Expert Systems With Applications 71 (2017) 125–137

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Automated reasoning based user interface

Paweł Kapłański[a,b], Alessandro Seganti[a,*], Krzysztof Cieśliński[a], Aleksandra Chrabrowa[a], Iwona Ługowska[c]

[a] Cognitum, Warsaw, Poland
[b] Gdansk University of Technology, Gdansk, Poland
[c] Maria Sklodowska-Curie Memorial Cancer Center and Institute of Oncology in Warsaw, Warsaw, Poland

ARTICLE INFO

Article history:
Received 29 April 2016
Revised 20 November 2016
Accepted 22 November 2016
Available online 23 November 2016

Keywords:
Model view controller
Reasoning
Semantic web
OWL/RDF
Model- driven engineering
Decision support system
User interface

ABSTRACT

Motivation: The ability to directly trace how requirements are implemented in a software system is crucial in domains that require a high level of trust (e.g. medicine, law, crisis management). This paper describes an approach that allows a high level of traceability to be achieved with model-driven engineering supported by automated reasoning. The paper gives an introduction to the novel, automated user interface synthesis in which a set of requirements is automatically translated into a working application. It is presented as a generalization of the current state of the art model-driven approaches both from the conceptual perspective as well as the concrete implementation is discussed together with its advantages like the alignment of business logic with the application and ease of adaptability. It also presents how a high level of traceability can be obtained if runtime support of automated reasoning over models is applied.

Results: We have defined the Automated Reasoning-Based User Interface (ARBUI) approach and implemented a framework for application programming that follows our definition. The framework, called Semantic MVC, is based on model-driven engineering principles enhanced with W3C standards for the semantic web. We will present the general architecture and main ideas underlying our approach and framework. Finally, we will present a practical application of the Semantic MVC that we created in the medical domain as a Clinical Decision Support System for GIST cancer in cooperation with the Maria Sklodowska-Curie Memorial Cancer Center and Institute of Oncology in Warsaw. The discussed expert system allows the expert to directly modify the executable knowledge on the fly, making the overall system cost effective.

1. Introduction

The development and maintenance of User Interfaces (UI) is an expensive process that is crucial for most of the businesses. Furthermore, with the spread of new interaction techniques such as touch or vocal input makes the implementation of a modern system UI becomes a critical part of the application lifecycle. Moreover, the ability to directly trace how the requirements map to the software implementation is crucial for domains that require a high-level traceability (e.g. medicine, law, crisis management). Furthermore, all modern system should be simple not only to create but also to modify following many change-requests.

Model-Driven Engineering (MDE) answers to these questions. If we see a software system as a realization of an abstract model, then MDE can be understood as the way to transform the abstract model to the actual implementation of the system.

In this article we present, developed by us, Automatic Reasoning Based User Interface (ARBUI) approach together with its implementation: the Semantic MVC. We show how we implemented the ARBUI approach and we give an example of a practical use where the Semantic MVC framework has been useful: the Clinical Decision Support System for Gist Cancer (GIST-CDSS).

1.1. Methodology

Selected research methodology combines Literature Review and Pilot Study with Action Research (AR), that is intended to explore improvements of the specific process. AR is a cycle of actions (Susman & Evered, 1978) presented in Fig. 1.

* Corresponding author.
E-mail addresses: pawel.kaplanski@cognitum.eu (P. Kapłański), a.seganti@cognitum.eu (A. Seganti), k.cieslinski@cognitum.eu (K. Cieśliński), a.klimek@cognitum.eu (A. Chrabrowa), iwonalugowska@coi.waw.pl (I. Ługowska).

http://dx.doi.org/10.1016/j.eswa.2016.11.033
0957-4174/

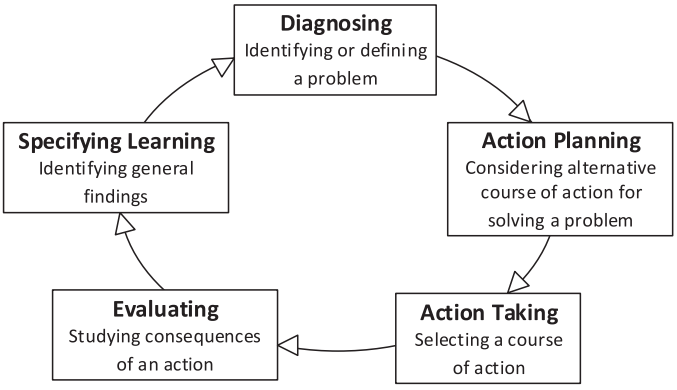

Fig. 1. Action Research Cycle.

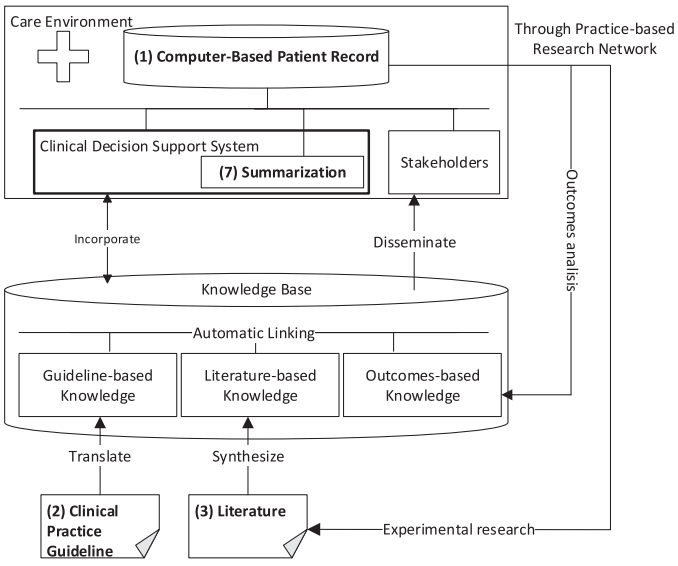

Fig. 2. Knowledge and functionality involved in the use of CDSSs to support evidence-based medicine (after Sim et al., 2001).

We followed all the steps of AR and in this article we present results of our research.

Diagnosis. The diagnosed problem lies in the implementation of specific Clinical Decision Support System for Gist Cancer (GIST-CDSS) as a part of a pilot study devoted to Gastrointestinal Stromal Tumors (GIST). Published in literature, systematically developed statements designed to assist medical practitioners and patients with decisions about appropriate health care for specific clinical circumstances (Field & Lohr, 1990) are known as Clinical Practice Guidelines. Because they are very formal, the automation of a decision support can be implemented, and the computer can make use of patients' clinical data, follow its own algorithm, and present the information relevant to the current clinical situation (Stahl, Rouse, Ko, & Niland, 2004). In other words, basing on the guidelines automated deductive reasoning tools helps a therapist to provide evidence-based diagnosis(4) that is logically followed by a (6) therapy (see Fig. 2). Moreover, clinical guidelines are being updated each causing cost-prone maintainability issue of the CDSS.

Action Planning. To address the problem, we performed Literature Review (see Section 3) searching for common UI practices. We created a conceptual model of UI infrastructure (see Section 2), that helped us to understand major differences between existing UI architectures.

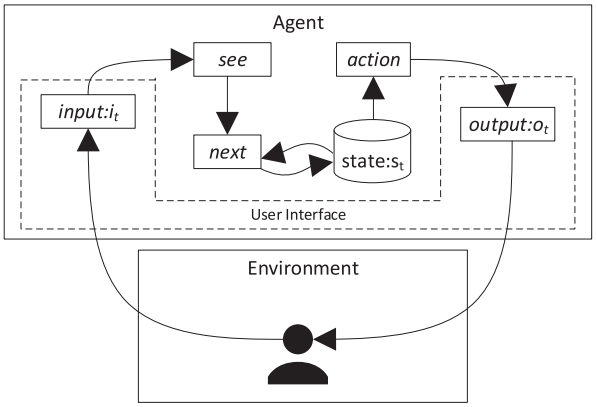

Fig. 3. An agent with its internal state.

Action Taking. Review of the literature led to the conclusion that we need to generalize existing architectures, resulting in the development of ARBUI architecture (see Section 4). The aforementioned conceptual model allowed us to prove that ARBUI cannot be reduced to any of existing architectures.

Evaluating. We implemented ARBUI architecture (see Section 6) and ultimately deployed it as a GIST-CDSS.

Specifying Learning. Discussion and lessons learned are presented in Section 7.

2. Conceptual model of user interface

This section presents a conceptual model of User Interface (UI) that helped us focus on the main differences between different types of UI architectures, ultimately proving that ARBUI approach is a generalization of currently available, related, state of the art approaches.

Let's consider a reactive agent as a general model of a computer program. It can maintain its state Woolridge and Wooldridge (2001) via an internal stand-alone data structure, and interact with its environment that in our case is the user of UI (Fig. 3). The perception of an agent is realized by its *see* function, that forwards the percept to the *next* function changing the internal state of the agent. The agent control loop is defined in the following way:

1. Start with the initial internal state $s \leftarrow s_0$.
2. Observe the environment state e generated by the user with UI *input* function $e \leftarrow input(i)$, and generate a percept $p \leftarrow see(e)$.
3. Update the internal state via the *next* function $s \leftarrow next(s, p)$.
4. Select an action via the *action* function $a \leftarrow action(s)$, and display an output of UI back to the environment (the user) $o \leftarrow output(a)$.
5. GOTO 2.

We can represent the control loop as a pair of dynamic equations (1), where the symbol: $\overleftarrow{user}$ represents the interaction of the user with the given agent.

$$\begin{aligned} s_{t+1} &= next(s_t, see(input(i_t))) \\ o_{t+1} &= output(action(s_{t+1}))) \\ i_{t+1} &\overleftarrow{user} o_{t+1} \end{aligned} \tag{1}$$

Let's define two functions: (a) *control* that combines *next, see* and *input*, and (b) *view* combining *action* and *output* in the following ways (2):

$$\begin{aligned} next(s_t, see(input(i_t))) &= control(s_t, i_t) \\ \text{output}(action(s_{t+1})) &= view(s_{t+1}) \end{aligned} \tag{2}$$