PwC Internet Advertising Revenue Report 2008/Q2
http://www.iab.net/media/file/IAB_PWC_2008_6m.pdf

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part II, LNCS 5871, pp. 1069–1086, 2009.

increasing. The main challenge in all these approaches is to personalize ads as much as possible. Google's AdSense being the most popular by far - automatically analyzes the content of Web pages to dynamically determine which ads are the most relevant to serve there. If a user is on a site reading about LCD televisions, they show him/her ads for retailers who sell them - without the publisher or the advertiser (or even the ad network) having to explicitly specify anything. There are many variations of this type, which use various forms of metadata for discovering what the visited Web page is about, including relying on the semantic metadata, once it will be broadly available.

In general, an ad will be consumed not only if it is relevant for a given context of the user (e.g. the content of the visited Web page: LCD televisions), but also if it is relevant for the current interest of the user (e.g. energy saving aspects of the LCD televisions). Indeed, focusing only on determining the content of a Web page for contextualizing/personalizing ads leads to over-generalization of ads, which decreases the probability that the user will pay attention to an ad, although it seems to be very relevant for the Web page the user is visiting.

In order to get the attention of a user, an ad should be as specific as possible regarding his current interest. For example, there is a big difference between the current interests of two users who are visiting the same Web site about LCD televisions, but one is reading technical characteristics of the device and another is focusing attention on the text related to the energy saving issues. Obviously, different ads should be delivered to these two users.

Consequently, differentiating between interests of the users who are visiting the same Web page becomes the key issue for the successful advertisement. Indeed, the concept of "differentiating" is crucial for determining what is very relevant for a user, in the following sense: that what is distinguishing a user from other users (i.e. an average user) is the best (most specific) descriptor of his/her interest. We are following here the analogy from the brick and mortar environment, in which an experienced shop assistant is approaching a customer after: 1) observing his/her behavior for a while, 2) determining the specificities in his/her behavior which can be used as events indicating what he/she wants and 3) having collected enough indicators that he/she needs support for (i.e. that he/she will pay attention to it). The background is that the shop assistant wants to determine that what is distinguishing the customer from an average one in order to sell/suggest to him the most appropriate solution (tailored to his/her interests). The shop assistant is trying to recognize some pattern of events in the user's behavior and act correspondingly. Or another comparison: if someone is coming into the shoe-shop it is not a big deal to recognize that he would like to buy shoes, but rather to discover smoothly which model he/she is interested in. In this paper we present such an approach for the Web advertisement.

The approach is based on the extraction of simple events from the user interaction with a Web page (e.g. reading a particular Web text) and their combination in order to discover the user's interests. We use Complex Event Processing technologies for efficient generating and combining events and semantic technologies for building their semantic interpretations, which will be used by an ad provider.

We present the architecture and implementation of such an approach. Furthermore, we demonstrate the advantages of 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, that we see as a necessary factor for the future Semantic Web applications.

This paper is structured as follows: In Section 2 we will describe the requirements for generating short-term profiles from annotated Web pages including means of processing these events to detect complex situations. In the subsequent section about our client-side reactive rule language we will name JSON-Rules as a key technology for our architecture. We will then present implementation details of our architecture in Section 4. The whole approach will be evaluated for performance and usefulness in the advertising scenario in Section 5, and we will discuss related work and conclude the paper in the last remaining sections.

2 User Profiling

Essentially there are two main approaches in Web advertising. The first is called *contextual advertising* [1], and is driven by the user's context usually in the form of keywords extracted from the Web site content or related to the user's geographical location and other contextual factors. The second approach is based on the user's behavior, collected through the user's Web browsing history (i.e., *behavioral targeted advertising*). Both are discussed in more detail in Section 6 including their disadvantages.

On the one hand, our approach utilizes semantics to cure major drawbacks of today's contextual advertising, discussed in Section 2.1. On the other hand, our approach addresses drawbacks of behavioral targeted advertising by realizing short-term profiling using client-side Complex Event Processing techniques. This is discussed in Section 2.2. Following up on this, Section 2.3 and 2.4 illuminate the life cycle of our approach.

2.1 Semantics for Profiling

In order to better understand events from Web clients and make sense of what happened we must enrich the content of events when they are produced. 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 receiver 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 useful 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[2], Creative Commons[3], FOAF[4], GeoRSS[5] and OpenCalais[6]. Regarding the format of structured data, RDFa [2], eRDF[7] and Microformats[8] 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 the ad content is created based on a current user's attention, i.e. an event pattern being recognized in his/her behavior. In order to accomplish this, we need as much (meta-) information as possible about the content of the Web page. Therefore we assume the semantically enriched Web content such that the 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 produce fragments of semantic annotations giving a high-level description of the content, and providing the basic context of data contained in the widgets. For instance on a news portal which would like to incorporate semantic advertising one widget could be used for listing all news belonging to one subcategory, e.g., politics, another one for arts, etc.. We use these annotations to discover user's interests when detecting complex events (cf. Section 2.2).

In Figure 1 we show an RDFa example of the semantic description for an arts event[9] listed in a widget related to musicals. Semantic descriptions are generated

[2] Dublin Core: `http://dublincore.org`
[3] Creative Commons: `http://creativecommons.org`
[4] FOAF: `http://foaf-project.org`
[5] GeoRSS: `http://georss.org`
[6] OpenCalais: `http://opencalais.com`
[7] eRDF: `http://research.talis.com/2005/erdf`
[8] Microformats: `http://microformats.org`
[9] An *event* in the sense of a gathering of people.

```
<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">Her carpet bag is packed, her umbrella
            is unfurled, and come this Fall, 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

from knowledge which was contained in schema information available only on the server. Using RDFa the data becomes self-describing, even after it is embedded in HTML and transmitted to Web clients. 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.

Extracting the context for Web advertisement can also be handled in an environment where semantic annotations and relationships already exist. For instance, a semantic media wiki[10] is a wiki that has an underlying model of the knowledge described in its pages. Therefore, it is easy to extract metadata and more complex relations from contained information. This metadata can be used to get the context of information a user is currently interested in, and further to offer relevant advertisement.

2.2 Complex Event Processing for Profiling

In order to detect nontrivial situations of interest, simple events are combined into more complex events. This is the task of Complex Event Processing, that we describe in the context of the Semantic Advertising use case. To target specialized advertising we want to detect the behavior of a user on a Web page. Two main ingredients are required for this task. On the one hand, the page content must be

[10] Semantic media wiki: `http://semantic-mediawiki.org`

semantically enhanced as discussed above. This serves the purpose of describing what exactly a user is looking at. On the other hand, statistical data from past users is required to define what is interesting behavior to an ad provider. Such behavior is modeled as sequences of events observed from past usage of the Web site.

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. The rules use a format which is easily processed in JavaScript. They are explained in more detail in Section 3. The event part of such a rule contain complex event expressions. The condition part contains complex conditions (which are evaluated in our client-side implementation of the Rete algorithm [3]). The action part contains different possible types of actions, i.e. either arbitrary client-side code or declarative modifications to the working memory or firing of further events.

The rules on the client-side 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. Event-triggered rules, in turn, create intermediate events in a hierarchy of event abstraction. These events are subsequently accumulated until sufficient interest (according to the ad provider) is recorded, i.e. 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 and hence 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) 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 the client and server. Below, 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.

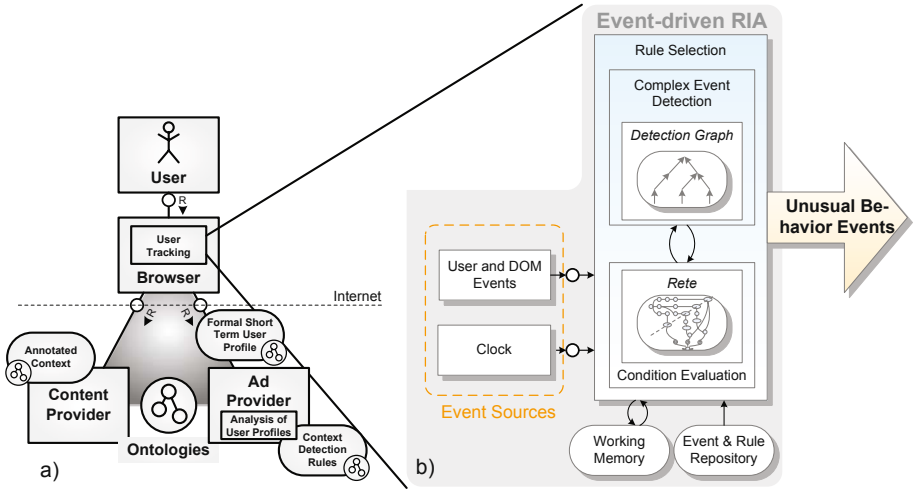

Fig. 1. Architecture: a) Logical Architecture b) Client-side User Behavior Analysis

2.3 Design Time

The design time of our application life-cycle is concerned with providing the most important requirements mentioned above — semantic enhancement of Web pages and complex event patterns, generated from statistical data.

Creating semantically enhanced Web content is vital for determining what exactly a user is looking at. For Web sites with underlying schemata it is feasible to provide a semantic vocabulary and embed it using RDFa. An example of such a semantic description for a musical event was given in Section 2.1. Another class of Web sites which can generate semantically annotated pages is sites which use semantics on the server side. This knowledge can be embedded in the delivered pages using RDFa. Examples are semantically enhanced wikis [4].

We expect such annotations on various parts of a Web page, for example on the granularity level as proposed with the Semantic Web Widgets (SWW) in Section 2.1. Using annotations on page elements allows us to catch the user's attention in a more detailed fashion, especially on mixed content pages such as Web portals which consist of several SWW (that are in the view of the user at the same time). A traditional keyword-counting approach will detect a diluted, averaged form of content for such a portal page.

On the other hand we anticipate annotations to be mostly used on elements at, or not far below the level of single widgets or paragraphs. Reasons for this are of practical nature, in keeping the number of events manageable. Handling too much detail might have further adverse effects at this point, creating a large number of event types which are almost never used (created or consumed). There might, for example, be no measurable interaction of the user with a certain word in a Web page, whereas the surrounding paragraph might encounter detectable mouse clicks or mouse hovering/movement.

Furthermore, in order to form complex event expressions, these annotations are combined with a temporal model. Such expressions group the user's atomic actions into temporal contexts like e.g. sequences of clicks. Determining sequences of interest is based on analyzing historical (log) data statistically. By using data mining algorithms for click streams such as [5], historical data is transformed into knowledge about unusual sequences of interaction such as clicks. Subsequently, the corresponding complex event expressions can be created. This process can be done automatically. A simple sequence along with its confidence might be "politics" followed by "flowers" with a low confidence of "2%". This means that from previous users only a fraction of 2% have looked at a politics widget followed by looking at a flowers widget. This pattern in the users behavior can be treated as unusual, i.e. his/her interests for "politics" and "flowers" are distinguished from the interest of others, so that this can be used for developing a very personalized ad. In fact, we argue that more information content (for generating ads) is stored in the exceptional behavior, than in the usual/-expected one. A simple explanation is that expected behavior is too general to detect what is specific in the behavior of the customers (cf. example from the brick and mortar environment from Introduction).

Such an ad will very likely attract the attention of the user, since it directly corresponds to his short-term profile. Further processing of e.g. the time interval within the two participating events could be envisioned.

Each complex event expression is embedded in an event-condition-action rule with the probability as the consequence. The consequence forms another event which is processed further by higher-level rules.

2.4 Run Time

At run time the task is to detect the user's short-term profile and then to deliver it to an ad provider. We will illustrate the procedure with an example. At first the user enters our site. The pages are semantically annotated and the event processor is invoked after loading first page. The up-to-date rules are then automatically fetched from the server once the event processor is fully initialized on the client.

Subsequently all rules are incorporated into a client-side event graph. This graph [6] is used for detecting patterns. It resembles the nesting levels of event expressions. The top of the graph contains nodes entailing the highest rule actions e.g., requesting new advertisements. The inner graph nodes model the user actions leading to the activation of the top nodes. At the bottom of the graph are the atomic events (e.g. single mouse clicks, etc) which are detected directly from the user's interaction. The graph and its creation is discussed in more detail in Section 4.

To be notified of user interactions, each atomic event node registers itself with elements of the Web page (DOM nodes). This is necessary to receive events created by the user in a browser window. The subscriptions are updated by the event processor if the page content is changed. The subscriptions are specified in a declarative manner by our rule language. Such a simple event can be the

subscription to e.g., all clicks from all elements annotated with a certain keyword, i.e. ontological concept. To use different concepts on the client side, we have developed a mechanism to export the class hierarchy of an OWL ontology (cf. [7]). Using this knowledge we can decide which elements to subscribe to, even when a more general concept is specified in rules.

Once the user interacts with a page element of interest, its associated subscriptions are triggered. For each triggering an associated event object is created. Event context is recorded from the page element e.g. the Semantic Web Widget. Context includes all semantic annotations about the page element in question. The specifics of the context extraction are discussed in detail in [8]. The event objects including context are fed into the event detection graph to detect complex patterns. This starts at the bottom nodes of the graph. To facilitate further event-driven execution, these event objects are propagated upwards in the graph if they successfully match the pattern of their parent nodes. If they do not fulfill the pattern and cannot be used in any future matches, events are discarded according to the semantics of Snoop [9].

Once the accumulated simple events have created enough unusualness, the higher-level rules fire. The consecutive actions of these rules transmit the completed short-term user profile to the ad provider. The profile contains the user model complete with all participating events, so that the model can be restored on the server.

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, the language targets 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[11], because it is natively usable in 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 Snoop [9] operators. The condition part consists of conjunctive predicates over facts 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 expense 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

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

documented in [9] and [6]. Additionally we define further event operators $Mask(E_1, \text{condition})$ and $Thres(E_1, \text{threshold})$ as follows:

Filter is modeled after the event masks from ODE [10]. The filter enforces a condition on each occurrence of event E_1. This allows e.g. for fine-grained content-based filtering/masking of events.

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.

The event operators in our rule language are represented and serialized in the rule files as tree nodes. Simple (atomic) events form the leaves. Derived events form inner and top nodes. Such hierarchical representation allows a lean, abstract syntax without constructs from concrete syntax (like parentheses) compared to textual event expressions.

A condition in our language may use comparison operators used on variables from the working memory and direct literal values. The condition part is a conjunction of predicates. Comparison operators are $<$, $>$, $=$, $<=$ and $>=$. Variables specify items from the working memory.

Rule actions are JavaScript code blocks or events to be triggered on rule execution. A code block has access to the set of events 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 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 adds a click handler to a DIV element 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 [11,12] which detects, aggregates and finally submits the user profile in form of one or more complex events.

4 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

```
{
    "meta": {
        "rule": "Politics->Science=>2%"
    },
    "event": {
        "type": "SEQ",
        "children": [
            {
                "type": "DOM",
                "selector": "div[property=dc:keywords][content~=politics]",
                "event": "click"
            },
            {
                "type": "DOM",
                "selector": "div[property=dc:keywords][content~=science]",
                "event": "click"
            }
        ]
    },
    "action": [
        {
            "type": "EVENT",
            "trigger": "unusual",
            "parameters": {"probability": 0.02}
        }
    ]
}
```

Listing 2. Example of a single Rule

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 [6]. 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. The graph is built starting at the leaves, bottom-up. The simple event types from the available rules are stored in a hash map, and form the leaves of the tree. The hash keys are the event names. Each hash value (i.e. leaf) has a list of parents containing pointers to inner tree nodes. These in turn carry references to their parents.

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 evaluation of the associated condition is started.

The conditions are also stored in a hierarchical structure, our client-side implementation of the Rete algorithm [3]. The implementation is capable of

evaluating conjunctive predicates of propositional logic. Quantifiers and negation are currently not supported, as we did not require them in our use case so far.

Rule execution is done by inspecting the action parts in the rule specification. Explicit triggering of events is possible as well as the direct execution of JavaScript code and manipulations of the working memory. For every explicit event that is specified by name, a new simple event occurrence is created and fed into the detection graph at the leaf of the corresponding event type. As all leaves are stored in a hash map, finding the leaf to a name is a simple hash lookup. For every JavaScript-code action that is specified in the action part of the rule, a new function[12] is created at runtime. The set of events which triggered the rule is passed to the function. Thus, the rule action may employ the data from the constituent events in its computation. That includes the occurrence and duration times, the number and sequence of events, and the parameters carrying all values collected at the occurrence of the events.

5 Evaluation

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). A widget is annotated using RDFa with 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.

For each evaluation session the rules automatically load together with the evaluation Web site. The JavaScript rule engine registers each rule with its specified event types from the widgets. Event occurrences are processed according to the rules. Notably, events are collected at the level of the threshold operators until enough unusualness is collected from a long enough sequence of events.

[12] Functions are first-class citizens in the JavaScript language and can dynamically be created and passed along as parameters.

We then halt the processing and present the keywords from their browsing session to the user. 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., [13]) 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.

Furthermore, we conducted a performance evaluation of our approach. It showed that our engine can process an average of 64 events per second[13] depending on the complexity of the event patterns. 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. Details of the evaluation can be found in [12].

6 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. We will explain them and compare them to our approach. The approach from [14] 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 built-in notion of continuous

[13] In a test environment using a 2.4 GHz Intel Core2 CPU, a Mozilla Firefox 3.0.3 browser and the Firebug profiler.

queries. This means that the approach from [14] as well as others such as [15] 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 also no focus on client-side events which occur in a browser, e.g. from humans interacting with a Web document.

Another event processing language for the Web is presented in [16]. It is likewise an event-condition-action (ECA) rule-based approach, but 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.

In Web advertising there are essentially two main approaches, *contextual advertising* and *behavioral advertising*. Contextual advertising [1] 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 [17]), this approach still often leads companies to investments that are wasting their advertising budgets, brand promotion and sentiment.

[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`

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 purchases unrelated good or services. Further on, there are problems with cookies. Computers are sometimes shared, and if cookies are enabled, users get to see ads governed by other user's cookies. For this and other reasons, cookies cannot be seen as means for exclusive capturing of user's *personalized* preferences. 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 approach in terms of personalization and identification of the current 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 2.1). Such profiles are automatically detected, are always up-to-date and fully personalized.

Complex Event Processing today is still mostly a feature of enterprise applications since the advent of reactive rules in the 1980s [18]. However, depending on the sources of events it is preferable to move the point of the detection as close as possible to the origin of events. In a Web-browsing scenario we therefore propose client-side event detection. This means moving the task of Complex Event Processing to the client, in this case the browser. In doing so, we reduce the latency which would otherwise be incurred by transmitting events over the Internet. Furthermore, the volume of transmitted events is decreased because most events might not take part in any patterns and hence relaying them would be meaningless.

The work from [19] 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.

Use of Complex Event Processing generally involves challenges such as having expressive operators, access to the necessary event sources as well as efficient detection algorithms [20] (page 146f.). For the Web client there are specific further challenges such as the choice for an appropriate client-side programming language which were addressed in Section 4.

7 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 Web. We envision the future of the Web as a huge, decentralized event repository (so called Event cloud), which will contain information about the real-time activities of different Web users. Such an event cloud will enable different kinds of processing of real-time information, making the Web really active i.e., the environment can react and adapt itself on the signals sensed from the environment. For our use case we identified some drawbacks in current approaches to Web advertising, and proposed a novel approach that uses *short-term user profiling* for realizing event-driven advertising. Our new approach is based on light-weight semantics and Complex Event Processing (CEP). For CEP we presented a client-side event processor which detects complex events on the Web client governed by rules which are created, updated and stored on the Web server but executed on the clients. The rules are used to capture the current user's interest when browsing the content. Semantics is used to improve analysis of data presented on a Web page, and hence to extract the context of the user's interest more accurately. Combined, semantic annotations and detected complex events enable an ad provider to deliver finer-grained personalized advertising.

For future work we envision the realization of more complex event patterns to detect a user's behavior also on a higher level of abstraction. For instance, a user may look at particular items on a Web page, then switch to another related item, and go back to the first one. In this particular situation, a user is likely *comparing* those two items. Other navigational patterns are conceivable. In addition to more sophisticated client-side capabilities we plan to elaborate on further event processing taking place on the server-side, propagating the client events to multiple ad providers and resulting in a distributed event-based system in addition to the client-side part presented in this paper.

Acknowledgments. We would like to thank Yan Li for his work on the user evaluation.

References

1. Kenny, D., Marshall, J.: Contextual marketing–the real business of the Internet. Harvard Business Review 78(6), 119–125 (2000)
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. Forgy, C.L.: Rete: a fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19, 17–37 (1982)
4. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic wikipedia. In: WWW 2006: Proceedings of the 15th international conference on World Wide Web, pp. 585–594. ACM, New York (2006)
5. Fortuna, B., Grobelnik, M., Mladenić, D.: Visualization of text document corpus. Special Issue: Hot Topics in European Agent Research I Guest Editors: Andrea Omicini 29, 497–502 (2005)
6. 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, Los Altos, CA 94022, USA, September 12–15, 1994, pp. 606–617. Morgan Kaufmann Publishers, San Francisco (1994)
7. Schmidt, K.-U., Stojanovic, L., Stojanovic, N., Thomas, S.: On enriching ajax with semantics: The web personalization use case. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 686–700. Springer, Heidelberg (2007)
8. Stühmer, R., Anicic, D., Sen, S., Ma, J., Schmidt, K.-U., Stojanovic, N.: Lifting events in rdf from interactions with annotated web pages. In: ISWC 2009: Proceedings of the 8th International Conference on The Semantic Web (2009)
9. Adaikkalavan, R., Chakravarthy, S.: Snoopib: Interval-based event specification and detection for active databases. Data Knowl. Eng. 59(1), 139–165 (2006)
10. Gehani, N.H., Jagadish, H.V., Shmueli, O.: Event specification in an active object-oriented database. SIGMOD Rec. 21(2), 81–90 (1992)
11. 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)
12. 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)
13. Liu, B.: Web Data Mining. In: Data-Centric Systems and Applications. Springer, Heidelberg (2007)
14. Bry, F., Eckert, M.: Rule-based composite event queries: The language xchangeeq and its semantics. In: Marchiori, M., Pan, J.Z., de Marie, C.S. (eds.) RR 2007. LNCS, vol. 4524, pp. 16–30. Springer, Heidelberg (2007)
15. 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)
16. 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., Hacid, M.S., Mylopoulos, J., Pernici, B., Babaoglu, Ö., Jacobsen, H.A., Loyall, J.P., Kifer, M., Spaccapietra, S. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1553–1570. Springer, Heidelberg (2005)
17. Schonfeld, E.: Proximic signs deals with yahoo and ebay to turn product listings into contextual ads; taking on adsense (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)

18. Dayal, U., Buchmann, A.P., McCarthy, D.R.: Rules are objects too: A knowledge model for an active, object-oriented databasesystem. In: Dittrich, K.R. (ed.) OODBS 1988. LNCS, vol. 334, pp. 129–143. Springer, Heidelberg (1988)
19. 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)
20. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Boston (2001)