Improving UI Integration with Formal Semantics

Heiko Paulheim and Florian Probst
SAP Research CEC Darmstadt
Bleichstrasse 8
64283 Darmstadt, Germany
{heiko.paulheim,f.probst}@sap.com

ABSTRACT

There are different strategies to integrate software systems: integration on the data layer, on the business logic layer, and on the user interface layer. The latter area, recently gaining attention by the rising popularity of Web 2.0 mashups, can benefit heavily from formal models and ontologies. In this paper, we argue why integration on the user interface level requires formal semantics. We present a framework using ontologies for user interface integration and discuss the relevant research tasks and links to related research fields.

ACM Classification Keywords
D.2.2 Software Engineering: Design Tools and Techniques—*User Interfaces*; D.2.13 Reusable Software: Reuse models

General Terms
Design, Algorithms

Author Keywords
Ontologies, User Interfaces, Integration

INTRODUCTION

Software applications are typically built in three layers: the data layer, the business logic layer, and the user interface layer. Therefore, three integration approaches can be identified: integrating on the data layer (and developing a common business logic and UI), integrating on the business logic layer (and developing a common UI), and integrating on the UI layer [6] (see Fig. 1). The latter has two important advantages: From an engineering point of view, existing UIs can be reused, thus increasing the overall degree of reuse, and modularization is facilitated [18]. From a usability point of view, users do not have to learn new UIs [17].

Integration on the user interface layer results in an integrated *system*, consisting of individual *applications*, whose UI as well as business logic and data layer (or a subset thereof) are preserved. The different applications behave as they would in a non-integrated system. In addition, cross-application interactions are possible, such as highlighting related information in different applications, or dragging and dropping objects from one application to another. Therefore, the integrated system is more than the sum of its parts. Common UI integration approaches, such as portals and mash-ups, have drawbacks and limitations when it comes to developing such interactions: they require manual coding in languages such as JavaScript which in most cases creates dependencies between the individual applications, thus contradicting the idea of a modular system and complicating system maintenance (e.g. by exchanging components) [6].

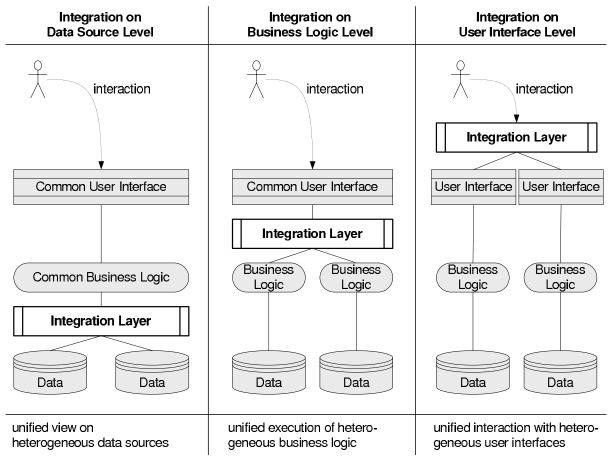
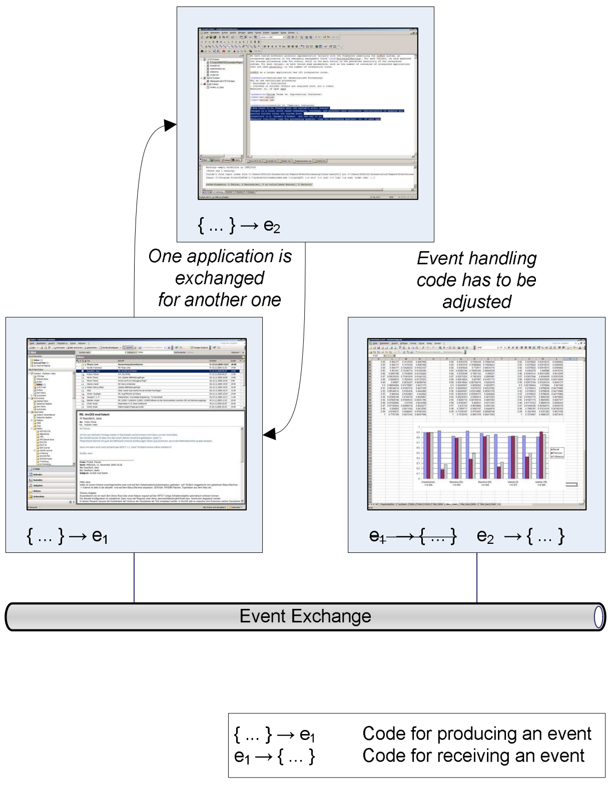

Integration on Data Source Level
Integration on Business Logic Level
Integration on User Interface Level
interaction
Common User Interface
Common Business Logic
Integration Layer
Data
User Interface
Business Logic
unified view on heterogeneous data sources
unified execution of heterogeneous business logic
unified interaction with heterogeneous user interfaces

Figure 1. Three different approaches for integrating software systems [17]

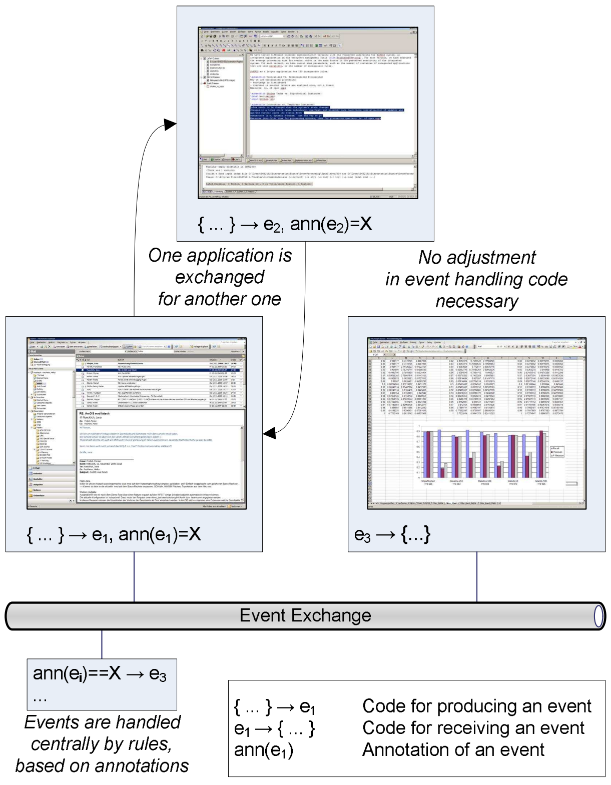
WHY FORMAL SEMANTICS ARE NEEDED

In integrated systems, applications need to react to events that take place in other applications. UI integration approaches are thus most commonly developed based on event exchange mechanisms, where different variants exist, such as centrally-mediated or component-to-component communication [6]. Without a sophisticated event processing approach, those mechanisms create dependencies between the integrated user interface components: applications rely on events of a certain type to happen, and specialized code is written to react to that sort of event. Thus, when one of the integrated application is exchanged for another one, the reacting application needs to be adapted as well (see Fig. 2(a)). Therefore, the principle of modularity is violated.

Furthermore, an event message which is sent from one application to another has to be understood by the receiving application, and this understanding has to be mutual for the sender and the receiver. This raises the question: "How do we capture the meaning of an event?" A common model for the contents of the event


{ ... } → e2
One application is exchanged for another one
Event handling code has to be adjusted
{ ... } → e1
e1 → { ... } e2 → { ... }
Event Exchange
{ ... } → e1 Code for producing an event
e1 → { ... } Code for receiving an event

(a) Point-to-point event processing without annotations


{ ... } → e2, ann(e2)=X
One application is exchanged for another one
No adjustment in event handling code necessary
{ ... } → e1, ann(e1)=X
e3 → {...}
Event Exchange
ann(ei)==X → e3
...
Events are handled centrally by rules, based on annotations
{ ... } → e1 Code for producing an event
e1 → { ... } Code for receiving an event
ann(e1) Annotation of an event

(b) Centrally mediated event processing with annotations

Figure 2. Different variants of implementing event based processing in integrated user interfaces. Without annotations, the event processing code has to be altered every time one application is exchanged for another one. Annotations and rules can be used to implement an integrated user interface in a modular way such that applications can be exchanged without causing further changes.

as well as its metadata, such as time and place, is required [21] to semantically *annotate* the events. In the semantic web community, ontologies are a well-researched mechanism which can be used to annotate e.g. data sources and web services [5, 10].

Ontologies defining the possible types of events, the objects that may be contained therein, and the components which can send and receive those events, can provide a formal model for information exchange between integrated applications. Events as well as applications and their components can then be annotated by using those ontologies, thus providing a mutual understanding of the applications to be integrated and the messages they send to each other. Event processing rules evaluating those annotations can be used to build more modular integrated user interfaces, since they dispose the need for adjusting event processing code upon changes in other integrated parts (see Fig. 2(b)).

Therefore, we propose using ontologies to create formal descriptions of UIs, which can be used in UI integration to provide the basis for a mutual understanding of exchanged messages and pave the way to modular integrated user interfaces [17, 18].

AN ONTOLOGY OF THE DOMAIN OF USER INTERFACES AND INTERACTIONS

For describing and understanding an event, information such as the following are necessary:

- What sort of action caused the event?
- Which information objects are involved in the action?
- Which component was the action performed with?
- Who performed the action?

Some additional information might become necessary when formulating rules for event processing as shown in Fig. 2(b):

- Is a reacting component (i.e. a receiver for the triggered event) available?
- In which state is that component (e.g. activated)?
- Do any domain specific constraints hold on the objects involved in an action?

These questions can be used to derive the necessary core concepts of the required ontology or ontologies [9]. In our approach, we have chosen to use three types of ontologies (see Fig. 3) [18]:

- An ontology of the domain of user interfaces and interactions defines basic concepts such as interactive components, actions, and information objects (see below).
- An ontology of the integrated application's real world domain (such as banking, travel, etc.) defines the real world concepts represented by the information objects the application deals with.

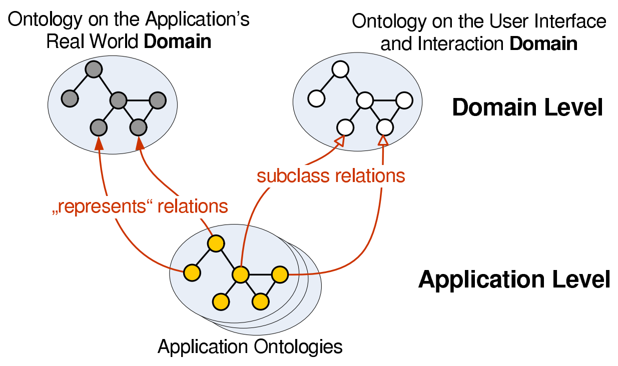

Ontology on the Application's Real World Domain
Ontology on the User Interface and Interaction Domain
Domain Level
subclass relations
„represents" relations
Application Level
Application Ontologies

Figure 3. Two domain ontologies and several application ontologies are used for integration on the user interface level [17].

- Based on these domain-level ontologies, one application-level ontology per integrated application is developed, defining the UI components and interactions of this UI by extending the base concepts from the user interfaces and interactions ontology, as well as the information objects that this UI uses.

While the UI and interactions domain ontology is part of the framework, the real world domain ontology is not, and there are no links between them. Therefore, our framework can be used to develop integrated systems for different real world domains by using an (existing or custom-built) real world domain ontology. All application ontologies use the two domain-level ontologies to describe the individual applications and their capabilities. Thus, the domain level provides the common ground and background knowledge upon which the application ontologies are defined [10].

The application ontologies also contain integration rules, which are used to facilitate an intelligent event processing mechanism, as described above. Following the Event-Condition-Action (ECA) pattern [1], those rules use concepts from the different ontologies to define the events triggering an interaction, the conditions under which those interactions are possible, and the actions (i.e. new events) to be triggered within an interaction.

Fig. 4 shows an excerpt of the ontology of the user interfaces and interactions domain. Each integrated *application* consists of several (potentially nested) *interactive components* and *supports* certain interactions. Each *interaction* has a *trigger* and an *effect*, each of which can be either a *user action* or a *system action* (at the moment, we do not further distinguish the individual users or system components who have actually performed the action), which are further classified (not shown in the figure) into different user actions (such as *drag*, *drop*, *select*, etc.) and system actions (such as *highlight*, *hide*, *delete*, etc.). Each *action* can involve one or more *information objects*, which represent things in the real world. It is important to notice that this modeling approach distinguishes things in the real world and information

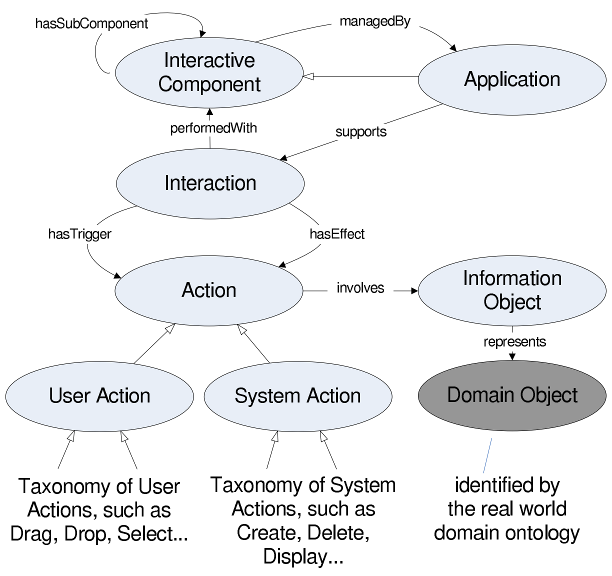


Figure 4. The core concepts of the ontology of the user interfaces and interactions domain. The dark grey ellipse marked "Domain Object" indicates a placeholder which can be filled by concepts from a real world domain ontology.

objects in the system (the first being represented by the latter), which is necessary for accurately describing the system functionality, but often neglected [3, 13].

PROTOTYPE FRAMEWORK

We have developed a prototype of an integration framework for showing the feasibility of UI integration based on ontologies, based on Java and the OntoBroker [16] reasoner. Fig. 5 shows the basic architecture of our integration framework. The application ontologies of each integrated application – which can be implemented in Java as well as in other technologies, such as Adobe Flex – are evaluated at run-time: When an event occurs, such as a select action performed by a user, an event annotated with the respective concepts in the ontologies is sent to the ontology processing unit. It reads the event, uses the reasoner to compute the reactions that have to be performed by other applications, and notifies them with new events (such as a command for highlighting information objects related to the one selected by the user).

On the implementation level, every application may use its own class model for handling data, and the *original* information objects involved in an action are sent along with the corresponding event. Those information objects have to be annotated with elements from the real world domain ontology, so the reasoner and other applications can analyze and process those objects correctly. Annotations of information object classes and their attributes are stored in a repository.

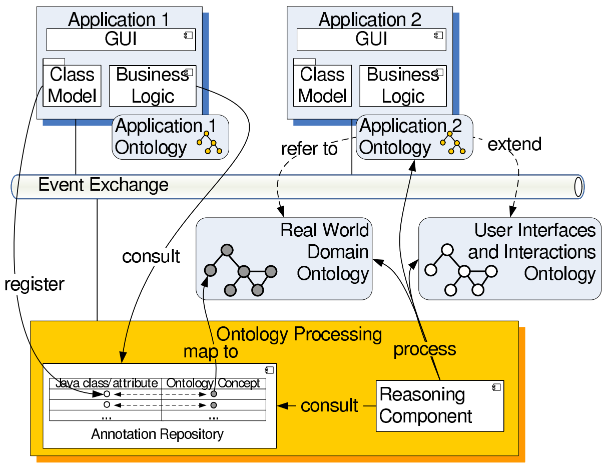


Figure 5. Framework architecture [17]

RESEARCH TASKS

From our previous work in user interface integration, we have identified three central research tasks which are relevant for the research community concerned with semantic and model-driven user interfaces.

First and foremost, **a framework for semantic models of user interfaces** which is versatile enough to cover the vast variety of user interfaces, from WIMP to tangible UIs, is needed. Such a framework is also necessary for model-driven UI development: In both cases, an adequate modeling depth and level of detail have to be determined, and both areas would benefit from standardized semantic models for user interfaces. Reverse engineering mechanisms turning existing user interfaces into models would also be desirable (yet probably utopian for the near future, at least in a generic fashion).

Second, we employ reasoning on user interfaces at run-time. There are other approaches where run-time reasoning can be employed, such as UI personalization and adaptation, or implementing context-sensitive interactions. In UI programming, the reactivity of the resulting UI is a paramount criterion, which poses significant **performance and scalability challenges** to the use of ontology reasoners. First experiments with different performance optimization strategies show that our framework can guarantee an acceptable performance way below the "magical" two seconds [15, 20].

Third, semantic models may also be employed to provide intelligent user assistance. We have shown the example of highlighting relevant drop locations and augmenting them with tooltips [18]: whenever a user drags an object, the corresponding drop locations are highlighted, and tooltips indicate what would happen if the user dropped the object there. The same integration rules as discussed above are evaluated for generating the user assistance. Here, research tasks such as **intuitive visualizations and/or verbalizations of ontologies** and rules have to be addressed.

RELATED WORK

The need for formal models for user interface integration has been pointed out in [22], where the authors name four requirements for those formal models: they are supposed to be simple, formal, human readable, and modular. While simplicity lies in the eye of the beholder, ontologies meet at least the other three criteria.

A universal approach using ontologies for user interface integration has not yet been presented. There have been examples for employing ontologies in portal and mashup development [2], which help showing related data in different applications at the same time. Another approach shows how data links can be established for automatic form-filling [7]. Unlike in the approach presented in this paper, these examples are restricted to data visualization and form-filling, i.e. no further cross-application interactions are possible.

There are several fields related to user interface integration in which ontologies are successfully employed. Ontologies may be used to capture and track the requirements in user interface engineering [8], and as input models to code generators for automatic generation of user interface code [14]. Futhermore, they may be used to develop repositories for annotated components and patterns to assist the UI developer [11, 12].

SUMMARY

The integration of user interfaces by using ontologies is an important and promising area of application for formal models in user interfaces. Like for other areas, two main ingredients are needed: precise formal models, and high-performance implementations of semantic processing mechanisms that scale up to larger integrated UIs. As integration requires an unambiguous and mutually understood information exchange between the integrated applications, semantic annotation of the exchanged information is essential.

The UI integration framework described in this paper has been successfully used for building the integrated emergency management platform *SoKNOS*, where twelve applications addressing different tasks in emergency response have been integrated on the user interface level [19]. In the SoKNOS project, an ontology for the real world domain of emergency response has been developed, which is used for the application ontologies and their integration rules [4].

Acknowledgements

The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research under grant no. 01ISO7009.

REFERENCES

1. D. Anicic and N. Stojanovic. Towards Creation of Logical Framework for Event-Driven Information Systems. In J. Cordeiro and J. Filipe, editors, *ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise*

Information Systems, Volume ISAS-2, Barcelona, Spain, June 12-16, 2008, pages 394–401, 2008.

2. A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandecic. The Two Cultures: Mashing Up Web 2.0 and the Semantic Web. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM.
3. N. Arora, R. Westenthaler, W. Behrendt, and A. Gangemi. Information Object Design Pattern for Modeling Domain Specific Knowledge. In *1st ECOOP Workshop on Domain-Specific Program Development*, 2006.
4. G. Babitski, F. Probst, J. Hoffmann, and D. Oberle. Ontology Design for Information Integration in Catastrophy Management. In *Proceedings of the 4th International Workshop on Applications of Semantic Technologies (AST'09)*, 2009.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
6. F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul. Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing*, 11(3):59–66, 2007.
7. O. Díaz, J. Iturrioz, and A. Irastorza. Improving portlet interoperability through deep annotation. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 372–381, New York, NY, USA, 2005. ACM.
8. E. Furtado, J. J. V. Furtado, W. B. Silva, D. W. T. Rodrigues, L. da Silva Taddeo, Q. Limbourg, and J. Vanderdonckt. An Ontology-Based Method for Universal Design of User Interfaces. In *Task Models and Diagrams For User Interface Design (TAMODIA 2002)*, 2002.
9. M. Grüninger and M. S. Fox. Methodology for the Design and Evaluation of Ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995*, 1995.
10. N. Guarino, editor. *Formal Ontology and Information Systems.* IOS Press, 1998.
11. H.-J. Happel, A. Korthaus, S. Seedorf, and P. Tomczyk. KOntoR: An Ontology-enabled Approach to Software Reuse. In K. Zhang, G. Spanoudakis, and G. Visaggio, editors, *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE)*, pages 349–354, 2006.
12. S. Henninger, M. Keshk, and R. Kinworthy. Capturing and Disseminating Usability Patterns with Semantic Web Technology. In *CHI 2003 Workshop: Concepts and Perspectives on HCI Patterns*, 2003.
13. E. Klien and F. Probst. Requirements for Geospatial Ontology Engineering. In F. Toppen and M. Painho, editors, *8th Conference on Geographic Information Science (AGILE 2005)*, pages 251–260, Estoril, Portugal, 2005.
14. B. Liu, H. Chen, and W. He. Deriving User Interface from Ontologies: A Model-Based Approach. In *ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 254–259, Washington, DC, USA, 2005. IEEE Computer Society.
15. R. B. Miller. 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*, pages 267–277, New York, NY, USA, 1968. ACM.
16. ontoPrise. OntoBroker Website. `http://www.ontoprise.de/de/en/home/products/ontobroker.html`, 2009.
17. H. Paulheim. Ontologies for User Interface Integration. In A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 973–981. Springer, 2009.
18. H. Paulheim. Ontology-based Modularization of User Interfaces. In G. Calvary, T. C. N. Graham, and P. Gray, editors, *Proceedings of The 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'2009)*, pages 23–28. ACM, 2009.
19. H. Paulheim, S. Döweling, K. Tso-Sutter, F. Probst, and T. Ziegert. 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*, pages 1435–1449, 2009.
20. B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *ACM Computing Surveys*, 16(3):265–285, 1984.
21. U. Westermann and R. Jain. Toward a Common Event Model for Multimedia Applications. *IEEE MultiMedia*, 14(1):19–29, 2007.
22. J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera. A framework for rapid integration of presentation components. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 923–932, New York, NY, USA, 2007. ACM.