MarcoFlow: Modeling, Deploying, and Running Distributed User Interface Orchestrations

(Demonstration Proposal)

Florian Daniel, Stefano Soi, Stefano Tranquillini, Fabio Casati

University of Trento, Povo (TN), Italy
{daniel,soi,tranquillini,casati}@disi.unitn.it

Chang Heng, Li Yan

Huawei Technologies, Shenzhen, P.R. China
{changheng,liyanmr}@huawei.com

Abstract. This demo introduces the idea of *distributed orchestration of user interfaces* (UIs), an application development approach that allows us to easily bring together UIs, web services and people in a single orchestration logic, language, and tool. The tool is called *MarcoFlow*, and it covers three main phases of the software development lifecycle: design (by means of a dedicated, visual editor), deployment (by means of a set of code generators), and execution (by means of a distributed runtime environment for UI orchestrations). We showcase the benefits of MarcoFlow in each of the phases by developing and running a practical and expressive application for the management of home assistance and by explaining, in each phase, which are the challenges, which the intuitions, and which the solutions. The demo targets the development of *mashuplike applications that require (distributed) process support* and, hence, targets researchers and practitioners interested in mashups, lightweight process design, web services, and innovative (and free) ways of providing process support.

1 Introduction

After workflow management (which supports the automation of business processes and human tasks) and service orchestration (which focuses on web services at the application layer), web mashups [1] feature a significant innovation: *integration at the UI level*. Besides web services or data feeds, mashups indeed reuse pieces of UIs (e.g., content extracted from web pages or JavaScript UI widgets) and integrate them into new web pages or applications. While mashups therefore manifest the need for reuse in UI development and for suitable UI component technologies, so far they only produced rather simple applications, most of the times consisting of only one web page and of little utility.

This demo complements the concepts introduced in [2], where we argue that there is a huge spectrum of applications that demand for development approaches that are similar to those of mashups but that go far beyond single page applications and in fact support multiple pages, multiple actors, complex navigation structures, and – more importantly – process-based application logic or navigation flows.

As of today, there is no single development instrument that allows one to develop this kind of applications using one language and one environment only. Filling this gap is the goal of MarcoFlow.

2 Demo scenario

In this demo, we want to develop an application that supports the scenario graphically described in Figure 1: A *patient* can ask for the visit of a home assistant (e.g., a paramedic) by calling (via phone) an operator of the home assistance service. Upon request, the *operator* inputs the respective details into his management console (steps 1 and 2; the console consists of four individual UI components) and inspects the patient's data and personal health history in order to provide the assistant with the necessary instructions to assist the patient (3, 4). The *home assistant* views the description in his own application (5, 6, 7; four UI components), visits the patient, and files a report about the provided service (8). The report is processed by the *back-end system* and archived (9). If no further exams are needed, the UI orchestration ends (10, 11). If exams are instead needed, the respective details need to be sent to the operator (11, 12) so that he can book the exam in the local hospital, asking the patient for confirmation (again via phone). Upon confirmation of the exam booking, the system also archives the booking (13, 14), which terminates the responsibility of the home assistance service (15).
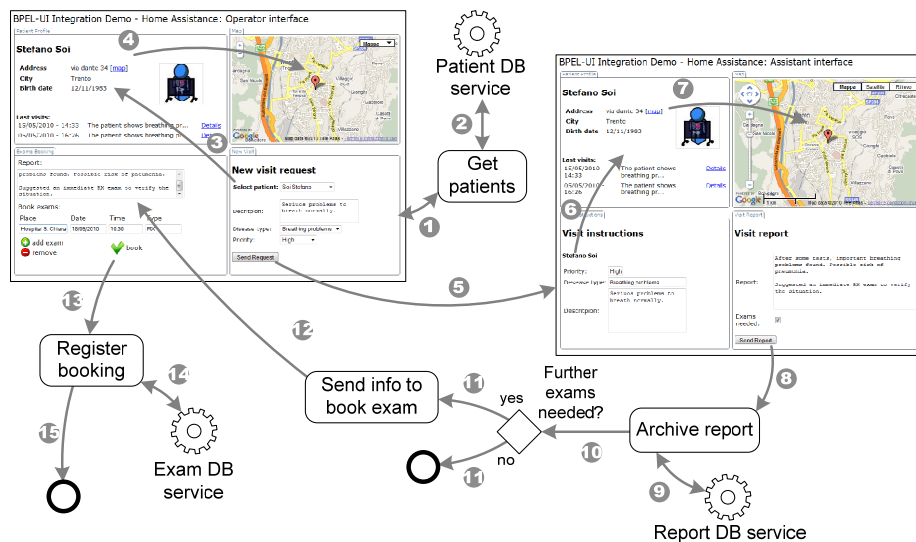

Figure 1 The distributed UI orchestration to be developed during the demo. The gray arrows indicate synchronization or orchestration points; they number labels indicate their order in time.

Our goal is to develop an application that supports this process. This application includes, besides the process logic, two mashup-like, web-based control consoles for

the operator and the assistant that are themselves part of the orchestration and need to interact with (and are affected by) the evolution of the process. Furthermore, the UI is itself component-based and created by reusing and combining existing UI components (each of the two pages in Figure 1 is, for instance, composed of four UI components). For each new request, the operator starts a new instance of the application, raising the need for correlation of services and UI components.

If we analyze the scenario, we see that the envisioned application (as a whole) is *distributed* over the Web: The UIs for the actors participating in the application are composed of UI components, which can be components developed in-house (like the *New Request* component) or sourced from the Web (like the *Map* component); service orchestrations are based on web services. The UI exposes the state of the application and allows users to interact with it and to enact service calls. The two applications for the operator and the assistant are instantiated in different web browsers, contributing to the distribution of the overall UI and raising the need for synchronization.

3   MarcoFlow: An Environment of Distributed UI Orchestration

The *key idea* to approach the coordination of (i) UI components inside web pages, (ii) web services providing data or application logic, and (iii) individual pages (as well as the people interacting with them) is to split the coordination problem into two layers: *intra-page UI synchronization* and *distributed UI synchronization and web service orchestration*.

Figure 2 shows the (simplified) architecture of the MarcoFlow environment, which aids the development and execution of the demo scenario. The architecture is partitioned into design time, deployment time, and runtime components, according to the three phases of the software development lifecycle supported by MarcoFlow.

The **design** part comprises the *BPEL4UI editor* that supports BPEL4UI [2], the composition language we use to specify distributed UI orchestrations. The editor is an extended Eclipse BPEL editor with (i) a panel for the specification of the pages in which UI components can be rendered and (ii) a property panel that allows the developer to configure the web pages, to set the properties of UI partner links, and to associate them to place holders in the layout.

The **deployment** of a UI orchestration requires translating the BPEL4UI specification into executable components: (i) a set of *communication channels* that mediate between the UI components in the client browser and the BPEL engine; (ii) a *standard BPEL specification* containing the distributed UI synchronization and web service orchestration logic; and (iii) a set of *UI compositions* (one for each page of the application) containing the intra-page UI synchronizations. This task is achieved by the *BPEL4UI compiler*, which also manages the deployment of the generated artifacts in the respective runtime environments.

The **execution** of a UI orchestration requires the setup and coordination of three independent runtime environments: (i) the interaction with users and intra-page UI synchronization is managed in the client browser by an *event-based JavaScript runtime framework*; (ii) a so-called *UI engine server* runs the web services implementing
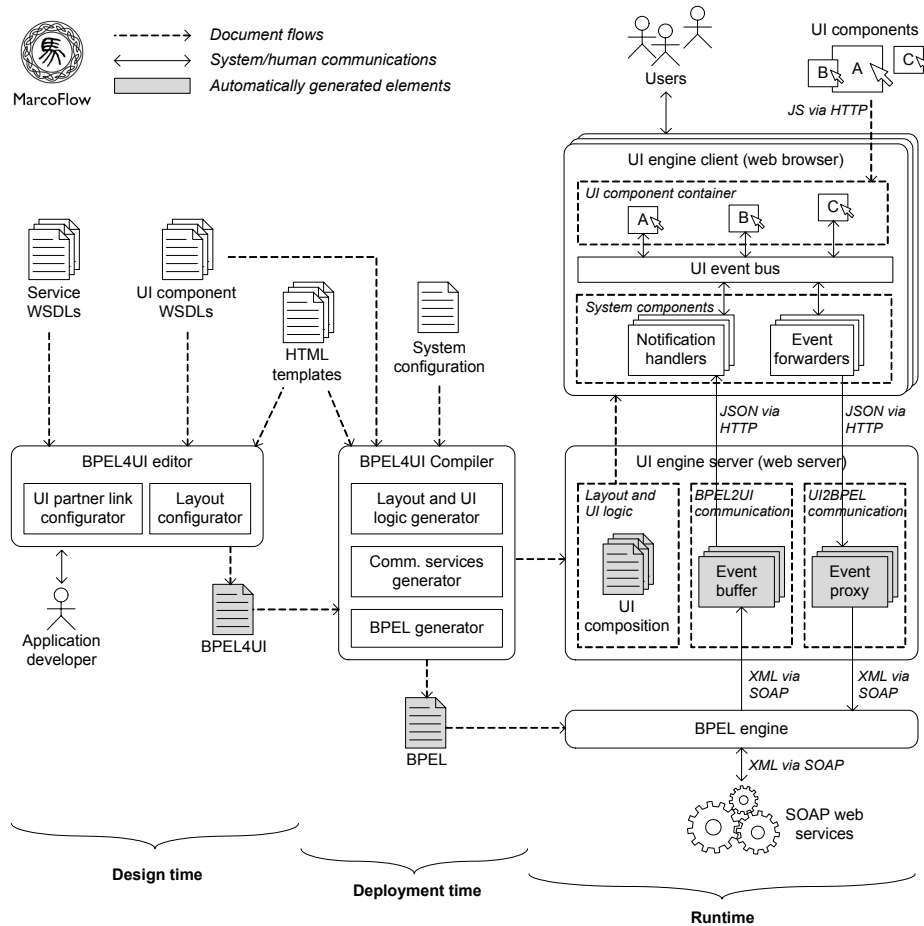
Figure 2 From design time to runtime: overall system architecture of MarcoFlow

the communication channels; and (iii) a *standard BPEL engine* manages the distributed UI synchronization and web service orchestration.

The MarcoFlow system shown in Figure 2 is fully implemented and running. A patent application for parts of the system has been filed.

4 Demonstration Script

During the demo we intend to develop the UI orchestration described in Section 2 from scratch and run it live, while interacting with the audience and answering questions and clarifying details. The planned demo flow is as followed:

1. *Goals of MarcoFlow*: explanation of the motivation that led us to the development of the distributed UI orchestration approach and of the goals that we pursued in doing so.
2. *The home assistance scenario*: explanation of the scenario described in Section 2, the development challenges it implies, and the basic assumptions we have regarding people, user interfaces, and web services.
3. *The MarcoFlow approach*: explain the key idea behind the MarcoFlow approach to UI orchestration, i.e., splitting the coordination problem into intra-page UI synchronization and distributed UI synchronization and web service orchestration.
4. *System architecture*: description of the architecture and the internals of Marco-Flow. The description highlights which components support which phase in the development cycle: design, deployment, and runtime.
5. *The basic components*: here we explain how the UI components and web services we compose look like and how they can be abstractly described in a homogenous fashion, using standard WSDL and a new JavaScript binding for UI components.
6. *Designing distributed UI orchestrations*: explanation and live demo of the visual, BPEL4UI editor. The goal here is to develop step by step the UI orchestration of the reference scenario and to highlight modeling conventions and patterns.
7. *Code generation and deployment*: as highlighted when explaining the system architecture, MarcoFlow is a complex system in which many of the components are generated automatically starting from a BPEL4UI specification. Here we explain which components are created, how, and where they are deployed, in order to provide for the necessary execution support.
8. *Running the application*: after the deployment, the application is ready for execution. Running an application is supported by a simple web console for users that allows them to easily start UI orchestrations (the operator) and to participate in already running UI orchestrations (the assistant). Here we show how users can interact with their individual web pages and how these and the UI components inside the pages are orchestrated as part of the overall process logic. Special emphasis is given to the distinction of the two actors, i.e., the operator and the home assistant.
9. *Conclusion*: finally, we conclude the presentation with a recap of the demonstration and the MarcoFlow system and we outline our ideas for future works.

A demo of the tool is available at *http://mashart.org/marcoflow/demo.htm*

References

1. J. Yu, B. Benatallah, F. Casati, F. Daniel. Understanding Mashup Development and its Differences with Traditional Integration. *IEEE Internet Computing*, Vol. 12, No. 5, September-October 2008, pp. 44-52.
2. F. Daniel, S. Soi, S. Tranquillini, F. Casati, C. Heng, L. Yan. From People to Services to UI: Distributed Orchestration of User Interfaces. *BPM'10*, Hoboken, NJ, USA.

MarcoFlow: Modeling, Deploying, and Running Distributed User Interface Orchestrations

(Demonstration Paper)

Florian Daniel, Stefano Soi, Stefano Tranquillini, Fabio Casati

University of Trento, Povo (TN), Italy
{daniel,soi,tranquillini,casati}@disi.unitn.it

Chang Heng, Li Yan

Huawei Technologies, Shenzhen, P.R. China
{changheng,liyanmr}@huawei.com

Abstract. This demo introduces the idea of *distributed orchestration of user interfaces* (UIs), an application development approach that allows us to easily bring together UIs, web services, and people in a single orchestration logic, language, and tool. The tool is called *MarcoFlow*, and it covers three main phases of the software development lifecycle: design (by means of a dedicated, visual editor), deployment (by means of a set of code generators), and execution (by means of a distributed runtime environment for UI orchestrations). MarcoFlow targets the development of *mashup-like applications that require (distributed) process support* and, hence, targets researchers and practitioners interested in mashups, lightweight process design, web services, and innovative (and free) ways of providing process support.

1 Introduction

After workflow management (which supports the automation of business processes and human tasks) and service orchestration (which focuses on web services at the application layer), web mashups [1] feature a significant innovation: **integration at the UI level**. Besides web services or data feeds, mashups indeed reuse pieces of UIs (e.g., content extracted from web pages or JavaScript UI widgets) and integrate them into new web pages or applications. While mashups therefore manifest the need for reuse in UI development and for suitable UI component technologies, so far they produced rather simple applications consisting of one web page and of little utility.

This demo complements the concepts and solutions introduced in [2], where we argue that there is a huge spectrum of applications that demand for development approaches that are similar to those of mashups but that go far beyond single page applications and in fact support multiple pages, multiple actors, complex navigation structures, and – more importantly – process-based application logic or navigation flows. We call this type of applications **distributed UI orchestrations**, as (i) both components and the application itself may be distributed over the Web, (ii) in addition to traditional web services we also integrate novel JavaScript UI components, and (iii) services and UIs are orchestrated in an integrated fashion.

Challenges and contributions. Developing distributed UI orchestrations implies the coordination of individual actors and the development of a *distributed* user interface *and* service orchestration logic. Doing so requires (i) understanding how to *componentize UIs and compose them into web applications*, (ii) defining a logic that is able to *orchestrate both UIs and web services*, (iii) providing a language and tool for *specifying distributed UI compositions*, and (iv) developing a runtime environment that is able to *execute distributed UI and service compositions*.

Innovativeness of the tool. As of today, there is no single development instrument that answers these challenges and allows one to develop UI orchestrations using one language and one environment only. *BPEL* [3] focuses on web services only. *BPEL4People* [4] adds human tasks and actors as first-class concepts, but without supporting the development of suitable UIs. *Model-driven web design instruments*, such as WebRatio [5] or VisualWade [6], allow the development of advanced web applications, without however facilitating reuse of UI components sourced from the Web. *Portals and portlets* [7], instead, focus specifically on reuse, but they fail in supporting service integration and process flows. *Mashup tools* [1] support the integration of UIs and of services, but they typically do not support complex orchestration patterns (if not hand-coded). In mashArt [8], we did some first steps into that direction, but without considering multi-user and distributed UI support.

Significance to the BPM field. With MarcoFlow, we go one step beyond state-of-the-art BPM and service composition and propose an original model, language and running system for the composition of distributed UIs. The approach brings together UIs, web services and people in a single orchestration logic and tool and supports the development of mashup-like applications that require (distributed) process support, a kind of application that so far was not supported by BPM practices and software.

2 Distributed UI Orchestration

The **key idea** to approach the coordination of (i) UI components inside web pages, (ii) web services providing data or application logic, and (iii) individual pages (as well as the people interacting with them) is to split the coordination problem into two layers: *intra-page UI synchronization* and *distributed UI synchronization and web service orchestration*. UIs are typically event-based (e.g., user clicks or key strokes), while service invocations are coordinated via control flows. In this demo and in [2], we show how to describe UI components in terms of standard WSDL descriptors, how to bind them to JavaScript, and how to extend the standard BPEL language in order to support the two above composition layers. We call this extended language *BPEL4UI*.

Figure 1 shows the simplified meta-model of **BPEL4UI**. Specifically, the figure details all the new modeling constructs necessary to specify UI orchestrations (gray-shaded) and omits details of the standard BPEL language, which are reused as is by BPEL4UI. In terms of standard BPEL [3], a UI orchestration is a *process* that is composed of a set of associated *activities* (e.g., sequence, flow, if, assign, validate, or similar), *variables* (to store intermediate processing results), *message exchanges*, *correlation sets* (to correlate messages in conversations), and *fault handlers*. The services or UI components integrated by a process are declared by means of so-called

partner links, while *partner link types* define the roles played by each of the services or UI components in the conversation and the *port types* specifying the operations and messages supported by each service or component.
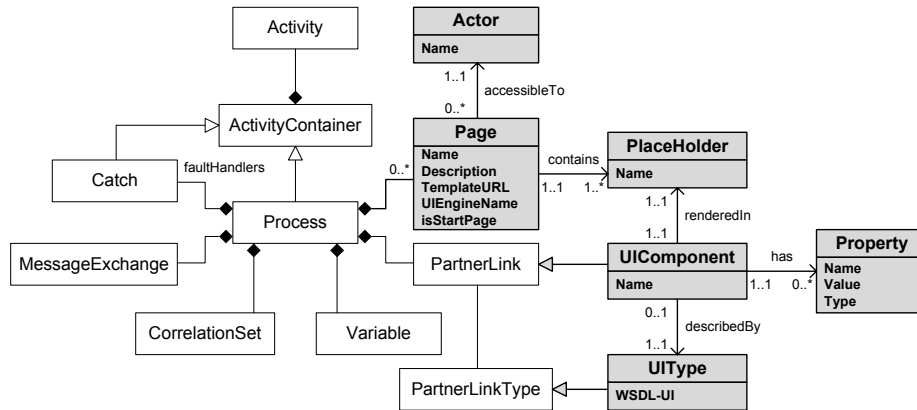

Figure 1 Simplified BPEL4UI meta-model in UML. White classes correspond to standard BPEL constructs; gray classes correspond to constructs for UI and user management.

Modeling UI-specific aspects requires instead introducing a set of **new constructs** that are not yet supported by BPEL. The constructs, illustrated in Figure 1, are: *UI type* (the partner link type for UI components), *page* (the web pages over which we distribute the UI of the application), *place holder* (the name of the place holders in which we can render UI components), *UI component* (the partner link for UI components), *property* (the constructor parameters of UI components), and *actor* (the human actors we associate with web pages).

It is important to note that although syntactically there is no difference between web services and UI components (the new JavaScript binding introduced into WSDL to map abstract operations to concrete JavaScript functions comes into play only at runtime), it is important to distinguish between services and UI components as their *semantics* and, hence, their usage in the model will be different. A detailed description of the new constructs and their usage can be found in [2].

As for the **layout** of distributed UI orchestrations, defining web pages and associating UI partner links with place holders requires implementing suitable HTML templates that are able to host the UI components of the orchestration at runtime. For the design of layout templates we rely on standard web design instruments. The only requirement the templates must satisfy is that they provide place holders in form of HTML DIV elements that can be indexed via standard HTML identifiers following a predefined naming convention, i.e., *<div id="marcoflow-left">… </div>*.

The main **methodological goals** in implementing our UI orchestration approach were (i) relying as much as possible on existing *standards*, (ii) providing the developer with only *few and simple new concepts*, and (iii) implementing a runtime architecture that associates each concern to the *right level of abstraction and software tool* (e.g., UI synchronization is handled in the browser, while service orchestration is delegated to the BPEL engine).
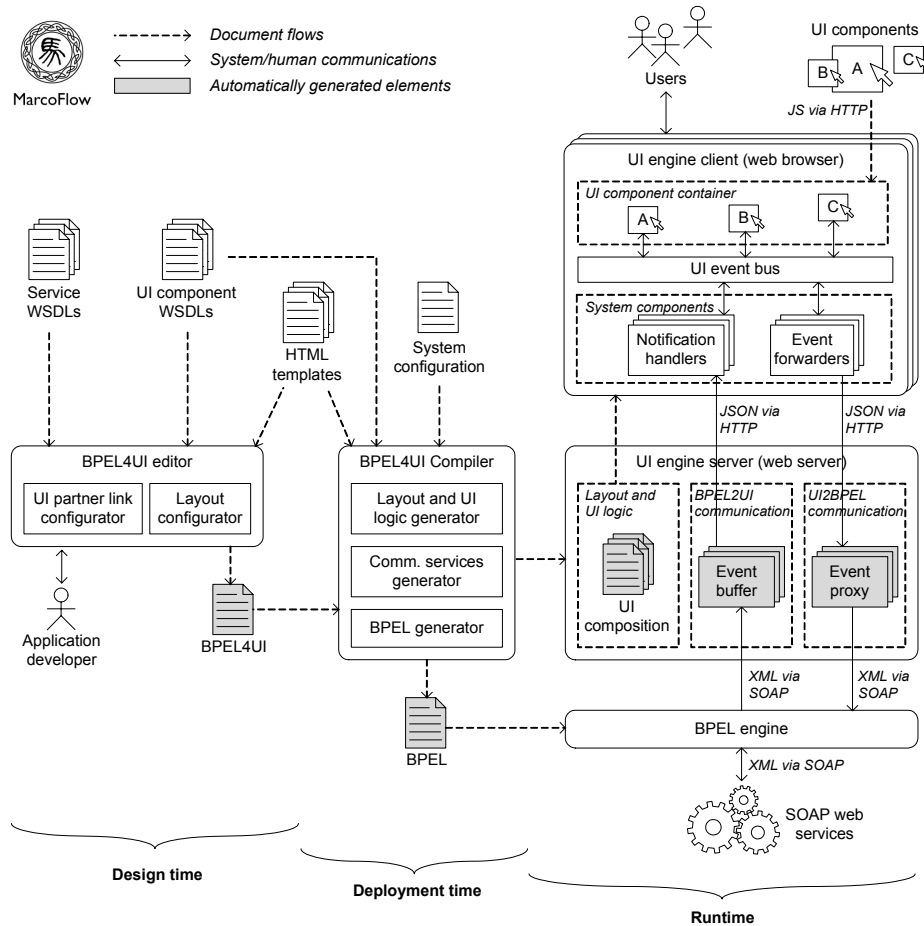
Figure 2 From design time to runtime: overall system architecture of MarcoFlow

3 The MarcoFlow Environment

Figure 2 shows the (simplified) architecture of the MarcoFlow environment, which aids the development and execution of distributed UI orchestrations. The architecture is partitioned into design time, deployment time, and runtime components, according to the three phases of the software development lifecycle supported by MarcoFlow.

The **design** part comprises the *BPEL4UI editor* that supports BPEL4UI [2], the composition language we use to specify distributed UI orchestrations. The editor is an extended Eclipse BPEL editor with (i) a panel for the specification of the pages in which UI components can be rendered and (ii) a property panel that allows the developer to configure the web pages, to set the properties of UI partner links, and to associate them to place holders in the layout.

The **deployment** of a UI orchestration requires translating the BPEL4UI specification into executable components: (i) a set of *communication channels* that mediate

between the UI components in the client browser and the BPEL engine; (ii) a *standard BPEL specification* containing the distributed UI synchronization and web service orchestration logic; and (iii) a set of *UI compositions* (one for each page of the application) containing the intra-page UI synchronizations. This task is achieved by the *BPEL4UI compiler*, which also manages the deployment of the generated artifacts in the respective runtime environments.

The **execution** of a UI orchestration requires the setup and coordination of three independent runtime environments: (i) the interaction with users and intra-page UI synchronization is managed in the client browser by an *event-based JavaScript runtime framework*; (ii) a so-called *UI engine server* runs the web services implementing the communication channels; and (iii) a *standard BPEL engine* manages the distributed UI synchronization and web service orchestration.

The MarcoFlow system shown in Figure 2 is fully implemented and running. A patent application for parts of the system has been filed.

4 Demo scenario

A example of how MarcoFlow can be used for the development of a distributed UI orchestration is available at *http://mashart.org/marcoflow/demo.htm*. The demo in form of a video illustrates in few minutes the main features of MarcoFlow in the context of a simple home assistance management application. Particular emphasis is given to the three development phases supported by the tool (design, deployment, and runtime) and to the use of the final application by the different actors involved in the distributed process logic.

References

1. J. Yu, B. Benatallah, F. Casati, F. Daniel. Understanding Mashup Development and its Differences with Traditional Integration. *IEEE Internet Computing*, Vol. 12, No. 5, September-October 2008, pp. 44-52.
2. F. Daniel, S. Soi, S. Tranquillini, F. Casati, C. Heng, L. Yan. From People to Services to UI: Distributed Orchestration of User Interfaces. *BPM'10*, Hoboken, NJ, USA.
3. OASIS. Web Services Business Process Execution Language Version 2.0, April 2007. [Online]. *http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html*
4. Active Endpoints Inc., Adobe Systems Inc., BEA Systems Inc., International Business Machines Corporation, Oracle Inc., SAP AG. WS-BPEL Extension for People (BPEL4People), Version 1.0. June 2007.
5. R. Acerbis, A. Bongio, M. Brambilla, S. Butti, S. Ceri, P. Fraternali. Web Applications Design and Development with WebML and WebRatio 5.0. *TOOLS'08*, pp. 392-411.
6. J. Gómez, A. Bia, A. Parraga. Tool Support for Model-Driven Development of Web Applications, *WISE'05*, pp. 721-730.
7. Sun Microsystems. JSR-000168 Portlet Specification, October 2003. [Online]. *http://jcp.org/aboutJava/communityprocess/final/jsr168/*
8. F. Daniel, F. Casati, B. Benatallah, M.-C. Shan. Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. *ER'09*, pp. 428-443.