RZ 3530 (# 99543) 01/19/04
Computer Science 6 pages

Research Report

Linking Workflow with Web Front End

Christian Hörtnagl

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
Email: hoe@zurich.ibm.com

Research
Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich

LINKING WORKFLOW WITH WEB FRONT END

Christian Hörtnagl[1]

Abstract
This hot spot paper presents a software system for enabling pervasive data gathering in electronic workflows, and explains also its wider utility for linking electronic workflows with common web pages.

1. Introduction

Electronic workflows tie together organizational resources for performing automated business processes. Their efficiency increases when resources can be mobilized as quickly as possible. For user-facing steps this means that data must be routed to key personnel without delay and that some participants should be reachable at all times and locations. For instance, a funding request may require approval by a line manager who is presently out-of-office, but who could be reached by mobile phone. If assistants have to place urgent voice calls, this may intrude uncomfortably on spare time, as far as the individual is concerned, and it also interrupts the automatic workflow, as far as overall business efficiency is concerned (lessens speed of execution, traceability, security, correctness).

Hence we envision a better solution, *formidable*, where users of mobile devices such as laptops, PDAs, and mobile phones can also participate in electronic workflows; it brings such devices into the normal scope of business process execution and puts them roughly on the same functional level as desktop computers. Specifically, the technical solution consists of middleware and tools, and allows user-facing workflow steps to be executed on mobile devices in a way that is simple to program and deploy, and straightforward to use. The increasing relative number of mobile devices over desktop PCs [7] and the strong importance of electronic workflows among business applications both combine to make this an important area for investigation.

Our general approach is to furnish a declarative web GUI for user-facing workflow steps, and to have this web front end actuated by appropriate middleware, as well as rendered by regular markup browsers on mobile devices. With the advent of XML for capturing semi-structured data, declarative programming models have received renewed attention as an alternative to programmatic ones. In the context of XML, we refer to declarative models as those whose (simple) logic is entirely encapsulated inside markup tags, not normal programming code. We recognize that declarative programming is not a cure-all for all problem sets; for instance, a programmatic Java client will probably fare better for RFID or biometric sensor reading on a mobile device than a declarative combination of

[1] IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland, hoe@zurich.ibm.com

extension tags and embedded scripts. We suggest that the trade-off between declarative and programmatic clients (GUIs) is in general determined by the kind of manipulated data and by the complexity of application-specific logic that must execute on a device.

With workflows, typical data are semi-structured, highly textual (e.g. ERP data), and therefore generally of the kind that is readily captured by XML; the complexity of required core logic is at the level of data gathering, and is well met by markup dialects such as HTML (forms) or XForms (Basic) [2]. We believe that a declarative web GUI is appropriate for user-facing workflow steps for the following combined reasons:

- Current mobile devices normally have built-in browsers for mobile internet access. Hence there is native support for markup manipulations such as rendering (including graphical adornment with normal (X)HTML and CSS), HTTP transport, declarative XSLT transformation, schema validation, and single sign-on. A solution that aligns with these techniques can reduce its footprint by leveraging functionality that sits in device ROMs. Markup-based clients are also easier to install (download as web pages) and advertise e.g. in UDDI registries.

- Current workflows are deployed as choreographed web services whose sequence is described with languages such as BPEL [1]. Since the meta-information and artifacts of web services (and BPEL instructions themselves) form XML documents, there exists a tooling opportunity for simplifying programming towards “drawing a web page” instead of “writing a program”. This is further explained in Section 3.

- Since it effectively combines application-specific markup data with generic, shrink-wrapped transformations, there is a tighter sandbox around the exposed GUI than for instance with Java Virtual Machines. Especially in the absence of markup scripts, manipulations are strictly constrained to GUI interactions, and hence there are fewer security exposures.

- Our solution is compatible with minimum markup capabilities (e.g. WAP browser), but also allows scaling up to alternative realizations (e.g. self-contained XForms for offline and multi-modal operation). In this way it accounts for heterogeneous platforms and incremental technology evolutions in the mobile device space. And as a consequence, normal web pages also qualify as potential clients and triggers for workflows (see Figure 1).

The remainder of the paper is organized as follows. In Section 2, we introduce an example scenario by way of motivation and illustration. Section 3 describes a tentative solution architecture, and Section 4 covers related work. Finally, we indicate future work and draw some conclusions in Section 5.

2. Scenario

Figure 1 presents a representative example scenario: employees seeking hotel and flight information on a travel web site can launch a related business trip workflow by simply pressing a button (i.e. traversing a hyperlink). Multiple web services and human agents

participate in the ensuing business process: among those inside the employees' own organizational domain, a registration service manages trip requests and authorizations; accounting checks statements and issues final reimbursements; managers give approval, and employees again collect and report expense information during or after their trips.

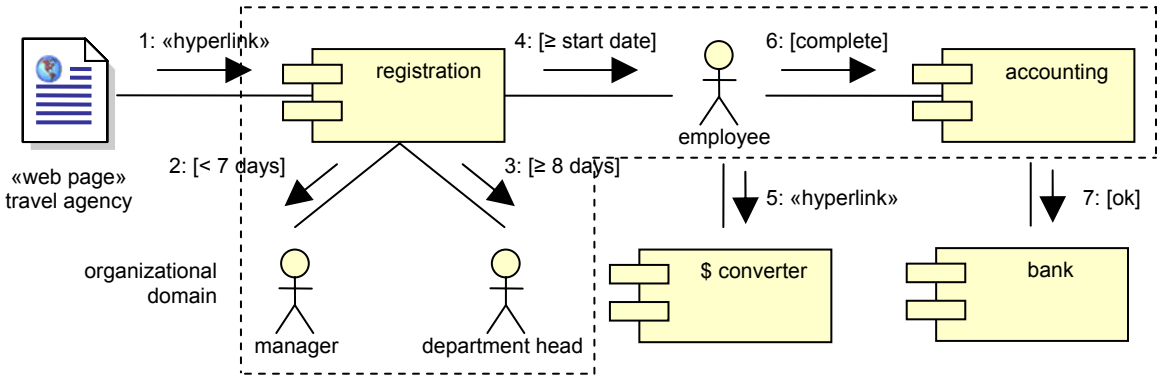

Figure 1 Business trip scenario.

The scenario combines web pages and web services in multiple different ways: web services (drawn as UML components) are workflows' building blocks, and workflows can be triggered from web pages. Web pages (especially their contained forms) provide user interfaces for human actors (individuals or teams), and web services can be accessed from inside those web pages to do form completion and validation tasks (e.g. currency conversion). During offline operation, some of these tasks may be automatically deferred.

The given scenario assumes that actors will be able to use a variety of devices, including mobile ones, and a variety of (wireless) connectivity options. For instance, traveling employees may fill in expense forms on their PDAs already while being on-the-road and while working offline: as part of the desired workflow integration, appropriate forms are automatically provided and input data are forwarded when their devices reconnect to a web portal, receive e-mail, do on-demand synchronization, or by other configurable means. All interactions must be under role-based access control, transactional, and secure across organizational domains [6].

3. Architecture

Our tentative solution architecture comprises a runtime component, *formidable-dock* (shown on the left in Figure 2), and a tooling component, *formidable-tools* (shown on the right). Both share access to a single data store, which is shown in the middle.

The desired flexibility regarding different technology choices is manifest in a number of multiplied sub-components each: the runtime component acknowledges different transport mechanism between itself (it is deployed in the backend) and client devices (e.g. HTTP, SMTP, SyncML); the tooling component acknowledges different configurable artifacts as sources (e.g. BPEL, WSDL, UML diagrams similar to Figure 1), markup technologies as targets (e.g. XForms inside XHTML, HTML), user directories (e.g. LDAP), and stores (e.g. DB2 relational database). Sub-components are designed as Enterprise Java Beans (EJBs) and Eclipse plug-ins [4] respectively; none of them need to

be deployed on eligible clients. Next to this static decomposition, flexibility is also achieved by using multiple dynamic XSLT transformations in the system core, hence the reference to declarative programming in Section 1.

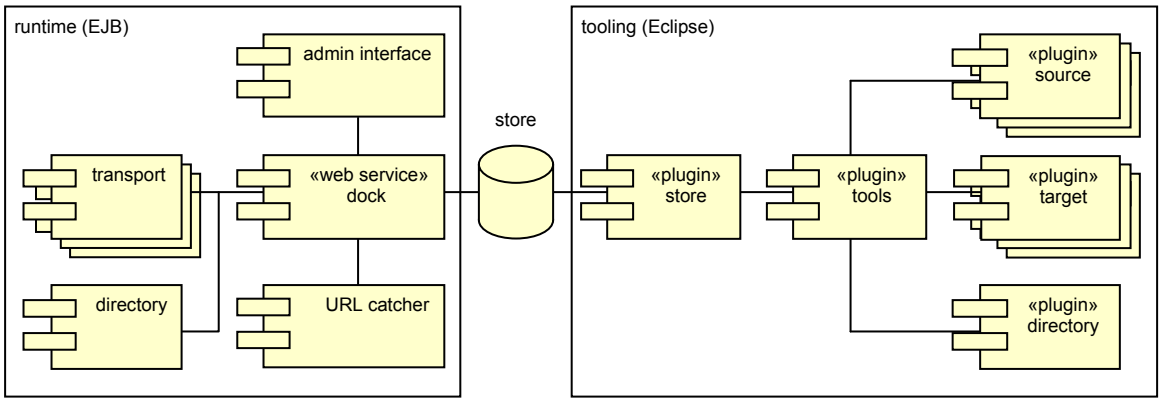

Figure 2 Conceptual architecture.

The tooling component accounts for the fact that forms can be derived in a semi-automatic process from web service descriptions, because data that travel inside SOAP messages, and forms' data models are naturally related by reference to XML schemas that can be linked in a deterministic way. For instance, data that arrive from the registration service in Figure 1 are isomorphic to what then goes into the forms that employees fill in (default values and labels); and the document that results from electronic forms completion aligns with what is later forwarded to the accounting service.

Our designed programming tools do not store verbatim forms for use by the runtime component, but rather instructions for how to obtain such forms from data instances. These instructions are in the form of transformations, and are managed in parallel for all configured targets; the transformations are reasonably similar in structure to the desired forms, and the additional level of indirection allows keeping algorithmic parts (both in tooling and runtime) uniform across all targets. Specific tools make sure that data models (and hence transformations) remain faithful to the interfaces specified by adjacent web services. To that end, tools first generate transformations from web service descriptions (two descriptions in case of user-facing workflow steps; one in case of isolated web services). In a second step, target-specific visual editors allow optional refinement of the exact visual markup representation, and they also leave specific data models intact; developers only manipulate widgets as representations for markup, and are not exposed to any details pertaining to the underlying transformations. When a developer requires arrangements that cannot be rendered with a given target, then this target is transparently excluded from the scope of the given form.

The core piece of the runtime component offers a docking interface as web service, and hence it can be referenced in arbitrary workflows: this is how user-facing workflow steps can be specified in canonical BPEL. The dock changes its WSDL service description and behavior depending on which forms are currently marked as active in the store (it also accepts `xsd:any` and special operations for communicating error conditions at all times).

Future developments of the BPEL standard may lead to tighter integration between the dock and workflow execution engines.

The following comprises a high-level view of what happens when a SOAP message arrives at the dock as part of a given workflow execution: the dock removes the SOAP envelope to expose contained data, and looks up stored transformations, depending on the exposed schema (namespace) and required targets. Currently, we apply all configured targets and transports in parallel (e.g. send forms by e-mail and upload them to portal), although in the future we want to take user context information into better account, for determining exact device characteristics (target), and for guidance on how to route forms to individuals (transport). After successful lookup, transformations are applied to bag data inside target forms and prepare them for (secure) transport.

The tools also associate each stored form with a user role (or with XPath instructions for how to extract a user role from arriving data). The dock consults the configured directory to find out transport-specific routing parameters, such as user credentials for accessing a download portal, or e-mail addresses. As a security precaution, there is also a fixed parent role constraining what values the role resolution may yield (e.g. required organizational unit). A guard condition, which is also stored with each form, determines at which time data are routed onwards (e.g. either after filled-in forms return for the first time via any configured transport, after a timeout, or when a conditional expression holds). Before actual forwarding, further manipulations extract filled-in data from returned forms and attach new envelopes.

In addition to the dock and technology-specific plug-ins, the runtime component also features an administrative interface (e.g. for resolving routing errors), and a URL catcher. The second manages a pool of URLs whose requests trigger actions such as workflow initiation or complex form completion for target technologies with relevant constraints (e.g. lack of SOAP).

4. Related work

This work relates to past assessments of declarative vs. programmatic programming models. For the problem at hand, we have concluded that given the meta-information (e.g. description) and artifacts belonging to participating components (e.g. data that are exchanged between them), a declarative GUI and programming model that is based on XML transformations is feasible and advantageous. This work also relates to (markup) transcoding [5] techniques for accommodating different device characteristics, although we transform tags in models and controls, and not images nor rendering instructions in views (e.g. XForms used as target, not source).

Our approach shares some general characteristics with a commercial solution from Microsoft in combining workflow and markup rendering: InfoPath uses a single proprietary format with Microsoft Office as its current rendering engine, whereas we incorporate an open set of industry standards, and employ web browsers (non-exclusively). By delegating rendering and related functions to existing web infrastructure, our workflow handling solution can specifically encompass mobile devices, legacy web

pages, and a defined upgrade paths to richer user interfaces, applicable as platform capabilities evolve (e.g. multi-modality and voice browsers; graphics animation in SVG).

The evolution of web services sees an ongoing debate over REST-full [3] and REST-less architectural styles. The first e.g. emphasize independence and scalability, and are embodied by HTTP. The second e.g. emphasize remote, typed calls with XML serialization, and are embodied by conventional use of SOAP. This solution can be seen as an instance that combines aspects of both styles (the first for web-based GUI; the second for backend integration; URL catcher acts partly as REST gateway), with benefits from use of common XML artifacts.

5. Conclusions and outlook

In this paper we have presented a novel means for integrating workflow and the web that has distinct advantages in terms of ease-of-deployment and ease-of-use. We have argued that the GUI requirements of electronic workflows, especially in relation to mobile devices, are well met by declarative markup forms, and have presented a solution architecture that accounts for this both in terms of tooling and runtime support, without introducing strong dependency on a particular markup technology.

Near-term future work will consist of refining the design and completing a software prototype. For longer-term future research, we foresee opportunities for context-based data routing based on user presence information, combinations with single-sign-on security, and multi-modal user input (e.g. voice-enabled XForms). There are also opportunities for adding native GUI-related semantics to the current BPEL standard.

Enabling markup forms as GUI for use in workflows offers an attractive end-to-end solution that consistently relies on markup technologies: XML dialects now capture the meta-information and artifacts of participating workflows and web service, and much application data; they also frame SOAP messages and GUI rendering instructions. A declarative programming model that mostly relies on markup transformations serves well to tie these pieces into a conceptually simple solution that allows for efficient tooling, and exhibits a consistent web front end for workflows that can also reach mobile devices.

6. References

[1] ANDREWS, T. et al., Specification: Business Process Execution Language for Web Services, Version 1.1, 5 May 2003.

[2] DUBINKO, M., RAMAN, T. V., XForms 1.0 Basic Profile, W3C Candidate Recommendation, 14 October 2003.

[3] FIELDING, R. T., Architectural Styles and the Design of Network-Based Software Architectures, PhD Thesis, University of California, Irvine, 2000.

[4] GAMMA, E., BECK K., Contributing to Eclipse: Principles, Patterns, and Plug-Ins, Addison-Wesley, 2004.

[5] MOHAN, R., SMITH, J. R, LI, C.-S., Adapting Multimedia Internet Content for Universal Access, IEEE Transactions on Multimedia, March 1999.

[6] RAMAN, B. et al., The SAHARA Model for Service Composition across Multiple Providers, International Conference on Pervasive Computing (Pervasive 2002), Zurich, Switzerland, August 2002.

[7] Why mobile is different, The Economist, Oct 13th 2001.