Exploiting E-C-A Rules for Defining and Processing Context-Aware Push Messages

Thomas Beer[1], Jörg Rasinger[1], Wolfram Höpken[1], Matthias Fuchs[1], Hannes Werthner[2]

[1] eTourism Competence Center Austria (ECCA), Technikerstrasse 21a
A-6020 Innsbruck, Austria
{firstname.lastname}@etourism-austria.at

[2] Institute for Software Technology and Interactive Systems
Vienna University of Technology, Favoritenstrasse 9-11/188
A-1040 Vienna, Austria
hannes.werthner@ec.tuwien.ac.at

Abstract. The focus of this paper is to show that the E-C-A paradigm offers an excellent approach for specifying the behavior of *context-aware information push services*. Such a service enables its operator to provide the users with tailored messages related to their current situation (context). The paper introduces CAIPS, an implementation of such a service for the tourism domain. The underlying E-C-A rules are presented and the design of the associated rule-engine is described. The engine's rule-interpreter is based on event-notification services and the object-oriented query-language HQL. The paper further presents a graphical high-level editor which supports business-experts in "writing" the CAIPS E-C-A rules. The presented approach enables the rapid development of new tailored messages (related to the user's context) without the need to modify the underlying application, i.e. without the trouble of writing new code for new message types.

Keywords: event-condition-action, reaction rules, context-awareness, push, interpreted rules

1 Introduction

In general two different information dissemination approaches can be distinguished, namely *push* and *pull* [6, 19]. The traditional pull approach requires that users know a priori where and when to look for information or that they spend an inordinate amount of time polling known sites for updates. Push-based services in turn, relieve their users from these burdens, i.e., the user is not forced to search for relevant information himself and automatically receives the information which is of a high relevance to him, thereby reducing the effort for gathering information [6, 9].

Context-aware services aim at tailoring services to the (user's) context [20]. A *context-aware information push service* (CAIPS) is therefore defined as a service which actively provides its users with tailored information (also denoted as *push-message*) regarding the topical situation (also referred to as *context*). A typical push-message in the tourism domain would be, for example, a message informing tourists about suitable indoor-events in the case of disadvantageous weather changes.

This article proposes an approach for the exploitation of E-C-A rules for specifying the behavior of context-aware information push services. Section two gives an overview of the CAIPS system. The rule-engine's design and its utilized rules are presented in section three and four. Section five presents the rule-editor, which supports domain experts in creating CAIPS E-C-A rules. Conclusions and future work are presented in section six.

2 Context-Aware Information Push Service (CAIPS)

This section introduces CAIPS, an implementation of a context-aware information push service for the tourism-domain [1]. One of the key requirements of the proposed system is "ease of use" for its operator. Put differently, the tourism expert can straightforwardly define *when* to send *what* to *whom*. The proposed approach to address this requirement is to define push-processes through E-C-A *rules*. Rules are a well known technique in the field of knowledge representation [5, 24]. They provide an excellent trade-off between readability of the knowledge representation and formal requirements. Conditional clauses (i.e. "if *condition* then *action*" clauses) were already used in pre-Christian times to express activity-instructions [17]. It therefore can be assumed that rules are well known to prospective users. Using rules addresses one of the prior objectives when designing such a system, namely the similarity to human thinking and the continuous improvement of usability [3].

In CAIPS, the tourism-expert utilizes E-C-A rules to *describe* both, the message's content and the situation, when the message is sent (cf. section 3.2). Fig. 1 illustrates the message definition and processing of CAIPS. The operator (tourism-expert) creates new message-types using E-C-A rules. He is thereby supported by a high-level graphical editor (cf. section 5), allowing him to easily specify the *message-firing-situation* (cf. section 3.2) and the message-content. As an integrated part of the etPlanner-framework [16] tourists may subscribe to these message-types within the (mobile) web-application. In the course of the subscription the users must indicate both, information preferences (e.g. sending time, communication channel) and preferences regarding message content (e.g. message should contain a *music-event* recommendation). This information is used for creating personalized content dynamically at the time of the push-rule execution. This approach stands in contrast to conventional *publish-subscribe systems* [22] where the message is produced and the content is matched to the preferences of the potential recipients. Instead, the CAIPS approach produces *tailored messages on demand*.

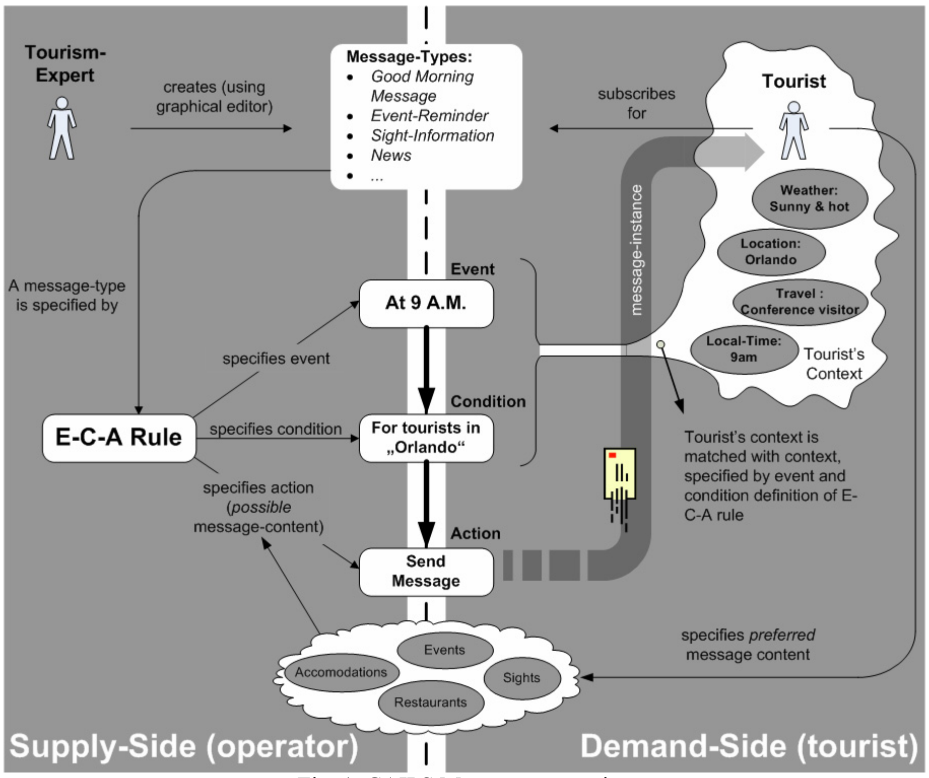


Fig. 1. CAIPS Message processing

The message generation and sending process (i.e. the rule interpretation) is controlled by the CAIPS rule-engine, an integral part of the CAIPS system. The next section describes its design in detail.

3 CAIPS: Rule-Engine

The rule-engine is responsible for interpreting rules and subsequent message-generation. As such, CAIPS is based upon concepts known from knowledge-based systems (KB systems). An important feature all KB systems have in common is the separation between knowledge about the domain of discourse (knowledge-base) and its processing (also referred to as reasoning) [3]. The design of the CAIPS Rule-Engine (RE) is illustrated in Fig. 2.

3.1 Rule-Interpreter

The rule-interpreter is one of the fundamental building-blocks of the rule-engine. It interprets the rules according to the underlying fact-base and thereby controls the application’s behavior. It is composed of three sub-components: the *Event-Handler*, the *Condition-Evaluator,* and the *Action-Manager*.

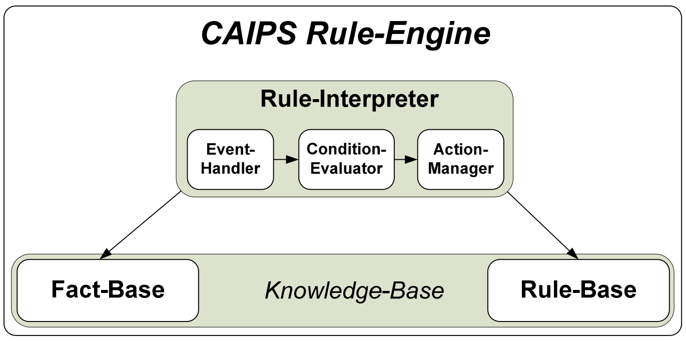


Fig. 2. Rule-Engine's Core Components

The event handling process is realized similar to a (distributed) event-notification service (ENS) [14]. Such a service informs entities (subscribers) about *events* they are interested in. Similar to [14] an *event* is defined as a state transition of an information object at a certain point of time, e.g. a user approaches a certain location, or a certain point of time is reached. An ENS is composed of event-triggers (services which report an event) and an event-handler. CAIPS distinguishes *external* and *internal* event-triggers. External events are triggered by external services, such as a weather change which could be reported by a weather server. Internal events are triggered “internally” by a state change within the fact-base, such as a change to the user’s profile. The message-based (asynchronous) communication between event-trigger and –handler enables the decoupling of event-generation and the subsequent handling-process.

The *Event-Handler* determines the rule(s) which are configured to react to the incoming event. This is done by a matching algorithm which “compares” incoming event-messages with the rule’s event-profile. This event-profile (which is formalized based on an event language similar to [14, 21]) denotes the rule’s interest in the incoming event (cf. section 4).

After identifying the “matching” rules, the condition evaluation process is triggered. The CAIPS condition-evaluator transforms the rule’s condition definition (cf. section 4) into the appropriate query-language and delegates the query processing to the related query engine. Clearly, the utilized query language must be compliant to the fact-base’s implementation (cf. section 3.2). CAIPS uses the object-oriented query-language HQL [13] to access the object-oriented fact-base. The query-results, comprising a number of potential receivers, are then forwarded to the action-manager.

The action-manager determines the appropriate action-processor according to the rule's action definition and subsequently triggers the action-execution. The action-processor completes the remaining tasks, i.e. it creates and sends the personalized messages [1].

3.2 Knowledge Base

The knowledge-base stores the “behavioral” knowledge of CAIPS, its *static* (i.e. the facts) and its *procedural* knowledge (i.e. the rules) [18]. It can be separated into two further sub-entities, namely the *fact-base* and the *domain & general knowledge* part (i.e. the rule-base). The fact-base contains problem specific knowledge, i.e., it represents the current state of the problem domain (i.e. the user’s context). The domain & general knowledge part stores domain specific and general knowledge about objects and their relationships, and heuristic knowledge which constitutes the “art of good guessing” in the domain (also known as rules-of-thumb or good judgment)[9]. In CAIPS the heuristic is constituted by the E-C-A rules.

The fact-base is based on object-oriented and domain modeling [8] design techniques. The facts (i.e. the user’s context) are represented by objects which are stored using an object/relational mapper [12]. The acquisition of coarse grained context data from the various sensors available in the application domain and its translation into objects is realized by appropriate context integration- and preprocessing-services [1].

The fact-base provides information for specifying the *message-firing-situation (MFS)*. The MFS occurs when the user’s context state (his "real-world situation") matches a predefined state, specified by the rule’s event- and condition statement.

The rule-base is the central repository for maintaining the rules. It provides functionality for creating, updating, deleting, and persisting CAIPS E-C-A rules. Currently the rule-base is implemented using a XML-based repository.

4 CAIPS Rule Markup Language

The utilized rule language is composed of query- and event-languages similar to the language presented in [21]. An example rule is presented at the end of this section. Each rule consists of three sections, the *event*, the *condition*, and the *action* definition.

Event

CAIPS utilizes an event language similar to the proposed language in [14]. This language enables the specification of primitive and composite events. Compared to [23], CAIPS supports the specification of *passive* events only; active events are covered by utilizing internal event-triggers (cf. section 3.1) which are “configured” outside of the rule-definition (cf. section 5). The event section specifies the event-profile (compare *client-profile* in [14]) which is utilized for filtering all rules which match an "incoming" event. The profile consists of a set of key-value pairs and an event type which categorizes the occurred event. An example XML representation of such an event-profile specification is shown in lines 2 to 8.

Condition

The condition definition is used to determine the set of users for which the occurred event is of high relevance. Restrictions are specified to narrow down the potential set of receivers (i.e., the users who will obtain generated push-messages). An example of a trivial condition statement is shown in lines 9 to 14.

Action

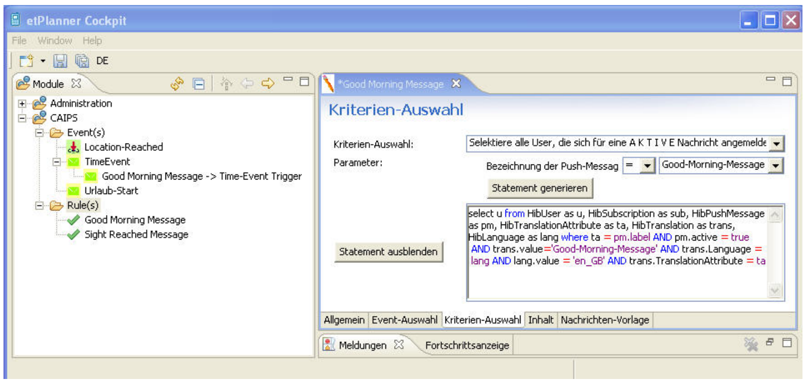
The action-definition specifies the action to be executed. The action, shown in lines 15 to 41, shows the definition of a "sendMessageAction". The embedded elements are used to configure the action-implementation. More precise, the parameters specify information to be passed onto the message generator.

The following example illustrates the representation of an E-C-A rule in CAIPS:

```
1:    <rule active="true" id="397705181623228736">
2:        <event id="3977051816232287345”>
3:            <type>TimeEvent</type>
4:            <param>
5:                <key>cron</key>
6:                <value>20 45 07 08 2007</value>
7:            </param>
8:        </event>
9:        <condition id="345789152443985792">
10:           <conditionType>HQL</conditionType>
11:           <conditionstatement>from User as u where
12:               u.prefs.identifier='GoodMorningMessage'
13:           </conditionstatement>
14:       </condition>
15:       <action id="98101636422723184" type="SendMessageAction">
16:           <content>
17:               <recommendation>
18:                   <type>Event</type>
19:                   <parameter>
20:                       <customerProperty>
21:                           <attributeName>startdatefrom</attributeName>
22:                           <attributeValue>&system.RuleExecutionDate</attributeValue>
23:                       </customerProperty>
24:                       <customerProperty>
25:                           <attributeName>classificationvalue</attributeName>
26:                           <attributeValue>&user.sight.classification</attributeValue>
27:                       </customerProperty>
28:                       <customerProperty>
29:                           <attributeName>city</attributeName>
30:                           <attributeValue>&user.address.city </attributeValue>
31:                       </customerProperty>
32:                       <customerProperty>
33:                           <attributeName>startdateto</attributeName>
34:                           <attributeValue>&system.RuleExecutionDate</attributeValue>
35:                       </customerProperty>
36:                   </parameter>
37:               </recommendation>
38:               <template>
39:                   <templateidentifier>GoodMorningMessage</templateidentifier>
40:               </template>
41:           </content>
42:       </action>
43:   </rule>
```

5 Rule Wizard

In section 2 the importance of supporting domain-experts in creating the CAIPS E-C-A rules was emphasized. CAIPS addresses this requirement by a graphical editor offering a high-level interface for creating CAIPS rules. The editor provides wizards for each part (event, condition, action) of a CAIPS rule, i.e. the business-expert can create a rule by a few “clicks” only. As an example, the event profile is built by selecting an “event” from a drop-down list. Further, the definition of condition-statements is facilitated by using forms. The approach is comparable to the "query by forms" technique presented in [11]. The screenshot in Fig. 3 illustrates its usage within the CAIPS rule-editor. The underlying HQL statement, which represents the condition definition (cf. sections 3.1 and 4), is generated according to the chosen form fields. Expert users may edited the condition definition directly.



Kriterien-Auswahl

Kriterien-Auswahl: Selektiere alle User, die sich für eine A K T I V E Nachricht angemelde

Parameter: Bezeichnung der Push-Messag = Good-Morning-Message

```
select u from HibUser as u, HibSubscription as sub, HibPushMessage
as pm, HibTranslationAttribute as ta, HibTranslation as trans,
HibLanguage as lang where ta = pm.label AND pm.active = true
 AND trans.value='Good-Morning-Message' AND trans.Language =
 lang AND lang.value = 'en_GB' AND trans.TranslationAttribute = ta
```

Fig. 3. CAIPS rule-editor, creating a rule’s condition definition

The editor also offers a wizard for the creation of internal event-triggers (cf. section 3.1). A prominent example is the creation of a time-event-trigger, which may be used to define events based on a time-schedule (comparable to cron-triggers).

6 Conclusion & Future Work

The paper demonstrated that interpreted E-C-A rules provide an excellent approach for controlling the behavior of context-aware push applications. It presented their applicability by using such rules in CAIPS, a prototypical implementation of a context-aware push service for the destination of Innsbruck. First field trials indicate high potential of the proposed service [1]. Based on these field trials further implementations for *Dolomiti-Superski* [7] (one of the largest skiing destination in Europe) and *Hofer-Reisen* [15] will take place until the end of 2007.

Future research will include an evaluation how the proprietary CAIPS rule-language may be complemented or even substituted by *Reaction RuleML* [23]. Furthermore it will be evaluated how the design of the fact-base effects tourism-experts in creating a rule's condition section, e.g. using conceptual queries [4] instead of query by forms.

References

1. Beer, T., Höpken, W. R., Zanker, M., Rasinger, J., Jessenitschnig, M., Fuchs, M., & Werthner, H. (2006). An Intelligent Information Push Service for the Tourist Domain. European Conference on Artificial Intelligence (Workshop on Recommender Systems), Riva del Garda.
2. Beer, T., Fuchs, M., Höpken, W. R., Rasinger, J., & Werthner, H. (2007). CAIPS: A Context-Aware Information Push Service in Tourism. In: Sigala, M., Mich, L., Murphy, J. (eds.): Information and Communication Technologies in Tourism 2007, Wien, Springer-Verlag, pp.129-140.
3. Beierle, C. and G. Kern-Isberner (2003). Methoden wissensbasierter Systeme. Hagen, Vieweg.
4. Bloesch, A. C. and T. A. Halpin (1996). ConQuer: A Conceptual Query Language. 15. Int. Conference on conceptual modeling, Cottbus, Germany, Springer LNCS.
5. Brachman, R. J. and H. J. Levesque (2004). Knowledge Representation and Reasoning. San Francisco, Morgan Kaufmann.
6. Cheverst, K., K. Mitchell, et al. (2001). Investigating Context-aware Informaiton Push vs. Information Pull to Tourists. MobileHCI'01 workshop on HCI with Mobile Devices, Lille, France.
7. Dolomiti Superski, http://www.dolomitisuperski.com
8. Evans, E. (2004). Domain-Driven Design. Boston, MA, Addison-Wesley.
9. Feigenbaum, E. A. (1992). Expert Systems: Principles and Practice.
10. Franklin, M. J. and S. B. Zdonik (1998). Data In Your Face: Push Technology in Perspective. ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, ACM Press.
11. Halpin, T. (2001). Information Modeling and Relational Databases - From Conceptual Analysis to Logical Design, Morgan Kaufmann.
12. Hibernate O/R Mapper, http://www.hibernate.org/
13. Hibernate Query Lang., http://www.hibernate.org/hib_docs/reference/en/html/queryhql.html
14. Hinze, A. (2003). A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service. Freie Universität. Berlin, Mathematik u. Informatik. PhD thesis
15. Hofer Reisen, http://www.hofer-reisen.at
16. Höpken, W., Fuchs, M., Zanker, M., Beer, T., Eybl, A., Flores, S., Gordea, S., Jessenitschnig, M., Kerner, T., Linke, D., Rasinger, J., & Schnabl, M. (2006). etPlanner: An IT framework for comprehensive and integrative travel guidance. In: Hitz, M., Sigala, M., Murphy, J. (eds.): Information and Communication Technologies in Tourism 2006, Wien, Springer-Verlag, pp.125-134
17. Jaynes, J. (1976). The Origins of Consciousness in the Breakdown of the Bicameral Mind. Princeton.
18. Karagiannis, D., & Telesko, R. (2001). Wissensmanagement. Konzepte der Künstlichen Intelligenz und des Softcomputing: Oldenbourg.
19. Kendall, J. E. and K. E. Kendall (1999). Information Delivery Systems: An Exploration of Web Pull and Push Technologies. Communications of the AIS 1(4).
20. Kranenburg, H. (2006). A context management framework for supporting context-aware distributed applications. Communications Magazine, IEEE 44(8): 67-74.
21. May, Wolfgang, Alferes, José Júlio, Amador; Ricardo (2005). Active Rules in the Semantic Web: Dealing with Language Heterogeneity. First International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005), Nov 10-12, 2005. Galway, Ireland, LNCS 3791, Springer, pp. 30-44.
22. Mühl, G. (2002). Large-Scale Content-Based publish subscribe systems. Informatik. Darmstadt, Technische Universität. PhD thesis
23. Reaction RuleML, http://ibis.in.tum.de/research/ReactionRuleML/
24. Sowa, J. F. (2000). Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove, CA, Brooks Cole Publishing.