Heiko Paulheim

Ontology-based Application Integration

Foreword by Johannes Fürnkranz

Springer

Ontology-based Application Integration

Heiko Paulheim

Ontology-based Application Integration

Foreword by Johannes Fürnkranz

Springer

Heiko Paulheim
Knowledge Engineering Group
Technische Universität Darmstadt
Germany
paulheim@ke.tu-darmstadt.de

ISBN 978-1-4614-1429-2 e-ISBN 978-1-4614-1430-8
DOI 10.1007/978-1-4614-1430-8
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011936126

This book is an extended version of the dissertation “Ontology-based Application Integration on the User Interface Level” at Technische Universität Darmstadt, D17.

Foreword

There is probably no invention in the history of mankind that had such a profound impact on our lives in such a short time as the World Wide Web. Twenty years ago, Tim Berners-Lee has developed the first versions of HTML which allowed to weave documents into the large hypertext document that we know today. It was soon realized that the potential of this technology is not limited to connecting texts, but may serve as a backbone for a world-wide knowledge base called the *Semantic Web*. Again, Tim Berners-Lee helped to pioneer the vision of data and knowledge being publicly available in a formalized, machine-processable form. Based on standards like RDF or OWL, knowledge and semantics may be freely exchanged between heterogeneous applications. The number of facts stored in public knowledge repositories, so-called *ontologies*, is increasing at a rapid scale. Linked open data are on the verge of permeating our everyday lives.

Now we are facing the next revolution. Not only documents or knowledge will be connected, but computer applications are no longer running on personal computers, but on centralized servers which can be accessed via Web interfaces from a large variety of processors in smartphones, TVs, cars, household appliances, and more. For the end user, this not only relieves them of the burden of the update and maintenance of their software, but allows them to access their applications in a uniform way, everywhere and at every time.

A grand challenge for web-based software design is to integrate different heterogeneous applications into a homogeneous new system that utilizes the familiar existing components but allows a transparent data exchange between these components. Such *Mash-Ups* can be realized at the code level, by reprogramming functions of the individual applications, or at the data or business logic level by formalizing the service description and access of the applications, e.g. in the form of Web Services. Both ways have the disadvantage that aspects of the application have to be reprogrammed in order to allow a standardized data exchange.

This book shows how ontologies and semantic web technologies can be employed to solve the practical problem of integrating applications on the user interface level. It shows how the relevant concepts of user interfaces, such as components and interactions, can be captured in a highly formalized ontology, and puts a strong emphasis on practical aspects of the implementation of an integration framework based on that ontology, such as the scalability of semantic event processing approaches, or the support of seamless cross-technological interactions. Never losing the focus on the end user, it further explores the possibilities ontologies provide to enhance the usability of integrated applications.

The book describes this innovative approach in all aspects. It provides an excellent introduction into ontologies and their applications in user-interface and application design, so that the book can be read without extensive prior knowledge in these areas. All presented concepts and techniques are illustrated with a case study that demonstrates the design of an integrated application for the management of catastrophes, which has been developed in a research project with different partners from the industry and academia, led by SAP Research. The book is thus of interest to both, researchers in ontologies who are looking for an interesting application, and for practitioners who want to find out techniques for combining different applications in a non-intrusive knowledge-based way. I am confident that it will become a key publication in its area.

Johannes Fürnkranz
Darmstadt, July 2011

Acknowledgements

While working on the topic of ontology-based application integration on the user interface level throughout the last three years, I have had valuable support and input from many people, without whom this book would not have become what it is.

Johannes Fürnkranz has encouraged me to pursue the topic of ontology-based application integration, agreed to take over the supervision of my dissertation, and has been continuously supporting me with valuable advice with respect to my research and numerous other topics. Jürgen Ziegler has agreed on acting as my secondary supervisor and has given me very valuable feedback in various discussions.

The topic covered by this book has been evolving out of and largely pursued within the research project *SoKNOS* at SAP Research in Darmstadt. First and foremost, Florian Probst has largely helped me in shaping the topic and taught me how to be formally precise when crafting ontologies in many fruitful (and sometimes exhausting) discussions. Many people both in Darmstadt and in other institutions involved in the SoKNOS project have been creating the atmosphere in which this research work could grow, especially Sebastian Döweling, Karen Tso-Sutter, Anna Lewandowski, and Thomas Ziegert, and everybody else who contributed to the SoKNOS project, in particular Daniel Oberle, Grigori Babitski, and Jörg Hoffmann at SAP Research Karlsruhe, Simon Bergweiler at DFKI, Alexander Walkowski, Christoph Stasch, and Florian Daiber at University of Münster, Stephan Braune and Martin Thoma at B2M, Marcus Hoffmann and Thorsten May at Fraunhofer IGD, and Alexander Behring and Andreas Petter at TU Darmstadt.

After the SoKNOS project was finished, I have continued pursuing my research in the *AdiWa* project and was happy to once more find a stimulating environment with great researchers, especially Benedikt Schmidt, Birgit Zimmermann, Christian Kuhn, Eicke Godehart, and Todor Stoitsev. During my whole time at SAP Research, Knut Manske, Nicole Scholl, and Anna Wypior have been supporting me with all the administrative work, Bettina Laugwitz has taught me interesting details about statistical evaluation that I had long forgotten about, and Andreas Faatz has been giving me valuable advice on very different issues far more than once.

During the work on this book, it was my great pleasure to supervise a lot of very talented students. Rene Abraham helped compiling an initial survey of user interface description languages. Atila Erdogan worked on the implementation of the Flex container for Java user interfaces, and Lars Meyer evaluated different implementation alternatives for improving the performance of the integration framework, as well as he worked on the implementation and evaluation of the Semantic Data Explorer. Roland Plendl contributed to the implementation of the rule-based object exchange mechanism, and Tobias Wieschnowsky built a prototype of a graphical user interface integration tool.

For the prototype implementation of the approach discussed in this book, I have used the system *OntoBroker*, and some people supported me in getting non-standard features implemented and told me about secret configuration options which are not mentioned in the official manuals. At OntoPrise, Saartje Brockmanns and Roman Korf have been patiently answering my questions and revealed various hidden functionalities, and Michael Erdmann has pointed me to using Skolem terms for event processing rules.

It was already two years before I started to dive into the topic of ontology-based application integration that I began working with ontologies. Michael Rebstock and Janina Fengel have sparked my interest in that topic and worked with me for two years. Since then, I have been discussing research ideas with countless colleagues at workshops, conferences, and other occasions, and always gained a lot from those discussions.

Last, but not least, my wife Carolin and my family have been continuously supporting me during the last years. Thank you for everything.

Heiko Paulheim
Darmstadt, July 2011

Contents

Chapter 1
Introduction

Abstract While software engineering traditionally has been concerned mostly with the development of individual components, there has been a paradigm shift during the past decades. Assembling applications from existing components and integrating applications to complex systems has become more important with a growing number of existing artifacts. In the future, application integration on the user interface level will drastically reduce development efforts and create customizable, seamlessly integrated systems. However, currently existing approaches and frameworks are still not capable of fully harvesting the benefits of this new style of system development.

1.1 Vision

During the past decades, software engineering has changed. The number of existing applications and software artifacts, such as modules, libraries, components, and services, has grown rapidly. Software systems have become more and more complex. At the same time, the pressure to produce those complex software systems in ever shorter periods of time has grown.

These shifts in the basic parameters has influences on the software engineering process itself. An analysis presented by Blechar (2010, p. 25) identify a

> "trend away from net new *development* of business software systems to *composition* of business software systems."

With the advent of *web mashups* in the course of the *Web 2.0* , the vision of rapid integration of applications, including their user interfaces, has rapidly gained momentum. In 2007, Gartner analysts stated that

> "By 2010, Web mashups will be the dominant model (80 percent) for the creation of composite enterprise applications. Mashup technologies will evolve significantly over the next five years, and application leaders must take this evolution into account when evaluating the impact of mashups and in formulating an enterprise mashup strategy." (Gartner, 2007)

One year later, they once again strengthened the business need for such mashup technologies:

> "Enterprises are now investigating taking mashups from cool Web hobby to enterprise-class systems to augment their models for delivering and managing applications. Through 2010, the enterprise mashup product environment will experience significant flux and consolidation, and application architects and IT leaders should investigate this growing space for the significant and transformational potential it may offer their enterprises." (Gartner, 2008)

In the same year, Forrester analysts predicted

> "that the enterprise mashup market will reach nearly $700 million by 2013; while this means that there is plenty of money to be made selling mashup platforms, it will affect nearly every software vendor. Mashup platforms are in the pole position and ready to grab the lion's share of the market – and an entire ecosystem of mashup technology and data providers is emerging to complement those platforms. Those vendor strategists that move quickly, plan a mashup strategy, and build a partner ecosystem will come out on top." (Young et al, 2008)

In 2010, Gartner analysts stated that

> "composite applications enable increased operational and decision-making efficiency by supporting a single integrated view of a critical business entity – e.g., customer, supplier, product, patient and taxpayer – whose data are scattered across multiple databases and applications" (Blechar et al, 2010, p. 25)

and predicted that

> "most organizations will benefit from using new development methodologies and tools focused on providing sustainable agile AD [Application Development], including support for the creation of composite applications and enterprise mashups." (Blechar et al, 2010, p. 25)

In the same report, the analysts also found that

> "composite applications increase the complexity of the computing environment and result in dependencies that demand cautious management." (Blechar et al, 2010, p. 25)

These quotations point to a need for appropriate, professional tools and mechanisms for integration user interfaces. In the future, a typical software engineer will spend less time on coding and more time on choosing components from an inventory and assembling them to a new product. Existing software applications will be reused and recombined to new products, avoiding the reinvention of the wheel and allowing faster and cheaper software production.

Even nowadays, applications are rarely be built from scratch, starting with an empty sheet of paper and ending up with a one-size-fits-all system. Instead, developers choose from a shelf of ready-to-use components and compose them to complex systems. While this composition is currently a manual task, in the future, there will be intelligent tools that require only little manual efforts, and deliver customized integrated systems to the end users. For the end user, however, these applications will not look like rag rugs, but rather like one-of-a-piece products, allowing interactions that do not show the seams where the original components were stitched together.

1.2 Challenges

The current state of the art still faces some major challenges for harvesting the promised benefits created by the mashup style of developing applications (Ogrinz, 2009, pp. 10 and pp. 311). Today, creating a mashup from different applications involves a lot of manual work and requires deep knowledge about the applications integrated. Badly documented interfaces, heterogeneous data formats and technologies, and the lack of adequate and intelligent tool support make mashup development complicated and difficult, thus, the benefits of mashup development are to large extent eaten up by its costs.

Software components, however, typically do not come with matching screws and bolts that make a seamless assembly an easy task. While numerous solutions have been developed which integrate software components using technologies such as web services, those solutions most often target at system integration at a lower level, i.e., the data or business logic level. Integrating existing user interface components is not possible with such technologies.

So far, existing tools for user interface integration are still very limited. In order to result in a maintainable and customizable software system, it is essential that the components forming an integrated system are only *loosely coupled*. Tight coupling introduces dependencies between the integrated components and thus leads to a system which is hard to maintain and hinders fast adaptations to the resulting software. On the other hand, the end user will want to experience a software product as all of a piece and not find any flaws when interacting with the integrated system, i.e., a *seamlessly integrated* application. As discussed by Schefström (1999, p. 24), these two requirements conflict with each other, leading to the so-called *integration dilemma*: strong cohesion and loose coupling are hard to achieve at the same time. This conflict is particularly strong on the user interface level when implementing seamless interactions.

Especially when it comes to the integration of user interface components, current approaches still lack mechanisms for seamlessly integrating components and allowing cross-component interactions. The problem gets even more complicated when dealing with technologically heterogeneous user interface components, such as Flex, Java, and Silverlight components, developed with different programming languages.

When application integration is performed on the user interface layer, it typically involves the acquisition of knowledge about the applications' internal functionality as well as many hacks and workarounds. When integrating heterogeneous user interface components, code is written in different programming languages, using different paradigms and mechanisms for event processing, data conversion, etc. Translating between those mechanisms requires code that is most often scattered across the integrated system, which leads to code tangling and a monolithic architecture that is hard to maintain.

1.3 Approach

For showing that formal ontologies improve the integration of applications on the user interface level, the book demonstrates the development of a formal ontology of the domain of user interfaces and interactions, following established ontology engineering methodologies and building on the foundations of upper ontologies. A prototype implementation will prove the feasibility of applying ontologies and rules to the given integration problem.

That prototype implementation will be the basis for different experimental evaluations. For showing that an implementation with reasonable performance is possible, different architectural variants are implemented, and their runtime behavior is measured. The possibility of integrating applications based on heterogeneous technologies is shown in a proof-of-concept prototype. Furthermore, the prototype implementation is used in a running case study showing the integration of a larger-scale emergency management system.

In addition to the the integration framework itself, the benefit of ontologies in integrated user interfaces is shown with a tool for exploring information contained in integrated applications, which is evaluated in a quantitative user study.

1.4 Contributions

In this book, we will introduce an approach for application integration on the user interface level which uses ontologies for formally describing user interface components and the data they process. A middleware based on semantic technologies is employed to fully decouple the individual components and still facilitate seamless integration of heterogeneous user interface components. This approach will show that the idea of ontology-based integration, which has been applied to database integration and business logic integration so far, carries over to the user interface level as well.

This book will provide several contributions to the research community. One central artifact is a detailed formal ontology of the domain of user interfaces and interactions, which allows for describing user interfaces based on strict formal foundations. This ontology may not be used in a middleware for application integration on the user interface level, but also for other purposes requiring a formal model of a user interface, such as providing assistance to the end user, or facilitating user interface adaptation, etc.

Data exchange between applications is a major cornerstone in system integration. This book discusses an approach for dynamically annotating data objects for facilitating exchange at run-time, which is more flexible concerning heterogeneous data models than today's state of the art approaches are.

The works on efficient, high-performance event exchange in user interfaces provide a discussion and thorough evaluation of several architectural alternatives for facilitating reasoning on data from running software systems. As performance today is often a major obstacle for employing ontologies and semantic technologies in real world scenarios, the results presented on that topic will help building scalable and usable ontology-based systems of different kinds.

Finally, a user study on ontology-based information exploration will show how ontologies and semantically annotated information may be used for assisting users with complex knowledge gathering tasks, even if those users do not have any prior knowledge on ontologies and semantic technologies. While ontology-based systems often do not find their path out of the research labs, this evaluation points out a way of carrying the benefits of ontology-based systems from specialized researchers to end users.

1.5 Outline of the Book

This book is divided in three parts. Part I discusses the background and state of the art in application integration on the user interface layer in Chap. 2. Chapter 3 introduces ontologies and discusses their role in application integration. Chapter 4 shows the state of the art of employing ontologies in user interface development.

Part II reports on an in-depth study of employing ontologies for application integration on the user interface layer. Chap. 5 introduces a general framework for application integration on the user interface level. This chapter serves as a basis for discussing various aspects of application integration on the user interface level in more detail in four chapters, which are widely independent from each other:

- Chapter 6 shows the development of an ontology used for formalizing user interface components and for annotating the events passed between the different components.
- Chapter 7 discusses the annotation of class models for facilitating the exchange of data objects between user interface components, i.e., bridging *conceptual* heterogeneities.
- Chapter 8 is devoted to the impact of performance on user interface integration and evaluates different architectural variants with respect to performance.
- Chapter 9 specifically deals with the problem of *technologically* heterogeneous user interface components.

The book is accompanied by a running case study dealing with SoKNOS (Döweling et al, 2009; Paulheim et al, 2009), an integrated emergency management tool that has been developed in a consortium of different companies and research institutions, lead by SAP Research. In that project, the approach discussed in this book has been applied on a larger scale. For each of the Chaps. 5, 6, 7, 8, and 9, the implementation of the approach in SoKNOS is discussed.

Part III gives an outlook on future developments in application integration on the user interface layer. Chapter 10 introduces a tool for information exploration built on top of the framework developed throughout the book. In a user study, it shows how information exploration can be significantly improved in an integrated user interface combined with a suitable visualization. Chapter 11 discusses how end users can be enabled to build custom tailored, ad hoc integrated user interfaces, and introduces the prototype of a tool which hides the complexity of formal ontologies and rules under the mask of an easy-to-use interface.

The book closes with a summary and an outlook on open and related research questions in Chap. 12.

Part I
System Integration, Ontologies, and User Interfaces

Chapter 2
Application Integration on the User Interface Level

Abstract This chapter starts with a definition of user interface integration and discusses how user interface integration differs from other application integration approaches in Sect. 2.1. Current state of the arts approaches are discussed, such as portals in Sect. 2.2, mashups in Sect. 2.3, as well as other popular solutions and academic prototypes in Sects. 2.4 and 2.5. The review of the state of the art discusses the recent advances in the field and points out some currently existing drawbacks in Sect. 2.6, which will be addressed in this book.

2.1 Fundamentals

Application integration on the user interface level is one technique of integrating software applications. Other terms encountered in the literature are, e.g. *front-end composition*, *integration on the glass* (Blechar, 2010, p. 50), and *integration on the presentation level* or *presentation integration* (Yu et al, 2007, p. 923). However, *user interface integration* is the most frequently used term[1], which we will therefore use throughout this book.

2.1.1 Levels of Application Integration

Application integration can be performed on different levels, following the widely accepted three layer model introduced by Fowler (2003, pp. 19). Figure 2.1 gives an overview on different views on application integration from the literature.

Daniel et al (2007, p. 60) follow the notion of Fowler (2003, pp. 19) that software usually comes in three layers, the data source layer, the business logic layer

[1] Google lists about 410,000 hits for "user interface integration" and "UI integration", compared to 47,000 for "presentation integration", 42,000 for "integration on the glass", and 8,600 for "front-end composition".

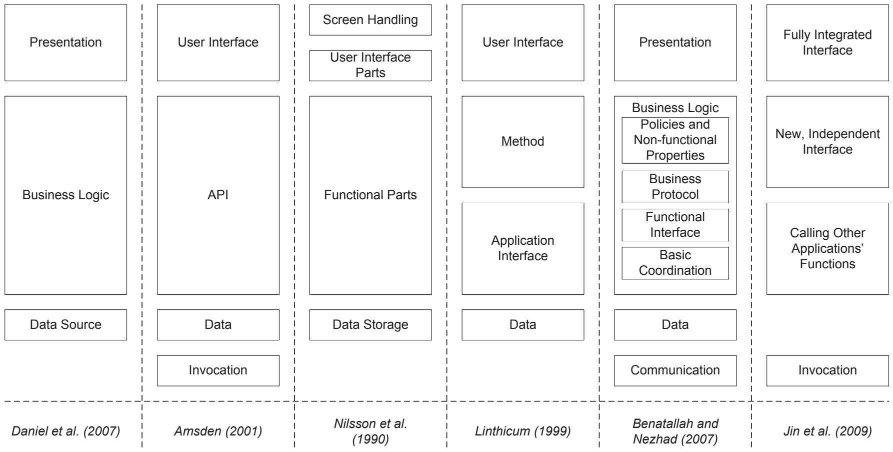

Fig. 2.1: Different levels of integration: an overview of classifications from the literature.

(originally called *domain layer* by Fowler, but *business logic layer* has become more widespread[2]), and the *presentation layer*. Consequently, the authors derive that there are three layers on which applications can be integrated, which leads to the simplest model of system integration: an *integration layer* can be placed on top of each of the layers, thus facilitating application integration on the data source layer, on the business logic layer, and on the presentation layer.

Amsden (2001) introduces another variation of integration: one application may *invoke* another one, i.e., start it via access to the underlying operating system. It is arguable whether this is really a way of *integration*, since the two applications only run side by side without any interaction after the invocation.

Nilsson et al (1990, p. 442) introduce a separation of integration on the user interface layer: they distinguish the integration of *user interface parts* from the integration on the *screen handling* layer. With architectures such as *X Window* (Scheifler and Gettys, 1986, pp. 79), the implementation of the UI components (called *user interface part* by the authors) is separated from the implementation of the display of and interaction with those components (which is what the authors call *screen handling*). Thus, the authors propose two different strategies of integration: on the UI components and on the screen handling layer.

Linthicum (1999, p. 18) discusses several ways of *enterprise application integration*, i.e., the integration of applications of different enterprises. He distinguishes two types of integration on the business logic level: *application interface integration*, and *method integration*. Application interface integration means that one application calls methods from another one. In contrast, method integration includes that the underlying process models are exchanged, and more complex patterns of interaction

[2] About 950,000 Google hits for "business logic layer", compared to 80,000 hits for "domain layer".

between applications, going beyond singular method calls, are supported, such as contract negotiations between companies (Paulheim et al, 2011a).

Benatallah and Nezhad (2007, p. 119) provide an even finer-grained distinction of integration on the business logic layer. Besides Linthicum's distinction of application interface integration (called *functional interface* by the authors) and method integration (called *business protocol* by the authors), they introduce the need for additionally coordinating the message exchange itself (called *basic coordination*) as well as policies such as privacy policies and quality of service agreements between systems. Furthermore, the authors introduce the communication layer as another layer of integration, thereby stressing that when integration distributed applications, the communication protocol heterogeneities must be overcome (Rebstock et al, 2008, pp. 28).

For the rest of this book, we will refer to *user interface integration* as *integration on the user interface* or *integration on the presentation layer*. To provide meaningful, useful integration, the integration layer has to be aware of the data processed by and the operations that can be performed with different user interface components. Thus, regarding the distinction introduced by Nilsson et al (1990), the focus will be on integration on the component layer, however, the screen handling layer will be regarded when performing cross-technological UI integration, as discussed in Chap. 9.

2.1.2 Definition of User Interface Integration

Application integration on the user interface layer, or UI integration for short, is the technique of assembling

> "applications by reusing their own user interfaces. This means that the presentation layer of the composite application is itself composed, at least in part, by the presentation layers of the components" (Daniel et al, 2007, p. 61).

Composing the user interface out of reused user interface components, however, is only the first step. An integrated user interface does not only display different user interface components next to each other, it also has to allow interactions between those components. The integrated user interface has to provide more value than the sum of the integrated applications (Westermann and Jain, 2007, p. 20) – otherwise, any effort of integration would not be justified, as the user could just run the individual applications in parallel instead.

Therefore, UI integration also includes *coordination*, *synchronization*, and *interaction* between the integrated applications. Actions performed with or state changes occurring in one of the integrated applications can cause reactions in other applications. If this sort of integration is pushed so far that the user can experience the integrated application in as being one of a piece, we use the term *seamless integration* (Paulheim and Erdogan, 2010, p. 303).

Jin et al (2009, p. 5) discuss four levels of UI integration: applications launching other applications, applications calling other applications' functions, several applications sharing one (new) user interface, and fully integrated user interfaces. In the terminology used in this book, only the latter would be regarded as UI integration; the others are merely integration on the invocation level (in Amsden's terminology), and the business logic level.

In the context of enterprise application integration (EAI), the term *user interface-level integration* is used with a different meaning: for systems that have no publicly available API and do not provide direct access to the business logic and data storage, methods such as screen scraping and input emulation on the systems' user interfaces are used to get the data out of those systems (Linthicum, 1999, pp. 79). In contrast, user interface integration, as used in this book, is about facilitating interactions between user interfaces, not about using user interfaces as an entry point for getting data out of IT systems.

Another notion of *user interface integration* has been introduced by Schefström (1999, p. 18): the author does not focus on the composition of different user interfaces, but on the harmonization of the layout and look and feel of integrated applications.

2.1.3 Benefits of Application Integration on the User Interface Layer

There are two main benefits for performing application integration on the user interface level: increasing the usability of software systems, and reducing development efforts for those software systems.

From an end user's point of view, any system that is integrated on a deeper level than the user interface will come with an individually developed user interface (Daniel et al, 2007, pp. 60). Thus, the user will be confronted with a new, unfamiliar user interface and thus have to learn how to work with the system. An integrated system with applications retaining the familiar user interfaces, on the other hand, will result in a steeper learning curve, as the user can continue working with familiar interfaces.

From a software engineer's point of view, reusing existing user interface components, as opposed to developing a new user interface from scratch, means saving time. The user interface is the most expensive part of a software system, the portion devoted to the user interface ranges from 50% (Myers and Rosson, 1992, p. 199) to 70% (Sergevich and Viktorovna, 2003, p. 89) of the total development effort. More sharply phrased: without an approach for integration on the user interface level, the degree of reuse will never exceed 50%. UI integration can therefore reduce development efforts of integrated software systems drastically.