End-User Development of Enterprise Widgets

Michael Spahn[1] and Volker Wulf[2]

[1] SAP AG, SAP Research, Bleichstr. 8, 64283 Darmstadt, Germany
Michael.Spahn@sap.com
[2] University of Siegen, Hölderlinstr. 3, 57076 Siegen, Germany
Volker.Wulf@uni-siegen.de

Abstract. Companies are operating in a dynamic environment, resulting in a continuous need of adapting used information systems to changing business processes and associated information needs. Viewed from a micro-perspective, business users are managing and executing business processes on a daily basis, but are not able to adapt used software to their individual needs and working practice. In this paper, we present an End-User Development (EUD) approach and prototypic environment, enabling business users to create enterprise widgets tailored to their personal information needs without the need of programming knowledge, by mashing up enterprise resources using a lightweight visual design paradigm. The approach especially considers extensibility of building blocks for widget creation even by small and medium sized enterprises (SMEs) using existing knowledge. We give evidence on the applicability of the approach in real enterprise contexts, by providing first results of an evaluation in three German SMEs.

Keywords: End-User Development, Mashup, Widget.

1 Introduction

In our today's business world, companies use enterprise software systems like Enterprise Resource Planning (ERP) systems to support and facilitate their business processes. As companies are not static and evolve like the environment of competitors, markets and customers surrounding them, a continuous need of adapting these systems to new requirements, business processes and associated information needs, exists. Due to a lack of resources and expertise, especially small and medium sized enterprises (SMEs) suffer from their inability to adapt used enterprise software to their needs [1]. They are forced to adapt themselves to the possibilities offered by the used enterprise software or to delegate adaptations to IT professionals, resulting in long and costly adaptation processes [2, 3]. As organizational and technological development are closely correlated, the inability of adapting the technical systems used, is limiting the organizational development possibilities of the companies as well [4]. As a consequence, the ability of organizations to optimize their business processes and to innovate to gain unique competitive advantages is limited.

Viewed from a micro-perspective, business users as the end-users of enterprise systems know best about changing requirements and needed adaptations, as they are managing and executing business processes on a daily basis. As end-users are domain experts but not necessarily IT professionals, they are not able to adapt the used enterprise systems to their individual needs on their own. End-users are forced to indirectly influence the adaptation processes by communicating their needs to IT professionals. IT professionals are confronted with the requirements of users expressed in their domain language, which have to be interpreted and transformed into models and technical solutions matching the capabilities of the enterprise software systems. This process is not only costly and time-consuming, but also error prone due to a possible misinterpretation of requirements [5]. Furthermore many useful adaptations to the individual working practice of end-users are dropped due to limited budget, resources and expertise. One approach to improve this situation is to better enable end-users to adapt the used enterprise systems on their own. At this, the inherent challenge is to reduce the expertise tension, existing in a two-dimensional continuum of job-related domain knowledge and system related development knowledge [6].

We are approaching this challenge from an End-User Development (EUD) perspective. EUD can be defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create or modify a software artifact" [7]. Crucial preconditions of successfully enabling EUD are on one hand systems, which are flexible enough to be adaptable in a technical sense, and on the other hand methods to leverage this flexibility at the hand of end-users [8, 9]. In the last years the dissemination of the Service Oriented Architecture (SOA) [10] paradigm led to an increased technical flexibility. SOAs, i.e. implemented using the widespread Web Services Stack, provide means to increase flexibility by recombining existing functionality of software systems using loose coupling of services, enabling the orchestration of new software solutions from existing functionality. SOAs offer rich possibilities for IT professionals, but as they are based on complex standards like SOAP, WSDL, UDDI, or BPEL, the technical flexibility is not leveraged at the hand of end-users. In this context a new type of web-based applications, known as *mashups*, has been gaining momentum. Novel lightweight design principles are currently about to emerge, allowing to mash up data from different resources into a single integrated tool and thereby creating a new and distinct service, that was not originally provided by either resource used. Popular examples for consumer mashup environments are e.g. Microsoft Popfly [11] or Yahoo! Pipes [12]. Gartner even identified mashups and composite applications as one of the top ten strategic technologies for 2008 [13].

We conducted preliminary studies in SMEs and identified several kinds of data-centric adaptation problems that end-users face in their work context [14]. We believe, that some of these adaptation problems can be adequately addressed by providing EUD tools based on a mashup design paradigm. Business users should be enabled to create data-centric tools that are tailored to their individual information needs, by mashing up enterprise resources. To further investigate that solution approach, we set up a prototypic EUD design environment for the creation of *widgets*, which can be mashed up of enterprise resources in a very lightweight way. Widgets are small, interactive applications for displaying data, packaged in a way to be executable on a users' machine [15]. By using a very lightweight mashup design paradigm and encapsulating mashups as widgets, we enable end-users to develop small, interactive applications using enterprise resources and to deploy these applications directly to their machine, without the need of any programming knowledge. Our approach especially considers extensibility of building blocks for widget creation even by SMEs using existing knowledge. Thereby the whole chain of developing building blocks and composing building blocks to widgets can be put in the hands of SMEs. To be able to evaluate the usefulness and practicability of our solution approach in real work environments, we deployed our solution to three German SMEs. We investigated if business users are able to create widgets using the lightweight design approach, and if practical problems can be addressed using such simple applications like widgets. Furthermore we classified types of end-users and analyzed how they collaborate to create solutions of practical value.

The remainder of the paper is organized as follows. In Section 2 we present the conducted preliminary empirical studies and explain our research and solution approach. In Section 3 we describe the developed EUD environment for widget creation with regard to conceptual layers, architectural components, user interface and design paradigm. First results of the evaluation phase are discussed in Section 4, before we summarize and conclude our paper in Section 5.

2 Research Approach and Preliminary Studies

An important aim of our research is to create EUD approaches and tools, which are applicable in real enterprise environments. This intent is considered by the setup of our research approach. The work presented in this paper is based on a research approach consisting of three steps, which can be characterized as follows: (i) Conducting empirical research to identify end-user adaptation problems relevant in real enterprise environments. (ii) Developing a solution approach to address identified adaptation problems. (iii) Evaluate the solution approach in real work contexts to determine its applicability for the intended purpose. In the following subsections we discuss each of the steps briefly.

2.1 Preliminary Empirical Studies

In preparation of designing EUD tools, we conducted a series of semi-structured interviews based on qualitative research methods [16] in three German midsized companies that use ERP systems to support their business processes. The addressed companies were two companies from production industry (137 and 140 employees) and one larger software vendor (500 employees). Semi-structured face-to-face interviews were conducted in an exploratory way to get a deep insight into operational tasks and work practices. The interviews focused on data-centric aspects of how end-users access information stored in the ERP system, how they flexibly process the data, and what problems they face, especially with regard to EUD related activities. With regard to EUD related activities our studies revealed, that it is common for business users to create individual information artifacts supporting them in their operative tasks. The most important information artifacts that end-users create are spreadsheets, tailored to the end-users' individual information needs. End-users commonly use predefined queries of the ERP system to import enterprise data into spreadsheets. Due to the complexity of the data model exposed by the ERP system and the complexity of the tools for query creation provided by the ERP system, most end-users are not able to create custom queries on their own to import data from the ERP system as desired.

Additionally, end-users are not able to create any kind of custom information artifact providing interaction on live data of the ERP system. Many end-users have to access a certain set of data relevant for their individual working tasks from the ERP system many times a day. In many cases, this data can not be accessed from a single location within the graphical user interface (GUI) of the ERP systems, forcing users to access multiple locations and cumbersomely collect the needed data. As they are not able to create any kind of customized GUI or interactive application, providing access to relevant data at a glance, many working tasks can only be executed in an inefficient and cumbersome manner.

Complementary to the interview studies, a participatory design workshop (PDW) was held to investigate how typical business users manage to design a software artifact using a very lightweight box and wires design paradigm. A PDW puts users in the role of designers and requires them to collaborate actively in a solution-oriented design process [17]. In the PDW business users had to design an information artifact representing a tool that should support users in an analytic task, which was taken from their work context. To create the solution, the users could add boxes to the design space that represent data or functionality. Boxes had output ports and input ports and could be connected by drawing lines to define data or control flow between the boxes. The boxes and wires model was left underspecified in a way that no concrete instructions were given to the users on how to formally specify the meaning of the used design elements. The underspecified semantics enabled to observe how users intuitively use design elements. As the participatory design workshop revealed, end-users intuitively thought of boxes as a representation of tabular data, organizing data in rows and columns. By connecting boxes, business users related data from different boxes with each other and defined the data flow of the solution. The users had no problems using the boxes and wires design paradigm itself to specify a solution, but had problems to express what data a box should represent. The users decided to specify the data by giving a short description on how they would access the data in their used ERP system. More details on the results of our preliminary empirical studies are given in [14].

2.2 Creating a Solution Approach

In our preliminary studies we identified two main problems that end-users face with regard to data-centric EUD activities. First, end-users create custom spreadsheets and rely on getting relevant business data from the ERP system by using queries, but face considerable challenges when trying to create custom queries for their individual information needs. Second, end-users need to access the same set of data within the GUI of the ERP system over and over again in a cumbersome manner, and are not able to create custom interactive applications that provide the information relevant for individual working tasks at a glance. We developed prototypic EUD environments addressing both problems identified. With regard to EUD of queries for complex enterprise information systems, we developed a prototype called "Semantic Query Designer" (SQD), which is build on an ontology-based middleware and provides sophisticated visualization, navigation, search and query building possibilities. As we are focusing on the second identified problem in this paper, we refer to [18], providing a detailed description of SQD.

With regard to EUD of custom, interactive applications providing access to data relevant for individual working tasks, we set up the prototypic EUD environment

"Widget Composition Platform" (WCP). The WCP enables business users to create custom widgets without any programming knowledge by mashing up enterprise resources in a visual design environment and deploying the created widgets to their machines. The WCP uses a very lightweight mashup design paradigm based on a simple box and wires framework. Boxes represent services that provide enterprise data that is in most cases rendered as a table. As our preliminary studies revealed, business users are able to learn simple design paradigms like boxes and wires in quite a short time and are able to use services providing data in a tabular format in an intuitive way for designing software artifacts.

For the realization of a visual mashup design environment, we could build on an early internal prototype of SAP Research that we modified and extended to suit our needs. Although we could build on a certain framework, we faced considerable challenges with regard to realizing a solution that could be deployed and used within SMEs for two main reasons. First, SMEs often use ERP systems that are not (yet) service-enabled and thus cannot provide enterprise resources as services that are accessible using standard web protocols, which is a an essential precondition for deploying developed mashups as a widget on common widget runtimes. Second, even if enterprise systems were exposing a fixed set of resources as services, this set could not be extended without programming skills and thus would limit the creatable solutions to combinations of predefined services. To address these challenges, we implemented a middleware which is able to wrap resources from not service-enabled ERP systems and provide these as services in a way accessible using standard web protocols. To enable SMEs to extend the provided services without any programming skills, new services must be creatable by SMEs using existing knowledge. As at least some advanced end-users in SMEs exist, that are able to create queries within the ERP system, we enabled the implemented middleware to wrap queries stored within the ERP system and expose these as services. Using the existing skills of query creation to enable the creation of new building blocks for widget creation, limits the entry barriers to flexibly use the new technology and can be seen as a kind of gentle slope of complexity approach [8, 19] that puts the whole chain of widget creation in the hand of SMEs. Additionally, by enabling the use of queries as services, a relation between the prototypes SQD and WCP is established, as end-users are able to use SQD for easy query creation, and WCP to mash up these queries to widgets. The conceptual layers, architectural components, and GUI of the WCP are described in detail in section 3.

2.3 Evaluating the Solution Approach

To be able to evaluate the practicability and usefulness of our solution approach in real work environments, we deployed the WCP environment to three German SMEs. In this evaluation phase we observed the usage, created widgets and services as well as the collaboration of different types of end-users. Additionally we conducted a questionnaire-based survey among the employees to refine the results of our preliminary studies and to get feedback on our WCP-based solution approach from a broad end-user base. In section 4 we provide first promising results of the evaluation phase by describing practical use cases of WCP-created widgets within SMEs, classify end-users by observed behavior related to widget creation, and discuss selected analysis results of the questionnaire-based study.

3 Widget Composition Platform

The WCP is a web-based EUD environment that enables business users to mashup enterprise resources in a visual design environment in a very lightweight way and to deploy the created mashups in the form of widgets to their local machines. In the following subsections we describe the WCP environment in detail. We describe the conceptual layers that are implemented by the WCP environment. We explain of which architectural components the WCP environment consists and how these components work together. Finally we describe the provided GUI and explain how end-users are able to create widgets by mashing up enterprise services.

3.1 Conceptual Layers

On a conceptual level, the WCP and created widgets are based on a layered architecture. Significant components of this conceptual architecture are classified and structured in the *widget stack* depicted in Fig. 1. The widget stack consists of five basic layers: resource layer, application programming interface (API) layer, wrapper layer, service layer, and mashup / widget layer. With regard to Fig. 1, we will describe the layers bottom-up, as each layer requires the functionality provided by its subjacent layer.

Resource Layer. The resource layer consists of all resources that can be integrated in mashups. Resources can be data, like customer master data stored in an ERP system, or functionality, like locating an address and returning an according image of a map provided by a map application. Resources are managed and provided by systems that can be internal or external to an organization. Internal systems might be various enterprise systems, like ERP or Customer Relationship Management (CRM) systems, custom applications tailored to the needs of the enterprise, or general purpose relational database management systems (RDBMS). External resources might be provided by systems accessible to a closed user group, like e.g. systems of suppliers, customers or B2B market places, or by systems publicly accessible over the internet, like e.g. map services or stock quotes.

API Layer. The API layer consists of the well-defined interfaces provided by the systems managing the resources. Depending on the systems, different formats and protocols may be needed to call the APIs and access the resources. APIs of modern service-enabled systems may be exposed as web services that can be called using standard web protocols, while legacy systems may expose APIs only as libraries that can be linked into source code and communicate with the system using proprietary protocols. Required formats of input parameters and formats of returned results vary accordingly and range from XML structures to proprietary binary formats.

Wrapper Layer. To abstract from the heterogeneity of APIs and respective protocols and formats, the wrapper layer provides a unified service model, consumable by the service layer. The abstraction is provided by wrapper components that transform requests to the abstract service model to concrete requests using the respective API, protocol and parameter format of the addressed resource. Raw results received from the API are transformed to a format that can be processed within the unified service

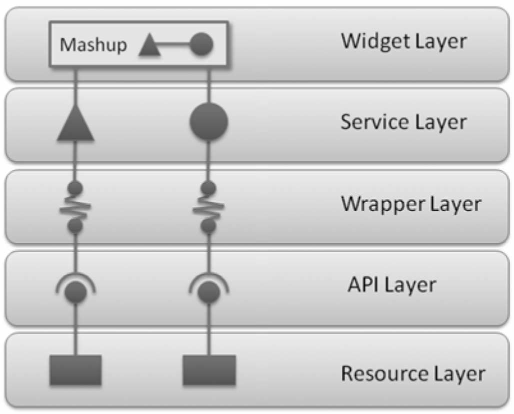


Fig. 1. Conceptual layers of widget stack in the WCP environment

model, e.g. data structures storing tables, lists or even images. Not each resource requires its own wrapper component. Generic wrappers are able to wrap certain types of resources, like SAP ERP queries or RSS feeds, in a generic way. These generic wrappers are exposed as service types in the service layer.

Service Layer. The service layer provides a repository of services that can be mashed up in the mashup environment. All services are parameterized instances of service types, relating services to a certain wrapper component in the wrapper layer. For example, a service "Customer data" providing data from a SAP ERP query is an instance of the service type "SAP ERP Query" that relates the service to an according wrapper component. The service instance uses parameters to specify which concrete query should be accessed inside the SAP ERP system via the wrapper component. Services provide further configuration possibilities that vary depending on the type of service. For example, the desired type of data visualization (e.g. rendering data as a table or a list), or the set of usable data filters that are exposed in the mashup environment, can be configured. Services can be further described by adding information like name and description text.

Mashup / Widget Layer. The mashup and widget layer is adding mashup functionality to the unified service model of the service layer and provides the functionality to deploy mashups as widgets. Mashups are defined by specifying a wiring of design elements, interconnecting design elements and defining the desired data flow. Design elements can be services or additional UI elements (like e.g. text boxes). The wiring defines how data from one design element is used to parameterize calls to other (dependent) design elements. Dependent design elements react on data updates of connected design elements and update their data accordingly. Mashups can be encapsulated as widgets and deployed as self-contained applications to individual runtimes, like e.g. the Yahoo! Widget engine [20]. The mashup creation and widget deployment functionality is provided by the mashup and widget layer to end-users by an integrated EUD environment.

To enable business users to create custom widgets supporting their individual working tasks, relevant business data needs to be available as services in the service layer. Therefore wrapper components and service types need to be implemented to be able to wrap resources from relevant enterprise systems. Any implementation of the widget stack should consider that at least some users within the organization are able to extend the services using existing knowledge. Therefore, employees should be able to create at least some types of resources and wrap these resources as services, without the help of IT professionals. If an implementation does not consider this aspect, its practical applicability will be limited by design. As an example, the usefulness of enabling the wrapping of a fixed set of web services would only be limited, as employees usually are not able to create customized web services due to a lack of programming skills. In contrast to this, the wrapping of queries enables at least some employees to create desired queries for data retrieval and flexibly include data into widgets.

3.2 Architectural Components of the Widget Composition Platform

We set up a prototypic system, instantiating the conceptual widget stack laid out in the previous subsection. Fig. 3 provides an overview of the system components, which are discussed in the following.

The central component of the system is the WCP, a web application which is implemented using Java technology and is run inside a web application server. The WCP provides an integrated graphical end-user development environment for widget creation. Its GUI can be accessed by calling a certain URL using a web browser. All services that can be mashed up within a widget are managed by a service repository. A widget deployment component within the WCP is responsible for creating source code, encoding the currently created widget for multiple runtime environments. During the process of widget composition within the WCP GUI, this component generates code for the WCP browser runtime libraries, so that the widget is fully functional within the GUI that is rendered inside a browser. If the user decides to deploy the developed widget to a widget runtime engine, the component generates code specific to that runtime environment and packages the widget to a file of according structure and format. To persist data, like the service repository, the personal repository of end-users' created widgets, or login and access right information, the WCP is using a RDBMS.

On the client side, end-users access the GUI of the WCP by using a web browser to call a certain URL. The installation of client-side software or browser plugins is not required to use the WCP. This simplifies the provisioning of the WCP to end-users and makes it very easy for end-users to start with the development of individual widgets. Within the browser, widgets are run based on a WCP runtime library implemented in JavaScript using common web standards. This enables a rendering of the widget and providing full widget functionality inside the browser during the development process without the need of any additional runtime environments installed to the client. Only if the end-user wants to use the developed widget within a certain widget runtime on the client side, this runtime environment needs to be installed on the client and the widget deployed to the runtime environment. For deployment, the WCP is delivering a single file that contains the widget packaged in the specific format required by the runtime environment.

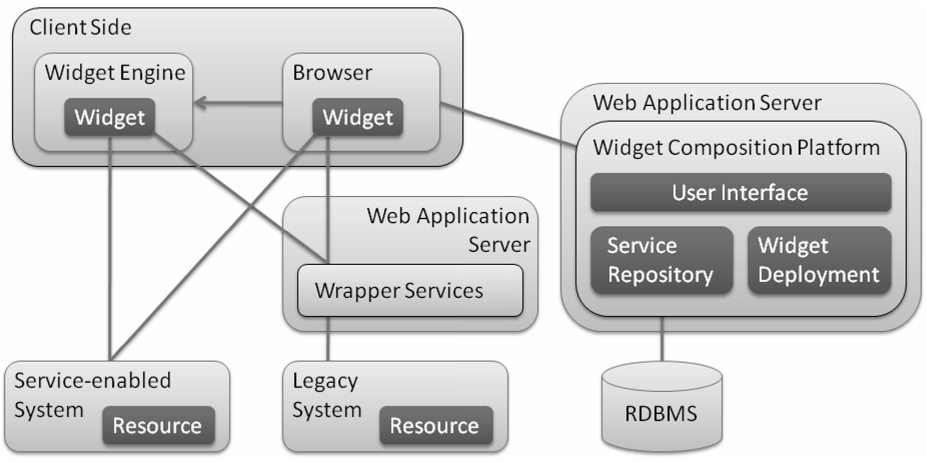


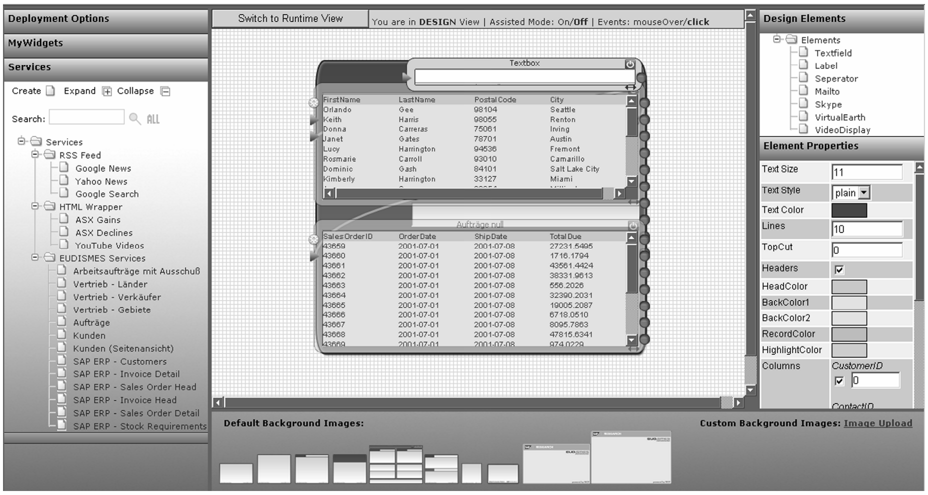
Fig. 2. Architectural components of the Widget Composition Platform

According to the widget stack described in the previous subsection, widgets access the resources that are mashed up within the widget via wrapper components. As wrapper components are encoded directly into the generated widget code, they are subject to the same technical restrictions as the technology used to implement widgets. As widgets need to run within a browser during the development process, wrapper components are limited to calls that can be realized using web standards from within a browser. If resources are managed by service-enabled systems, wrapper components are able to address the resources directly using standard web protocols. If resources are managed by legacy systems, that do not provide an API addressable using standard web protocols, wrapper components are assisted by wrapper services. Wrapper services are deployed to a dedicated web application server and provide access to legacy systems by encapsulating relevant parts of their API with a Representational State Transfer (REST) [21] based API that can be consumed by wrapper components. In our concrete setup we implemented wrapper services to access queries within not service-enabled versions of SAP ERP systems by encapsulating SAP Remote Function Calls (SAP RFC) using SAP Java Connector (SAP JCO), and execution of queries expressed in Structured Query Language (SQL) to RDBMS or Excel files using Java Database Connectivity (JDBC). By this, wrapper components are able to access resources within such systems using simple web protocols, keeping the needed technology on the client side as simple and lightweight as possible.

3.3 GUI of Widget Composition Platform

The WCP provides a browser-based GUI representing an integrated development environment (IDE) enabling the visual development of widgets without any programming knowledge. The GUI does not depend on any browser plugins and can simply be consumed by accessing an URL. A screenshot of the GUI is depicted in Fig. 3.

The GUI is separated into several panes. On the left side, a list of all services contained in the service repository is provided. The list groups services according to their service types. Users are able to add services to a mashup, simply by dragging and



FirstName	LastName	Postal Code	City
Orlando	Gee	98104	Seattle
Keith	Harris	98055	Renton
Donna	Carreras	75061	Irving
Janet	Gates	78701	Austin
Lucy	Harrington	94536	Fremont
Rosmarie	Carroll	93010	Camarillo
Dominic	Gash	84101	Salt Lake City
Kimberly	Harrington	33127	Miami

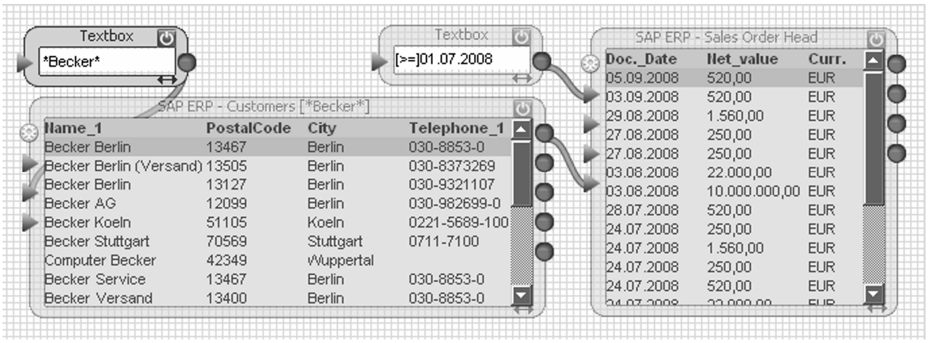
SalesOrderID	OrderDate	ShipDate	Total Due
43659	2001-07-01	2001-07-08	27231.5495
43660	2001-07-01	2001-07-08	1716.1794
43661	2001-07-01	2001-07-08	43561.4424
43662	2001-07-01	2001-07-08	38331.9613
43663	2001-07-01	2001-07-08	556.2026
43664	2001-07-01	2001-07-08	32390.2031
43665	2001-07-01	2001-07-08	19005.2087
43666	2001-07-01	2001-07-08	6718.0510
43667	2001-07-01	2001-07-08	8095.7863
43668	2001-07-01	2001-07-08	47815.6341
43669	2001-07-01	2001-07-08	974.0229

Text Size	11
Text Style	plain
Text Color	■
Lines	10
TopCut	0
Headers	✓
HeadColor	▨
BackColor1	▨
BackColor2	▨
RecordColor	▨
HighlightColor	▨
Columns	*CustomerID* ✓ 0 *ContactID*

Fig. 3. GUI of the Widget Composition Platform

dropping the service to the design pane, located in the middle of the GUI. Besides services, additional design elements, like text boxes, can be dragged from the upper right of the GUI into the design pane. If an element in the design pane is selected, its properties are visible and modifiable in a properties pane on the right. Properties that can be modified include visual properties (such as color or font faces), but – more importantly – structural properties of services. For example, if service data is rendered as a table, then columns can be disabled or enabled to show exactly the desired data, or the number of displayed rows can be limited. From the background pane at the bottom of the GUI, the user is able to select a background image, enhancing the visual appearance of the created widget. Custom background images, e.g. tailored to the corporate identity of an organization, can easily be added by uploading an image in a common format like JPG. After adding services or design elements to the design pane, they are immediately populated with live data and provide runtime interaction possibilities. All design decisions create immediate effects, thus blurring design time and runtime and enabling development close to a WYSIWYG manner, increasing the confidence of the user in creating the desired results.

To mashup services, a wiring has to be defined using a simple box and wires design paradigm. As depicted in Fig. 4, services and other design elements offer input ports and output ports that can be connected to define the wiring. Input ports are visualized as orange triangles on the left side of elements and output ports are visualized as blue circles on the right side of elements. Elements are connected by drawing a line, originating from an output port of a source element to an input port of a target element. Whenever data in the source element changes, the new data is pushed as input to the target element, which updates itself accordingly. The update of data in target elements might again push new data to dependent elements, which in turn might trigger updates. Services update their data by executing a service call, which is parameterized according to the current input values. In the given example, a service providing customer master data is connected to a service providing sales order data. When selecting a customer in the customer table, the connected sales order service is automatically



SAP ERP - Customers [*Becker*]

Name_1	PostalCode	City	Telephone_1
Becker Berlin	13467	Berlin	030-8853-0
Becker Berlin (Versand)	13505	Berlin	030-8373269
Becker Berlin	13127	Berlin	030-9321107
Becker AG	12099	Berlin	030-982699-0
Becker Koeln	51105	Koeln	0221-5689-100
Becker Stuttgart	70569	Stuttgart	0711-7100
Computer Becker	42349	Wuppertal	
Becker Service	13467	Berlin	030-8853-0
Becker Versand	13400	Berlin	030-8853-0

SAP ERP - Sales Order Head

Doc._Date	Net_value	Curr.
05.09.2008	520,00	EUR
03.09.2008	520,00	EUR
29.08.2008	1.560,00	EUR
27.08.2008	250,00	EUR
27.08.2008	250,00	EUR
03.08.2008	22.000,00	EUR
03.08.2008	10.000.000,00	EUR
28.07.2008	520,00	EUR
24.07.2008	250,00	EUR
24.07.2008	1.560,00	EUR
24.07.2008	250,00	EUR
24.07.2008	520,00	EUR

Fig. 4. Wiring of services and design elements

updated to only show sales orders of the selected customer. In case of the depicted services, input ports correspond to filters on table columns and output ports to the column values of the currently selected row. In the given example, two text boxes are used to enable the user to interactively define values that are pushed to services as input. The text box connected to the customer data service restricts the shown customers according to the given name pattern. The text box connected to the sales order service restricts shown sales orders to such sales orders that have been received at or after a given date. In addition to text boxes, other design elements exist, that provide additional functionality, if served with according data. Examples are design elements that use e-mail addresses or Skype names as input and provide the ability to send an e-mail or establish a Skype call just by clicking on an envelope or telephone symbol.

If realizing during the creation process, that a resource needed for the current mashup is missing in the service repository, an according service can be defined using the "Create" option of the service pane. To define a service, the service type needs to be selected (e.g. SAP ERP query) and then the resource that should be encapsulated by the service specified (e.g. by providing the SAP ERP system storing the query, and the name of the query). After saving the service to the repository, it can instantly be used in the mashup process.

At any point of time, the user is able to switch from the design time mode to a runtime mode. Although the widget is fully functional even in design time mode, the runtime mode hides all visual elements that are only needed at design time, like input ports, output ports, or connections, and prevents modifications to the widget. This way, the widget is presented as it would appear if deployed to an external widget engine. Widgets can be saved to and loaded from a personal widget repository managed by the WCP. This functionality is accessible through the "MyWidgets" pane on the left of the GUI. Using the "Deployments Option" pane, the widget can be deployed to multiple widget runtime environments, like Yahoo! Widget Engine [20] or Microsoft Windows Vista Sidebar [22]. When selecting deployment, the WCP is returning a single file, which can either be directly opened and thereby gets deployed to the client-side widget engine or saved as a file, that can easily be sent via e-mail or moved to a file share, to share the widget with colleagues. Widgets that are deployed to a client-side widget engine, run independently of the WCP and are small, self-contained, interactive applications.

4 Evaluation

The aim of the WCP approach is to enable business users to create custom widgets tailored to their individual information needs, without the need of any programming skills. To achieve this aim, a very lightweight composition approach was chosen, to mash up enterprise resources. This enables the creation of simple solutions by simple means. Additionally, advanced end-users are able to create new enterprise services to be mashed up within the WCP by creating and wrapping according resources within enterprise systems. With regard to validity of this approach, several questions arise. Are business users of SMEs able to create widgets using the WCP? Are widgets able to address practical problems in real work contexts, although they are very simple software artifacts composed in a very simple way? Are advanced end-users able to create and wrap enterprise resources as new services to extend the available building blocks for widget creation? What types of end-users exist with regard to widget usage and development, and how do they collaborate? To be able to evaluate our solution approach in real work environments, we deployed the WCP environment to three German SMEs. Additionally, we conducted a questionnaire-based survey among the employees to refine the results of our preliminary studies and to get feedback on our WCP-based solution approach from a broad end-user base. In the following, we describe the setup of the practical evaluation of the WCP, discuss first results of the practical use within SMEs, describe a first classification of end-users based on observed EUD behavior, and discuss first results of the questionnaire analysis. As the evaluation phase is still ongoing at the time of writing, all results have to be considered as preliminary results.

4.1 Setup of Evaluating WCP in Practice

The WCP environment has been deployed to three German midsized companies, which use an SAP ERP system to support their business processes. The companies are two midsized companies from production industry (137 and 140 employees) and one larger software vendor (500 employees). In the respective companies 57, 80 and 350 employees use a PC. Among those, 50, 70 and 116 employees have access to the SAP ERP system. 18, 60 and 70 employees use the SAP ERP system on a regular base. In two of the three companies the initial WCP installation was realized in cooperation with the IT department, and in one company with the person responsible for IT concerns, as no dedicated IT department existed. After installation, the WCP could be accessed by every employee having access to the internal network of the company.

In each company one advanced end-user was nominated as a contact person, responsible for all concerns with regard to the WCP. This person should act as an evangelist to promote the usage of the new technology, as well as provide support to end-users, if questions arise. In a first phase, the contact person was given an introduction into the WCP and should experiment freely with it to get more familiar with its concepts and usage. We defined five distinct services encapsulating SAP ERP data as a starting point for experiments. The services provided common data, like data related to customers, sales orders and invoices. To increase the motivation to go for experiments, we included some appealing external services that could be used, like a service for visualizing addresses on a map, a YouTube video service, Google news, and a stock quote service. In a second phase, we discussed any problems that might have arisen in the first phase and provided help to solve these problems. After that, we discussed potential use cases of widgets within the company with the contact person. In a third phase, we encouraged the contact person to act as an evangelist and promote the usage of the WCP by approaching employees related to the discussed use cases, giving an introduction to the WCP, and motivating its usage. If the contact person was not motivated enough or had problems acting as an evangelist, we gave additional support or acted as evangelists on our own to push the dissemination of WCP technology. During the third and ongoing phase we conduct accompanying interviews with the employees, which are using the WCP for the creation of individual widgets. We investigate what use cases exist, how end-users are able to use the WCP and widgets to address practical problems, and how different end-users collaborate to create solutions.

4.2 Adoption of WCP and Widgets in Practical Use Cases

As the evaluation phase is still in progress at the time of writing, a final analysis of the results cannot be given yet. Nevertheless we are able to discuss some of our first preliminary results, which are very promising. In the following we exemplarily discuss two use cases which describe how business users adopt widgets that have been created.

Use Case 1: Sales Support Widget. A user in the sales department is in charge of answering questions of customers related to sales orders. Customers contact the user by phone to get information related to the content and status of sales orders. Inquiries comprise e.g. questions about whether or not certain goods have been ordered in a certain sales order, or what the current state of sales order processing is. To be able to answer such questions, the user has to access multiple locations inside the GUI of the SAP ERP system. In a first step, the user needs to access customer master data to uniquely identify the customer. In a second step, the user accesses sales order header data to filter sales orders of the respective customer and the sales order of question. In a third step, the individual items of the sales order are accessed to view ordered goods, the ordered quantity and their status. If multiple sales orders are of interest, the user needs to switch back and forth between the sales order list and its details. The process of incrementally gathering required information from the GUI by accessing multiple locations and the need to switch back and forth between them is rated to be cumbersome by the user, especially as the user has to access the information many times a day.

The user was approached by the local evangelist and was shown the WCP. The user started to experiment with the WCP. Using the services we defined as standard demo services during installation, the user was able to create a suitable widget for his needs. The widget shows customer master data, sales order header data and sales order details as three distinct tables. Using text boxes the user added filters for customers by name or customer number, and filters on the order date of sales orders. The user deployed the widget to his desktop to make it easily accessible and uses it to quickly access sales order related data if customers ask for them. By using the filters on customer master data, a customer can be uniquely identified in a fast and comfortable way. By clicking on this customer in the table, all sales orders of the customer are shown in another table, directly beneath the customer table. The user then restricts the shown sales orders to the ones ordered at or after a certain order date. By clicking on a sales order, all relevant details are shown in a third table. The user configured the tables to only show data relevant for him, resulting in a compact overview providing required data at a glance. To view details of other sales orders in question, the user simply clicks on the sales order in the sales order table.

The created widget is used by the user about 40 times a day to answer standard questions of customers related to sales orders. As the user does not need to access multiple locations within the complex GUI of the SAP ERP system, but gets all relevant data at a glace, he is able to answer standard customer inquiries in approximately the half of the time compared to using the GUI of the SAP ERP system. Because of this practical value, the user sent the widget to a colleague, who needs to access such data occasionally. The user has absolutely no programming skills and was able to create a custom widget for supporting his individual working task within three hours after having seen the WCP for the first time. The user told us that experimenting with the WCP was fun for him. He perceived the WCP to have an appealing user interface and the composition of widgets to be simple, comprehensible and easy to learn.

Use Case 2: Material Lookup Widget. A user in charge of procurement needs to access certain information related to material many times a day. For instance, identifying a material by its material number and getting the quantity currently in stock and the quantity already scheduled for production. To get a first overview of the material status, the user determines a constant set of information, which she considers to be standard for her work context. Similar to the previous use case, relevant information are widely spread within the GUI of the SAP ERP system and have to be gathered in a cumbersomely manner.

As the user was approached by the local evangelist and was shown the WCP, she immediately thought of building a widget to access her standard set of material related information. As no predefined services existed that could deliver relevant data, the evangelist discussed with the user which data was actually needed and searched for possible data sources like tables and queries within the ERP system. Using a larger query as a template, the evangelist managed to create a SAP query joining five distinct tables and providing most of the requested data. The query was wrapped as a service for the WCP and was used by the user to create a suitable widget.

The widget is rather simply structured and just consists of a text box and the created service. The text box defines a filter on the material number to the service. As the service just returns a single record, the results are not rendered as a table with a single row, but as a list, showing all attributes of the record with according values as rows. The user configured the service to show only the most relevant data and arranged all design elements neatly to create an appealing widget. The widget was also deployed on the PC of an employee working in the raw material warehouse, who did not have a direct access to live data from the SAP ERP system before. By using the widget, he is able to access the most important data related to material in a very easy way, without the need of having to learn and understand a complex enterprise system.

4.3 Types of End-Users

Based on first insights obtained by accompanying interviews and observations, we distinguish different types of end-users with regard to widget usage and development. An according segmentation of end-users is given by the following classification: widget consumers, widget creators, and service creators.

Widget Consumers. Widget consumers are end-users which are only consuming widgets, but do not create widgets. Some users only use computers occasionally and are not very familiar with complex information systems. They receive widgets from more experienced colleagues and use them to access data in an easy and comfortable way without the need of learning and understanding one or more complex information systems. By this, the value of information stored in enterprise systems is leveraged, as it enables more end-users to access them and make better informed decisions. Additionally this removes the need of asking colleagues for data and thus makes data access more efficient. Another class of widget consumers consists of end-users that could create widgets for their own, but are not motivated enough to engage in widget creation. On the other hand, they willingly use widgets that are created by other end-users that turn out to be useful to support own working tasks.

Widget Creators. Widget creators are end-users that create widgets for themselves or others to support individual working tasks. They are motivated by multiple reasons to create widgets. One main reason is to simplify data access for own working tasks and thereby making these working tasks more efficient and less cumbersome. Another motivation is to provide others with especially tailored widgets, to improve data supply to others engaged into the same business processes. By this, the amount of inquiries for data is reduced and business process may be executed more efficiently. Providing better tools to support individual working tasks of processes is especially relevant for end-users being responsible for certain processes and thus motivates them to act as widget creator to optimize these processes and improve process performance. End-users successfully using widgets motivate others to engage in widget creation as they are motivated by the success of others and want to use such optimization possibilities for their own working tasks and processes. Other end-users are technology-savvy and start widget creation for the reason of having fun experimenting with new technology. They discover the usefulness for own tasks or tasks of others while experimenting.

Service Creators. Service creators are advanced end-users that are able to create new resources that can be wrapped and added to the service repository, from where they can be used for widget creation. By extending the service repository with new services, they enable widget creators to address more use cases and create more widgets that are more precisely tailored to the individual needs of end-users. Without service creators, widget creators would be limited to a certain set of predefined services which limits the amount of creatable solutions. With regard to the WCP environment service creators are end-users having the skills of creating, modifying or at least locating suitable queries inside the SAP ERP system that match existing information needs of widget creators.

4.4 Questionnaire-Based Evaluation of WCP

To refine the results of the preliminary studies and to acquire feedback on the WCP-based approach on a broad end-user base, we conducted a questionnaire-based survey among the employees of the companies participating in the evaluation of the WCP environment. Besides the WCP related aspects, the questionnaire addressed many more aspects like IT-related skills of employees, satisfaction with provided enterprise software, experience with EUD related activities, as well as working practices and problems with regard to individual information processing and access. For the sake of brevity, we focus on WCP related aspects only in the following. As not all employees participating in the survey knew the WCP environment, we provided the questionnaire online and embedded a small video presenting the WCP. The video was shown to the participants after having explained a small use case and showed the employees how to create a suitable enterprise widget and deploy it to the desktop in less than three minutes. As common office PCs mostly do not provide sound, short explanations were given by fading text in the video. Based on the video, the questionnaire asked several questions directly related to the WCP approach. Video presentation can be seen as a viable medium for demonstrating systems in a user acceptance testing context as it enables subjects "to form accurate attitudes, usefulness perceptions, quality perceptions and behavioral expectations (self-predictions of use)" [23], which are important factors for technology acceptance according to the Technology Acceptance Model [24].

Each of the three participating companies was asked to send the URL to access the questionnaire to 33 randomly chosen users of the SAP ERP system. Finally we received 73 filled and analyzable questionnaires. With regard to a total of 236

Table 1. Questions related to the WCP environment and widgets with according results

Question	*Results*
How do you rate the difficulty level of the shown method of widget creation?	0% too difficult, 13.8% difficult but manageable, 39.7% passable, 22.4% easy, 20.7% very easy, 3.4% declared not to answer
Do you think you could manage to create a widget using the shown method on your own?	72.5% yes, 6.9% no, 17.2% uncertain, 3.4% declared not to answer
Do you think custom-made widgets can provide benefits in real work contexts?	69.1% yes, 5.3% no, 21.3% uncertain, 4.3% declared not to answer
Do you think, you could ease or accelerate your work by using widgets tailored to your needs?	Scale from 0 (would not provide any benefits for me) to 5 (would help me a lot): 6.4% 0, 5.3% 1, 17.0% 2, 26.6% 3, 26.6% 4, 7.4% 5, 9.6% declared not to answer, 1.1% did not answer
Is there any personal, typical work situation that spontaneously comes to your mind, in which a widget would be of help for you?	47.9% yes, 27.6% no, 14.9% uncertain, 8.5% declared not to answer, 1.1% did not answer
If your last answer was "yes", would you be willing to create a widget for that purpose on your own?	58.5% yes, 8.5% no, 17.0% uncertain, 12.8% declared not to answer, 3.2% did not answer
Would you be willing to accept learning efforts to learn the creation of widgets?	5.3% no, 7.4% yes, up to half an hour, 11.7% yes, up to an hour, 19.1% yes, several hours, 21.3% yes, a day, 28.8% yes, several days, 5.3% declared not to answer, 1.1% did not answer
Can you think of using complete, predefined widgets in your everyday work, if they provide data relevant to you?	84.0% yes, 2.1% no, 8.5% uncertain, 4.3% declared not to answer, 1.1% did not answer

employees working with the SAP ERP system in these companies, we achieved a coverage of about 30.1% of the addressed test population. Table 1 lists some of the questions which have been asked in the questionnaire after the participants watched the video demonstrating the WCP environment, together with the according results.

The results are promising. The participants rate the WCP not to be too difficult. 82.8% rate the perceived difficulty level to be passable, easy or very easy. 72.5% think that they can manage to create widgets on their own using the WCP environment. 69.1% think that custom-made widgets are able to provide benefits in real work context. On a scale ranging from 0 (widgets do not provide any benefit for me) to 5 (widgets would help me a lot), 60.6% of participants rated the ability of widgets to ease or accelerate their own work with at least 3. 47.9% of the participants state, that a typical, personal work situation spontaneously comes to their mind, in which widgets would be helpful. 58.5% of these participants would be willing to create a widget on their own for that work situation. 88.3% of the participants are willing to accept learning efforts to learn the creation of widgets. 69.2% of these participants would accept learning efforts ranging from several hours to several days. 84% of the participants would use complete, predefined widgets in their every day work, if they provide data relevant to them.

The results indicate that typical business users of SMEs, which are using an ERP system in their daily work, believe that widgets are able to provide benefits in real work contexts. Many business users see application possibilities in their own, personal work context and would like to create a widget for that purpose on their own, even if they have to accept learning effort to achieve this aim. The vast majority of business users believes to be able to create widgets with the WCP environment and rates its usage to be not difficult. Overall, this can be seen as quite a positive and promising feedback with regard to the WCP environment and the practical applicability of widgets. According to the Technology Acceptance Model [24], the high degree of perceived usefulness can be seen as a strong indicator for acceptance.

5 Summary and Conclusion

In this paper, we presented an EUD approach enabling business users to create enterprise widgets tailored to their personal information needs without the need of programming knowledge. The approach is based on a prototypic, web-based EUD environment called WCP, which enables end-users to mash up enterprise resources in a visual design environment in a very lightweight way using a simple box and wires design paradigm and to deploy the created mashups in the form of widgets to their local machines. The ability of creating custom widgets enables business users to create small, interactive applications to access relevant business data in a fast and convenient way, without the need of starting heavyweight and complex enterprise systems and cumbersomely collect data from multiple locations within the GUI of such systems. This creates a new experience for end-users, as they were not able to create interactive tools providing live interaction on enterprise resources to support their individual work tasks on their own before. The approach especially considers extensibility of building blocks for widget creation by SMEs using existing knowledge, as it enables to wrap and expose enterprise resources (e.g. SAP ERP queries), which can be created by advanced end-users, as services that can be mashed up in a widget. Thus, the whole chain of widget development is put in the hand of SMEs.

We described our research approach targeted at creating EUD solutions applicable in real enterprise contexts, and motivated the need and setup of our approach with results of our preliminary empirical studies. With regard to the set up of our solution approach, we discussed the conceptual layers of the widget stack, which is instantiated by the WCP environment, and explained the architectural components of the WCP and how these work together. We presented the GUI of the WCP and explained how end-users are able to create individual mashups of enterprise resources using a simple box and wires design paradigm, and how these mashups can be deployed as widgets to the local machines of the end-users. With regard to evaluation of the approach, we presented first results of a practical evaluation of the EUD environment in three German SMEs. We provided use cases from real enterprise contexts, and gave a classification of end-user types based on our observations. Additionally, we presented results of a questionnaire-based survey conducted on a broad end-user base, which document a very promising feedback with regard to our approach.

The provided use cases from real enterprise contexts demonstrate that business users are able to create custom widgets supporting their individual work tasks using the provided EUD environment. The use cases gave evidence that widgets can be used to address practical problems in real work contexts, although they are very simple software artifacts composed in a very simple way. The results of the questionnaire-based survey support these conclusions, as they show that a broad base of typical business users believe that widgets are able to provide benefits in real work contexts, see application possibilities in their own work context, and would like to create a widget for that purpose on their own, even if they had to accept learning effort to achieve this aim. The vast majority believes to be able to create widgets with the WCP environment and rates its usage to be not difficult.

Beyond this, we observed in practical situations that advanced end-users are able to create and wrap enterprise resources as new services to extend the available building blocks for widget creation. By this, the whole development chain of widget creation can be put in the hands of SMEs and allows them to act as autonomous creators of services and widgets, without the need of external IT professionals. With regard to our aim of creating EUD approaches for enterprise environments, which can be managed even by SMEs and enable business users to better adapt software to their individual working tasks and work practice, we consider these first results as being very promising.

With our research, we contribute to the field of EUD in various ways. We give an example of how the technical flexibility of SOAs can be leveraged at the hand of end-users using approaches like mashups and widgets. We provide indications, taken from our evaluation, that a simple box and wires design paradigm, combined with a design environment blurring design time and runtime can be easily learned and used by typical business users to orchestrate simple software artifacts. We show that EUD approaches enabling business users to create simple software artifacts like widgets are able to address practical problems in real enterprise work contexts. Additionally, we provide indications from our observations, that by enabling SMEs to create enterprise resources using existing knowledge and wrapping these resources as building blocks for the EUD of software artifacts, the whole development chain of software artifacts can be put in the hands of SMEs, thus reducing the need of external IT professionals and at the same time increasing flexibility in adapting the used software infrastructure to individual and changing needs.

Acknowledgments. We would like to thank Alexander Dreiling and Kathrin Fleischmann of SAP Research for providing an early prototype of the WCP we could branch and extend. The presented research was funded by the German Federal Ministry of Education and Research (BMBF) under the project EUDISMES (number 01ISE03C).

References

1. Roth, A., Scheidl, S.: End-User Development for Enterprise Resource Planning Systems. In: Informatik 2006, pp. 596–599. GI (2006)
2. Brehm, L., Heinzl, A., Markus, M.L.: Tailoring ERP Systems: A Spectrum of Choices and their Implications. In: 34th Annual Hawaii International Conference on System Sciences (HICSS-34). IEEE, Los Alamitos (2001)
3. Markus, M.L., Tanis, C.: The Enterprise System Experience: From Adoption to Success. In: Zmud, R.W. (ed.) Framing the Domains of IT Research: Glimpsing the Future through the Past, pp. 173–207. Pinnaflex (2000)
4. Wulf, V., Rohde, M.: Towards an integrated Organization and Technology Development. In: Designing Interactive Systems 1995 (DIS 1995). ACM, New York (1995)
5. Gallivan, M.J., Keil, M.: The User–Developer Communication Process: A critical Case Study. ISJ 13, 37–68 (2003)
6. Beringer, J.: Reducing Expertise Tension. Commun. ACM 47, 39–40 (2004)
7. Lieberman, H., Paternò, F., Wulf, V.: End User Development. Springer, Heidelberg (2006)
8. Spahn, M., Dörner, C., Wulf, V.: End User Development: Approaches towards a flexible Software Design. In: 16th European Conference on Information Systems (ECIS 2008), pp. 303–314. CISC (2008)
9. Wulf, V., Pipek, V., Won, M.: Component-based Tailorability: Towards highly flexible Software Applications. IJHCS 66, 1–22 (2008)
10. Erl, T.: Service-oriented Architecture: Concepts, Technology, and Design. Prentice-Hall, Englewood Cliffs (2005)
11. Microsoft Popfly, `http://www.popfly.com/`
12. Yahoo! Pipes, `http://pipes.yahoo.com/`
13. Gartner Identifies the Top 10 Strategic Technologies for 2008 (2008), `http://www.gartner.com/it/page.jsp?id=530109`
14. Spahn, M., Dörner, C., Wulf, V.: End User Development of Information Artefacts: A Design Challenge for Enterprise Systems. In: 16th European Conference on Information Systems (ECIS 2008), pp. 482–493. CISC (2008)
15. Caceres, M.: Widgets 1.0 Requirements. W3C Working Draft. W3C (2008)
16. Kvale, S.: Interviews: An Introduction to Qualitative Research Interviewing. Sage Publications, Thousand Oaks (1996)
17. Muller, M.J.: Participatory Design: The third Space in HCI. In: The Human-Computer Interaction Handbook: Fundamentals, evolving Technologies and emerging Applications, pp. 1051–1068. Erlbaum (2003)

18. Spahn, M., Kleb, J., Grimm, S., Scheidl, S.: Supporting Business Intelligence by Providing Ontology-based End-User Information Self-Service. In: 1st International Workshop on Ontology-supported Business Intelligence (OBI 2008). ACM, New York (2008)
19. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-tailorable Systems: Pressing the Issues with Buttons. In: SIGCHI Conference on Human Factors in Computing Systems (CHI 1990), pp. 175–182. ACM, New York (1990)
20. Yahoo! Widgets, `http://widgets.yahoo.com/`
21. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, USA (2000)
22. Lal, R.: Creating Vista Gadgets. Sams (2008)
23. Davis, F.D.: A Technology Acceptance Model for empirically testing new End-User Information Systems: Theory and Results. PhD thesis, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA (1986)
24. Davis, F.D.: Perceived Usefulness, perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly 13, 319–340 (1989)