Supporting User-Centric Business Processes with WADE (Extended Abstract)

Federico Bergenti
Università degli Studi di Parma
43124, Parma, Italy
federico.bergenti@unipr.it

Giovanni Caire, Danilo Gotta
Telecom Italia S.p.A.
10148, Torino, Italy
{giovanni.caire,danilo.gotta}@telecomitalia.it

ABSTRACT

In this paper we present the latest developments of *WADE (Workflows and Agents Development Environment)* that provide concrete support for a better realization of the innovative paradigm of *agent-based BPM (Business Process Management)*. We discuss the new functionality that WADE offers to enable the rapid and effective realization of *user-centric business processes*, i.e., business processes that are tightly integrated with the work of users and that are mainly driven by user interactions. Such processes are met frequently in practice and WADE seamlessly accommodates Web and Android users by means of dedicated views.

Categories and Subject Descriptors

I.2.11 **[Computing Methodologies]**: Distributed Artificial Intelligence – multiagent systems, languages and structures, coherence and coordination

General Terms

Management, Design, Languages

Keywords

Agent-based business process management, user-centric business processes, WADE

1. INTRODUCTION

The extensive use of WADE in mission-critical applications (see the concluding section and [1] for some examples) has witnessed the notable importance of user interactions in the scope of workflows. This is not surprising and we acknowledge that the idea of workflows has its origins in the management of the work of people. Nonetheless, we believe that the common approach of treating user interactions as *yet another type of event* does not adequately capture the importance and the high frequency of them.

So called *user-centric workflows* are introduced in WADE version 3.0 as a means to capture workflows that *(i)* frequently need to interact with users, and *(ii)* are mainly intended to gather information and provide feedback to users. WADE now lifts user interactions to a higher level and it provides specific tools and features to manage them effectively. The design guidelines for such a recent development of WADE are as follows:

- The description of the information to provide to users and the related input to acquire from users must be independent of the device that the user is concretely accessing;
- Any element of such a description must be extensible in order to let developers provide more specific descriptions of both input and output information;
- The software application that the user accesses must be replaceable by any custom application, once the communication with the WADE platform is correctly set up; and
- No device is privileged and developers must be able to describe workflows in full generality, if they really want.

From such very generic guidelines we could easily choose the *Model-View-Controller (MVC)* [2] architectural design pattern as the coarse grained model around which we designed the new *interactivity package* of WADE. The new WADE interactivity package provides the Java classes of the model of interactions (see Section 2.1) and a number of *visualizers* (see Section 2.2) intended to be integrated in the application shipped to users.

2. WADE USER-CENTRIC WORKFLOWS

In order to fully exploit the power of user-centric business processes, the developer of a workflow should first inform WADE that the workflow itself needs to interact with users. This is accomplished by realizing a workflow class that extends the `InteractiveWorkflow` class rather than the common `Workflow` class. Such an `InteractiveWorkflow` class is a specific subclass of `Workflow` that provides the needed machinery to link a workflow instance to a visualizer. WADE ensures a one-to-one correspondence between a user and an instance of an `InteractiveWorkflow`, and therefore an `InteractiveWorkflow` has just one user at a time.

When an `InteractiveWorkflow` is connected to a visualizer, it is requested to provide the visualizer with a description of the information to present to the user and with a related description of the possible user inputs. Such a mechanism is concretely driven by the workflow developer who can freely use the new method `interact()` that `InteractiveWorkflow` provides. Such a method is supplied with an `Interaction` object that contains the following parts: *(i)* an abstract description of the information to be presented to the user with some abstract requirements on the way information is presented, e.g., by indicating how a set of

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4–8 June 2012, Valencia, Spain.

labels should be aligned on the device screen; *(ii)* an abstract description of the information that the user is allowed to return in a response; *(iii)* an abstract description of the constraints that the user response must meet to be considered valid; and *(iv)* a list of possible abstract actions that the user is allowed to choose as valid responses.

Upon executing the `interact()` method, the workflows is put into a *SUSPENDED* state to allow the corresponding visualizer to present the information to the user and to enable the user to provide feedback by means of one of the available response actions. The visualizer is on duty for showing the information in the best possible way and for allowing the user to provide its response. The visualizer is also responsible for the correctness of the provided response because it is in charge of checking the constraints that identify valid responses.

Once the user has validly compiled its response and chosen one of the available response actions, the visualizer returns user response to the workflow instance in terms of a copy of the original `Interaction` object that now contains relevant user input and from which the developer can extract the user response easily. Such an approach allows developers retrieving response information from where they originally decided they should be contained. Moreover, it ensures no redundant information is sent back in responses.

2.1 A Model of Interactions

In the WADE nomenclature, an interaction is both an abstract description of the information to be provided to users and a means to allow users constructing responses. Therefore, WADE provides a set of Java classes that are used to describe interactions with such a dual meaning. Such classes are designed using the standard approach adopted in modern user interfaces and they are structured in a containment tree. They are divided into the following major groups:

- Passive elements, e.g., labels and pictures, that are leafs of the containment tree intended to describe the information to be provided to users;
- Information elements, e.g., text areas and menus of various types, that are leafs of the containment tree and that are meant to provide the user with a means to provide responses;
- Containers, e.g., list and grid panels, that are designed to aggregate a group of children in order to describe their relative position in an abstract manner;
- Actions, that describe the types of responses the user can select; and
- Constraints, that concretely provide check procedures to ensure the correctness of responses.

With the notable exception of constraints, all such Java classes are purely descriptive and they are simple containers for information flowing between an `InteractiveWorkflow` and a visualizer. They are designed to maintain the clear separation of concerns of the MVC design pattern.

All such classes describe the model of an interaction, while the relative controller is implemented by the adopted visualizer, which also generates on the fly the relative view. Such an approach ensures, among other things, that developers are free to add new visualizers and that no visualizer is privileged.

Constraints are peculiar in the scope of the MVC pattern because they are intended to validate input. They represent a pluggable part of the controller because they are responsible for updating the view upon changes in the model, e.g., by marking invalid components with an error notification. WADE provides a set of general purpose constraints that can be used, e.g., to make sure a mandatory menu has at least one item selected or to warrantee that the text in a text field conforms to a given regular expression.

2.2 Available Visualizers

At the time of writing WADE provides two visualizers meant to accommodate two important classes of users: Web users and Android users. Web users are allowed to activate new interactive workflows and to connect to suspended workflows by means of a dedicated visualizer developed using the ZK toolkit [4]. ZK is a very popular toolkit to develop AJAX applications in Java and it is easily interfaced with WADE. The ZK visualizer instantiates one JADE agent on the server side of the Web application for each and every Web session, and it ensures agents are properly connected with the WADE platform. The client side of the ZK application is meant to: *(i)* present information to the user; *(ii)* provide selectable actions in terms of buttons; and *(iii)* ensure constraints are met before passing any response to workflow agents. The chosen approach ensures a lightweight client that is only in charge of realizing the user interface on the fly and of validating constraints. ZK provides a proprietary communication mechanism between the client browser and the server side of the application which is completely hidden in the deep internals of ZK, thus becoming transparent to developers.

The Android visualizer is developed along the lines of the ZK visualizer and we ensured that the internals of the two visualizers are designed using the same architecture and adopting closely related approaches. The major difference with the ZK visualizer is that the Android visualizer is a single Android application that hosts: *(i)* a JADE container in *split mode* (see JADE documentation for details [3]) which is created in the scope of the WADE platform; *(ii)* the agent needed to connect the user with the workflow; and *(iii)* the visual components that are used to dynamically assemble and render the user interface. No proprietary communication mechanism is needed in this case because the agent and the visual components share some memory of the Android device.

3. REFERENCES

[1] Caire, G., Gotta, D., and Banzi, M. 2008. WADE: A Software Platform to Develop Mission Critical Applications Exploiting Agents and Workflows. In *Proc. 7th Int'l Conf. Autonomous Agents and Multiagent Systems*, 29-36.

[2] Fowler, M. 2003. *Patterns of Enterprise Application Architecture*, Addison-Wesley.

[3] JADE – Java Agent Development framework. Available at http://jade.tilab.com

[4] ZK. Available at http://www.zkoss.org