Ontology-Based Application Integration on the User Interface Level

Heiko Paulheim*1, Universität Mannheim

* Correspondence author: heiko@informatik.uni-mannheim.de

Summary Application integration can be performed on three levels: the data source level, the business logic level, and the user interface level. Approaches of the latter kind include, e.g., portal and mashup solutions. Deeper integration offering rich interaction possibilities usually leads to strong dependencies between applications, and requires deep knowledge of those applications. To reduce such dependencies, ontologies have been proposed as an inter-lingua mediating between the applications. While there have been approaches for using ontologies for integration on the data source and business logic level, this dissertation is the first approach for ontology-based application integration on the user interface level. It discusses the development of suitable formal ontologies, the use of reasoning and rule-based event processing, as well as aspects such as performance and seamless cross-technological interaction.

Keywords ACM CSS → Software and its engineering → Integration frameworks, ACM CSS → Computing methodologies → Ontology engineering, ACM CSS → Theory of computation → Semantics and reasoning, ACM CSS → Human-centered computing → User interface programming

1 Dr. Paulheim earned his doctorate at the Faculty of Computer Science of the Technische Universität Darmstadt. The examiners were Prof. Dr.-techn. Johannes Fürnkranz, Technische Universität Darmstadt, and Prof. Dr.-Ing. Jürgen Ziegler, University of Duisburg-Essen. The dissertation of Dr. Paulheim titled "Ontologiebasierte Applikationsintegration auf Nutzerschnittstellenebene" has been recommended to the GI-Dissertation Award 2011 by the Technical University of Darmstadt.

Introduction Following the classical three-tier model by Martin Fowler, software applications typically consist of three layers: the data source layer, the business logic layer, and the user interface layer [2]. Thus, there are three kinds of approaches to application integration: integration on the...
data source level (i.e., integrating data sources and developing a new business logic and user interface), integration on the business logic level (i.e., integrating business logics and developing a new user interface), and integration on the user interface level (i.e., reusing all three layers of the integrated applications).

Integration on the user interface level is a desirable approach both from an end user’s and a developer’s perspective. End users working with integrated applications that reuse existing user interfaces, which they may already be familiar with, will experience steeper learning curves. Developers, on the other hand, will save the majority of efforts of developing a new user interface – around 50% of the efforts in software development are devoted to the user interface [3].

There are quite a few approaches for performing application integration on the user interface level. For example, portal and mashup frameworks allow for integration of web-based user interfaces. However, most of the existing solutions – both open source and commercial – share a common set of shortcomings. There are hardly any abstract models used, instead, integration can only be performed with deep knowledge about the integrated applications, and by introducing deep dependencies. Furthermore, the support for overcoming conceptual and technological heterogeneity is most often very limited.

Similar challenges exist on the data source and the business logic level. Many approaches have been discussed in the past that propose the use of ontologies, i.e., formal, machine-interpretable models of a domain, as an inter-lingua to reduce the dependencies between applications [8]. Ontologies have been used for database integration as well as for semantic web services, but no ontology-based approaches for integration on the user interface level have been proposed so far.

2 Approach

Application integration on the user interface level means to reuse all layers of the integrated applications, including their user interface, and facilitate interactions across the borders of applications. Such interactions may include displaying related information in one application when selecting an object in another one, or dragging and dropping objects from one application to another. In order to facilitate cross-application interaction, integrated applications need to know about each other’s capabilities, states, and information objects. For example, when selecting a customer object in an application A1, another application A2 should react by displaying all transactions related to that customer. To implement such an interaction, A2 needs to be informed that a customer object has been selected. It needs to have access to the customer information object (even if it uses a different data model to define customers), and it needs to keep track of the state of A1 (e.g., be informed if another object is selected), or if customer information object itself is changed.

Implementing such an information exchange in an ad hoc way would lead to tight coupling and a large number of dependencies. The approach taken in this work thus aims at decoupling applications by the use of formal ontologies.

2.1 Ontologies, Rules, and Reasoning

Many integration approaches on the user interface level are based on event exchange mechanisms [1]. Ontologies can be used for decoupling using those approaches, as they may provide a common formal language for the events being exchanged. We use ontologies to provide a formal annotation of the event type (e.g., something has been selected), the involved user interface component (e.g., a button), and the corresponding information object (e.g., a customer).

An ontology of the domain of user interfaces and interactions describes the categories of interactive components and the interactions a user can perform with them. A complementary real world domain ontology is used for defining objects such as customers and transactions, which are specific to the integrated application’s domain. As depicted in Fig. 1, relying on the ontology-based annotations helps decoupling the integrated applications.

To further reduce the dependencies between the integrated applications, rules and reasoning may be used. Rules can be defined based on the ontologies to describe abstract notions of interactions. For example, a map component may define a rule that it displays objects that have a position, without having to define those objects a priori. That means that the map component may also support interaction with applications processing object types that were not known at implementation time.

Reasoning may be used to further externalize business knowledge from the application. For example, an interaction may be defined only for premium customers, where the knowledge whether or not a customer is a premium customer can be defined in a domain ontology and resolved by a reasoner.

2.2 Describing Interfaces and Information Objects

For describing user interfaces, we have developed the UI² (user interfaces and interactions) ontology, which defines around 80 different categories of user interface components, and more than 60 categories for describing interactions. This ontology is one of the largest and most concise ontologies of the domain, and it covers a large design space of interactive systems, including tangible and speech interaction [5]:

Information objects, such as customers and transactions, are defined in a domain-specific real world ontology which is not part of our approach, since it is not generic, but depends on the domain in which our

2 The ontology is available from http://www.ke.tu-darmstadt.de/resources/ui2-ontology
Ontology-based Application Integration

Figure 1 Improving modularity with event annotations [5].

approach is applied, such as banking or travel. Each information object to be shared between applications is described using such an ontology.

Since existing approaches to using ontologies for sharing objects explicitly or implicitly assume a certain degree of similarity between the applications’ class models and the ontology used, they are rather limited when it comes to integrating applications with a large degree of conceptual heterogeneity, i.e., strongly diverging data models. In our approach, we propose a rule-based mechanism for exchanging objects between applications, which is more flexible than the currently predominant annotation-based approaches [7]. The integration framework makes the conversion of objects between different data models completely transparent, allowing each application to work only with its own data model.

3 Framework

The overall integration framework provides an event bus, to which all components of the framework connect. The most important component is an application container, which encapsulates an application and handles all the communication for that application. Containers also provide special functionality, such as coordinating drag and drop interactions across applications.

All events are processed centrally by a component which does not only hold all the ontologies for describing applications and information objects, but also all the integration rules. It connects to the applications to get data and status information at run-time (see below), computes reactions to events, and notifies the applications which are expected to react to an event produced in another application. Since no application reacts to an event from another application directly, the coupling between the applications is very loose.

3.1 Performance

When dealing with interactive applications, the reaction times of the integrated application become essential. On the other hand, running a general-purpose reasoner for dispatching events can soon become a bottleneck for performance.

A particular problem can occur with interactive systems that often change their state. A straightforward approach would always inform the reasoner on (potentially irrelevant) status updates. In [4], we have shown that such straightforward approaches do not scale to the requirements of interactive systems. We have proposed a model that only creates the relevant instance information from the integrated applications at reasoning time, thus allowing high performance event processing. With a prototype implementation using OntoBroker3 as a reasoning engine, we have shown that the event processing time is below 500 ms even under extreme conditions.

3.2 Seamless Interaction

Integrating applications on the user interface level poses particular challenges when supporting seamless interac-

---

tions – such as drag and drop between applications – and applications that are technologically heterogeneous. For example, implementing drag and drop between Java and non-Java applications without any special support leads to strong coupling of the respective application.

Our container-based approach, on the other hand, allows for providing containers for different technologies, and thus encapsulating heterogeneous applications while facilitating seamless interaction. In [6], we have shown that even interaction mechanisms such as drag and drop can be implemented between heterogeneous applications, such as a Java and a Flex application, without giving up the loose coupling paradigm.

4 Conclusion and Outlook

This article has given a short glance at the work on ontology-based application integration at the user interface level laid out in [5]. We have discussed why using ontologies is beneficial, and how a general framework can be designed. The book covers further aspects, such as ontology-based visualization in integrated applications, and tool support for performing the integration.

Our work done so far concentrates on classic single-user WIMP (windows, icons, mouse, pointing) user interfaces, although the framework and ontology is designed in a general way to cover multi-user interaction as well as other forms of user interfaces. Examining the capabilities to extend the framework for working with multiple users on multiple devices would be an interesting aspect of future work. Another interesting direction of future research is an integration at run-time, were users can use ad hoc ontology-based descriptions of applications they wish to integrate.

In summary, our work has shown that the use of ontologies as formal models can improve both the development and maintenance process of integrated applications, as well as their usability.

References


Received: March 8, 2013

Dr. Heiko Paulheim graduated with a PhD from the Technical University of Darmstadt in 2011. Before completing his PhD, he worked on projects dealing with semantic integration at University of Applied Science in Darmstadt, and at SAP Research. Since 2011, he has been a postdoctoral researcher at the universities of Darmstadt and Mannheim. His current research focus is on applying methods from data mining and machine learning to Linked Open Data.

Address: Universität Mannheim, Institut für Informatik und Wirtschaftsinformatik B6, 26, D-68159 Mannheim, e-mail: heiko@informatik.uni-mannheim.de