applied sciences

MDPI

Article

Rapid Requirements Elicitation of Enterprise Applications Based on Executable Mockups

Milorad Filipović *, Željko Vuković, Igor Dejanović * and Gordana Milosavljević *

Faculty of Technical Sciences, University of Novi Sad, 21000 Novi Sad, Serbia; zeljkov@uns.ac.rs
* Correspondence: mfili@uns.ac.rs (M.F.); igord@uns.ac.rs (I.D.); grist@uns.ac.rs (G.M.); Tel.: +381-21-485-4562 (G.M.)

Abstract: Software development begins with the requirements. Misunderstandings with customers in this early phase of development result in wasted development time. This work investigates the possibility of using executable UI mockups in the initial phases of functional requirements elicitation during the development of business applications. Although there has been a lot of research in the field in recent years, we find that there is still a need to improve model-driven tool design in order to enable customer participation from the initial phases of requirement specifications based on working prototypes. These prototypes can directly be reused in the rest of the development process. To meet the goal, we have been developing an open-source solution called Kroki that enables rapid collaborative development. We conducted a series of 10 joint user sessions with domain experts from different domains and backgrounds, resulting in the prototype specifications ranging from 7 to 20 screen mockups accompanied with domain models, developed in two-hour time frames. In this paper, we present our tool design that contributes to rapid joint development, and the results from the user sessions.

Keywords: requirements elicitation; executable mockups; model-driven engineering; code generation

Citation: Filipović, M.; Vuković, Ž.; Dejanović, I.; Milosavljević, G. Rapid Requirements Elicitation of Enterprise Applications Based on Executable Mockups. *Appl. Sci.* **2021**, *11*, 7684. https://doi.org/10.3390/app11167684

Academic Editor: Andrea Prati

Received: 1 July 2021
Accepted: 16 August 2021
Published: 21 August 2021

1. Introduction

The adoption of agile development methodologies has become a widespread practice in software companies in recent years. Techniques such as continuous delivery, iterative and incremental development, user stories, and user-on-site have boosted delivery time and improved stakeholders' insight into the development process. However, there is still a need to improve the requirements of engineering practices of agile methods, especially for applications where a user interface (UI) plays an important role [1,2]. The communication gap between the stakeholders and the development team is still hard to overcome.

A well-known technique in requirement engineering for narrowing the communication gap is the usage of prototypes. The prototype can be anything from a low-fidelity paper mockup to an executable application that the customer can operate [3]. In the early design phases, the purpose of the low-fidelity UI mockups is to ensure that required business functions are supported and that customers can perceive whether some information is missing. High-fidelity UI design is not necessary for this phase since the development team could be caught up in cosmetic discussions [4]. It is essential to enable efficient development and adjustments of mockups to foster communication and support changes of requests. When low-fidelity mockups are finished and approved by all involved parties, they can become a starting point for high-fidelity UI design [4].

Unfortunately, low-fidelity mockups created by paper and pencil or by general-purpose mockup tools often fail to identify some design inconsistencies and shortcomings. In our opinion, the best feedback could be gained when users can operate an executable prototype in a context that is close to their natural workplace environment. However, executable prototypes are considered expensive for development and hence traditionally left for the later phases [3].

The optimal solution for more reliable requirements elicitation could be to efficiently create low-cost executable mockups so that customers are able to perform a hands-on evaluation from the very start. Although executable, these mockups should be low fidelity in their visual appearance, to keep the focus on business features under development. To achieve this, we have been developing an open-source tool called Kroki (fr. *croquis*—sketch) [5,6] for the rapid prototyping of enterprise applications in collaboration with customers.

The core of Kroki is our EUIS (Enterprise User Interface Specification) DSL (Domain-Specific Language) [7]. EUIS DSL is developed to enable rapid sketching of enterprise application mockups and their simple integration with persistence layers in order to support code generation for all three application tiers (the user interface, the business logic, and the database). This enables a more realistic user experience while operating the prototype, since users tend to pay more attention and discover errors and inconsistencies more easily when mockups can be populated with their actual data [8].

In Kroki, mockup execution is provided by two aspect-oriented engines for web and desktop applications. Executing the prototype allows customers to evaluate the look and behavior of the developed system so that corrections can be made immediately. The verified specification can then be used as a basis for further manual development since the prototype code and developed models can be exported from Kroki to general-purpose modeling and programming tools. In addition, if Kroki's supported technological platform is appropriate, the generated prototype, accompanied by hand-written code, can evolve to the final application. This way, the effort invested in mockup development is not wasted.

Since Kroki does not have support for non-functional requirements, they have to be negotiated with customers before utilizing Kroki in mockup development.

In this paper, we present:

- A brief overview of the Kroki tool design that enables development agility needed for collaborative work with customers (details can be found in our previously published papers (http://www.kroki-mde.net/publications, accessed on 18 August 2021));
- An exploratory case study we performed to investigate what can be achieved if we start to use executable mockups from initial requirements elicitation sessions with customers. We conducted 10 two-hour development sessions with domain experts from different business domains and developed an executable application prototype ranging from 7 to 20 mockups in each session.

An insight into the design of this tool and the results of the study could be helpful for model-driven tool developers.

The paper is structured as follows. Section 2 gives an overview of Kroki. Section 3 presents the exploratory study. Section 4 discusses the results of the study. Section 5 reviews the related work. Section 6 concludes the paper.

2. Kroki Tool

Kroki is a rapid prototyping tool that is being developed to enable:

- Participation of customers in the development process from the initial phases of functional requirements specification;
- Direct reuse of as much information as possible from the developed prototypes to avoid waste of time and energy.

To see the Kroki overview video visit https://www.youtube.com/watch?v=r2eQrl11bzA (accessed on 18 August 2021).

The prototype can be specified using a Mockup editor, Lightweight UML class diagram editor, or Command console.

The mockup editor is mostly used in collaborative sessions with customers. It consists of a mockup drawing area, UI component palettes, and property editor panels used for setting the property values of mockups and their components (Figure 1). By drawing these components, the user is actually performing UI modeling since the editor is an implementation of a mockup-based concrete syntax of EUIS.

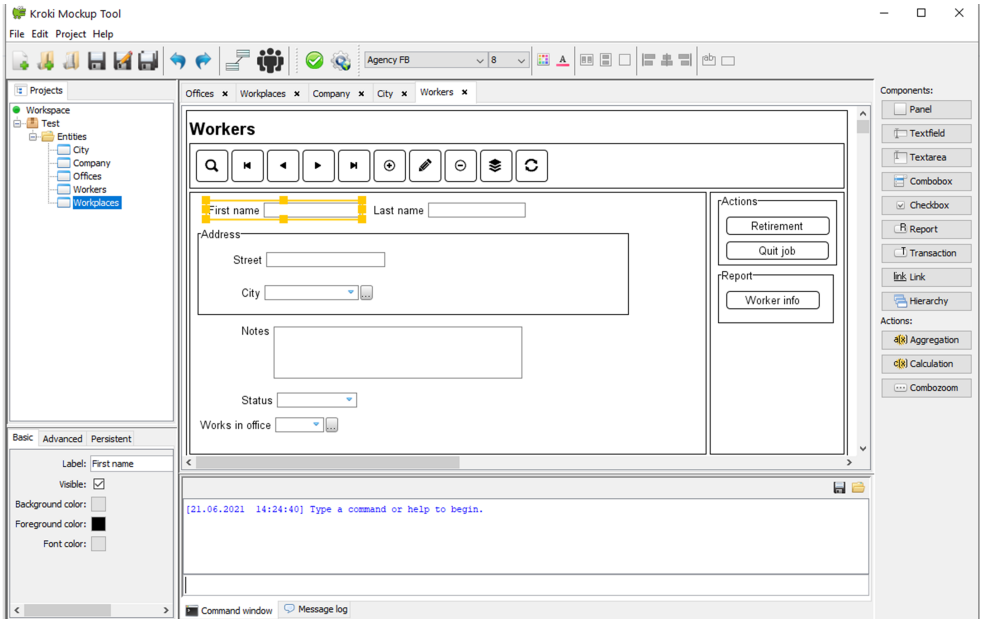

Figure 1. Kroki Mockup editor.

The property editor is divided into three tabs for the specification of basic, advanced, and persistence settings of the selected component (Figure 2). If not specified, default settings are used, so prototype execution is possible even when only basic values are filled in.

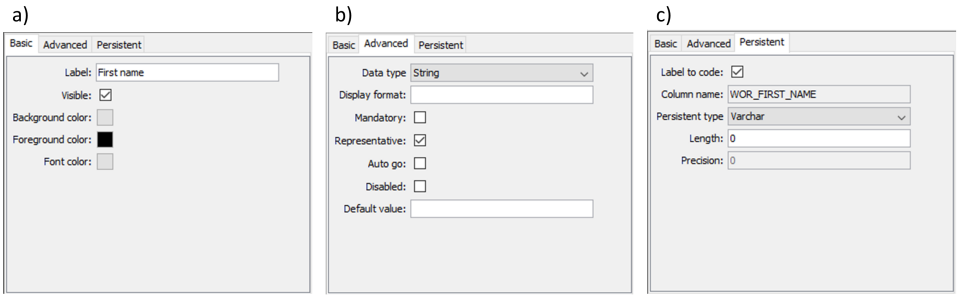

Figure 2. An example of (**a**) basic, (**b**) advanced, and (**c**) persistence settings for an `Editable` component.

Kroki's lightweight UML editor implements a UML-based graphical syntax of EUIS, which is designed to look like a simplified UML class diagram notation with stereotypes (Figure 3). While the UI drawing interface is tailored for direct communication with the end users, this editor is intended for the developers, who can use it to better view and specify the structure and navigation through the application, including composite panes structure. In our experience of exposing the customers to UML, this is best done while they are not present.

Both UML and Mockup editors manipulate the same prototype specification (Figure 4). Changes that are made in one representation are immediately visible in others. For newly-added elements, automatic layouting is performed [9].

EUIS DSL is based on our HCI (human-computer interaction) guidelines, which define functional and presentational features of coarse-grained building blocks of enterprise applications (types of forms, reports, and subsystems). A brief overview of these guidelines and corresponding EUIS meta-classes is given in Section 2.1.

The aspect-oriented engines that enable instant execution of the specified prototype are explained in Section 2.4.

Kroki's support for reusability, and import and export features is shown in Section 2.5.

The administration subsystem for the specification of user roles and their working environments resides in Section 2.6.

The details regarding Kroki's recommended usage are explained in Section 2.7.

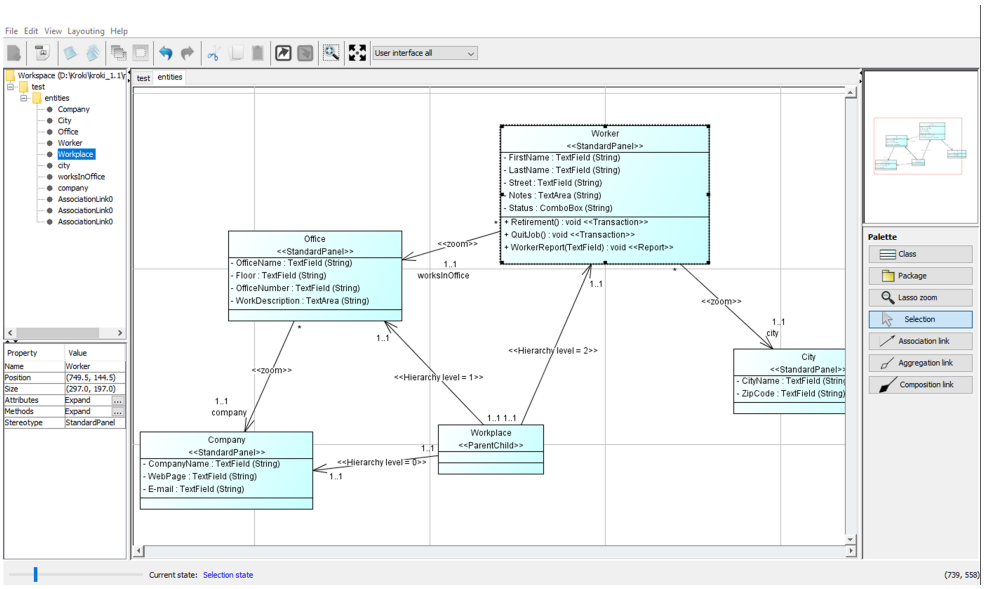

Figure 3. Kroki's lightweight UML editor.

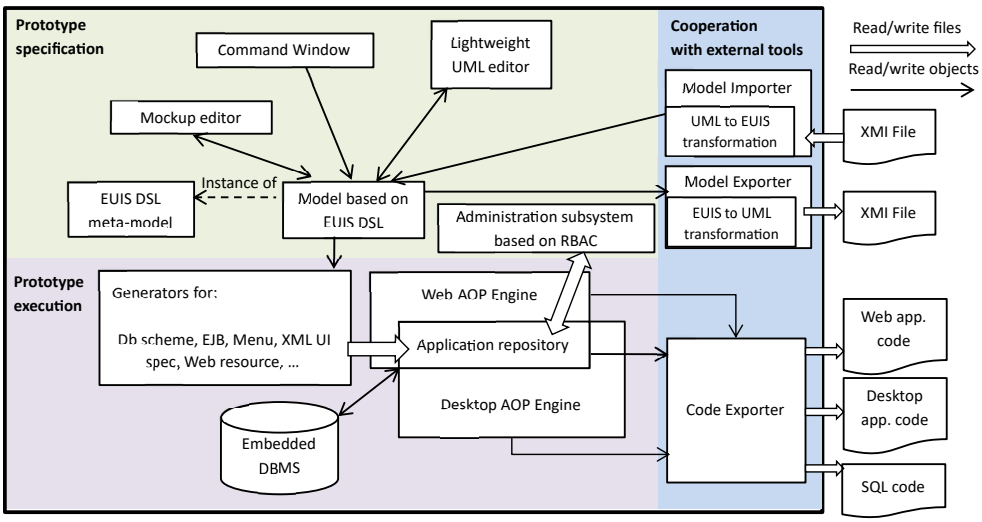

Figure 4. The architecture of Kroki.

2.1. EUIS DSL and Our HCI Guidelines

EUIS DSL is based on concepts described in our human-computer interaction (HCI) guidelines aimed at defining functional and visual features of business application UI components. We intended to specify the minimal set of coarse-grained building blocks to support simplicity of use, quick user training, and the automation of user interface construction. Our HCI guidelines are, with some modifications, in use since 1996 by several software companies that have adopted our approach to business application development. As an example, the experience of Lanaco (https://www.lanaco.com/en/lanaco-information-technology, accesed on 18 August 2021) has been described in [10].

The first formal specification of EUIS DSL was in the form of a UML profile [7]. EUIS profile was intended to provide a simple integration of the user interface model with the persistent model and the relational model by using a UML class diagram as a common foundation. The EUIS UML profile extends meta-classes from `UML::Kernel` package: `Class`, `Property`, `Operation`, `Parameter`, `Constraint`, and `Package`.

The most important concepts of our HCI guidelines with corresponding EUIS profile stereotypes are explained as follows, while more details can be found in [7,11].

A standard panel (stereotype `StandardPanel`, Figure 5a) performs CRUD (create, update, delete) and search operations on instances of one domain class. Operations common to most entities ("standard" operations) are represented by icons at the top of the panel, while buttons on the right-hand side enable invocation of specific operations. The specific operations are business transactions and reports (stereotypes `Transaction` and `Report`, Figure 5d).

The standard panel can be directly invoked or used as a building block for complex panels (descendants of `ContainerPanel` stereotype). A parent-child panel (stereotype `ParentChild`) is used to organize data with a hierarchical structure, where the standard panel represents one level in the hierarchy. The panel level is modeled by a hierarchical association (`Hierarchy` stereotype, Figure 5c).

Elements of the standard panel (UI components) can be attached to editable (stereotype `Editable`) or derived properties (stereotypes `Aggregated`, `Calculated`, and `Lookup`) of domain classes (Figure 5b). The formula used to calculate derived values is specified using OCL (*Object Constraint Language* [12]) and implemented using the Dresden OCL library [13].

Relationships among panels are represented by four mechanisms: *Zoom*, *next*, *activate*, and *hierarchy*, which are descendants of the `VisibleAssociationEnd` stereotype, Figure 5c). The *zoom* mechanism represents an invocation of the panel associated with the given domain class where the user can choose a row and pick its values to the fields of the previous panel. The *next* mechanism, invoked from the panel of a parent class, displays the panel associated with the child class in a separate view, with its data filtered according to the selected row of the parent panel.

The *activate* mechanism enables direct invocation of a panel by another panel, without restrictions on the data displayed. The invoked panel does not need to have an association with the current one. The *hierarchy* mechanism enables the development of parent-child panels.

The development of a business application prototype using the EUIS profile comprised the following activities (Figure 6):

- Development of a domain model using a class diagram;
- The model-to-model transformation of the domain model to a UI model based on the EUIS profile and persistence model;
- Manual adjustments of generated models;
- Code generation (model-to-code transformation) of the UI and persistence model.

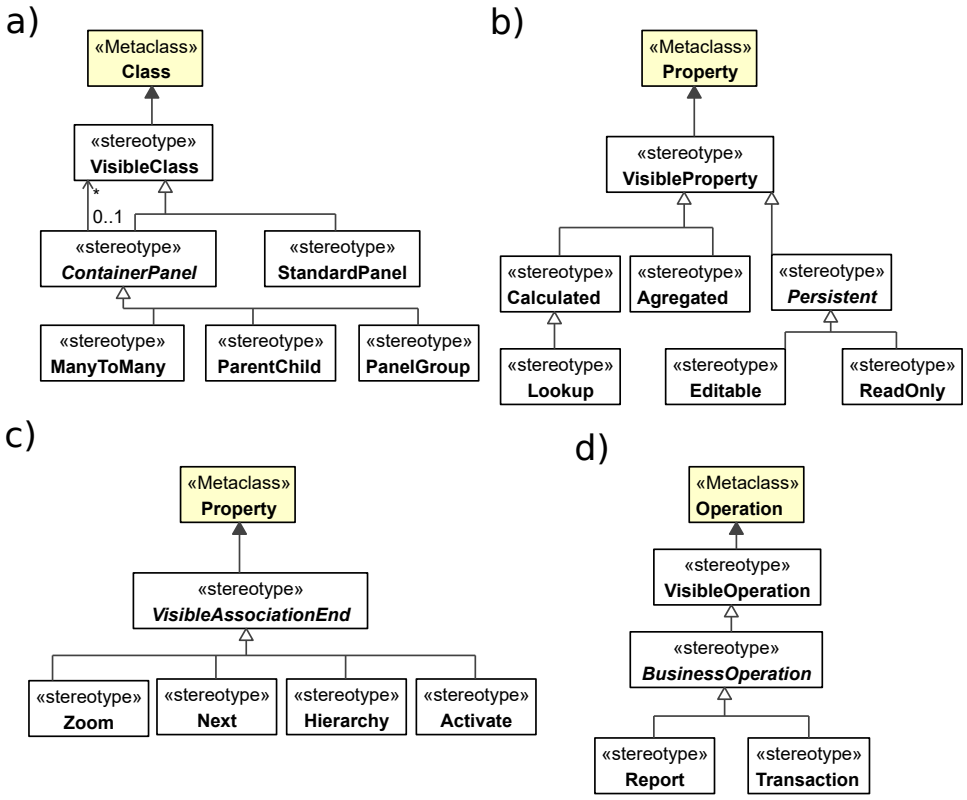

Figure 5. The most important elements of the EUIS UML profile: (**a**) Visible classes (panels), (**b**) visible properties, (**c**) visible association ends, and (**d**) visible operations.

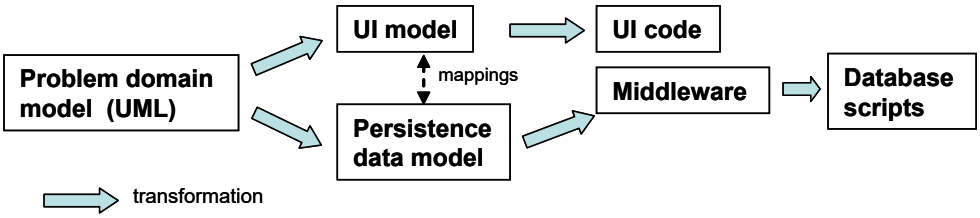

Figure 6. The development process using the EUIS profile is based on transformations. Reproduced with permission from [5], copyright IEEE, 2013.

However, since the development was conducted without customers during the domain model specification, we needed several iterations until mutual understanding was achieved. A mental gap had to be crossed when manually mapping user requirements to the domain model based on a UML class diagram [5].

2.2. Kroki Meta-Model

Kroki is designed as a tool for which the foundation is EUIS DSL—a stand-alone implementation of the EUIS UML profile. Its meta-model (Figure 7) consists of four smaller meta-models integrated by common meta-classes. Meta-classes from the UI meta-model and the `Persistence` meta-model reference meta-classes that implement a simplified `UML::Kernel` meta-model. The `Layout` meta-model is added to enable the design of mockup layouts (Figure 8).

The goal of the way the meta-model is designed is to make it possible for UI model elements to be viewed both as mockups through the Mockup editor and as "ordinary" classes with stereotypes using Kroki's lightweight UML editor. As a result, we have eliminated the need for model-to-model transformations, which were a part of our earlier approach with the EUIS profile and many other approaches presented in Section 5. Some of the benefits of this design is that the many views of the application specification remain constantly synchronized, without the need for round-trip transformations. Furthermore, the feedback loop is much shorter as the specifications can be executed immediately.

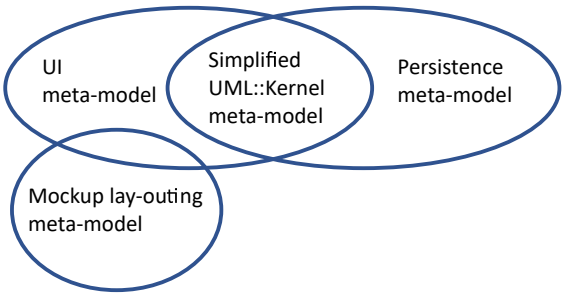

Figure 7. EUIS DSL meta-model structure. Reproduced with permission from [5], Copyright IEEE, 2013.

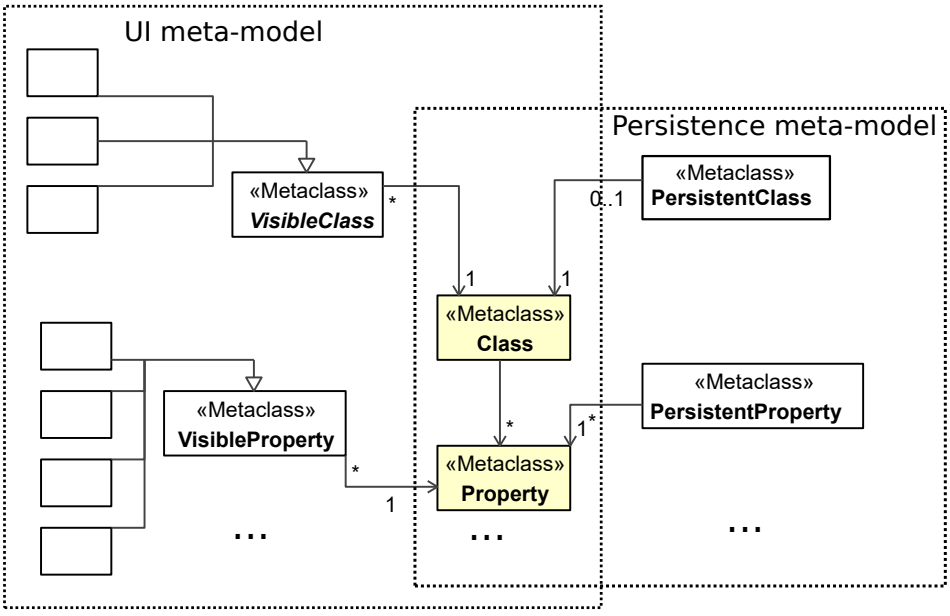

Figure 8. A sketch of UI meta-model and Persistence meta-model integration using common meta-classes from simplified `UML::Kernel` meta-model implementation.

2.3. An Example Specificaton

Figure 9 presents an example of a prototype specification in Kroki's lightweight UML editor in the form of the UI model. Classes `City`, `Office`, `Company`, and `Worker` are denoted as standard panels, while the class `Workplace` is denoted as a parent-child panel consisting of class `Company` at level zero (parent), `Office` at level one, and `Worker` at level two.

The UI model in Figure 9 is a result of mockups' sketching in Kroki's Mockup editor. Figure 10 presents a mockup for managing data regarding workers in an office.

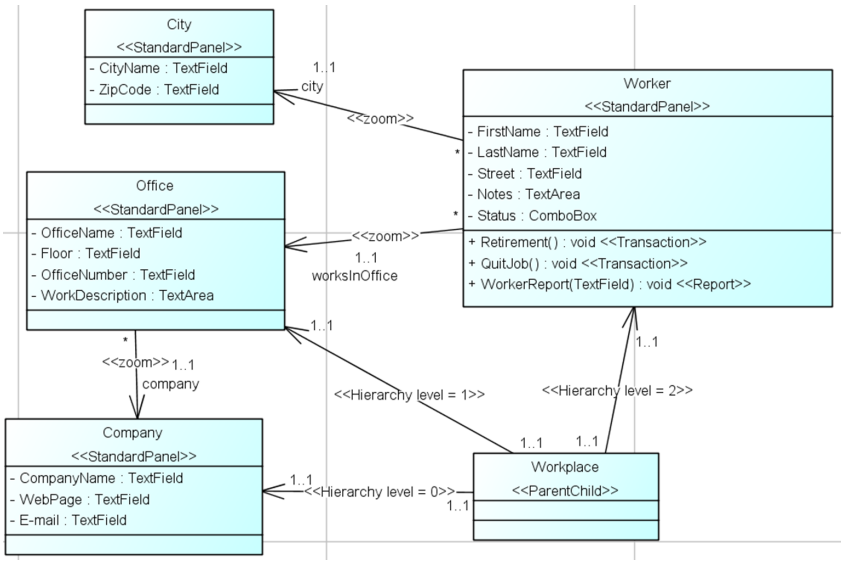

Figure 9. An example specification in the UML editor.

Figure 10. A mockup for managing worker data.

Figure 11 shows a parent-child Workplace mockup which is assembled of previously created mockups for `Company`, `Office`, and `Worker`.

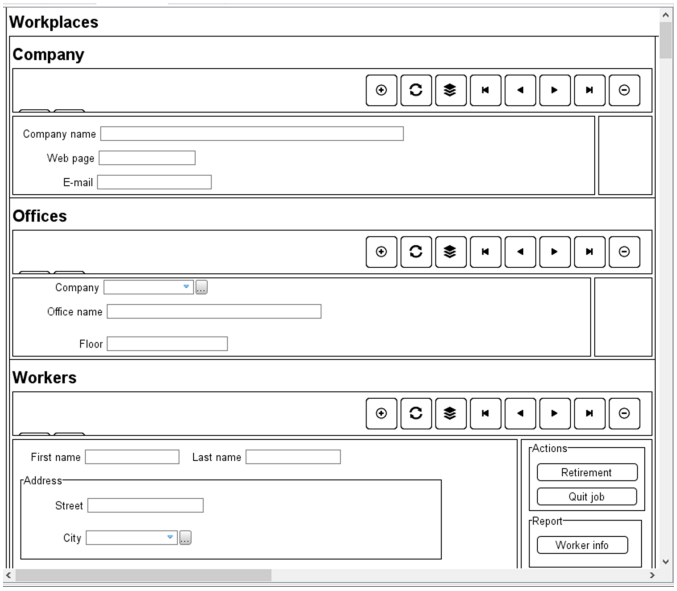

Figure 11. An example of a parent-child mockup.

2.4. Kroki AOP Engines

To enable the execution of prototypes in Kroki, code generator modules feed the specified model to aspect-oriented Java engines. This approach is based on the generic web or desktop Java applications that adapt their look and behavior according to the prototype specification [14] and defined user privileges.

The configuration files for these engines are generated for each project, while most of the prototype code and assets already exist in the application repository (Figure 4) in order to enable rapid collaborative development cycles with the customers. Each collaborative cycle includes gathering the specification or changes of the specification from the customer and presenting them with a running prototype.

A database schema created based on the generated JPA entities [15] can be deployed into an existing local or remote database. If the database connection parameters are not specified, the project will use a temporary, in-memory H2 database (H2 Database Engine, http://www.h2database.com, accessed on 18 August 2021). An example of a running web application that is the result of the described process can be viewed in Figure 12–14.

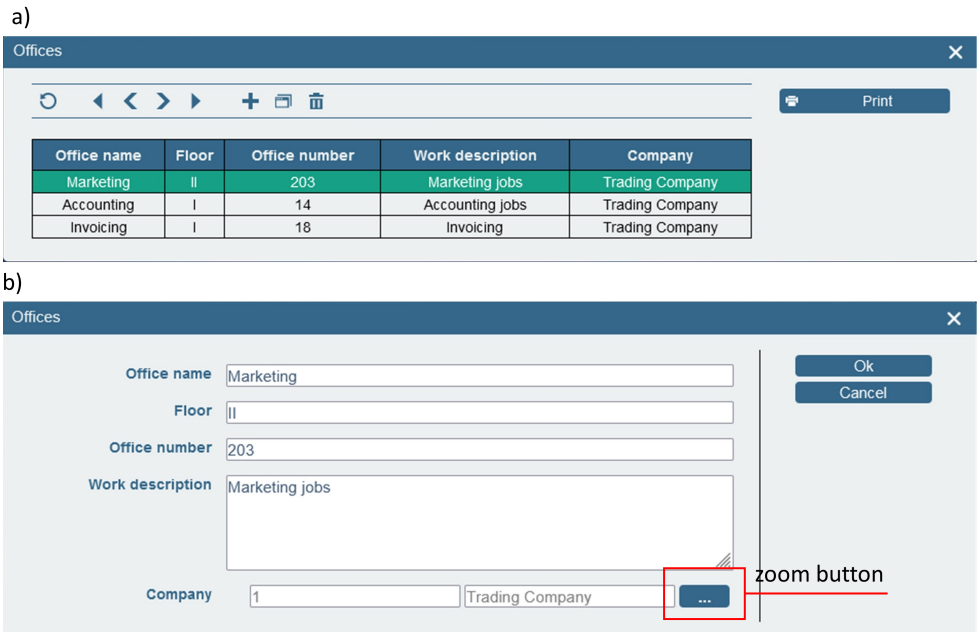

Figure 12. A web page for managing office data in (**a**) view mode and (**b**) edit mode.

Figure 13 presents a web page for managing workers' data in view mode.

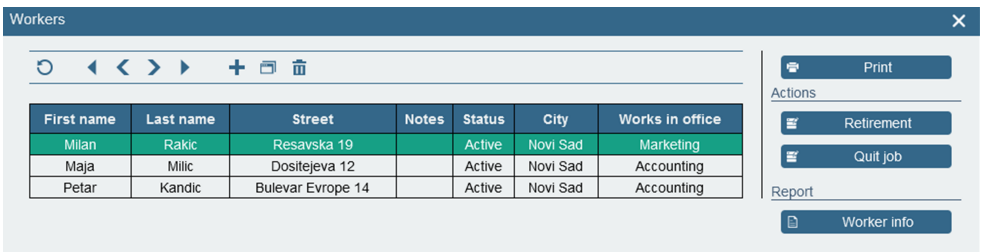

Figure 13. A web page for managing workers data in a view mode.

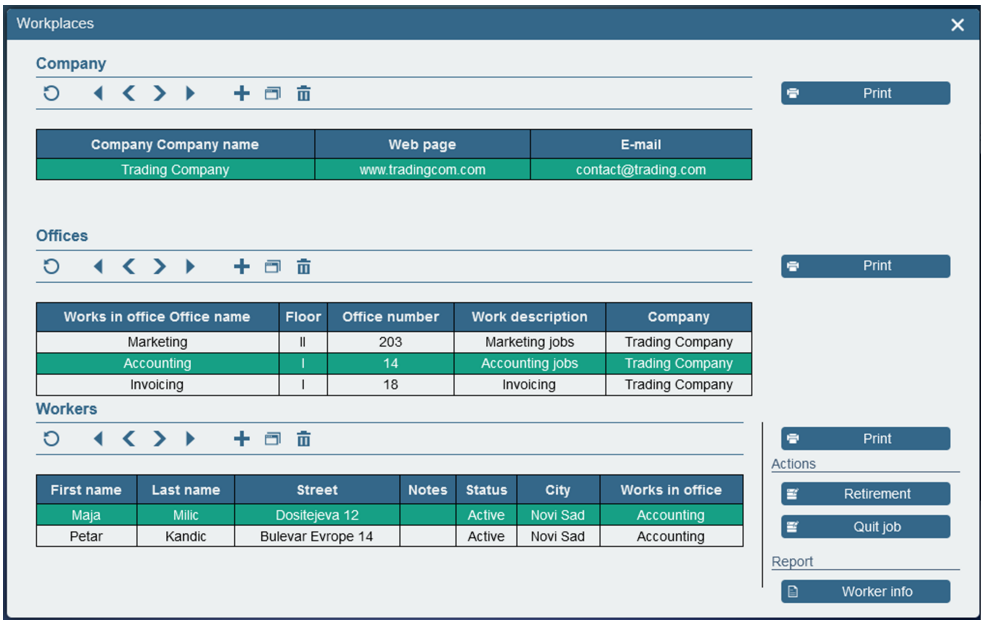

Figure 14. A parent-child web page of a prototype.

2.5. Import and Export Features

Kroki fosters model reusability by providing UML class diagrams to be imported from or exported to general-purpose modeling tools using the XMI format.

Generated prototype code can be exported as an Eclipse Java project. If we need to manually change or enhance the generated project, we can implement the changes using aspects. This way, there is no need to change the generated code directly, and subsequent code generation cannot damage handwritten changes.

The usage of Kroki's import and export features is described in more detail in [16].

2.6. Administration Subsystem

The administration subsystem consists of an Administration Manager and Menu Editor tools [17]. The administration manager enables the specification of user roles and their privileges in the developed system based on simplified RBAC (*Role-Based Access Control* [18]). The existence of the user role's privileges enables dynamic configuration of the mockups and application specifications during runtime, based on the needs of each user's role.

If user roles and privileges are not specified; Kroki generates a default user with all the privileges, so all specified features can be verified.

The Menu Editor enables the specification of the main menu structure for each user role defined in the Administration subsystem. If the menu is not specified, Kroki's Menu Generator creates the main menu and its sub-menus so that they reflect the package structure of the Kroki project.

2.7. Kroki's Recommended Usage

Before we can utilize Kroki, we need to perform an initial development phase to define non-functional requirements, the system's scope, perform decomposition to a set of well-defined subsystems, and create a plan for their implementation, integration, and deployment. Each subsystem can then be developed using Kroki, with or without the help of general-purpose modeling and programming tools.

Kroki is primarily designed to support the iterative and incremental prototype development of enterprise applications in collaborative sessions with customers, although it can be used in any development environment. If Kroki's supported technological platform is appropriate, the generated prototype, accompanied by handwritten code, can evolve to the final application. Otherwise, the prototype can be used only for functional

requirements elicitation, providing a data model and mockups specification as input to manual construction.

Similar to our previous solutions [19], Kroki is designed to create a fully functional prototype with as little generated code as possible. A large body of program code complicates management and reconfiguration and slows down prototype compilation and execution. The strategy of reducing the program code size is carried out through the Application repository, the aspect-oriented engines, and the administration subsystem as follows.

The developed mockups from all development sessions should be kept in a single project that can be logically separated into packages. User roles' working environments can be specified using the administration subsystem by choosing a subset of developed mockups and setting appropriate privileges and menu items for their activation. A single mockup can appear in multiple user roles' working environments.

Suppose different user roles need several versions of the same mockup, with added or removed UI components or operations. In that case, the development team should create a single mockup as a union of the needed UI components and use the administration subsystem to specify privileges for each UI component. The development team's responsibility is to recognize the overlapping requirements and address them by mockup adjustments and specification of user rights. We did not want to implement automatic identification, adjustment, and integration of separately developed mockups since this process is prone to errors and can be done in a semi-automatic way at best [20]. During execution, generic aspect-oriented engines instance only those elements (menus, forms, and UI components) with appropriate privileges according to the RBAC standard.

3. Exploratory Case Study Research

In order to verify if executable mockups can be used for requirements' specification from the start, we decided to conduct an exploratory case study [21,22] with participants from various business domains. The research question and the study proposition are defined as follows:

- **Research question:**Is it beneficial to organize collaborative development sessions based on executable prototypes in the early phases of requirements elicitation?
- **Study proposition:** Executable prototypes developed in tight cooperation with customers can enhance mutual understanding, increase customer satisfaction, and shorten the requirements elicitation phase.

So far, we were able to organize 10 sessions. A summary of these sessions is displayed in the following sections. Developed specifications and prototype screenshots are available in [23]. These specifications can be opened and executed by Kroki. The source code of Kroki itself can be downloaded from [6].

The participants were voluntarily joining the research. Our goal was to find the participants from different business domains, with different company positions and education levels. There was no restriction regarding age, gender, or computer literacy (see Tables 1 and 2). The participants were chosen from domains that we were not familiar with in order to avoid the influence of our domain knowledge. Participants were also not engaged in software design and construction in their workplace. They did not have previous knowledge of Kroki as a tool nor the prototyping approach on which we planned to base the communication on.

Since we had experience with the development of several ERP (*Enterprise Resource Planning*) systems, we did not choose participants from areas usually supported by ERPs (accounting, invoicing, human resources management, asset management, production, planning, etc.). They were also not chosen from our faculty services since we cooperated with them during our in-house information system development. Although we could more easily gather participants by recruiting our students, we did not engage them because they were not a genuine substitute for real-world customers [24]. We used students for testing the Kroki tool in [25].

The development team consisted of two researchers who conducted all sessions so as to avoid the impact of different researchers on the outcome. One of them operated Kroki during the sessions, while the other was more engaged in design decisions. Both of them were equally engaged in communication.

Participants were asked to come to their session with a software application in mind that they would like to have developed and some artifacts (documents, forms, reports, etc.) that could help us better understand their requirements. It could be a development of a new application or an improvement of an existing one that lacked some essential features. Some of the participants were customers during software development in their companies, so they came particularly well prepared and often had examples of documents they were operating with in their everyday tasks. After the sessions, they were able to compare our approach with their previous experiences.

The sessions were limited to a two-hour time frame. At the beginning of each session, we asked participants to briefly describe their domain, the main goals they wanted to achieve, and the application's main features. We also tried to understand the application boundaries, especially if the application was perceived as a part of an anticipated more complex system. When we felt that we acquired a basic understanding of participant's needs, we would start creating initial mockups for one of the features we agreed upon. Mockups were created using the Kroki mockup editor (Figure 3), so the participants were able to understand the process and suggest corrections. These mockups served as a foundation for further discussion and development.

3.1. Evaluation Instrument

Since we aimed to investigate if the usage of executable prototypes in the early phases of software development can enhance mutual understanding and increase customer satisfaction, we chose participant observing, a technique of qualitative research, combined with the questionnaires as a method of quantitative research, to enhance the validity of our observations. The combination of technical and human aspects in software engineering research demands both qualitative and quantitative methods in order to use the strengths of both [21]. Participant observation can capture firsthand behaviors and interactions through the collaborative work of a researcher and participants, which might not be noticed using only quantitative methods [21].

To investigate if executable prototypes can shorten the requirements elicitation phase, we planned to compare the time needed to acquire mutual understanding with the time spent in sessions in which we did not use mockups in communication with customers [19]. The applications developed in both types of sessions were based on the same user interface guidelines and created using code generators, so the comparison was possible. In addition, one of the researchers was engaged in both session types so they could compare their experience.

During study preparation, we conducted several focus groups aimed to design basic propositions, questionnaires, and metrics for qualitative research. Once the preparation was finished, we performed a zero session with a participant who was a marketing specialist in a large corporation. Before the zero session, our expectation was that we would need at least four hours to elicit requirements and create the prototype, but it helped us realize that the process could be finished in even less time and that a two-hour session time limit was adequate. Furthermore, a discussion with the zero-participant enabled us to improve the questionnaires and the events that we planned to observe.

We prepared three questionnaires:

- **A: Background questionnaire**—used to obtain some general background information about the participants (Table 1).
- **B: Development session evaluation (15 questions)**—a questionnaire with fixed and open-ended questions regarding participant's opinions on the collaborative design session with the developers (Table 3).

- **C: Developed prototype evaluation (eight questions)**—a questionnaire regarding quality and ease-of-use of the generated prototype (Table 4).

Each questionnaire consisted of a number of closed questions and, with the exception of the background test, a free section for comments and suggestions. Participants answered each question by choosing an item on the five-point agreement scale [21] with the terms "Strongly agree" and "Strongly disagree" on the opposite sides of the scale. Statements in some questions were negated, and the questions in the same groups have been randomized to minimize the systematic responses. In summaries, answer items were coded with numbers one to five, where one corresponds to the statement "Strongly disagree" and five to "Strongly agree".

Table 1. Background questionnaire.

General Info			
Gender Age Occupation Education			
Questions		Scale	
aQ1	How skilled are you in using a personal computer?	5-point	Inexperienced—Skilled
aQ2	How would you rate your skill in using graphical editors?	5-point	Inexperienced—Skilled
aQ3	Does the company you work for have any information system which you use as a part of your everyday job? If the answer is YES, how would you rate your skill in using it?	5-point	Inexperienced—Skilled
aQ4	Did you attend any of the courses in information management or software modeling during your education?	Yes/No	
aQ5	How many times have you got the chance to participate in a software development process for your company or to cooperate with the software development team as a user or a client?	5-point	Never—Many times

Table 2. Background questionnaire results. Education level is based on International Standard Classification of Education (ISCED) 2011.

Session	Gender	Age	Institution	Job Description	Education	aQ1	aQ2	aQ3	aQ4	aQ5
S1	F	44	Bank	Head of non-performing loans bank division	Master or equivalent	4	2	5	NO	4
S2	F	51	Primary school	Teaching coordinator	Bachelor or equivalent	5	5	5	NO	4
S3	M	43	Theatre Museum	Museum collection archivist	Upper secondary education	5	5	5	YES	3
S4	F	42	Insurance company	Insurance Frauds Inspector	Bachelor or equivalent	4	3	5	NO	3
S5	M	43	Insurance company	Insurance Chief Underwriter	Bachelor or equivalent	4	2	4	YES	5
S6	F	30	University	Assistant professor	Doctoral	5	5	5	YES	3
S7	M	28	Computer repair shop	Computer technician	Upper secondary education	5	4	5	YES	4
S8	F	62	Tax administration	Tax inspector	Bachelor or equivalent	5	5	5	NO	3
S9	F	46	Music school	Professor of violin	Master or equivalent	3	3	5	NO	1
S10	F	41	Moving company	CEO	Bachelor or equivalent	4	2	5	NO	4
Average		43				4.4	3.6	4.9		3.4

Table 3. Development sessions evaluation questionnaire.

Questions		Scale	
bQ1	Working together with the developers helped me express my requirements.	5-point	Strongly agree—Strongly disagree
bQ2	Time spent with the developers was too long.	5-point	Strongly agree—Strongly disagree
bQ3	During that time we did a lot of work.	5-point	Strongly agree—Strongly disagree
bQ4	This way of working was exhausting.	5-point	Strongly agree—Strongly disagree
bQ5	It was hard to understand the developers' questions.	5-point	Strongly agree—Strongly disagree
bQ6	The developers had a hard time understanding my needs.	5-point	Strongly agree—Strongly disagree
bQ7	During the session, I felt uncomfortable and incompetent.	5-point	Strongly agree—Strongly disagree
bQ8	It would be easier for me to deliver the requirements on paper without direct communication.	5-point	Strongly agree—Strongly disagree
bQ9	I think that sketching the program on paper would be a better way to communicate with the development team.	5-point	Strongly agree—Strongly disagree
bQ10	Running the application prototype helped me notice what I need.	5-point	Strongly agree—Strongly disagree
bQ11	Entering the data in the prototype makes finding errors easier.	5-point	Strongly agree—Strongly disagree
bQ12	Entering the data in the prototype takes a lot of time.	5-point	Strongly agree—Strongly disagree
bQ13	I would like to continue participating in the development of software for my company in this way.	5-point	Strongly agree—Strongly disagree
bQ14	I think that I could make a specification using Kroki tool by myself, without the developers.	5-point	Strongly agree—Strongly disagree
bQ15	If you have ever participated in software development with another team, please write down your impressions compared to this experience.	Free text	
bQ16	Comments and suggestions.	Free text	

Table 4. Developed application prototype evaluation questionnaire.

cQ1	The concepts of the application prototype are easy to understand.	5-point	Strongly agree—Strongly disagree
cQ2	The application elements are easy to use.	5-point	Strongly agree—Strongly disagree
cQ3	I find the standardized look of the forms intuitive.	5-point	Strongly agree—Strongly disagree
cQ4	Navigating the application was easy using the provided user interface.	5-point	Strongly agree—Strongly disagree
cQ5	I think that I could use the developed application without previous training.	5-point	Strongly agree—Strongly disagree
cQ6	I think that I could use the developed application without the written manual.	5-point	Strongly agree—Strongly disagree
cQ7	I think that such an application would help me in my job.	5-point	Strongly agree—Strongly disagree
cQ8	I like the look of the application.	5-point	Strongly agree—Strongly disagree
cQ9	Comments, suggestions:	Free text	

After the session, every participant received the questionnaires by email and was asked to take their time to answer the questions clearly and honestly. We chose this way of gathering answers in order to enable them to better concentrate on the survey instead of filling it right after the session when they were perhaps exhausted from the process or feeling obstructed by our presence.

The following section presents a summary of the notes and observations taken during each session. The sessions and their application domains (project titles) are listed in Table 5.

The developed models which can be executed using Kroki, screenshots of the application prototypes, and the questionnaires are available in [23].

Table 5. Development sessions' application domains.

Session	Project Title
S1	Exposure and Collateral
S2	Professional development database
S3	Theatre Museum of Vojvodina
S4	Claims management
S5	Road assistance
S6	Printing office project management
S7	Computer repair shop
S8	Tax documentation management
S9	Performance and competition management
S10	Moving and relocation

3.2. Exposure and Collateral

The participant was a 44-year-old bank executive working on extensive credit management. She had an application that supported her daily routine, but it lacked some navigation and reporting features. The current application could not provide a combined and summary overview if a bank's client had several loan accounts.

Firstly, we discussed data needed for the combined overview. We jointly created several simple mockups (standard panels) that enabled managing the essential information about clients, loan accounts, currencies, collaterals, etc.

Then we combined standard panels into a parent-child panel and activated the prototype execution. The participant helped us enter some data and realized that the parent-child panel was not organized properly, and that some information is missing. We returned to the Mockup editor, fixed the mockups, and executed the prototype once more. This time, the combined overview was specified correctly.

Since we had to develop and validate de facto just one feature, this session was quite short—it was finished in 70 min and resulted in seven developed mockups. The prototype development in two iterations took 25 min. The participant satisfaction was very high. Her comment about the participatory development (bQ15 in the development sessions evaluation questionnaire) was the following: "From my previous experience, the project lasted for months and included a lot of people with different levels of understanding and ways of thinking. This experience is fantastic compared to my previous ones".

3.3. Professional Development Database

This case involved a 51-year-old teaching coordinator in an elementary school. The participant was responsible for collecting and recording data about teachers' professional development activities (seminars, workshops, publications, etc.). Application for logging such information was non-existent, so she used multiple sources and teacher submissions to put together an annual report for each teacher's achievements.

The session had similar dynamics to the previous one. After the initial introduction to the participant's domain and required features (55 min), we jointly specified the standard panels for managing teachers' data (basic info, occupation, job position, office location, etc.) and details about the competence improvement program. Then we created a parent-child panel to integrate teachers with their accomplishments in the given school year. After fixing the minor issues that we have discovered after launching the prototype, the participant confirmed that she acquired what she had in mind. The prototype development took 40 min and resulted in eight developed mockups in two iterations.

The session took more time than the previous one with a similar result, but it was not due to problems in communication. The current participant was just more elaborate than the previous one. She rated the experience as very satisfying, particularly praising the functional and straightforward design of the web prototype: "I like the simplicity and the functionality of the user interface, without any redundant elements".

We find this input significant since she had formal education in the fine arts, so we expected her to give some remarks on the overall aesthetics of the prototype. The participant has collaborated with the programmers before and has stated the following regarding that:

"I have been involved in the development of multiple software programs as a customer before. This is by far the best experience. This way, the customer gets a better insight into needed functionalities and becomes directly responsible for the developed application features. This tailor-made concept is an answer to many client and developer doubts. My only concern is that some future clients may consider your work too easy, unaware of all the effort made in the background".

3.4. Theater Museum of Vojvodina

In this case, the participant was a 43-year-old museum collection archivist in the Theater Museum of Vojvodina in Novi Sad. The museum executive board was considering digitization, so he was very enthusiastic about taking part in the session and discussing the needed features of the future system.

After an introduction that provided us with a big picture (50 min), we estimated that we could start with the mockup development. We had to provide an application for managing data about theaters (basic info, available stages), events performed in theaters, and events' contributors (actors, directors, scriptwriters, etc.). Contributors were coming from multiple countries, so events had to be organized in different languages.

Besides basic data entry, it was necessary to obtain the bibliography (published articles, monographs, etc.) for the contributors, events, and theaters.

The prototype development lasted 75 min and resulted in 14 mockups developed in four iterations.

The participant enjoyed this kind of development and was satisfied with the results. He was keen to learn how to use Kroki to be able to add mockups by himself if the need arose.

When asked to comment about the developed prototype, he stated the following: "In two hours which I spent learning the tool's functionalities and developing the prototype, I was able to participate in application development and see instant results with great ease. Also, my requirements were very easily applied. This tool greatly reduces the time needed to create an application and, most importantly, simplifies the communication between the customer and the developers. The client needs to have an insight into the modeling process". We organized another session to teach him how to use Kroki, helped him to specify some additional features, and exported the model to XMI so that another development team could get a starting point for the project.

3.5. Claims Management

This 42-year-old participant worked in an insurance company in the fraud detection department. She wanted to enhance the existing application she was using at her job. Her work extensively relied on the efficiently organized forms and advanced navigation features, which were not satisfying in the current system.

An insurance claim was the main topic, accompanied by insurance policy, policy holder, and other participants in the claim. A claim could be supplied with a number of supporting documents, each classified by type. It was also the basis for the claim calculation, including financial reimbursements in a periodic manner. The insurance policies were classified according to the insurance products and comprised risk and tariff descriptions. The application also had to support responsibilities in the insurance company according to geographic regions and organizational hierarchy.

After we finished the required features overview (50 min), we jointly created standard panels for managing data about the presented concepts. Then we tried to organize them into composite panels in order to provide her with features she lacked in her existing application. The final prototype specification consisted of 17 mockups, developed in five iterations in 70 min.

The participant reported she collaborated in the software development before and stated that the experience from the session "cannot be compared to the previous ones" (in favor of the Kroki tool).

3.6. Road Assistance

This session involved a 43-year-old economist also working in an insurance company. He wanted to develop a mobile application that would allow users to put together and buy a custom car insurance package at any time. Considering that Kroki currently does not support mobile application prototyping, the emphasis was on developing a prototype that would serve as the back-end for such an application.

The application was focused on road assistance as an insurance product. It comprised detailed information about vehicle characteristics relevant from the standpoint of insurance and a policy holder. The road assistance product references a number of scopes of covers, categorized into packages, and includes tariffs with time validity periods and vehicle types.

The participant had attended a one-semester course in information systems during his education and has participated in the development process as a customer many times (he ranked himself as a five in the background survey). Thanks to his experience, there was little need for prototype execution during the session—he could easily comprehend all necessary details from the mockups. His requirements were noticeably clear, so it was straightforward for us to understand what he wanted.

This session was the most productive of all. At the beginning of the session, the participant introduced us to his business domain in 40 min. Then, we jointly created 20 mockups in 90 min in two iterations.

The participant stated that writing formal requirement specifications was a problem for him and found this approach very creative, simple, and time-saving.

3.7. Printing Office Project Management

The participant was a 30-year-old assistant professor with a degree in graphics design and engineering. As a member of a printing office at her faculty, she reported that a large number of printing office projects had problems because of a lot of minor mistakes made during the printing preparation. It was the consequence of informal communication between their customers and printing office staff, performed mainly by telephone and e-mail. The participant had expressed the need for project management software that would provide a custom workflow of printing preparation phases for every project. There was a known set of phases, and for every printing project there was a need to specify which phases were contained, and in which order.

This was challenging and we spent almost 60 min trying to understand the participant's needs in the first part of the session. However, once the mockups development started, we rapidly made progress and even came up with a useful new feature that the participant initially did not have in mind. It was a support for common project types with predefined set of phases.

The session resulted in 16 mockups developed in three iterations in 60 min.

3.8. Computer Repair Shop

In this case, the participant was a 28-year-old technician that was working in a computer repair shop. He was not satisfied with his current repair shop application, in which data was entering mostly in free text forms. The application's main purpose was to keep an informal track of performed repairs to help with their customer's complaints and reclama-

tions. Pricing and invoicing features were not supported—technicians used to read prices from printed lists.

In order to support technicians to work more efficiently, we provided support for managing data about employees, customers, repair shop services and their prices, receipts for received devices, etc.

The session was relatively short and lasted 80 min. The introductory part of the session took 45 min. The prototype development took 35 min in which we specified 13 mockups in two iterations. This was mainly the result of our familiarity with the domain of computer equipment. In addition, the participant knew precisely what he wanted.

3.9. Tax Documentation Management

The participant, a retired 62-year-old tax inspector, wanted to develop a web application that would help taxpayers organize needed documentation based on their tax profile. Throughout her career, she has encountered many taxpayers with incomplete or faulty tax documentation because they had trouble getting the needed information. This application would be available to all taxpayers and allow them to log in and manage all their tax-related activities. Inspectors could also use the application to track taxpayer's activity.

This session was challenging for us. We spent most of the time trying to communicate the domain specifics and the details of the requirements. We had an impression that there was a considerable amount of tacit knowledge that the participant could not convey. Furthermore, since the participant has been retired for two years and was currently occupied with a different career, she was constantly checking the official websites for new information, which also took more time. In an attempt to reach mutual understanding, we were making small iterations. We would create a mockup for a feature we thought that we comprehended, launched the application, and asked her how to populate it with data. Often, she did not know the answer, which was a sign that mutual understanding was not acquired.

The session was unsuccessful regarding our understanding of the requirements. However, the participant said that the developed application was what she had in mind and expressed a positive opinion in the questionnaires. We assumed that she probably thought she would hurt our feelings if stated otherwise. We had an interview with her a few days after the session, and she confirmed our assumption—the session was hard for her as well.

The initial part of the session lasted 40 min. The prototype development took 80 min and resulted in nine mockups developed in 14 iterations.

3.10. Performance and Competition Management

This participant, a 46-year-old professor of violin, wanted to develop an application that would help her keep records of students in her class. She wanted to be able to prepare and enter a specific program for each of her students based on their abilities and ambitions. The student program consisted of a list of compositions the student had to master in the given school year. She also wanted to track the class involvement in music competitions and other performances. A student group taking part in a competition or performance could be assembled of students and professors from different music schools. Students and professors could take part in several performing groups with different roles and instruments. It is important to know the time, place, participants, and composition that was performed.

She had the needed data in multiple documents, printed and digital, and the software that would help her keep it in one place would be beneficial.

After an initial discussion that took 45 min, we started with prototype development. We built 18 mockups in six iterations. It took 70 min.

Every time the prototype was launched, the participant got more ideas about what could be added next, so the prototype was gradually enhancing. The result was very satisfying for the participant. After the session, we installed the application prototype on her laptop since she found it helpful even in the current state of development. She commented that she really enjoyed the time spent in development and found the whole process highly creative. We had the same impression.

3.11. Moving and Relocation

The participant was a 41-years-old CEO of a company that provided moving and storage services. She wanted to develop a subsystem for managing external workers and their assignments to work orders. The subsystem aimed to keep data about workers such as their name, address, contact information, availability, skills, uniform size, etc. It was also essential to enable support to assign workers to work orders and record their work log (the time they spent on each type of service within the work order).

The introductory part of the session lasted 50 min. Then, we started with the collaborative prototype development. Firstly, we developed several simple mockups that enabled workers' information management and a parent-child mockup that enabled the manager to grasp the most important worker's information at a glance. The prototype was launched, we entered some data and realized that additional fields were needed and that the parent-child form should be organized differently. After the repair, the prototype was restarted, and the participant was satisfied with the result.

Then, we developed mockups for work orders, workers' assignments, and work log tracking. This part was much shorter, as we understood the domain better, and the participant had learned about our way of work.

The finished prototype consisted of 11 mockups developed in 60 min in two iterations.

The participant had detailed knowledge of her business domain, her requests were precise, and she did not lose time to recall what information the subsystem should provide. Communication was easy and understanding was mutual. After the session, she said she would like to have this type of development in her company. She complained about the rigid rules that their IT department implemented when new requests appeared, which implies that the entry of a request and its detailed description in the requests tracking system and waiting for its implementation, is a long time (even several months).

4. Discussion

We carried out 10 two-hour sessions with participants from various domains. During the limited time, we obtained a functional specification of the applications that the participants had in mind in 9 out of 10 cases. The size of applications ranged from 7 to 20 mockups. Based on our observations, participant experience was mostly positive, which greatly affected the commodity and the working pace, making the development less stressful for all parties. Results from Table 6 and 7 confirm our observations.

Table 6. Development sessions evaluation results.

Ses.	Project Title	bQ1	bQ2	bQ3	bQ4	bQ5	bQ6	bQ7	bQ8	bQ9	bQ10	bQ11	bQ12	bQ13	bQ14
S1	Exposure and Collateral	5	1	5	1	1	1	1	1	1	4	5	1	5	2
S2	Professional development database	5	1	5	1	1	1	1	1	1	5	5	1	5	1
S3	Theatre Museum of Vojvodina	5	1	5	1	1	1	1	1	1	5	5	1	5	2
S4	Claims management	5	1	5	1	1	1	1	1	1	5	5	1	5	5
S5	Road assistance	5	1	5	1	1	2	1	1	1	5	5	1	5	1
S6	Printing office project management	5	1	5	2	2	1	1	1	1	5	4	2	4	5
S7	Computer repair shop	5	1	5	2	1	1	2	1	2	5	5	1	5	1
S8	Tax documentation management	5	1	5	1	1	1	1	1	1	5	3	1	5	1
S9	Performance and competition management	5	1	5	1	1	2	2	1	1	5	5	1	5	1
S10	Moving and relocation	5	1	5	1	1	1	1	1	1	5	5	1	5	1
Average		5	1	5	1.2	1.1	1.2	1.2	1	1.1	4.9	4.7	1.1	4.9	2

Table 7. Developed application prototype evaluation results.

Session	Project Title	cQ1	cQ2	cQ3	cQ4	cQ5	cQ6	cQ7	cQ8
S1	Exposure and Collateral	5	5	5	5	3	5	5	5
S2	Professional development database	5	5	4	4	5	5	5	5
S3	Theatre Museum of Vojvodina	5	5	5	5	3	3	4	4
S4	Claims management	5	5	5	5	1	1	5	5
S5	Road assistance	5	5	5	5	5	5	5	5
S6	Printing office project management	5	4	4	3	2	4	3	4
S7	Computer repair shop	5	5	4	4	1	1	5	4
S8	Tax documentation management	5	5	5	5	1	1	5	5
S9	Performance and competition management	5	5	5	5	1	4	5	5
S10	Moving and Relocation	5	5	5	5	1	1	5	5
Average		5	4.9	4.7	4.6	2.3	3	4.7	4.7

Once the participants understood how their ideas can be turned into mockups, they were able to give more concise requirements and review some of the previous ones. By navigating the forms populated with their actual data, customers could spot some fundamental issues in the prototype that were overlooked in the previous discussion (missing data components, data components in wrong places, unsuitable forms' organization, inadequate navigation, etc.).

We noticed the tendency of less frequent prototype execution for more experienced participants (the ones that had previously participated in software development as a customer or the ones that work on executive positions in their company). During the sessions, those participants were able to comprehend most of the concepts and details during the mockup development.

We had a hard time understanding participants in some sessions and vice versa, which also reflects real situations in initial meetings with customers. However, as soon as mockup development and execution were started, mutual understanding was gained in 9 out of 10 sessions.

Table 8 shows the summary of development sessions. The average number of mockups per session in the specified prototypes is 13.3, which is expected given the time limitation of the sessions. The average time spent on each mockup is 4.67 min.

Table 8. Development session summary. The duration is given in minutes.

Session	Project Title	Number of Mockups	Number of Iterations	Session Duration	Prototype Development Duration	Mockups per Minutes	Successful?
S1	Exposure and Collateral	7	2	70	25	3.57	yes
S2	Professional development database	8	2	95	40	5	yes
S3	Theatre Museum of Vojvodina	14	4	125	75	5.36	yes
S4	Claims management	17	5	120	70	4.12	yes
S5	Road assistance	20	2	130	90	4.5	yes
S6	Printing office project management	16	3	120	60	3.75	yes
S7	Computer repair shop	13	2	80	35	2.69	yes
S8	Tax documentation management	9	14	120	80	8.89	no
S9	Performance and competition management	18	6	105	45	2.5	yes
S10	Moving and Relocation	11	2	110	70	6.36	yes
Average		13.3	4.2	107.5	59	4.67	

In our previous approach based on UML modeling with general-purpose modeling tools and code generation from UML class diagrams using AppGen [19], we needed from one to several days before we were able to generate an executable prototype (Figure 15, AppGen part). Our approach was, as soon as the requirements elicitation session with customers was finished, to proceed with a design session for the specification of a data model in the form of a class diagram. Vagueness and shortcomings in user requirements

were usually detected during the design, but we had to wait until the following requirement session to get the needed clarifications. Each session usually lasted several hours.

Development based on executable mockups gains a dramatic improvement (Figure 15, Kroki part) since we can perform several rounds of elicitation, design, and prototype validation in a single session, using the means understandable to all parties.

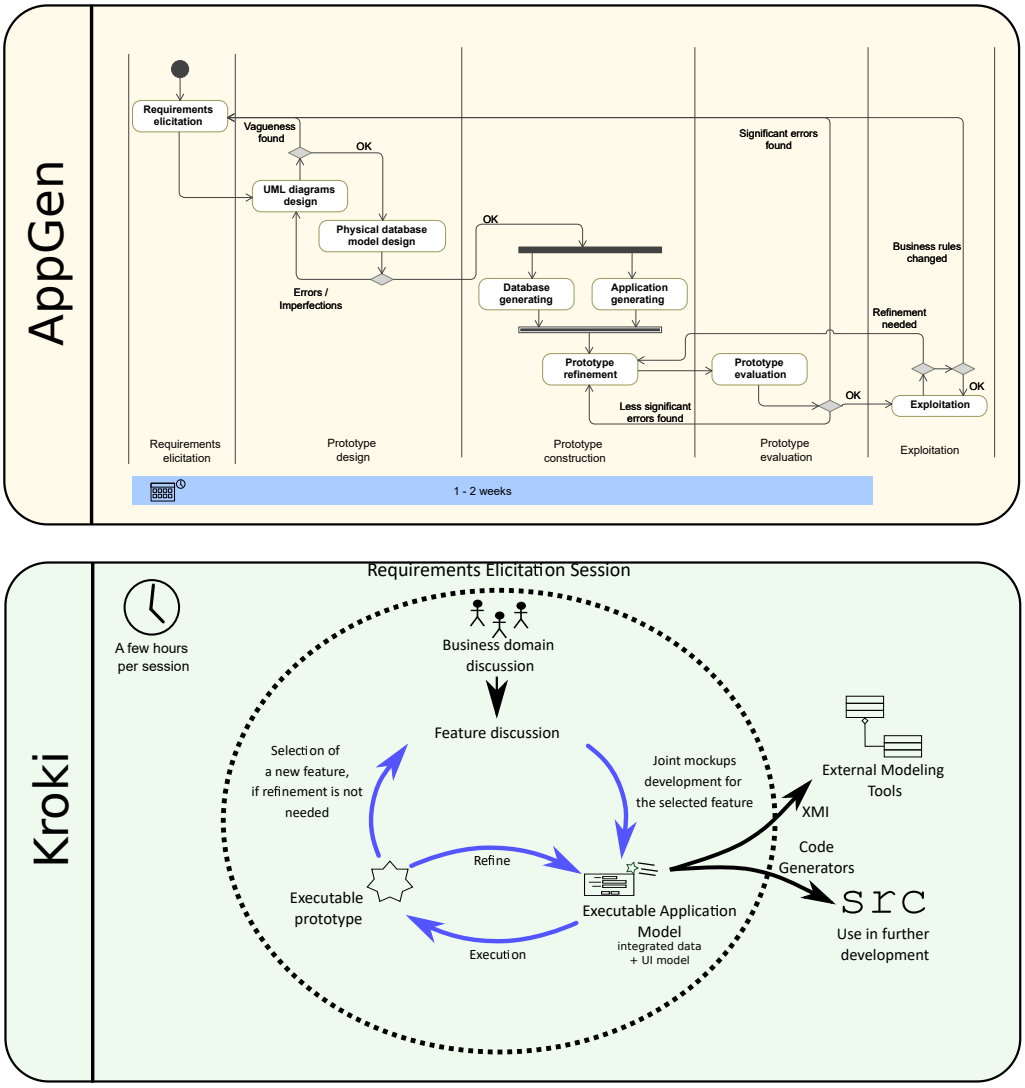

Figure 15. A comparison of development processes using the AppGen [19] and Kroki tool.

Of course, the development of an actual application would take more time. In the case of a real system, before we could use Kroki in development sessions, we would need to perform the initial phase to obtain the system's overview, decomposition, and non-functional requirements, as described in Section 2.7.

During the session, we were populating only basic settings (Figure 2)—advanced and persistence settings were omitted. We also needed at least an hour after each session to write down detailed requirements.

From our point of view, the sessions were very intensive. It took considerable effort to communicate with participants, memorize their requirements, and implement them in the mockups, having the domain model in mind. We performed sessions in pairs, which helped us to divide the work. For real development, we think that additional developers could be helpful to prevent quick tiring and potential errors. Still, the development team probably could not perform more than two sessions a day. In our opinion, this is enough, given the speed of work and the level of understanding acquired in such sessions.

In order to ensure the validity of our study, we used triangulation—a combination of data extracted from qualitative and quantitative methods: Participative observation, questionnaires, and comparison with the previous sessions in which the Kroki tool was not used. During sessions, both researchers were taking notes, so they were able to compare the notes and their impressions. Kroki's message log window was helpful for time tracking since it displays the time for every prototype launch. The study was carefully prepared, using several focus groups and a zero session. We performed 10 sessions. Nine of ten sessions were successful regarding acquiring mutual understanding.

The quality of the produced data model and development speed depends on the skills of developers participating in the sessions. Someone fluent in requirements elicitation and conceptual modeling will immediately create a quality solution that can be used for further development. An inexperienced developer will probably create a solution that needs additional refactoring, but it is still enough to serve as a basis for communication with customers.

5. Related Work

A mockup-driven enhancement to a standard agile development life-cycle, called MockupDD, is presented in [26]. It extends a technique and a tool introduced in [27]. A MockupDD process starts with the creation of user stories based on the brief requirements specification phase. Then, clients and the development team build screen mockups as a graphical representation of the user stories. Mockups are created using general-purpose mockup tools, imported, and annotated in order to produce a Structural User Interface (SUI) model. This model serves as a basis for obtaining a WebML model of a working web application.

MockupDD is intended to be a tool for building general-purpose web applications, so many of its concepts are specific to the web domain. Compared to our approach, MockupDD needs additional post-processing in order to transform mockups into SUI. In addition, MockupDD does not have the means to visualize data or UI models, which could lead to harder comprehension of the developed application.

The described approach is later evolved into a process called DataMock [28] which also uses user-annotated sketches to generate data models.

Similar variations of mockup-driven development can be found in [29]. Rough sketches of a storyboard are created during participative development and then scanned, manually annotated, and imported into Window/Event Representation (WED) using the AIDE tool. WED is a UI specification notation based on UML 2 state machines that can be used to specify the layout and the behavior of the UI. Each graphical element (a window, a dialog, etc.) is represented as a state, while the events on those elements initiate state transitions. AIDE generates XUL (XML User Interface Language) code which can be executed in a web browser.

In the aforementioned papers, mockups need to be imported and, using additional graphical or textual annotations, transformed into software models and then into working code. In contrast to this, screen mockups developed in Kroki are elements of our EUIS DSL, so no additional processing or transformation needs to be done to obtain a working prototype, which contributes to the development speed required for collaborative sessions.

In [30], a UI prototype is developed as a result of a series of semi-automatic transformations performed by the Marama AI tool. Taking written textual requirements specification as a starting point, the Essential Use Case (EUC) diagram is extracted and then mapped to an abstract Essential User Interface (EUI) model, which is a base for generating a concrete form-based UI. Each generated model has to be examined and manually adjusted before the next step is performed.

In [31], an approach is presented of using model-driven techniques in order to create a prototype for large scale Enterprise Information Systems. The process begins with a model of the data structure defined in one DSL, which is used to generate the backend part of the target application, as well as a basis for model-to-model transformation into another DSL, targeted toward GUI specification, which is subsequently used in order to generate the front-end part of the enterprise application. The result is a functional prototype of the system being developed, which can be used for validation with end users. Included mechanisms provide the means for this prototype to be further refined and not thrown away.

This approach has a similar vision to Kroki. One advantage that we see in Kroki is that it does not have the need for model-to-model transformation while work is done in the tool itself, but it enables the import and export of the specification as general purpose UML (*Unified Modeling Language* [32]) via XMI (*XML Metadata Interchange* [33]).

The paper [34] presents a first proof of concept that targets an open issue in the industry regarding automating the reuse of information included in prototypes developed with clients in the early stages of software development. Since those early prototypes are usually discarded, many times, the final product does not represent what the client expects. The authors propose to build a tool that transforms prototypes into requirements and user interface models. For the validation, they built a prototype consisting of two mockups using PowerPoint, exported it to XMI, imported it into Enterprise Architect, and discussed results with two companies.

We agree with the findings presented in [34]. Kroki enables the reusability of developed mockups both in data models and code. However, we do not use the transformation from mockups to models since the mockups are part of the concrete syntax of EUIS DSL.

A model-oriented technology called Umple that can be used for rapid prototyping is presented in [35]. Umple end-users need to create class and state machine models using textual syntax. Code generation uses many heuristics for transforming these models into a usable UI similar to our solution.

WebRatio [36,37] is a professional tool that started as a set of Eclipse plugins and grew into four separate platforms: Mobile, BPM, web, and enterprise. WebRatio models are based on IFML language, but the modeled concepts are on a lower abstraction level and harder to learn than the ones used in EUIS DSL and the Kroki tool. It is expected since WebRatio's purpose is to develop general-purpose and fully-functional business applications.

CUBA Platform [38] is another model-driven tool intended for the rapid development of web information systems. It is free and open-source. The generated CUBA Platform's application prototype has a similar look to the prototype developed using the Kroki tool. The prototype can be manually enhanced to support a rich HTML interface with advanced widgets such as data graphs, interactive maps, etc. The CUBA platform is not designed for collaborative development with customers because developers need to elicit requirements, transform them into a data model using dialogs for specification of persistent classes, and finally generate and customize the prototype.

Prototizer is an open-source tool that can generate a working prototype of a web application from a MagicDraw class diagram [39]. Prototizer supports the iterative development and provides integration with manually written code. Similar to the CUBA platform, the customers can take part when modeling and code generation is finished.

A DSL called *Application Specification Language* (ASL) can be used for the improvement of the requirements in the engineering process [40]. It provides constructs for the definition of `DataEntities`, which can then be grouped into `DataEntityClusters`. `UseCases` and user interface elements can also be defined in ASL. A series of model-to-model and model-to-code transformations can then be performed using the proposed approach, which enables model validation and generation of an executable application for the Django framework. The generated application can utilize the *Admin* site provided by Django for user permission control.

XIS-Web [41] is a model-driven approach implemented as a UML profile and XIS-Web framework comprising a set of integrated software tools. It has been built on top of the Sparx System Enterprise Architect and Eclipse Modeling Framework for the M2M and M2T transformations. XIS-Web supports a "smart" mode where the designer only needs to define the `Domain`, `BusinessEntities`, `Actors`, and `UseCases` view, while the rest is automatically generated based on a predefined set of UI patterns. The language and the framework do not provide the means to create UI in a WYSIWYG style (e.g., a form designer). The evaluation is done on two use-cases: (1) Management of TimeSlots booked by students and (2) management of personal documents. The use-cases were implemented both manually and using the XIS-Web framework. The percentage of generated lines of code vs. lines of code in manually-implemented applications ranges from 58% to 87%, depending on the use-case and the target language. The presented pilot study is based on a group of participants with at least a Bachelor in Computer Science or Software Engineering degree, as the language and the framework are oriented towards Web application developers.

Enterprise WAE (E-WAE) [42] is a lightweight UML extension for modeling of Enterprise Web Applications with a focus on the presentation tier and the ability to characterize navigation maps with RBAC (*Role-Based Access Control* [18]) support. It comprises 22 stereotypes. The PIM model is automatically transformed to PSM (currently Java Server Faces and ASP.NET). The approach enables a quick modeling of the presentation tier, thus having the appeal of mockup-based development but without the cost of mockup disposal. However, the layout of the UI forms must be defined manually.

Comparison of the discussed tools is presented in Table 9. Most of the tools require some form of a domain model as an input to code generators. This kind of approach enables the rapid development but is not suitable for collaborative sessions with customers. The developers must create a domain model based on elicited requirements and generate code before the customers can participate in the generated prototype evaluation.

The tools that take mockups as input need additional processing: Scanning or parsing, importing and annotating to transform mockups to software models. Mockups developed in Kroki need no additional processing to provide a working prototype, contributing to the development speed.

An overview of the benefits of incorporating mockup-driven techniques into the requirements specification process is given in [43,44]. In the survey presented in [43], the responses from 92 industry participants showed that low-fidelity prototyping is the most used UI technique (68.48%) in the requirements gathering phase. Still, the reusability of the created sketches is low in the later development phases. The same survey showcased a growing adoption of agile methodologies in the industry, with Scrum being most commonly used, followed by Extreme Programming. In addition to these findings, the main driving force for our work has been a growing interest in incorporating User-Centered Design (UCD) into existing agile workflows and a lack of empirical research on this topic [45,46].

In [45], a case study of a large software project in Germany is presented. The authors followed an agile development team called AgDev, which was working on a project for a government company called WaterWorks. The user involvement was fostered by conducting acceptance tests which took place in between the agile development iterations. The study gives valuable insight into the real difficulties of converting requirements into UI design due to the lack of collaboration between involved parties.

Table 9. Comparison of mockup tools.

Tool	Design Starts with Development of	Mockups Are Developed by	Needs Additional Manual Processing of the Mockups?	Requires Development of Mockup/Model Parsers?	Needs Model-to-Model Transformations?	What Parts of the Application Are Generated?	Supported Immediate Execution?
MockupDD [26]	Mockups	General purpose mockup tools, like Balsamiq or Pencil	Yes	Yes	Yes	Running prototype /MDWE Models (such as WebML)	No
DataMock [28]	Mockups	General purpose mockup tools	Yes	Yes	Yes	UML Class Diagrams	No
AIDE [29]	Mockups	Paper and pencil	Yes	No	No	Generated XUL (XML User Interface Language) code	No
Marama AI [30]	EUI (Essential UI) model developed in MaramaEUI editor	-	-	-	Yes	EUC (Essential Use Case) Model and a simple HTML prototype	No
MontiGEM [31]	Data structure model	-	-	-	Yes	Extendable working prototype with CRUD functionalities	No
SocietySoft [34]	Mockups	PowerPoint	No	Yes	Yes	Navigable UI prototype for EnterpriseArchitect	No
Umple [35]	Textual Data model	-	-	-	Yes	UI prototype with CRUD functionalities	Yes
WebRatio [36,37]	WebML data model	-	-	-	Yes	Running application	No
Prototizer [39]	MagicDraw class diagram	-	-	-	No	SQL scripts, model classes and UI code	No
ASL-based tools [40]	Textual Data and Use Case model	-	-	-	Yes	Django model classes with automatic admin (CRUD) pages	No
XIS Web [41]	Domain View, Business Entities View, Architectural View, and Use Cases View	-	-	-	Yes	Web prototype with CRUD functionality	Yes
Enterprise WAE [42]	UML diagrams with WAE UML profile	-	-	-	Yes	JSF or .NET web prototype	No
CUBA Platform [38]	Application specification in CUBA studio	-	-	-	No	Running Java Script web application	Yes
AppGen [19]	Data model	-	-	-	Yes	Desktop application, database scripts	No
Kroki	Mockups	Kroki mockup editor	No	No	No	Desktop or web application, Data model in XMI	Yes

Similar research presented in [46] discusses the trend of incorporating UCD into Scrum and Extreme Programming (XP). The study follows three development teams within the same company working on three different projects and aims to develop a set of principles helpful in conducting a user-centered development workflow. As the authors stated, a field study within just one company does not provide sufficient data for any meaningful validation. The proposed framework has to be regarded as a set of guidelines rather than a strict collection of rules. Nonetheless, the explicit emphasis on user involvement and prototyping (the designers must be willing to "feed the developers" with prototypes and user feedback on a cycle that works for everyone involved [46]) greatly corresponds with the principles that form the foundation of our approach.

In the family of experiments presented in [44], the effectiveness of augmenting Use-Case diagrams with screen mockups was measured using software engineering students. The provided study represents the replication of the previously conducted controlled experiment, which showed that screen mockups could almost double the efficiency of model comprehension. The replication study confirmed the results of the original research. Even though the experiments from [44] aim to quantify the impact of adopting screen mockups on users with a software engineering background and are focused on model comprehension improvement, it provides useful insight regarding the benefits of screen mockups usage in requirements engineering.

A recent study that addresses state-of-the-art in the field of prototyping tools and presents their comparative analysis is given in [47]. The publication underlines the benefits of using software prototypes as a communication tool and the problem of treating them as disposable artifacts that many stakeholders consider as a necessary but not beneficial step in software development. Based on this, the authors have set up a list of research questions and conducted a comparative analysis of the 10 selected prototyping tools in order to quantify whether they live up to the promise of faster and more automatic software prototype creation and reusability. Detailed results can be found in [47], but the main takeaway is that the research questions are poorly addressed in the tested tools: Only one tool has four positive answers to 10 available questions, while the rest have less than that.

Regarding our research, this paper provides beneficial insight into the recent state-of-the-art in the prototyping tooling scene. Still, some improvements could be performed to enhance the contribution to the research field:

- Tool selection could be refined to include academic tools and domain-specific tools;
- The addition of research questions regarding the developed prototypes' nature and quality could improve the survey.

The authors in [47] also recognize those shortcomings and stress that the provided research is just preliminary and will be used to lay the ground for more detailed ones in the future.

6. Conclusions

This paper presented a brief overview of the design of Kroki—a tool that enables collaborative sessions with customers. The needed agility is supported by:

- EUIS DSL based on our HCI guidelines, focused on a minimal set of course-grained UI components;
- The mockup-based concrete syntax that enable customers to actively take part in the prototype specification;
- The integrated meta-model of EUIS-DSL, which helps to avoid model-to-model transformations. The transformations usually slow down development and prevent immediate model execution;
- Aspect-oriented engines and code generators which produce a fully-functional, three-tiered application prototype even when minimal specification details are obtained (default code generation action exists for every non-specified functionality);

- Administration subsystem based on the RBAC standard that enables flexible access control policies and dynamic configuration of deployed application and its elements based on user roles; and
- The option of reusing artifacts across development phases, in order to reduce waste of time and effort.

We used Kroki in an exploratory case study performed to investigate what can be achieved if we were to start to use executable mockups from initial functional requirements elicitation sessions with customers. We conducted 10 two-hour development sessions with domain experts from different business domains and developed an executable application prototype ranging from 7 to 20 mockups in each session. Nine of ten sessions were successful regarding achieving mutual understanding and development of the application that the participant had in mind.

The development based on executable mockups gained a dramatic improvement compared to our previous approach based on class diagram specification with general-purpose modeling tools [19]. Using Kroki, we could perform several rounds of elicitation, design, and prototype validation in a single session.

6.1. Perceived Limitations

A focus on enterprise applications created with a minimal set of coarse-grained building blocks enabled the development speed and effective inclusion of participants in development sessions. Although this type of business application has been successfully tested in practice [10], it is possible that some teams or business domains would benefit from a different or extended set of building blocks.

Kroki's engines and code generators currently support the development of three-tiered Java web and desktop applications. It is sufficient for functional requirements elicitation based on executable prototypes. Still, it is constraining if the development team would want to use Kroki for automating the construction phase based on a different technological platform.

To achieve the best results regarding development efficiency and the quality of work, the development team must have at least one experienced developer who can efficiently map the requirements to mockups and the corresponding data model. Some form of automatic validation that could help detect redundancy and other "bad smells" during modeling could be needed for less experienced development teams.

Kroki currently does not support the specification of functional and non-functional requirements in textual form. Its development products are mockups and data model specification as well as generated code in Java.

6.2. Future Work

Since we achieved encouraging results, we plan to adjust our previous process described in [19] to incorporate joint mockups development and to apply it to the development of a real-world enterprise application.

We also plan to develop a module in Kroki for the specification of functional and non-functional requirements in textual form to integrate developed mockups with requirements.

Exporting specifications built in Kroki to WebML [37] is under development. As a tool intended for the automated development of general-purpose enterprise applications, WebRatio could be utilized for the construction phase using a technological platform not supported by Kroki.

Author Contributions: Conceptualization, G.M. and M.F.; methodology, G.M. and M.F.; software, M.F., G.M. and Ž.V.; validation, G.M. and I.D.; investigation, M.F. and Ž.V.; data curation, M.F.; writing—original draft preparation, M.F. and G.M.; writing—review and editing, M.F., I.D., Ž.V. and G.M.; visualization, M.F., I.D., Ž.V. and G.M.; supervision, G.M. and I.D. All authors have read and agreed to the published version of the manuscript.

Funding: The research is partially funded by Ministry of Education, Science, and Technological Development of the Republic of Serbia.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Kroki is a free and open-source software [6]. Data presented in this research is publicly available [23]. Additional information is available from corresponding authors upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Memmel, T.; Gundelsweiler, F.; Reiterer, H. Agile human-centered software engineering. In Proceedings of the BCS-HCI'07: 21st British HCI Group Annual Conference on People and Computers, Lancaster, UK, 3–7 September 2007 ; pp. 167–175.
2. Inayat, I.; Salim, S.S.; Marczak, S.; Daneva, M.; Shamshirband, S. A systematic literature review on agile requirements engineering practices and challenges. *Comput. Hum. Behav.* **2015**, *51*, 915–929. [CrossRef]
3. Mannio, M.; Nikula, U. *Requirements Elicitation Using a Combination of Prototypes and Scenarios*; Technical Report; Telecom Business Research Center Lappeenranta, University of Technology: Lappeenranta, Finland, 2001; ISBN 951-764-528-7.
4. Sukaviriya, N.; Sinha, V.; Ramachandra, T.; Mani, S.; Stolze, M. User-centered design and business process modeling: Cross road in rapid prototyping tools. In Proceedings of the IFIP Conference on Human-Computer Interaction, Rio de Janeiro, Brazil, 10–14 September 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 165–178.
5. Milosavljević, G.; Filipović, M.; Marsenić, V.; Pejaković, D.; Dejanović, I. Kroki: A mockup-based tool for participatory development of business applications. In Proceedings of the 2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT), Budapest, Hungary, 22–24 September 2013; pp. 235–242.
6. Kroki Tool. 2021. Available online: http://www.kroki-mde.net/ (accessed on 18 August 2021).
7. Perišić, B.; Milosavljević, G.; Dejanović, I.; Milosavljević, B. UML profile for specifying user interfaces of business applications. *Comput. Sci. Inf. Syst.* **2011**, *8*, 405–426. [CrossRef]
8. Lim, Y.k.; Pangam, A.; Periyasami, S.; Aneja, S. Comparative analysis of high-and low-fidelity prototypes for more valid usability evaluations of mobile devices. In Proceedings of the 4th Nordic Conference on Human-Computer Interaction: Changing Roles, Oslo, Norway, 14–18 October 2006; pp. 291–300.
9. Vaderna, R.; Dejanović, I.; Milosavljević, G. Grad: A new graph drawing and analysis library. In Proceedings of the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), Gdansk, Poland, 11–14 September 2016; pp. 1597–1602.
10. Kremenović, N.; Vukmir, S.; Dacešin, R. *Lanaco Monograph*; LANACO Information Technology Ltd.: Banja Luka, Bosnia and Herzegovina, 2015.
11. Milosavljević, G.; Ivanović, D.; Surla, D.; Milosavljević, B. Automated construction of the user interface for a CERIF-compliant research management system. *Electron. Libr.* **2011**, *29*, 565–588. [CrossRef]
12. OMG Object Constraint Language (OMG OCL). Version 2.4. 2014. Available online: https://www.omg.org/spec/OCL/ (accessed on 18 August 2021).
13. Demuth, B. The Dresden OCL toolkit and its role in Information Systems development. In Proceedings of the 13th International Conference on Information Systems Development (ISD'2004), Vilnius, Lithuania, 9–11 September 2004; Volume 7.
14. Filipović, M.; Kaplar, S.; Vaderna, R.; Ivković, Ž.; Milosavljević, G.; Dejanović, I. Aspect-oriented engines for Kroki models execution. In Proceedings of the 5th International Conference on Information Society Technology and Management (ICIST), Kopaonik, Serbia, 8–11 March 2015; pp. 502–507.
15. Java Persistence API. 2021. Available online: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html (accessed on 18 August 2021).
16. Filipović, M.; Vaderna, R.; Ivković, Ž.; Kaplar, S.; Vuković, Ž.; Dejanović, I.; Milosavljević, G.; Ivanović, D. Application of Kroki mockup tool to implementation of executable CERIF specification. *Procedia Comput. Sci.* **2017**, *106*, 245–252. [CrossRef]
17. Kaplar, S.; Filipović, M.; Milosavljević, G.; Sladić, G. Kroki Administration Subsystem Based on RBAC Standard and Aspects. In Proceedings of the 5th International Conference on Information Society Technology and Management (ICIST), Kopaonik, Serbia, 8–11 March 2015; pp. 61–66.
18. Ferraiolo, D.; Kuhn, D.R.; Chandramouli, R. *Role-Based Access Control*; Artech House: London, UK, 2003 .
19. Milosavljević, G.; Perišić, B. A method and a tool for rapid prototyping of large-scale business information systems. *Comput. Sci. Inf. Syst.* **2004**, *1*, 57–82. [CrossRef]
20. Vuković, Ž.; Milanović, N.; Vaderna, R.; Dejanović, I.; Milosavljević, G.; Malbaša, V. Semantic-aided automation of interface mapping in enterprise integration with conflict detection. *Inf. Syst. E Bus. Manag.* **2017**, *15*, 305–322. [CrossRef]
21. Shull, F.; Singer, J.; Sjøberg, D.I. *Guide to Advanced Empirical Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2007.
22. Flyvbjerg, B. Five misunderstandings about case-study research. *Qual. Inq.* **2006**, *12*, 219–245. [CrossRef]
23. Filipović, M.; Milosavljević, G. Rapid Requirements Elicitation Sessions Based on Executable Mockups. Dataset on Mendeley. 2021; V1. Available online: https://data.mendeley.com/datasets/kbh7hjttzy/1 (accessed on 18 August 2021).

24. Höst, M.; Regnell, B.; Wohlin, C. Using students as subjects—A comparative study of students and professionals in lead-time impact assessment. *Empir. Softw. Eng.* **2000**, *5*, 201–214. [CrossRef]
25. Alhaag, A.A.; Savic, G.; Milosavljevic, G.; Segedinac, M.T.; Filipovic, M. Executable platform for managing customizable metadata of educational resources. *Electron. Libr.* **2018**, *36*, 962–978. [CrossRef]
26. Rivero, J.M.; Grigera, J.; Rossi, G.; Luna, E.R.; Montero, F.; Gaedke, M. Mockup-driven development: Providing agile support for model-driven web engineering. *Inf. Softw. Technol.* **2014**, *56*, 670–687. [CrossRef]
27. Rivero, J.M.; Rossi, G.; Grigera, J.; Burella, J.; Luna, E.R.; Gordillo, S. From mockups to user interface models: An extensible model driven approach. In Proceedings of the International Conference on Web Engineering, Vienna, Austria, 5–9 July 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 13–24.
28. Rivero, J.M.; Grigera, J.; Distante, D.; Montero, F.; Rossi, G. DataMock: An Agile Approach for Building Data Models from User Interface Mockups. *Softw. Syst. Model.* **2019**, *18*, 663–690. [CrossRef]
29. Störrle, H. Model driven development of user interface prototypes: An integrated approach. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, Copenhagen, Denmark, 23–26 August 2010; pp. 261–268.
30. Kamalrudin, M.; Grundy, J. Generating essential user interface prototypes to validate requirements. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, 6–10 November, 2011; pp. 564–567.
31. Gerasimov, A.; Michael, J.; Netz, L.; Rumpe, B.; Varga, S. Continuous transition from model-driven prototype to full-size real-world enterprise information systems. In Proceedings of the 25th Americas Conference on Information Systems (AMCIS 2020), Online, 10–14 August 2020; pp. 1–10.
32. OMG. Unified Modeling Language(UML). 2021. Available online: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML (accessed on 18 August 2021).
33. OMG. MOF 2 XMI Mapping (XMI®). 2021. Available online: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI (accessed on 18 August 2021).
34. Sánchez-Villarín, A.; Santos-Montaño, A.; Koch, N.; Casas, D.L. Prototypes as Starting Point in MDE: Proof of Concept. In Proceedings of the WEBIST, Online, 3–5 November 2020; pp. 365–372.
35. Forward, A.; Badreddin, O.; Lethbridge, T.C.; Solano, J. Model-driven rapid prototyping with Umple. *Softw. Pract. Exp.* **2012**, *42*, 781–797. [CrossRef]
36. Acerbis, R.; Bongio, A.; Brambilla, M.; Butti, S. Webratio 5: An eclipse-based case tool for engineering web applications. In Proceedings of the International Conference on Web Engineering, Como, Italy, 16–20 July 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 501–505.
37. Brambilla, M.; Fraternali, P. Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience. *Sci. Comput. Program.* **2014**, *89*, 71–87. [CrossRef]
38. CUBA Platform. Available online: https://www.cuba-platform.com (accessed on 18 August 2021).
39. Hovsepyan, A.; Van Landuyt, D. Prototizer: Agile on Steroids. In Proceedings of the Flexible Model Driven Engineering Proceedings (FlexMDE 2015), Ottawa, ON, Canada, 29 September 2015; pp. 51–60.
40. Gamito, I.; da Silva, A.R. From Rigorous Requirements and User Interfaces Specifications into Software Business Applications. In Proceedings of the International Conference on the Quality of Information and Communications Technology, Faro, Portugal, 9–11 September 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 459–473.
41. Seixas, J.; Ribeiro, A.; da Silva, A.R. A Model-Driven Approach for Developing Responsive Web Apps. In Proceedings of the ENASE, Heraklion, Greece, 4–5 May 2019; pp. 257–264.
42. Cortés, H.; Navarro, A. Enterprise WAE: A Lightweight UML Extension for the Characterization of the Presentation Tier of Enterprise Applications with MDD-Based Mockup Generation. *Int. J. Softw. Eng. Knowl. Eng.* **2017**, *27*, 1291–1331. [CrossRef]
43. Hussain, Z.; Slany, W.; Holzinger, A. Current state of agile user-centered design: A survey. In Proceedings of the Symposium of the Austrian HCI and Usability Engineering Group, Linz, Austria, 9–10 November 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 416–427.
44. Ricca, F.; Scanniello, G.; Torchiano, M.; Reggio, G.; Astesiano, E. Assessing the effect of screen mockups on the comprehension of functional requirements. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2014**, *24*, 1–38. [CrossRef]
45. Kautz, K. Participatory design activities and agile software development. In Proceedings of the IFIP Working Conference on Human Benefit through the Diffusion of Information Systems Design Science Research, Perth, Australia, 30 March–1 April 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 303–316.
46. Chamberlain, S.; Sharp, H.; Maiden, N. Towards a framework for integrating agile development and user-centred design. In Proceedings of the International Conference on Extreme Programming and Agile Processes in Software Engineering, Oulu, Finland, 17–22 June 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 143–153.
47. Sanchez-Villarin, A.; Santos-Montaño, A.; Enríquez, J. Automatic Reuse of Prototypes in Software Engineering: A Survey of Available Tools. In Proceedings of the 15th International Conference on Web Information Systems and Technologies, Vienna, Austria, 18–20 September 2019; pp. 144–150. [CrossRef]