Efficient Semantic Event Processing: Lessons Learned in User Interface Integration

Heiko Paulheim

SAP Research
heiko.paulheim@sap.com

Abstract. Most approaches to application integration require an unambiguous exchange of events. Ontologies can be used to annotate the events exchanged and thus ensure a common understanding of those events. The domain knowledge formalized in ontologies can also be employed to facilitate more intelligent, *semantic* event processing, but at the cost of higher processing efforts.
When application integration and event processing are implemented on the user interface layer, performance is an important issue to ensure acceptable reactivity of the integrated system. In this paper, we analyze different architecture variants of implementing such an event exchange, and present an evaluation with regard to performance. An example of an integrated emergency management system is used to demonstrate those variants.

1 Introduction

Integrating existing applications to form new systems is an important topic in software development, both for the purpose of saving engineering and maintenance efforts and for enabling the cooperation of existing systems (within as well as across organizations) [1]. Application integration can be carried out on three different levels: the data source level, the business logic level, and the user interface level [2]. Integration on the user interface level, or *user interface integration* for short, has two significant advantages [3]:

- Existing applications' user interfaces can be reused. Since the development of the user interface consumes about 50% of a software's total development efforts [4], the degree of reuse can be raised significantly.
- Users already familiar with existing user interfaces do not have to learn how to work with new ones. Therefore, the usability of an integrated user interfaces can be higher than of a user interface developed from scratch.

Integrated applications, and especially the implementation of cross-application interactions, require an event exchange mechanism [2]. To facilitate integration, the events issued by each application have to be commonly understood. Therefore, an ontology formalizing the information contained in the events is required [5]. It can be used to annotate the events, thus facilitating unambiguous

event exchange. Using ontologies and the domain knowledge encoded therein also allows for a more sophisticated approach of dealing with events, called *semantic event processing* [6].

Such a sophisticated approach, e.g. incorporating an ontology reasoner, makes event processing a more complex and thus more time consuming task. But when dealing with UI integration and user interfaces in general, reactivity is an important factor with massive influence on the users' performance and satisfaction [7, 8]. Therefore, semantic event processing mechanisms have to be implemented in an *efficient*, high-performance way. Various options exist for such implementations: events can be processed in a centralized or a decentralized manner [2], and instance data can be made available to the reasoner via push or pull mechanisms. In this paper, we analyze those different implementation variations and evaluate them with respect to performance.

The rest of this paper is structured as follows: in section 2, we outline the basic concepts of semantic event processing. In section 3, we introduce a framework for UI integration and present an example for semantic event processing, which does not only demonstrate how ontologies can be used to facilitate cross-application interactions such as drag and drop, but also how to make those interactions more intelligent and comfortable for the user. In section 4, we compare the different implementation variations based on the framework introduced, and we show the impact on the system's performance with each variant. We conclude with a survey of related work, a summary, and an outlook on future developments.

2 Background

Following the survey in [9], event-driven approaches can be roughly categorized in *event detection* (dealing with the detection and creation of events) and *event processing* (dealing with reacting to those events, e.g. by creating new events and/or changing a system's state). Furthermore, logic-based and non-logic-based approaches can be distinguished, where the former encorporates formal logic to detect or process events, while the latter does not. Following this categorization, the work presented in this paper is a formal approach to event processing.

The term *semantic event processing* denotes the processing of events based on information on the semantics of that event [6]. The decisions in event processing may range from filtering and routing of events to the production of new events from the detected ones. An events' semantics may be comprised of information about the actor who caused the event, the objects with which the event was performed, and many more. Westermann and Jain propose a six-dimensional common event model, including temporal and spatial aspects as well as information about the involved actors and information objects [5]. As semantics can be described by using ontologies based on formal logics, semantic event processing is a subset of logic-based event processing.

One simple form of event processing systems are *publish-subscribe-systems*. Here, clients subscribe to events which deal with a certain topic or, more general, fulfill a certain set of conditions. Ontologies may be used to provide a hierarchy

of topics, in the simplest case. Sophisticated approaches can use more complex annotations of events and allow subscription not only on topics, but also on subgraphs of the annotations, e.g. by using SPARQL queries [10–12].

More advanced approaches of event processing do not only forward or discard events, but may also create new events or allow the triggering of actions if events occur under certain conditions, an approach known as *event-condition-action* (ECA) rules. There are several approaches to implementing event-driven systems based on ECA rules, e.g. in Datalog [13] or RuleML [14]. The approach presented in this paper uses F-Logic [15] for implementing event-processing rules.

3 A Framework for Integration on the UI Level

Application integration on the user interface level, or UI integration for short, means assembling applications in a way that their existing user interfaces are preserved. Typically, those interfaces are presented as individual parts on the screen within one common frame, such as a portal [16], or a mashup [17]. In each case, the user can simultaneously and parallely interact with different applications. Most current approaches to implementing cross-application interactions, such as drag and drop from one application to another, are still very limited: they require writing a larger amount of glue code in each of the applications to be integrated, most often leading to code-tangling and non-modular integrated systems [2]. In this section, we introduce an ontology-based framework for UI integration which aims at remedying those limitations by introducing centralized semantic event processing.

3.1 Framework Architecture

Our framework for UI integration is based on Java and uses OntoBroker [18] as a reasoner and rule engine. It can be used to integrate applications written in Java as well as applications which can be wrapped in Java components, e.g. Flex applications by using libraries such as JFlashPlayer [19].

The integrated applications are connected via an event exchange, where events can be sent in a directed or broadcast way. To allow a common understanding and sophisticated semantic event processing, each event is annotated with different information, such as the action that has caused the event, the component that this action was performed with, and the types of objects that are involved in the action (see Fig. 1). Events can than be analyzed by a reasoner, and re-distributed and further processed by using a rule engine [3].

As user interface integration requires formal and modular models of the integrated applications as well as the part of the real world for which the applications are built [20], we use ontologies in our framework for modeling the relevant parts of the applications as well as the real world [3]. With the help of those ontologies, the events can be annotated to make them universally and unambiguously understandable by all parties, and to allow sophisticated semantic event processing. In our framework, we use three types of ontologies:

1. An ontology of the user interfaces and interactions domain defines basic concepts such as user interface components and actions that can be performed with those components. Furthermore, it defines a basic category for information objects, which are objects in the application carrying information (and which are typically visualized in user interfaces). Events are annotated with this ontology to categorize the type of action underlying the event, and the reasoner uses this ontology for formulating the queries needed in event processing.
   The UI and interactions ontology is an integral part of our framework.
2. A real world domain ontology defines the objects from the applications' real world domain, such as banking, travel, etc. The real world domain ontology is used to annotate data objects passed between the integrated applications. Each data object and its attributes is annotated with concepts from the domain ontology. Furthermore, it provides background knowledge which can be used to formulate more elaborate interaction rules.
   The real world domain ontology is not part of our framework, since the framework is domain-independent. For integrating applications, an appropriate real world domain ontology needs to be created or reused.
3. For each integrated application, an application ontology defines this application's components and the interactions that are possible with them. The components and actions defined in the application ontologies are *subclasses* of the respective concepts defined in the user interfaces and interactions domain ontology. All applications and their components are instances of the components defined in these ontologies, and the information objects they process *represent* objects from the real world domain ontology.
   During the integration, a developer has to implement one application ontology per integrated application.

The application ontologies also contain interaction rules that define how the user can interact with the different integrated applications, formalized in F-Logic [15]. Those interaction rules are used to define cross-application behaviour.

The integration framework is currently used in the research project *SokNOS*[1], an integrated application in the emergency management field [21]. The SoKNOS system itself is a larger system which consists of 20 integrated applications. On average, there are nine integration rules per application, forming a total of 180 integration rules. In SoKNOS, we use an ontology of the emergency management domain [22] as a real world domain ontology.

Further details of the framework are introduced in section 4, where the different architectural variants are discussed.

3.2 Example Interaction Rules: Intelligent Drag and Drop

Examples for cross-application interaction include displaying related information in one application when selecting an object in another one [23], cross-application

[1] ***S****ervice-****o****rientiertierte Archite****K****turen für* ***N****etzwerke im Bereich* ***O****effentlicher* ***S****icherheit* (German for *service oriented architectures for networks in the field of public security).*

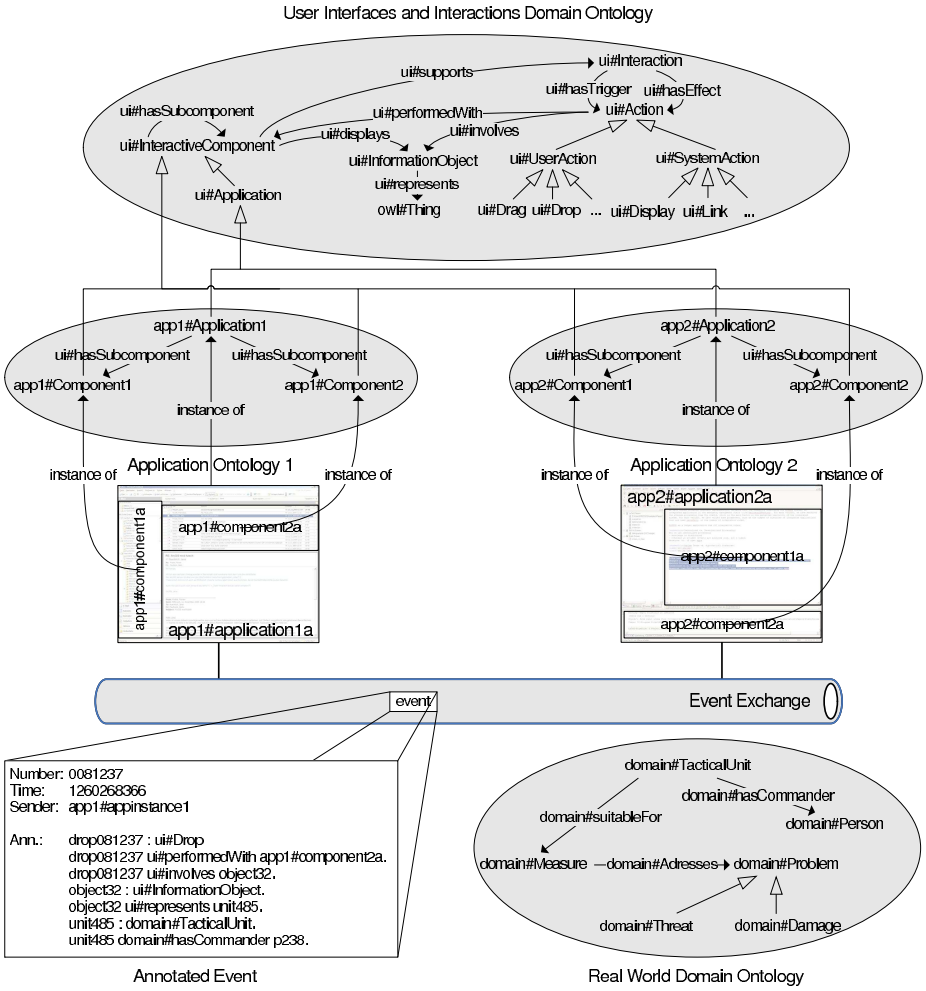


Fig. 1. The use of ontologies in our framework. Parts of the UI and interaction domain ontology, real world domain ontology and two application ontologies as well as their relations are shown. The event is annotated by using concepts from the different ontologies. A real world domain ontology from the emergency response domain is used in this example.

workflows (e.g. the creation of an object in one application requires entering data in another one), or dragging and dropping objects from one application to another one. In the following, we will introduce an example taken from the emergency response domain, which is implemented in the SoKNOS system [21].

The emergency response domain deals with entities like damages (such as fires, floodings, etc.), measures addressing those damages (such as fire fighting, building dams, etc.), and tactical units (such as fire brigade cars, helicopters, etc.). A useful interaction rule could be the following: if the user drops a tactical unit on a measure, the unit is allocated to that measure (which may e.g. result in issuing an order to that unit), given that the unit is suitable for fulfilling that measure. The (non-trivial) decision whether a tactical unit is suitable for a measure or not requires a certain amount of knowledge of the emergency response domain [22].

To formalize such a rule, we use three types of ontologies, as depicted above: an ontology of the user interfaces and interactions domain (prefixed *ui*), an ontology of the real world domain (here: emergency response, prefixed *domain*), and the application ontology of the integrated application which should support the interaction[2] (prefixed *app*). Each interaction rule contains a triggering event (the E in ECA, see section 5), an action to be performed as an effect (the A in ECA), and some conditions under which the interaction can be performed (the C in ECA). The example rule facilitating the drag and drop interaction, formalized in first order logic, looks as follows:

$$
\begin{aligned}
\forall c, t, io_1, io_2, u, m: \quad & app\#DisplayMeasureComponent\,(c) \wedge ui\#DropAction\,(t) \\
& \wedge ui\#InformationObject\,(io_1) \wedge domain\#TacticalUnit\,(u) \\
& \wedge ui\#InformationObject\,(io_2) \wedge domain\#Measure\,(m) \\
& \wedge ui\#represents\,(io_1, u) \wedge ui\#represents\,(io_2, m) \\
& \wedge ui\#performedWith\,(t, c) \wedge ui\#involves\,(t, io_1) \\
& \wedge ui\#displays\,(c, io_2) \wedge domain\#suitableFor\,(u, m) \\
\rightarrow \quad & \exists i, e : sys\#Interaction\,(i) \wedge ui\#LinkAction\,(e) \\
& \wedge ui\#supports\,(app, i) \wedge ui\#involves\,(e, io) \\
& \wedge ui\#hasTrigger\,(i, e) \wedge ui\#hasEffect\,(i, e) \qquad (1)
\end{aligned}
$$

The rule states that whenever a triggering event t is processed which states that an information object io_1 representing a tactical unit u is dropped on a component c displaying an information object io_2 representing a measure m, the resulting interaction i will have the effect e of linking the tactical unit to the measure. The last term of the rule's body – $domain\#suitableFor\,(u, m)$ – involves the usage of real world domain knowledge (i.e. the conditions under which a tactical unit is suitable for a measure). Thus, the semantics of the event

[2] As the example shows, the interaction is triggered by the drop action, not by the drag action. Therefore, the application supporting the interaction is the one where the object is dropped, not the one from which it is dragged. The interaction as it is defined above works regardless of which application the object is dragged from.

and the objects contained therein are used to provide an intelligent processing of that event. The rule also contains statements such as $ui\#displays\,(c, io_2)$, which need information on the system's state to be evaluated. In section 4.2, we will show alternatives of providing this information to the reasoner and rule engine.

An *intelligent* drag and drop interaction mechanism would not only allow the drag and drop itself, but also highlight the possible drop targets when the user starts dragging an object. Such a behaviour can be formalized as another interaction rule:

$$
\begin{aligned}
\forall c, t, io: \quad & ui\#InteractiveComponent\,(c) \wedge ui\#DragAction\,(t) \\
& \wedge ui\#InformationObject\,(io) \wedge ui\#involves\,(t, io) \\
& \wedge (\exists t_{hyp} : ui\#DropAction\,(t_{hyp}) \wedge ui\#involves\,(t_{hyp}, io) \\
& \quad \wedge \quad ui\#performedWith\,(t_{hyp}, c) \\
& \quad \rightarrow \quad \exists i_{hyp} : ui\#Interaction\,(i_{hyp}) \wedge ui\#hasTrigger\,(i_{hyp}, t_{hyp})) \\
\rightarrow \quad & \exists i, e : ui\#Interaction\,(i) \wedge ui\#HighlightAction\,(e) \\
& \wedge ui\#hasTrigger\,(i, e) \wedge ui\#hasEffect\,(i, e) \\
& \wedge ui\#performedWith\,(e, c) \qquad (2)
\end{aligned}
$$

This rules states that for any drag action t, if a corresponding (hypothetical) drop action t_{hyp} on a component c would serve as a trigger for *any* (hypothetical) interaction i_{hyp}, then this component is to be highlighted as an effect e of that drag action[3]. This rule is only defined once and fires for *every* drag and drop interaction performed with *any* object from *any* application, as no concepts from the domain ontology nor from any specific application ontology are referred to. It can thus be included in the user interfaces and interactions ontology. By using the natural language representation of the actions and objects involved in the computed events, the highlighted drop locations may also be augmented with tooltips (see Fig. 2).

4 Implementation Variants

We have tested different possible implementation variants with the framework described in section 3. For each variant, we have measured the average processing time for events, which is the main factor in the perceived reactivity of the integrated system. Throughout the experiments, we have varied some parame-

[3] Note that such a rule cannot be defined directly in most rule-based systems, due to the nested implication statement in the body. Therefore, further steps such as breaking down the rule into two or more rules are necessary for the actual implementation.

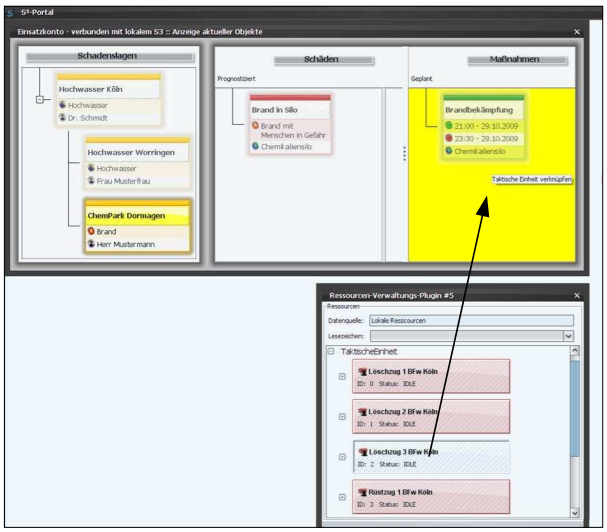


Fig. 2. A screenshot of intelligent drag and drop. Two integrated applications are shown: one for managing problems and measures addressing those problems (top), the other one for managing tactical units (bottom). If the user starts dragging a tactical unit (indicated by the arrow), the corresponding drop locations are highlighted. In addition, a tooltip is shown indicating the effect of dropping the object.

ters, such as the number of instances of integrated applications that are used in parallel[4], or the number of integration rules per application.[5]

4.1 Centralized vs. Decentralized Processing

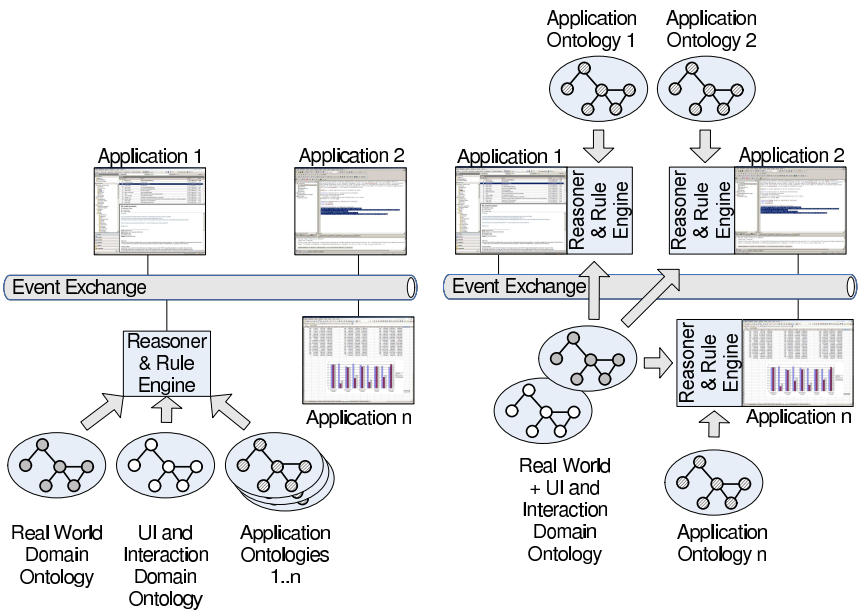
Semantic event processing involves operations such as event filtering or the creation of new events from those that are already known. Such operations may be performed either by one central unit, or in a decentralized way by each participant involved in the event exchange [2]. The two variants are depicted in Fig. 3.

Both approaches have advantages and drawbacks. A centralized event processing unit needs to know about each application ontology including the event processing rules defined therein (see Fig. 3(a)), thus leading to a large number of rules to be processed by one unit. This unit may become a bottleneck, and cross-dependencies between rules can slow the whole process down.

With decentralized event processing, on the other hand, each event has to be analyzed and processed multiple times, even if the result of such process-

[4] Note that this is not the number of applications that are integrated into one system, but the number of instances of those applications that are used in parallel, where the latter is usually lower than the former.

[5] The tests have been carried out on a Windows XP 64Bit PC with an Intel Core Duo 3.00GHz processor and 4GB of RAM, using Java 1.6 and OntoBroker 5.3.



(a) Implementation with global event processing

(b) Implementation with local event processing

Fig. 3. Framework architecture using global vs. local event processing. The global variant uses one central reasoning and rule engine which processes all domain and *all* application ontologies. The local variant uses several reasoning and rule engines which each process all domain ontologies and *only one* application ontology.

ing is that the event is discarded in most cases. Furthermore, common domain knowledge, which is an essential ingredient of semantic event processing, has to be replicated and taken into the processing process each time (see Fig. 3(b)). Those operations can also have negative impact on the overall event processing performance.

The measurements depicted in Fig. 4 reflect these mixed findings. Both approaches scale about equally well to larger numbers of integrated applications (and thus, larger total numbers of integration rules). For integrated applications with a smaller number of integration rules per application, global processing is about 40% faster than local processing; with a growing number of integration rules per application, the difference is not as significant, but global processing is still slightly faster.

The main reason why global event processing turns out to be faster is that each event has to be processed only once, not once per receiving application – this advantage is not trumped by the larger number of rules in a centralized approach.

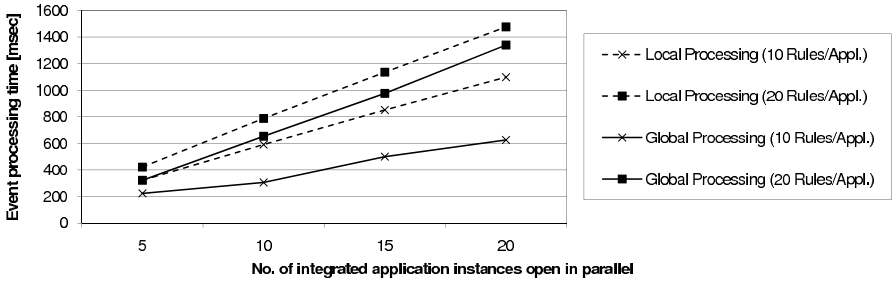


Fig. 4. Event processing performance comparison between using centralized and decentralized processing. Event processing time has been measured for 5 to 20 instances of integrated applications used in parallel, with 10 to 20 integration rules each.

4.2 Pushing vs. Pulling of Instance Data

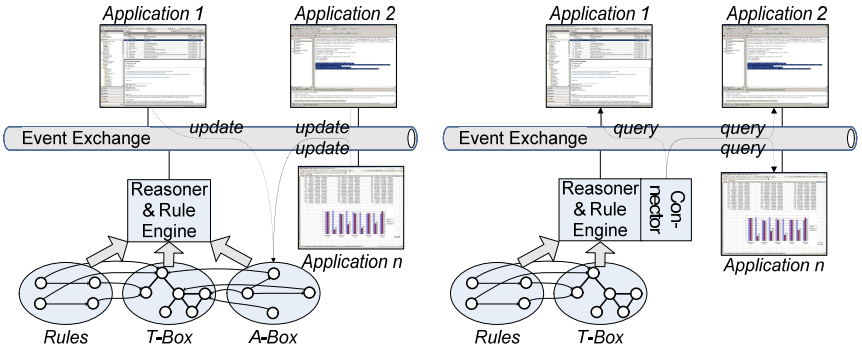
Ontologies have two parts, a T-Box, which contains the definitions of classes and relations, and an A-Box, which contains the information about instances of those classes. Reasoning about an integrated UI and events requires information about the system at run time, such as the application instances that are currently open, the components that constitute them, and the information objects they currently process (such as the example rule no. 1, which uses information about which components display which information objects as a condition). These kind of information are part of the ontologies' A-boxes.

There are two possible ways of implementing this A-box. A straight forward approach is to use an instance store for the instance data (see Fig. 5(a)). In this approach, integrated applications are responsible for sending (i.e. *pushing*) regular updates to assure that the instance store is always synchronized with the system its instances represent.

Another approach is to use the integrated system itself as an instance store. In this approach, an instance store connector is used to answer the reasoner's queries when needed. When called upon a query, it passes the query to the applications, collects (i.e. *pulls*) their answers and returns the instance data to the reasoner (see Fig. 5(b)).

Both approaches have their advantages and drawbacks. Pushing instance data into an instance store causes redundancy, since the data contained in the instance store is also contained in the integrated system itself. Furthermore, to make sure that each query is answered correctly, the updates sent by the applications and the queries issued by the reasoner need to be properly serialized. This may result in slower query execution if a larger number of updates is queued before the query. These problems are avoided when using a pull-based approach.

On the other hand, a pull-based approach includes that instance data is retrieved from external systems while a query is processed. Depending on the reactivity of those systems, the overall query processing time might also increase.



(a) Push-based implementation (b) Pull-based implementation

Fig. 5. Framework architecture using a push-based vs. a pull-based approach. Both variants are demonstrated in an implementation combined with global event processing. In comparison to Fig. 3, the individual ontologies' rules, T-boxes and A-boxes are subsumed in this figure.

A comparison of both implementations is shown in Fig. 6, using applications with ten integration rules each, and varying the number of applications that are integrated at the same time. For applications issuing five updates per second on average, both approaches scale equally well. For applications issuing ten updates per second, the pushing approach does not scale anymore.

With a pushing approach, the updates from the individual applications form an ever growing queue, slowing down the whole system until a total collapse. This behavior will occur with every reasoning system as soon as the frequency of updates exceeds the inverse of the time it takes to process an update. Thus, only approaches using the pulling approach scale up to larger integrated systems.

5 Related work

Few works exist which inspect the efficiency and scalability issues of using semantic events. In the field of *event detection*, the approach described in [6] uses modular event ontologies for different domains to enable more sophisticated semantic event processing mechanisms based on ECA rules. The authors propose to use reasoning to detect more complex event patterns in a stream of events described by ontologies, and to tell important events from non-important ones. The authors name scalability and real time processing as challenges, although no actual numbers are presented.

The idea of modular event ontologies is further carried out in [24] and [25]. The authors propose a modular approach where not only different ontologies, but also different languages can be used for individual parts of ECA rules. In this very versatile approach, applications on the semantic web can register their rules as well as the corresponding processing units. Annotated events are then processed in a distributed fashion by dynamically calling the registered processing

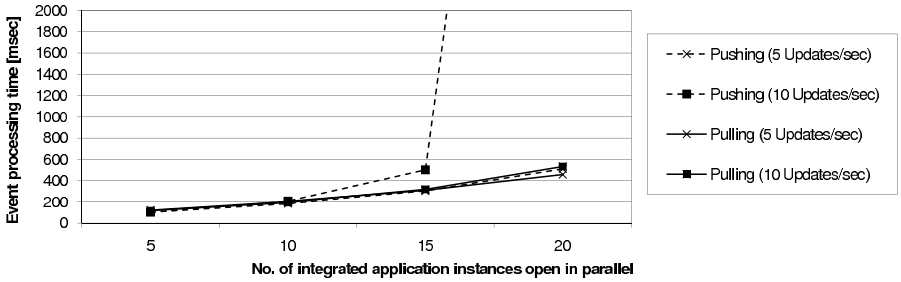


Fig. 6. Event processing performance comparison between pushing and pulling instance data. Event processing time has been measured for 5 to 20 instances of integrated applications used in parallel, with ten integration rules each, working at an update frequence of 5 to 10 updates per second. The figure has been cut at the two second mark where reasonable work is assumed to become impossible, as indicated in the HCI literature [7, 8]. Systems integrated from applications issuing 10 updates per second collapsed when integrating more than 15 applications using the pushing approach.

units. As such an approach involves (possibly remote) method calls during the event processing procedure, it may not be suitable in applications with real-time requirements.

The work discussed in [26] also uses events described with ontologies. The authors focus on mining information from a large database of events, allowing queries about characteristics such as the social relationships between the actors involved in events as well as the temporal and spatial relationships between events. Although real-time processing is not a necessary property in this case, the authors discuss the use of high-performance triple stores to allow fast query answering. A similar approach is described in [27], where operators for detecting complex events from a database of atomic events are introduced, based on the model by Westermann and Jain mentioned above.

In the field of *event processing*, there are a few examples of using semantic event processing in the user interface area. One approach is described in [28]. Annotated web pages can be used to create events that also carry information about the semantics of the involved objects. The authors present an approach for producing, processing, and consuming those events, and for constructing complex events from atomic ones. In their approach, the annotations may be used to formulate rules, but no reasoning is applied in processing the events. The approach is evaluated with respect to performance (although no variants are discussed) by means of the example of context-sensitve advertising, and it proves high scalability, but at the price of not incorporating domain knowledge and reasoning in the event processing mechanism at all (and thus not facilitating semantic event processing as defined above).

Another application of semantic event processing in the user interface area is shown in [29]. Here, it is employed for run time adaptation of user interfaces,

allowing the incorporation of domain knowledge in the reasoning process. The authors present an e-government portal and show the use case of presenting appropriate content based on the users context, and evaluate their approach with regard to run time, reporting event processing times between a few and a few thousand milliseconds, depending on the number of rules and the number of instances that are used to answer a query. The authors present performance measures that allow a direct comparison to our work: While the performance marks are about the same, our approach allows for formulating arbitrary rules on the ontologies, the approach presented in that paper allows only the use of the class hierarchy. Thus, examples as shown in section 3.2 would not be possible with that approach.

6 Conclusion and Outlook

Application integration on the user interface level is an emerging area of research. With the example of an intelligent drag and drop mechanism, we have shown how semantic event processing – i.e. event processing incorporating domain knowledge – can be employed to build more intelligent integrated systems.

Incorporating domain knowledge makes the event processing task more complex and thus more time consuming. As user interfaces require fast reactivity, our main focus was on the performance of the approaches, i.e. on the time to process an event. In this paper, we have introduced a framework for UI integration, which is capable of integrating Java-based as well as non-Java-based user interfaces, using ontologies and rules for the integration. Based on this framework, we have analyzed different variants of implementing a semantic event exchange based on the commercial off-the-shelf reasoner OntoBroker. We have introduced architectures for centralized and decentralized event processing, and for pushing and pulling instance data needed by the reasoning and rule engine.

While there are only minor differences between centralized and decentralized event processing, the pushing approach using a continuously updated instance store does not scale at all to larger and more complex integrated systems. Therefore, the key finding is that high-performance semantic event processing can only be implemented using the pulling approach.

For the work introduced in this paper, we have used a very basic and straight forward connector implementation (which still outperforms the pulling approach). In the future, we aim at improving this implementation, e.g. using more sophisticated caching approaches and rule sets that minimize the number of connector invocations. These actions may further improve the performance measures.

The framework for UI integration introduced in this paper is also subject to further research work. Current research addresses problems such as mediating between heterogeneous class models based on semantic annotation, and on incorporating more detailed context information, which can then be used to formulate more concise interaction rules (e.g. including restrictions based on the users' rights and context).

In summary, we have shown approaches to implement efficient semantic event processing, and discussed its implementation in the scenario of application integration on the user interface level.

Acknowledgements

The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research under grant no. 01ISO7009. The author would like to thank the reviewers for their helpful and constructive remarks.

References

1. Linthicum, D.S.: Enterprise Application Integration. Addison Wesley (1999)
2. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing **11**(3) (2007) 59–66
3. Paulheim, H.: Ontologies for User Interface Integration. [30] 973–981
4. Myers, B.A., Rosson, M.B.: Survey on user interface programming. In: CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM (1992) 195–202
5. Westermann, U., Jain, R.: Toward a Common Event Model for Multimedia Applications. IEEE MultiMedia **14**(1) (2007) 19–29
6. Teymouriana, K., Paschke, A.: Towards semantic event processing. In: DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, New York, NY, USA, ACM (2009) 1–2
7. Miller, R.B.: Response time in man-computer conversational transactions. In: AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I, New York, NY, USA, ACM (1968) 267–277
8. Shneiderman, B.: Response Time and Display Rate in Human Performance with Computers. ACM Computing Surveys **16**(3) (1984) 265–285
9. Schmidt, K.U., Anicic, D., Stühmer, R.: Event-driven Reactivity: A Survey and Requirements Analysis. In: Proceedings of the 3rd International Workshop on Semantic Business Process Management. (2008)
10. Wang, J., Jin, B., Li, J.: An ontology-based publish/subscribe system. In: Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, New York, NY, USA, Springer-Verlag New York, Inc. (2004) 232–253
11. Skovronski, J., Chiu, K.: An Ontology-Based Publish Subscribe Framework. In: Proceedings of the 8th International Conference on Information Integration and Web-based Applications & Services (iiWAS2006). (2006)
12. Murth, M., Kühn, E.: Knowledge-based coordination with a reliable semantic subscription mechanism. In: SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing, New York, NY, USA, ACM (2009) 1374–1380
13. Anicic, D., Stojanovic, N.: Towards Creation of Logical Framework for Event-Driven Information Systems. In Cordeiro, J., Filipe, J., eds.: ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise Information Systems, Volume ISAS-2, Barcelona, Spain, June 12-16, 2008. (2008) 394–401
14. 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)

15. Angele, J., Lausen, G.: 2. International Handbooks on Information Systems. In: Ontologies in F-Logic. Springer (2004) 29–50
16. Wege, C.: Portal Server Technology. IEEE Internet Computing **6**(3) (2002) 73–77
17. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing **12**(5) (Sept.-Oct. 2008) 44–52
18. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In Meersman, R., Tari, Z., Stevens, S.M., eds.: Database Semantics - Semantic Issues in Multimedia Systems, IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics (DS-8), Rotorua, New Zealand, January 4-8, 1999. Volume 138 of IFIP Conference Proceedings., Kluwer (1999) 351–369
19. Software, V.: JFlashPlayer Web Page. `http://www.jpackages.com/jflashplayer` (2009)
20. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM (2007) 923–932
21. Paulheim, H., Döweling, S., Tso-Sutter, K., Probst, F., Ziegert, T.: Improving Usability of Integrated Emergency Response Systems: The SoKNOS Approach. In: Proceedings "39. Jahrestagung der Gesellschaft für Informatik e.V. (GI) - Informatik 2009". Volume 154 of LNI. (2009) 1435–1449
22. Babitski, G., Probst, F., Hoffmann, J., Oberle, D.: Ontology Design for Information Integration in Catastrophy Management. In: Proceedings of the 4th International Workshop on Applications of Semantic Technologies (AST'09). (2009)
23. Eick, S.G., Wills, G.J.: High Interaction Graphics. European Journal of Operational Research **84** (1995) 445–459
24. Alferes, J.J., Eckert, M., May, W.: Evolution and Reactivity in the Semantic Web. In Bry, F., Maluszynski, J., eds.: Semantic Techniques for the Web. Volume 5500 of Lecture Notes in Computer Science. (2009)
25. Behrends, E., Fritzen, O., May, W., Schenk, F.: Combining ECA Rules with Process Algebras for the Semantic Web. In: RULEML '06: Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web, Washington, DC, USA, IEEE Computer Society (2006) 29–38
26. Aasman, J.: Unification of Geospatial Reasoning, Temporal Logic, & Social Network Analysis in Event-Based Systems. In: DEBS '08: Proceedings of the Second International Conference on Distributed Event-Based Systems, New York, NY, USA, ACM (2008) 139–145
27. Rafatirad, S., Gupta, A., Jain, R.: Event composition operators: ECO. In: EiMM '09: Proceedings of the 1st ACM international workshop on Events in multimedia, New York, NY, USA, ACM (2009) 65–72
28. Stühmer, R., Anicic, D., Sen, S., Ma, J., Schmidt, K.U., Stojanovic, N.: Lifting Events in RDF from Interactions with Annotated Web Pages. [30] 893–908
29. Schmidt, K.U., Dörflinger, J., Rahmani, T., Sahbi, M., Thomas, L.S.S.M.: An User Interface Adaptation Architecture for Rich Internet Applications. In Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M., eds.: The Semantic Web: Research and Applications. Proceedings of the 5th European Semantic Web Conference, ESWC 2008. Number 5021 in LNCS (2008) 736–750
30. Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K., eds.: The Semantic Web - ISWC 2009. Volume 5823 of Lecture Notes in Computer Science., Springer (2009)