Aspect-Oriented Engines for Kroki Models Execution

Milorad Filipović, Sebastijan Kaplar, Renata Vaderna, Željko Ivković, Gordana Milosavljević, Igor Dejanović

Faculty of Technical Sciences, Novi Sad, Serbia
{mfili, skaplar, vrenata, zeljkoi, grist, igord}@uns.ac.rs

***Abstract* – The paper presents an overview of techniques and mechanisms implemented in generic web and desktop engines that enable execution of application prototypes being specified by our Kroki tool. Unlike most other solutions where only a user interface skeleton is executable, Kroki's specifications can be tested through all three application tiers – the user interface, the business logic, and the database. Kroki is a mockup-driven tool that enables development of enterprise information systems based on participatory design. Since immediate execution is always possible, it can significantly contribute to decreasing a communication gap between the development team and users.**

I. INTRODUCTION

Kroki [1, 17, 18] is a rapid prototyping tool that enables users and developers to be concurrently engaged in the development of enterprise information systems. Kroki enables requirements elicitation based on executable prototypes, using the terms that are familiar to the end users - by enabling them to draw the user interface (UI) mockups. Contrary to the approaches where mockups are created by general-purpose drawing tools and then manually or semi-automatically transformed to formal models (which are prone to errors and can lead to information loss), mockups created by Kroki are already elements of the UI model. Kroki's mockup editor actually implements a concrete syntax of our EUIS (Enterprise User Interface Specification) DSL [13] for specifying UIs of enterprise applications at a high-level of abstraction. EUIS DSL also has a textual syntax implemented by Kroki's command window and a UML-like concrete syntax implemented by Kroki's UML lightweight editor (Figure 5) [18]. EUIS DSL supports specification of several types of forms and panels and their elements, where the corresponding layout and functionality are defined by our user interface guidelines.

In order to reduce waste of time and effort, a special attention is paid to the option of reusing artifacts across development phases. The reuse is supported by exporting class diagrams and application prototypes to general purpose modeling and programming tools, and by importing models from general-purpose modeling tools (Figure 1). Thus, a created prototype can be used for requirements elicitation and can also evolve to the final enterprise application using the preferred toolchain (currently supported target language is Java).

Kroki enables hands-on prototype evaluation based on executable engines that can be activated almost instantaneously at any given moment during the development phase. This helps narrow the gap between the user specification and the finished product by iterative and online evaluation based on a real working system. According to the principles of

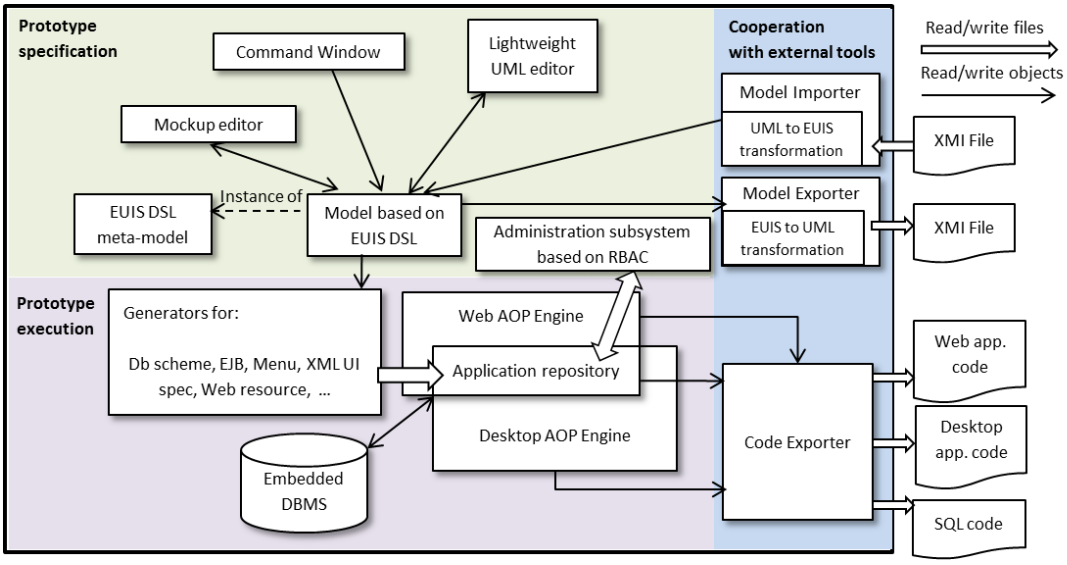

Figure 1. Kroki tool architecture

agile development, information gathering is most effective if it is based on something that works. Unlike most other solutions where only the UI skeleton is executable, Kroki's executable mockups can be tested through all three application tiers – the user interface, the business logic, and the database.

Generic enterprise engines can adapt their functionalities to each specified prototype on the fly. The basis of this adaptable behavior is the configuration data stored in the application repository (Figure 2) [14]. The application repository is a file directory that contains configuration files that provide information about the developed prototype that the generic engines need in order to obtain functions and look defined in Kroki editors. When the user chooses to execute the specified prototype, Kroki generates an application repository specific to the prototype and runs desired (web or desktop) generic enterprise application.

Figure 1 shows the architecture of the Kroki tool. As can be seen, two main parts of the architecture are the Prototype specification and the Prototype execution modules. The paper gives an overview of the prototype executions modules, primarily focusing on the web AOP engine and the application repository. More details about Kroki architecture can be found in [1].

The paper is structured as follows. Section 2 reviews the related work. Section 3 provides a detailed overview of Kroki's application repository with its static and generated parts. Basic mechanisms and principles of generic web engine used for prototype execution are given in Section 4. Section 5 provides additional options for extending built-in engine functionalities. Section 6 gives some final thoughts on the subject of the paper.

II. Related Work

This review of the related work deals primarily with the problems of mockup-driven development and applying aspect-oriented programming in web development.

The generic nature of the developed engine along with its need for adaptiveness leads to a lot of design challenges that make the standard object-oriented and model-driven approaches insufficient and error prone [3]. The shortcomings of traditional approaches are especially emphasized in the design of modern enterprise web applications which are expected to provide a rich user interface and high performance by default [3, 5]. In order to challenge those problems, aspect-oriented programming methods are incorporated into web application development more often than before [4, 5].

The benefits of AOP approach can be seen in [3] and [4], where the greater attention has been dedicated to design techniques of adaptive, context-aware web applications and the performance of the final product. Our approach, which is presented in this paper, follows these basic principles, but presents AOP web engine as a basis for model execution.

The main motivation for developing a generic web engine (in contrast to the standard methods which rely on code generators) was to increase interest in

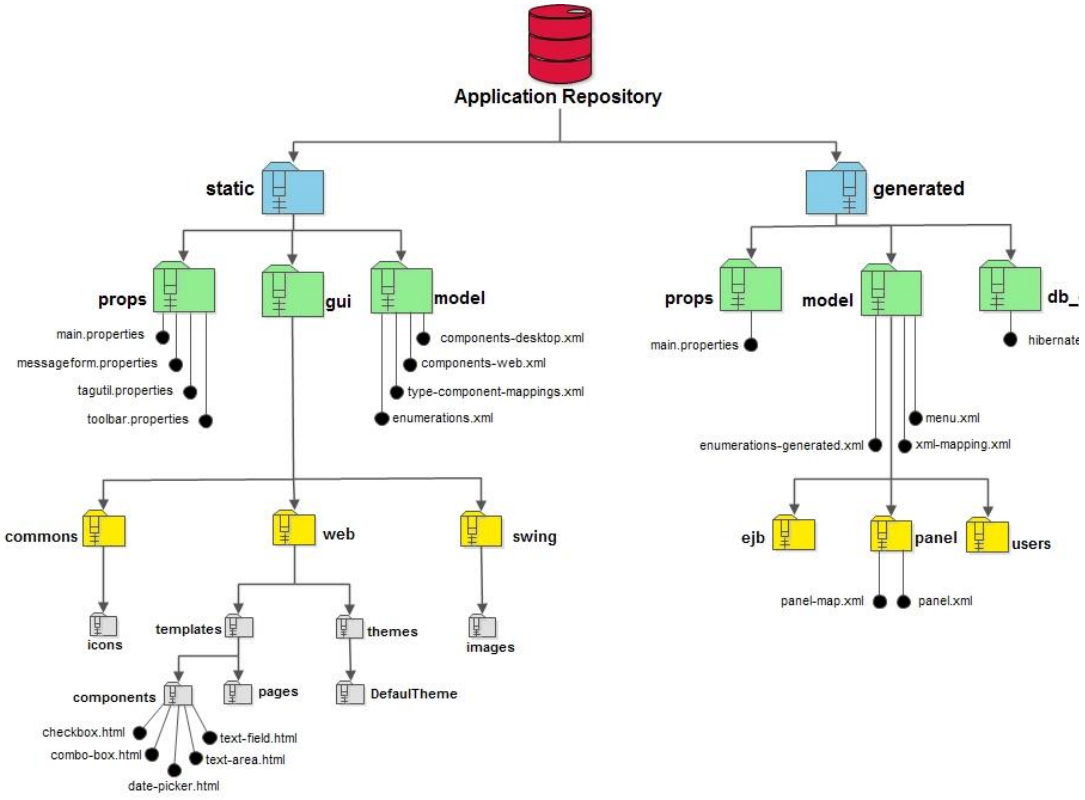

Figure 2. Application repository structure

adoption of agile development methods in web design as presented in [4, 5]. One of the most comprehensive solutions in the field of prototype-driven development is Umple tool, most notably its UIGU generation extension [7]. A slightly different approach is presented in [8] which presents window/event diagrams (WED) as reusable specification artifacts. WEDs combine UI mockups and state diagrams that enrich the UI specification with a prototype which has some basic functionalities. A large body of research deals with digitalization of hand-drawn UI mockups using shape recognition algorithms [8, 9, 10, 11, 12]. Unlike some of the solutions presented here where only the user interface skeleton is executable, Kroki's executable mockups can be tested through all three application tiers.

III. Application Repository

The application repository stores configuration data which is the basis for adaptive behavior of Kroki's generic engines. These data specify various parts of the enterprise system and their relations, look-and-feel resources (graphic icons, CSS, and HTML templates etc.), and other configuration artifacts needed for configuration of all generic engine layers.

The application repository structure is shown in Figure 2. It is composed of static and generated parts.

A static part of the repository contains general data that is independent of the concrete specification and as such is always the same (configuration files needed for engine core functionalities, look-and-feel artifacts for web and desktop engines, etc.). Main directories in this part of the repository are:

- **props** - Contains properties files with settings and string resources for web and desktop applications.
- **model** - Contains XML specifications of static parts of the engines. These static specifications mainly deal with the process of mapping programming language types to concrete GUI components
- **gui** - Contains look-and-feel resources for both web and desktop generic applications.

A generated part of the repository is created by Kroki generators and contains data about currently specified application prototype. Although the engine could take this data directly from Kroki model, we choose XML files as an intermediate step in order to provide independent functioning of the specified applications (after deployment). It's structure resembles the structure of the static part. The main difference is the lack of gui subdirectory which is due to enterprise systems using the same UI guidelines used as a basis for EUIS DSL specification [13]. Apart from the configuration files, the concrete EJB classes are being generated directly to the engine source code directory. Hence, they are not part of the application repository.

Main subdirectories of this part of application repository are:

- **props** - provides additional information that supplements the static properties with the data specific to the current specification (such as application name and description).
- **db_config** - contains the hibernate.cfg.ml file used by generic engines to configure the database connection used in the prototype execution phase. During the specification, each engine can be configured to use an existing database or to run embedded test database.
- **model** - contains XML descriptions of enterprise application elements organized in the following files and subdirectories:
    - **ejb** - contains XML specifications of EJB entities used in Kroki project. One XML file is generated for each EJB entity.
    - **panel** - contains XML definitions of standardized panels specified in Kroki project and mapping information (with which EJB entities the panel is associated with)
    - **enumerations-generated.xml** - XML specification of enumerations specified in the Kroki tool.
    - **menu.xml** - XML specification of the application's main menu. The structure of this menu reflects the structure of the packages and forms contained by the Kroki project, but can be overridden by Kroki's administration subsystem. Every user group can have their own main menu.
    - **xml-mapping.xml** - specifies which EJB class is associated with which XML description file in ejb subdirectory.
- **users** - contains XML description files for user rights administration module.

IV. Generic Web Engine

Kroki web engine is a generic web application developed in Java that adapts its look and behavior according to the configuration data stored in the application repository. This section presents its architecture in order to provide detailed insight into the engine's inner mechanisms.

Figure 3 shows conceptual architecture of the web engine. Upper half of the figure shows basic modules for initial data collection which are used by both the desktop and web engine. Lower part (with a blue background) displays web-specific architecture. As can be seen, the two parts are loosely coupled, so just the presentation layer is technology dependent. This section will cover some of the basic mechanism for obtaining executable prototype from Kroki

specification with the focus on the web engine. This presented explanation features as less as possible web-specific details, so it can be used to comprehend also Kroki's generic desktop engine since it lies on the same foundations.

is equivalent to the main form with the main menu in the desktop enterprise applications. It shows a page with a main menu from which the user can activate desired form associated with a specific enterprise entity.

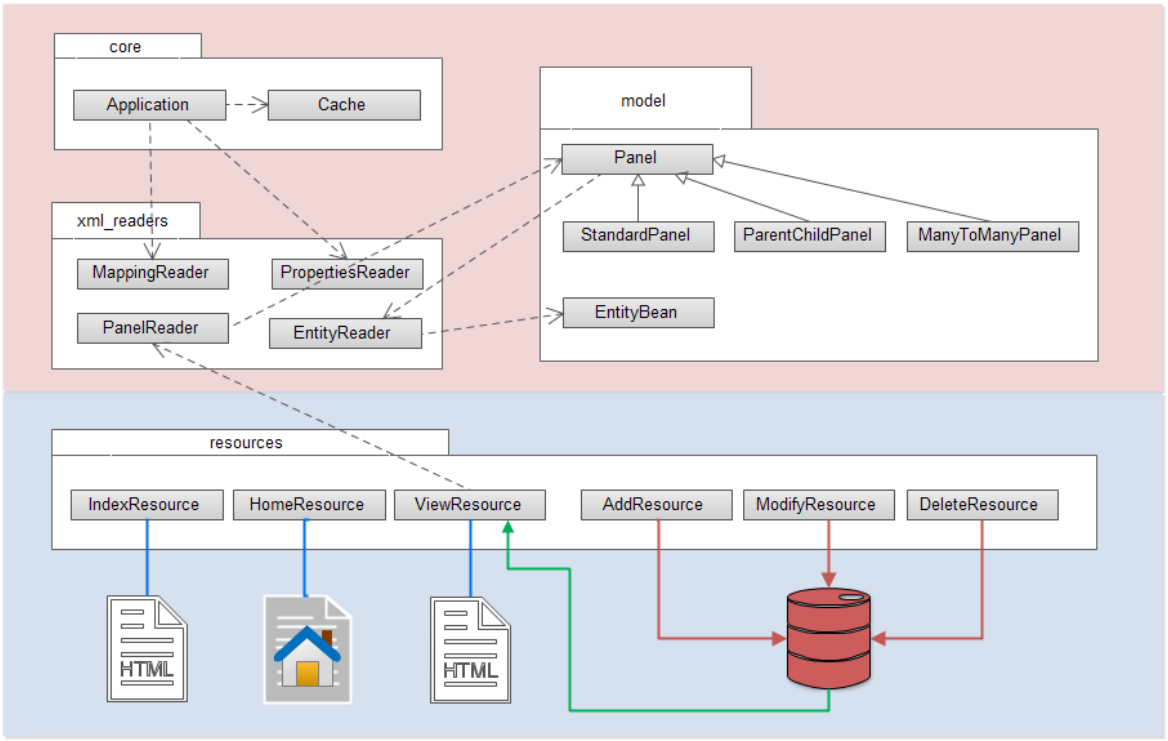

Figure 3. Generic WEB engine architecture

The main components of the engine are shown in the core package in Figure 3. Application module represents the main application class while all of the application's data is stored and managed by the Cache class. Upon startup, application loads the mapping data and project properties from the application repository using the corresponding readers from the xml_readers package. This reduces performance drops when executing large projects since only the mapping data is loaded into an application cache while the actual model data is loaded on demand. Each reader module reads the data from the corresponding configuration file stored in the application repository and stores it in the application cache. Once all the necessary data has been obtained, engine is ready to run.

Basic use scenario in enterprise system revolves around users manipulating data from the database via standard forms. Standard forms contain (one or more) standard panels with well-defined look and features (see [13] for details). The resources package contains the modules responsible for obtaining data for a specific standard form and presenting it to the user. Since our web engine is based on the Restlet web engine, all modules represent Restlet's resource classes. HomeResource handles the login requests and

These user actions are handled by the ViewResource module. It obtains corresponding panel data from the application repository via the PanelReader component. The panel specification contains only the representational aspect of one panel (layout specification and default panel controls), so in order for the given form to be functional, additional persistent data needs to be acquired. Each panel is associated with one EntityBean instance that it obtains via the EntityReader module. Combining the EntityBean and Panel data, the ViewResource displays the web form that conforms to the desired specification. The data that needs to be represented is wrapped into HTML elements using the Freemarker templates.

The basic steps in this process are illustrated in Figure 4. The corresponding Restlet resources handle other enterprise operations. For the sake of simplicity, Figure 3 only shows the basic CRUD (create, update, delete) resources. These modules don't have the explicit HTML representation, they just inform the user of the operation result via the simple text sent over an AJAX call.

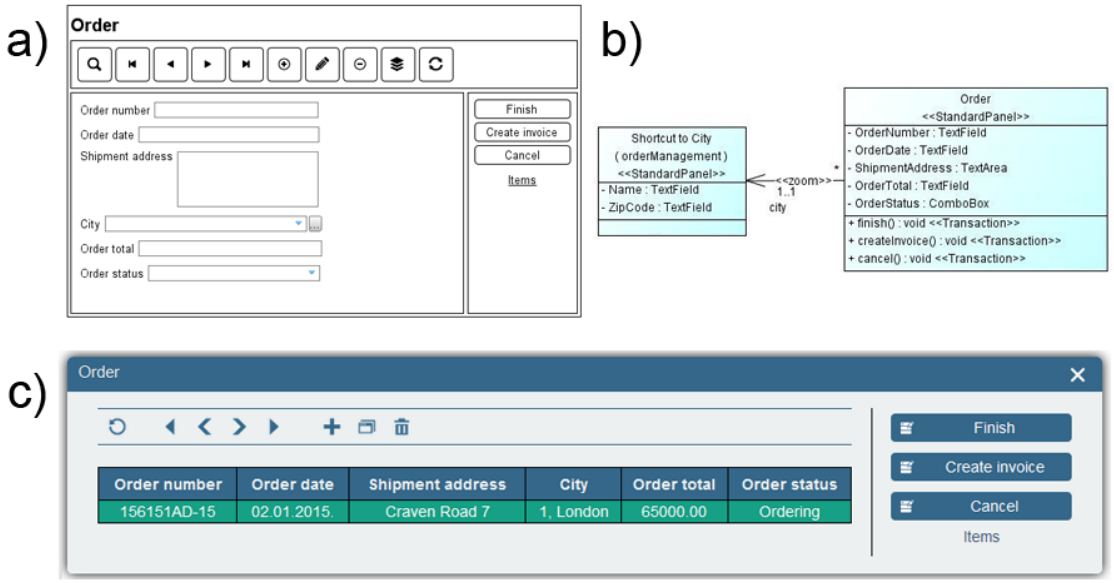

Figure 5 Example of a) a mockup specification b) a corresponding UML-like specification c) a resulting web form in a view mode

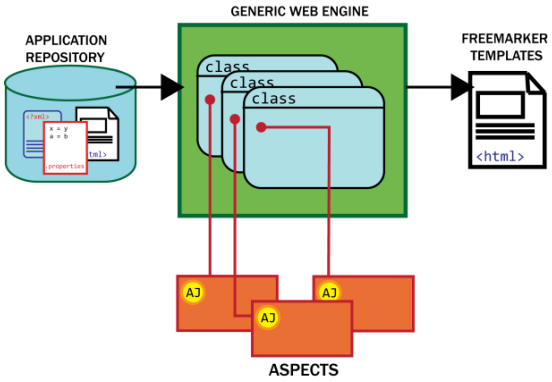

Figure 4. WEB engine overview

V. EXTENDING GENERIC ENGINE

Kroki engines offer standard enterprise functionalities over arbitrary data sets, so its main concern is to process and present data from the database in the predefined way. As a result of the fact that enterprise systems vary in their functionalities, it was necessary to develop mechanism for extending generic engines. Kroki generic engines use aspect-oriented programming techniques to capture run-time points of interests and react in a desired way.

In the web engine, this process is pretty straightforward. The web engine is developed using Restlet engine, so all of the web classes extend Restlet Resource class and are located in the resources package. Every resource class has `prepareContent` method that is invoked when a client request is sent to a particular resource and can be used to attach aspect functionalities. Restlet resources use map called `dataModel` to pass arbitrary data to HTML templates, so once attached to prepareContent, aspect can get access to the resource object and its corresponding `dataModel` (Figure 6). Also, all data contained in the application cache is available to the aspect.

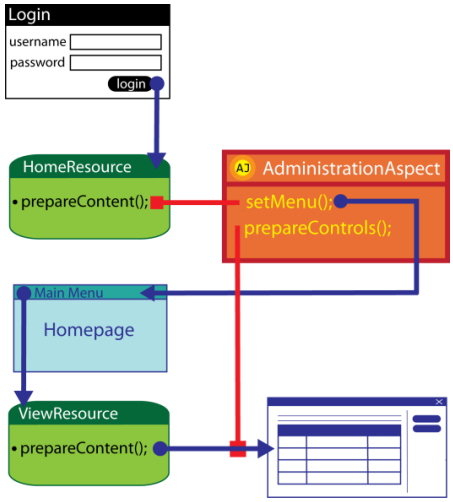

Figure 6. Aspects can change the content before pages are rendered

Listing 1 shows an aspect that modifies the main menu before it is presented to the user (one possible scenario for this is filtering main menu items based on user permissions created by Kroki administration subsystem). Since generic web engine is designed as one-page Ajax-based web application, once the user is logged in, the interaction in it's entirety takes place on the home page and Restlet resource in charge of this page. So, as mentioned before, in order to modify the main menu preparation process, we need to attach our aspect to `prepareContent` method of `HomeResource` class. Freemarker template looks up main menu list by the name `main_menu`, so it will be the name by which we will put our modified menu into `dataModel`. Listing 1 represent basic steps described above.

VI. CONCLUSIONS

The paper presented architectural solutions incorporated in the Kroki tool which enable prototype execution. Core elements of this rapid prototyping technique are the application repository and generic engines which create functional enterprise Java application based on the developed specification.

```
public aspect MainMenuAspect {
/* Create the pointcut that
   intercepts PrepareContent method in
   Home resource and obtain home resource object
*/
  public pointcut setMenu
        (HomeResource homeResource) :
  call(public void HomeResource.prepareContent())
        && this(homeResource);


  after (HomeResource homeResource) :
        setMenu(homeResource) {
        //Obtain main menu list from AppCache
        ArrayList<AdaptMenu> menus =
        AppCache.getInstance().getMenuList();

        //Do something...

        //Put modified main menu to data model
        homeResource.addToDataModel
                ("main_menu", menus);
}
```

Listing 1. Aspect extension example

The process does not include traditional code generation techniques where complete programming code (or most of it) is being generated. In our approach, engines are generic enough to cover basic business operations on provided data.

The engines based on aspect-oriented programming enable: (1) easier inclusion of future tools and features that can affect the specified enterprise application execution; (2) integration of generated and hand-written code during application evolution, when its code is exported to a general-purpose programming tool; (3) dynamic adaptation of application's look and behavior in accordance with the user rights defined in Kroki's administration subsystem.

Decision to use this approach is guided by years of research in model-driven development and generic enterprise systems [1, 13, 14, 15], which resulted in development of our EUIS DSL [13] and Kroki tool.

Now, we are planning to convey a research as large as possible including end-users, business specialists, students, and IT experts in order to get their feedback and measure their reactions while using the tool. We plan to base our experiment on Methodology Evaluation Model (MEM), which represents software engineering specific extension of earlier Technology Acceptance Model (TAM). The MEM model suggests that a certain methodology, in order to be successfully accepted and used, needs to satisfy both subjective and objective measures of usefulness and ease of use. In the experiment, we plan to measure performance based variables (actual efficiency and actual effectiveness) using cognitive load measurement approach, and perception based variables (perceived ease of use and perceived usefulness), like presented in [16].

REFERENCES

[1] G. Milosavljevic, M. Filipovic, V. Marsenic, D. Pejakovic, I. Dejanovic, Kroki: A mockup-based tool for participatory development of business applications.. SoMeT (p./pp. 235-242), : IEEE. ISBN: 978-1-4799-0419-8, 2013.

[2] T. Cerny, M. Macik, M.J. Donahoo, J. Janousek, Efficient description and cache performance in Aspect-Oriented user interface design, Computer Science and Information Systems (FedCSIS), 2014

[3] T. Cerny, K. Cemus, M. J. Donahoo, and E. Song. Aspect-driven, Data-reflective and context-aware user interfaces design. Applied Computing Review, 13(4):53–65, 2013

[4] J. L. H. Agustin, P. C. del Barco, A model-driven approach to develop high performance web applications, Journal of Systems and SoftwareVolume 86, Issue 12, 2013

[5] J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, F. Montero, M. Gaedke. 2014. Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering. Inf. Softw. Technol. 56, 6, June 2014

[6] J. Solano, Exploring How Model Oriented Programming can be Extended to the UI Level, PhD Thesis, University of Ottawa, 2010, http://hdl.handle.net/10393/28569

[7] A. Forward, O. Badreddin, T. Lethbridge, J. Solano, Model-driven rapid prototyping with Umple, Software: Practice and Experience, Volume 42, Issue 7, pages 781–797, July 2012

[8] H. Störrle, Model-driven development of user interface prototypes: an integrated approach, Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, pp 261-268, Copenhagen, Denmark

[9] T. Buchmann, Towards Tool Support For Agile Modeling: Sketching Equals Modeling, *Proceedings of the 2012 Extreme Modeling Workshop*, pp. 9-14. ACM, 2012.

[10] A. Coyette, S. Schimke, J. Vanderdonckt, C. Vielhauer, Trainable Sketch Recognizer for Graphical User Interface Design , *Human-Computer Interaction–INTERACT 2007*. Springer Berlin Heidelberg, 2007. 124-135

[11] A. Coyette, J. Vanderdonckt, In *Human-Computer Interaction-INTERACT 2005*, pp. 550-564. Springer Berlin Heidelberg, 2005.

[12] B. Plimmer, M. Apperley, Interacting with Sketched Interface Designs: An Evaluation Study, In *CHI'04 extended abstracts on Human factors in computing systems* (pp. 1337-1340). ACM.

[13] B. Perišić, G. Milosavljević, I. Dejanović, B. Milosavljević, "UML Profile for Specifying User Interfaces of Business Applications", Computer Science and Information Systems,Vol. 8, No. 2, pp. 405-426., 2011

[14] G. Milosavljević, B. Perišić, "A Method and a tool for rapid prototyping of large-scale business information systems", Computer Science And Information Systems, Vol. 02, pp. 57-82, 2004

[15] B Milosavljevic, M. Vidakovic, S.Komazec, G. Milosavljevic, User interface code generation for EJB-based data models using intermediate form representations,2nd International Symposium on Principles and Practice of Programming in Java, PPPJ 2003, Kilkenny City, Ireland, 2003

[16] S. Abrahão, E. Insfran, J. A. Carsí, M. Genero, "Evaluating requirements modeling methods based on user perceptions: A family of experiments", *Information Sciences*, Volume 181, Issue 16, pp. 3356-3378 (2011)

[17] Kroki, www.kroki-mde.net

[18] Kroki demo, http://youtu.be/r2eQrl11bzA