CMST 20(4) 139-150 (2014) DOI:10.12921/cmst.2014.20.04.139-150

Extracting Use Case Scenarios and Domain Models from Legacy Software

M. Śmiałek, K. Rybiński, N. Jarzębowski, W. Nowakowski, S. Blatkiewicz

Warsaw University of Technology
Warsaw, Poland
E-mail: smialek@iem.pw.edu.pl

Received: 22 May 2014; revised: 01 October 2014; accepted: 7 November 2014; published online: 11 January 2015

Abstract: Developing a new software system based on legacy software is a labor-intensive process. The main problem lies in preserving the application and domain (business) logic. This paper presents an approach to automate the process of extracting this logic, together with accompanying domain definitions. The approach is based on recording and processing the legacy system behaviour, observable through its user interface. The recovered application logic is represented with use case scenarios having precise sentences describing user-system interactions. These scenarios are tightly linked with domain models which are also created. The presented approach is supported by a tool chain. The central idea is to use a standard test automation system to capture test scripts. These scripts are then processed by a dedicated translation tool and translated into constrained natural language models. These models are machine processable, which allows for further automatic transformations even down to code. The paper presents the results of a case study where the approach and tools were used to migrate an old desktop system to a modern web technology application.

Key words: legacy system recovery, application logic recovery, model-driven requirements engineering

I. INTRODUCTION AND RELATED WORK

Many modern software systems use old, obsolete technologies. It is thus necessary to move them to new, efficient technologies that would allow for their further development. This includes the rising trend to substitute classical desktop systems with their web-based versions. Unfortunately, recovery of logic from such legacy software is very hard. This is usually caused by inability to comprehend and analyse the code which became tangled and twisted throughout the years of development. Thus, it is often much easier to write a new system from scratch instead of attempting to understand and modify the existing system.

This paper proposes a method for recovering important elements of legacy software independently of their code structure and details. Instead of analysing and reverse-engineering the code, we propose to reverse-engineer the system's user interface. Our method is accompanied by a tool for extracting application logic information from legacy systems to facilitate further easy migration into new system design. Application logic carries information about the user-system dialogue in relation to domain-specific data processing and platform-specific user interface appearance. In our solution, such information can be extracted from any legacy system by determining its observable behaviour. This information can then be stored in the form of requirements-level models written in the Requirements Specification Language (RSL) [1] which has precise formal specification of its syntax [2]. These models can be then transformed into architectural and design-level models or even into code [3].

The recovery process is illustrated in Fig. 1. It consists of three major steps: 1) recording test scripts, 2) transforming scripts to RSL, 3) manually correcting RSL models. In the first step, the legacy system is subject to "UI ripping" (recording the observable behaviour) as available in standard commercial test automation tools. The legacy system users

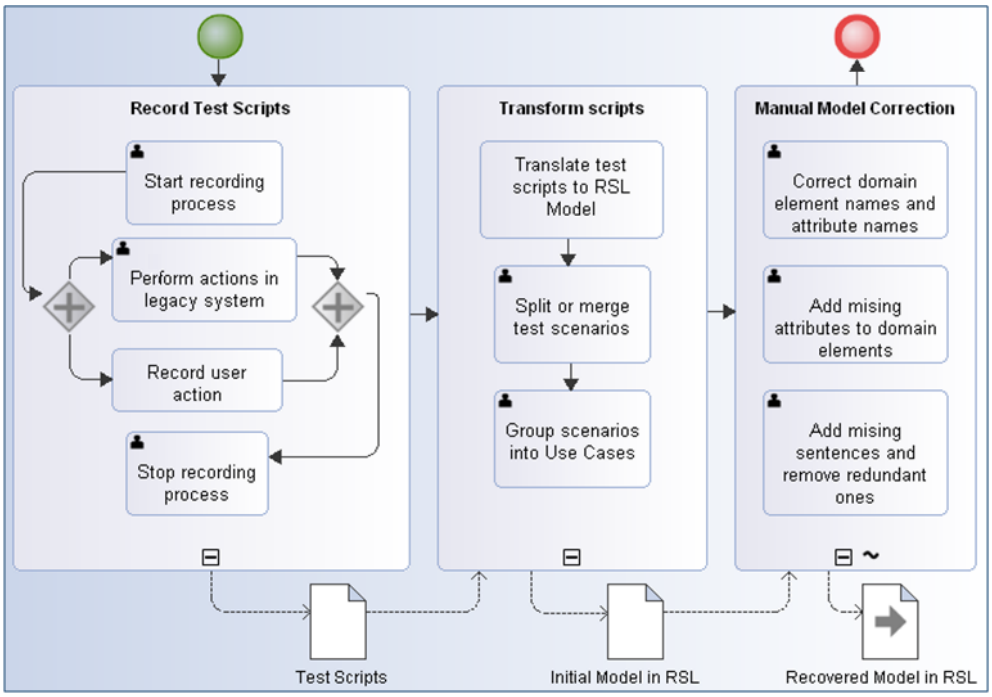

Fig. 1. Overview of the recovery process

work with it as normal but also record their activity using the test tool that integrates with the user interface. This produces processable test scripts (e.g., in the XML format). The scripts are then processed by a dedicated tool, called TALE (Tool for Application Logic Extraction), developed within the REMICS project [4]. The tool can process test scripts and turn them automatically into RSL models, consisting of use case scenarios and domain notions. These models can be further edited manually by merging scenarios and grouping them into use cases. The final step is to update the extracted RSL models in an RSL editor, called ReDSeeDS [5, 6]. This allows for correcting the domain model (naming, etc.), extending the models with new functionality, and changing the existing functionality.

This process is a part of a wider process to migrate to new architectures. Presenting the details of this wider process is out of scope of this paper and we refer the reader to work by Nowakowski et al. [7]. The proposed recovery process can also be compared to that proposed by Hungar et al. [8]. However, this solution generates low-level state models and is limited to recovery of telecommunication systems and concentrated on generating test cases. Here we present the details of the TALE tooling environment that is suitable for a wide range of business systems which exhibit high intensity of user-system interactions. The models generated by TALE are suitable for further transformation into modern technology code, at the same time preserving the application logic of the legacy system.

It can be noted that most solutions to knowledge recovery from legacy systems concentrate on retrieving code or detailed design artifacts. This was formalised through the introduction of the Knowledge Discovery Metamodel (KDM) [9], with MoDisco [10] and Netfective Blu Age [11] being one of its first implementations. Recently, KDM has been extended to cover also the application logic [12], which is relevant in the context of this paper. Other approaches include data analysis solutions like Data Reverse Engineering [13] or Database Reverse Engineering [14]. According to our best knowledge there are no similar solutions where application logic is recovered based on observable behaviour. Most work regarding reverse engineering of (graphical) user interfaces concentrate on testing purposes (see e.g., work by Memon et al. [15]) and direct migration into new user interface technology (see e.g., Stroulia et al. [16]). In the presented solution, the application logic is recovered in the form of requirements-level scenarios that facilitate discussion on the possible changes in functionality. By contrast, there seem to be no other advanced tools that focus on requirements recovery. Sparse examples include work by Fahmi and Choi [17] and the RETR initiative [18]. However, there seems to be no toolkit that supports these ideas. Recently, an approach by Repond et al. [19] proposes to formulate the recovered system knowledge in the form of use cases. Though, in contrast to the TALE tool, it does not offer any means to automate the recovery of scenarios. Some approaches, like the one by Bertolino et al. [20] propose synthesizing behaviour protocols that contain similar information to that of use case scenarios. However, this is based on using the existing implementation artifacts or design models, while our approach is independent on any “internals” of the legacy system.

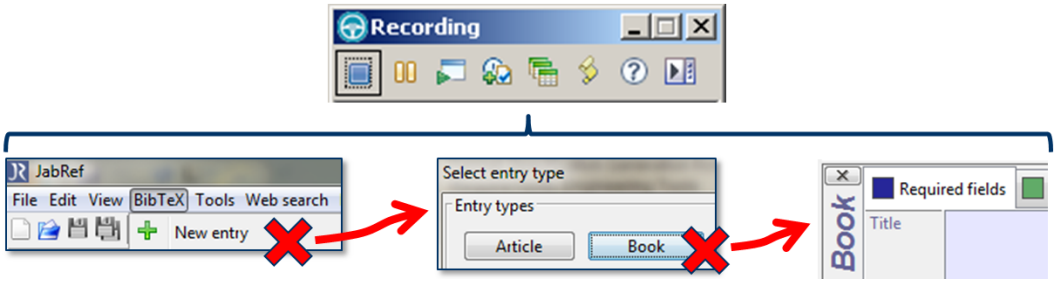

Fig. 2. Recording a GUI interaction scenario

II. TOOL CHAIN REQUIREMENTS

To capture legacy application logic we need to process and store information on all the significant paths through the user interface, including exceptional behavior (e.g., entering invalid data, operation cancellation). Thus, the important requirement of the new tool suite is to be able to record the various possible user-system interaction paths of a legacy system. One of such paths is illustrated in Fig. 2. Here we can see a short scenario for entering a new book entry using the JabRef reference manager [21]. We would thus want to "play-out" many such scenarios through normal usage of the system and record their steps and data exchanged with the user (cf. UI ripping). It can be noted that such recording is present in typical test automation tools.

The above requirements resulted in to choosing Rational Functional Tester (RFT) [22]. Its main purpose is the automation of functional and regression testing. Capturing and simulation of user actions can be performed for various user interface styles and technologies. The captured functionality ("test scripts") is editable and presented together with the UI screens. RFT uses an object map that maps between the script and the application under test, thus providing detailed information about the data objects engaged in the interactions. The test scripts and objects can be exported into machine processable scripts in XML. This is illustrated in Fig. 3 where the XML file contents reflect some elements of the scenario ("Book" button, "New Book" window) and the data ("Title" for books) from Fig. 2.

There were also attempts to utilise other test automation tools, but none of them met the presented requirements as fully as RFT. They capture only some details of the user-system interaction or store the captured information in a form difficult to process. HP Unified Functional Testing software [23] and SmartBear TestComplete [24] are some examples of the analysed tools. Unfortunately, the extent and form in which the captured objects are stored by these tools were not satisfactory for the recovery process. There were also additional problems with the range of supported languages and technologies. For example, we could not use tools such as Selenium [25] because they only support browser-based applications.

Test scripts recorded by RFT need further processing. Their purpose is not to capture application logic units but to capture linear paths through the system behaviour for further repeated automatic test execution. Thus, the new tools should be able to translate and merge such scripts into coherent human-readable high-level models representing units of application logic. It can be noted that this can be very well packaged into familiar use cases with their potential to be processed by the techniques of Model Driven Engineering (see e.g., work by Astudillo et al. [26]).

What is also needed is means to store the use case scenarios as models. This is offered by the RSL which is unique through its metamodel [2] that provides requirements representation details. It contains detailed constructs for scenario sentences and their parts (subjects, verbs, object). Moreover, its constructs greatly facilitate linking of the scenario elements with the domain elements. Such capabilities of the notation are not present in popular modelling languages like UML [27].

RSL has a comprehensive and mature tool suite – the ReDSeeDS Engine [6]. Using this suite we can store and process (edit) RSL models. The new tools should thus be able to create RSL models based on the recorded test automation scripts. This is illustrated in Fig. 4. The scenario from Fig. 3 is now translated from an XML file into a use case with a scenario containing 5 simple subject-verb-object sentences. This is supplemented by a domain model containing information on windows (e.g., "New Book window") and associated data (e.g., "New Book" with "Title").

Such translation should be done automatically. Moreover, it should be possible to merge several similar scenarios into use cases. The tool should also support modification and correction of the translation results (e.g., to improve the application's functionality). This is particularly helpful in situations where the person recording the scripts did not properly perform recording actions (e.g., made the wrong choice for the script beginning and end points). It can thus save time on repeating the script collection and transformation. Therefore, the tool should provide features for dividing and merging scripts to properly reflect the use case goals.

In summary, the analysis resulted in selecting RFT and ReDSeeDS Engine as the first and last component in the

recovery path. What was still necessary to develop was the tool to transform test-related scripts into application logic units (use cases with scenarios). This resulted in constructing TALE - Tool for Application Logic Extraction, as presented in further sections. It should be noted that RFT and (especially) ReDSeeDS were developed within the Eclipse framework [28], and thus it was a natural choice also for TALE.

III. TRANSFORMATION FROM RFT SCRIPTS TO RSL

III. 1. Source and target of transformation

Understanding the RFT script structure is fundamental to proper extraction of information. Each script is an XML file (see Fig. 3) containing activities grouped by the UI elements in which they have occurred. To understand its structure and prepare for transformation into RSL, the RFT script "language" has been reverse engineered into a metamodel[1]. Its simplified structure is presented in Fig. 5. The scripts are composed of test element groups (cf. **TestElements**). Every recorded window has its xsl:type set to **TopLevelWindowGroup**. Each element of this kind holds a reference to its description stored separately in a list of all windows (cf. **TopLevelWindow**). A window contains elements that the user has interacted with, typed as **ProxyMethod**s. Each such method has an **Action** – a click, a text input or key press. Actions have **Argument**s which refer to **TestElement**s that specify action attributes and values (e.g., entered text or pressed keys). At the window group or window element representation level, there can exist test elements typed as **ScriptMethod**s which represent calls to other scripts.

Test script recording also results in creating special structures containing objects with information on data elements and its types exchanged with the user. They are placed in separate XML files, called Object Maps. For brevity we will not present their structure here.

The above structure has to be transformed into RSL models which are also defined through a metamodel. Fig. 6 shows a small (and simplified) fragment of the metamodel for use cases scenarios (compare with the upper part of Fig. 4), which is relevant for the transformation. Any **RSLUseCase** can have many **ConstrainedLanguageScenario**s. Scenarios are composed of ordered sequences of **ConstrainedLanguageSentence**s, and particularly **SVOSentence**s. Such sentences contain a 'subject' which is a **NounPhrase** and a 'predicate' which is a **VerbPhrase**. Verb phrases contain sentence 'objects' which are also **NounPhrase**s.

The scenario metamodel is strictly associated with the metamodel for domain elements (compare with the lower part of Fig. 4), presented in a significantly simplified form in Fig. 7. The domain models consist of **Notion**s which refer to **NounPhrase**s as their names. Notions can contain many **DomainStatement**s which have **VerbPhrase**s as their names. Notions can be linked through **DomainElementRelationship**s.

III. 2. RFT script to RSL transformation algorithm

The recorded RFT scripts in XML format are parsed and transformed into a model compliant with the RSL metamodel. This model is stored within the ReDSeeDS repository (model storage) that is based on the JGraLab [30] technology. Since this repository is graph-based, this step involves creating a graph structure. From the user point of view, the translation results in a series of simple RSL constrained language sentences, user interface element notions and domain notions reflecting the contents of the XML file. In abstract syntax, this means parsing and translating linear scripts conforming to the metamodel in Fig. 5 into graphs conforming to the metamodel in Fig. 6 and 7. In concrete syntax, this means translating a script like in Fig. 3 into a model like in Fig. 4.

The transformation algorithm was coded in plain Java using a standard XML parser library (see Sec. III. 3.) and the JGraLab API. Its overview is presented in Fig. 8. The

[1] All the metamodels presented in this paper are written in MOF [29] which uses simplified class diagrams. Colouring of the diagrams is for aesthetics only.

```
…  <testElements xsi:type="com.ibm.rational.test.ft.visualscript:ProxyMethod" name="book" type="GuiTestObject"
role="Button" elementType="TestObject" domain="Java" controlName="Book" topLevelWindow="//@topLevelWindows.2">
      <action name="click"/>
</testElements>

<testElements xsi:type="com.ibm.rational.test.ft.visualscript:ProxyMethod" name="newBook"
type="TopLevelTestObject" role="Frame" elementType="TestObject" domain="Java" controlName="New Book"
topLevelWindow="//@topLevelWindows.1">
      <action name="inputChars">
        <argument>
          <testelement xsi:type="com.ibm.rational.test.ft.visualscript:Value" value="Title"
          elementType="Value" valueType="String"/>
        </argument>
      </action>  …
```

Fig. 3. Example RFT script file

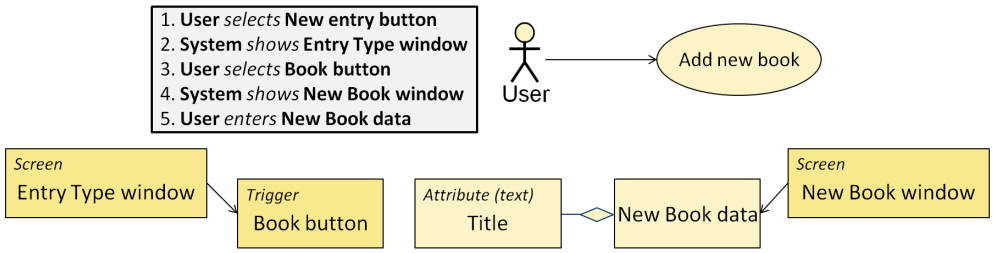

Fig. 4. Example RSL model

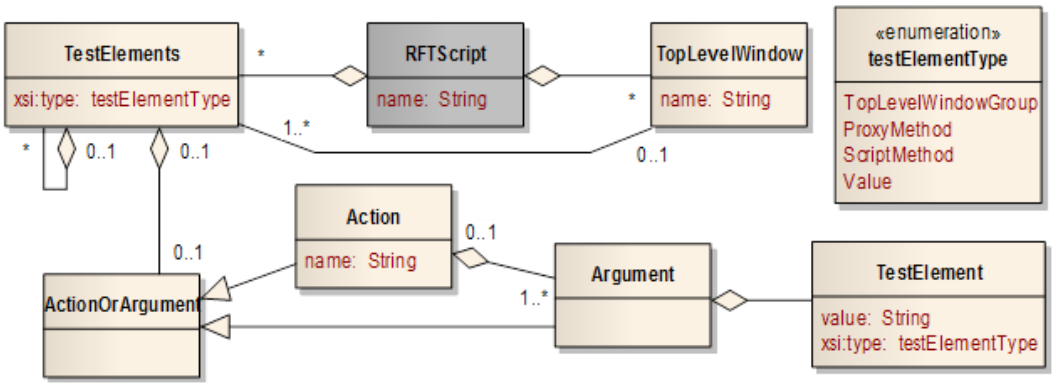

Fig. 5. Overview of the RFT script structure

algorithm is based on gathering information from consecutive TopLevelWindow groups. For each of such groups, a window presentation sentence is generated (cf. sentences 2 and 4 in Fig. 4). Furthermore, each element under the current top level window is processed. The recorded user actions referenced by these elements result in generating either UI element selection sentences (cf. sentences 1 and 3), or data input sentences (cf. sentence 5). For the latter situation, the associated Object Map file is parsed and sought for associated data objects. Based on this, appropriate RSL domain notions are generated.

It can be noted that the target sentences observe a simple subject-verb-object format (sometimes with two objects). The rule is that the subject is set to *system* for the window display sentences. In other cases the subject is set to *user*. The verb is set to *select* for the selection or button pressing sentences; *enter* is used for the data input sentences and *show* is used for the window display sentences. The sentence objects are set to either "window", "button" or "domain" notions. The object name is taken from the source script and an appropriate postfix is added (cf. "New Book window" vs. "New Book data").

As mentioned above, before any data input sentence (cf. sentence 5) is added, a domain notion, representing this data, is created. All the input data element types are identified inside the proper Object Map file and marked with appropriate data types. They are then added to the domain specification (cf. "New Book data" element). All the associated primitive values from the map are set as the domain notion's attributes (cf. "Title" attribute).

The full transformation results in creating a scenario for each of the RFT scripts. For all the scenarios, a single domain model is created and contains all the notions, (windows, buttons, domain elements) linked through appropriate relationships (cf. Fig. 4, bottom).

III. 3. TALE functionality and architecture

The transformation presented in the previous section can be executed as part of the functionality of the TALE tool. The user is prompted to select RFT script files and the tool creates appropriate scenarios and domain notions. The recovered scenarios are displayed in the so-called **Detached scenario list**. They are now subject to manual merging and combination into use cases. This allows for constructing complete and coherent models that fully reflect the observable functionality of the legacy system.

The TALE use case editor is illustrated in Fig. 9. Any "detached scenario" can be attached to an existing use case. New use cases can be created and freely edited. When attaching a scenario to a use case, the user can choose a reference scenario and point to a correct joining place. This also adds special condition sentences to both scenarios[2]. The previously (possibly erroneously) attached scenarios can also be detached back to the unassigned scenario list. The user can

[2] Detailed discussion on condition sentences is out of scope of this paper

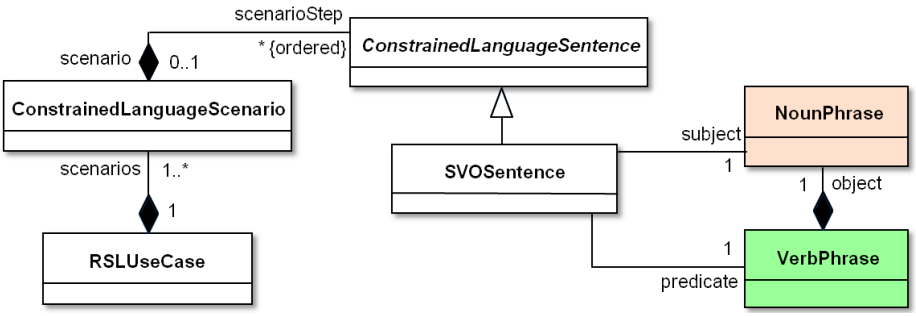

Fig. 6. Fragment of RSL metamodel for use case scenarios

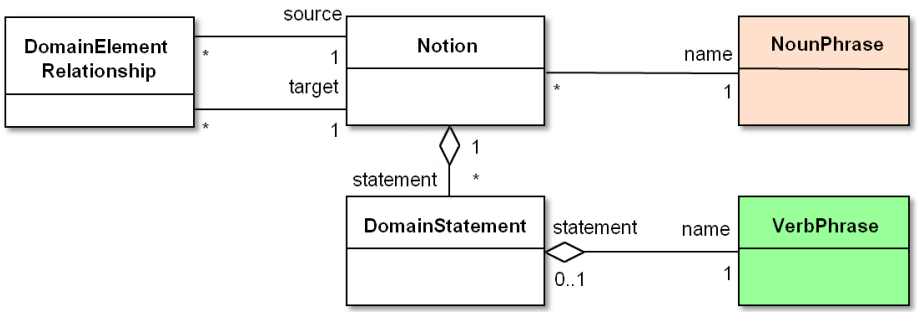

Fig. 7. Fragment of RSL metamodel for domain notions

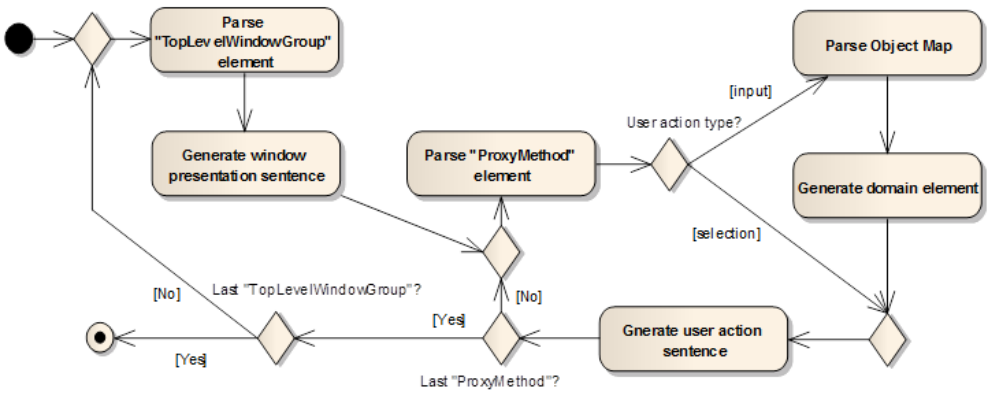

Fig. 8. Overview of the RFT to RSL translation algorithm

also delete scenarios from the list, join them or split them. It is also possible to move scenarios between use cases, merge use cases or notions and automatically find common scenario fragments. This last feature uses in its implementation the Rabin-Karp algorithm [31] for detecting same scenario fragments. Single sentences act as string patterns.

Fig. 9 illustrates the package structure of the edited model, maintained within the tool. Some packages contain the created use cases, some contain the domain notions (see left). The scenarios and domain notions can be further edited and extended according to newly emerging requirements (see top-right).

TALE was implemented as an extension for the ReDSeeDS tool, within the framework of Eclipse Rich Client Platform (RCP). The test script parsing algorithm implementation uses the Xerces Java Parser – an XML parser from Apache Xerces [32]. Worth noting is the possibility to switch between TALE and ReDSeeDS (arranged as perspectives) seamlessly since both tools are integrated within a single framework and they share the common RSL data model. Additionally, TALE uses GMF plug-ins for handling graphical diagrams with the underlying EMF model [33] modified to serve as a proxy layer for the JGraLab model.

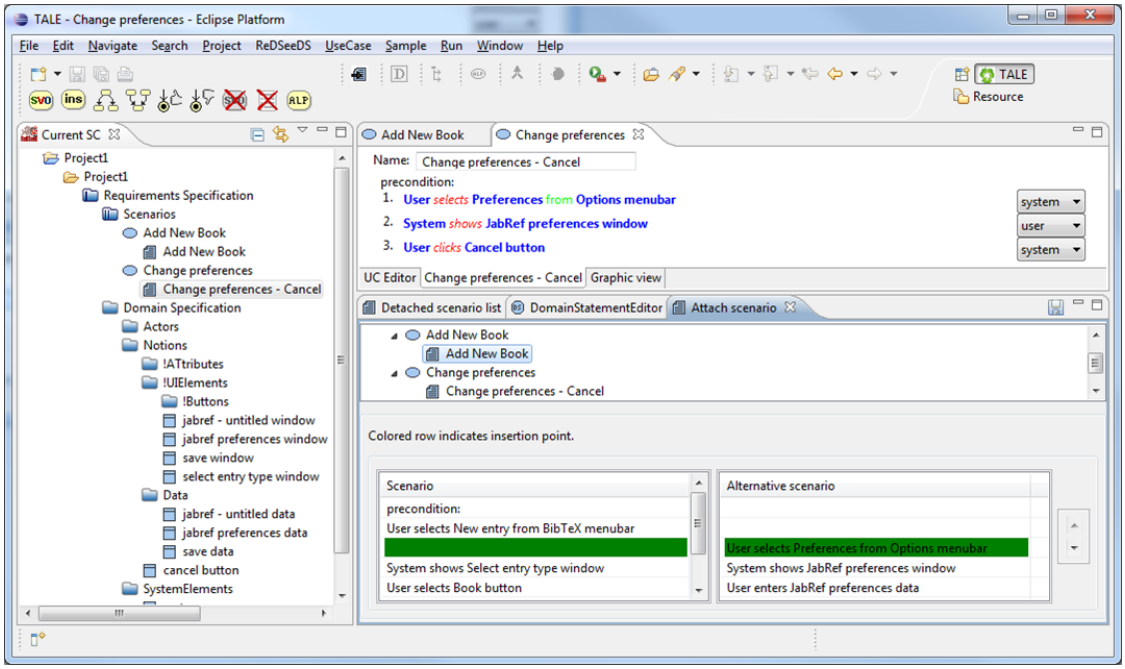

Fig. 9. Sample window of the TALE tool

The general structure of the tool components (Java plug-ins) is shown in Fig. 10. The relevant ReDSeeDS components are marked red (darker) and the TALE plug-ins are marked green (lighter). Generally, the system is functionally divided into domain logic and application logic (combined with the UI). The domain logic is handled by two components: *redseeds.scl.model* implements the original RSL metamodel (part of a broader SCL metamodel); *remics.recovery.model* implements script processing and transformation, communicating frequently with *redseeds.scl.model*. The main TALE observable functionality, as presented in this and previous section, is contained in two application logic components: *remics.script. loader* and *remics.recovery.manager*. These two components are supported by the RSL editor (*redseeds.editor.rsl*) and the project tree manager (cf. *redseeds.engine* and *remics.engine*). The *navigator.listener* component supplements this functionality by reacting to changes in UI elements and updating the current model.

IV. CASE STUDY EXAMPLE

The presented toolkit has been validated through recovering a non-trivial commercial system in the bank loan management domain. The system, called “SZOK” (in Polish: System Zarządzania Obsługą Kredytów), has been developed by

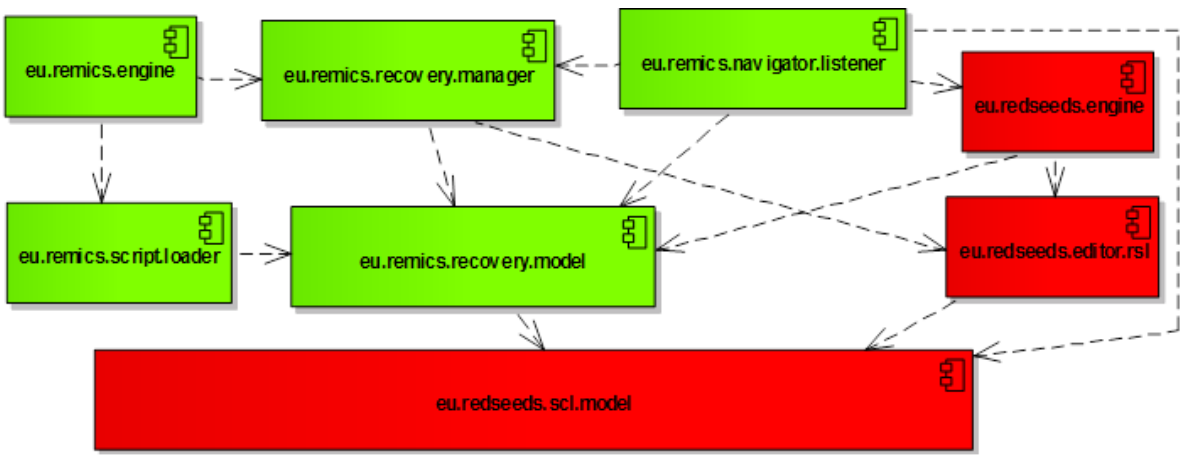

Fig. 10. TALE main architectural components

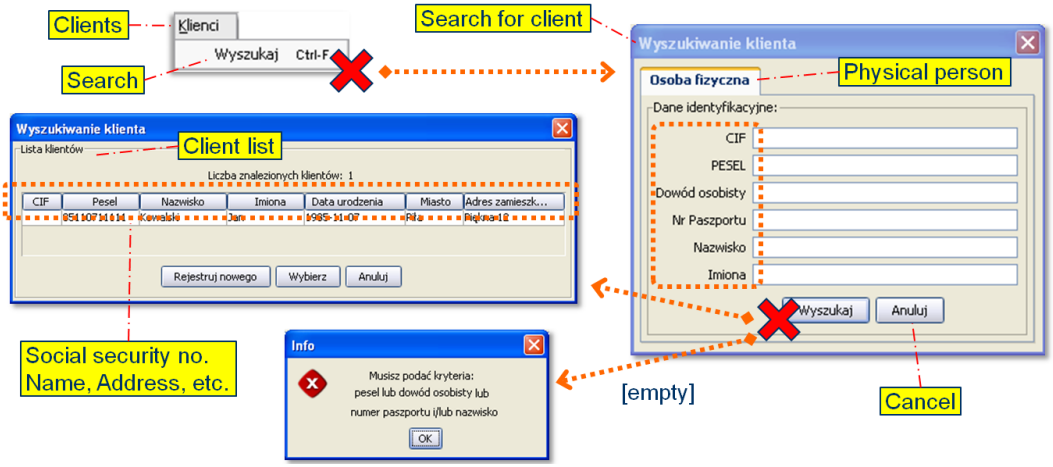

Fig. 11. Example UI behaviour recording for the case study

a Polish major software provider Infovide-Matrix. It was discontinued from further development in 2009, after nearly 10 years of development and commercial usage. The system became obsolete, although it was developed using technologies that are still in use: Java 1.5, SWING for the user interface, WebSphere Application Server and JDBC with Hibernate for database access. The main problem with the system is its use of SWING that is no longer treated as an ergonomic solution. Also, the architectural structure of the system and related code became obsolete and impossible to evolve.

For this reason, the only justified way to recover logic from SZOK was to reverse engineer its observable behaviour. This has led to using the TALE approach. The case study covered a significant part of the system's functionality and resulted in a model consisting of 50 full use cases, each with 2 or more scenarios. The study started with recording test scripts using RFT. This is illustrated in Fig. 11 for one example piece of functionality. This simple example shows two alternative scenarios associated with searching for clients[3]. Both scenarios start with selecting an option (Klienci → Wyszukaj; Clients → Search), and then showing a search criteria window. One scenario results in showing a client list (*pol. lista klientów*) and the other one (when the list is empty) shows an info message.

In the next step of the recovery process, the TALE tool has transformed the recorded scripts into an initial RSL model. This model was then manually modified by adding use cases to group scenarios and merging alternative scenarios under these use cases. The result of this activity is illustrated in Fig. 12. It shows a small fragment of the use case model with six use cases, connected through RSL's special «invoke» relationships (denoted by arrows between use cases, see [34]). Two scenarios that resulted from the recording shown in Fig. 11 were manually inserted as contents of the "Search for client" use case. The scenarios were merged and appropriate condition sentences ('cond:') were added. Finally, invocation sentences ('invoke/INSERT') that denote calling of invoked use cases, were introduced.

Together with use case scenarios, also the user interface and domain notions were recovered. This is illustrated in Fig. 13. It shows notions related to two of the windows shown in Fig. 11. These windows were turned into notions of type *Screen*. These notions are connected with appropriate data notions. One of them is typed as *Simple View*, as it denotes attributes for a single data element placed on a window. The other data notion is a *List View* because it offers a list of many elements shown on a window. It can be noted that both views share some of the attributes.

The recovered domain model is consistent both with the user interface elements of the legacy system and with the recovered scenarios. Fig. 13 can be compared with Fig. 11 and 12 to see this consistency. For example, the "wyszukiwanie klienta data" (*Eng. search for clients data*) notion contains attributes that reflect the fields in the window entitled "Wyszkiwanie klienta" in the legacy user interface.

The final step is to refine the RSL model to cater for possible modifications and extensions to the system's functionality. Often, the domain model needs manual refactoring due to required renaming of recovered notion names. This is done in the ReDSeeDS perspective of our tool suite and is illustrated in Fig. 12. Some notions were renamed and several use cases "wired" to compose for consistent application logic and navigation between various parts of the user interface.

The case study also involved migration to new technology. The result of this migration is shown in Fig. 14. The shown forms were generated completely automatically from the presented domain model. What is more, the generated system also followed the application logic according to the presented use case scenarios. As part of the case study, the generated code was updated with database access and some

[3] The system's user interface is entirely in Polish, so the figure provides some English translations.

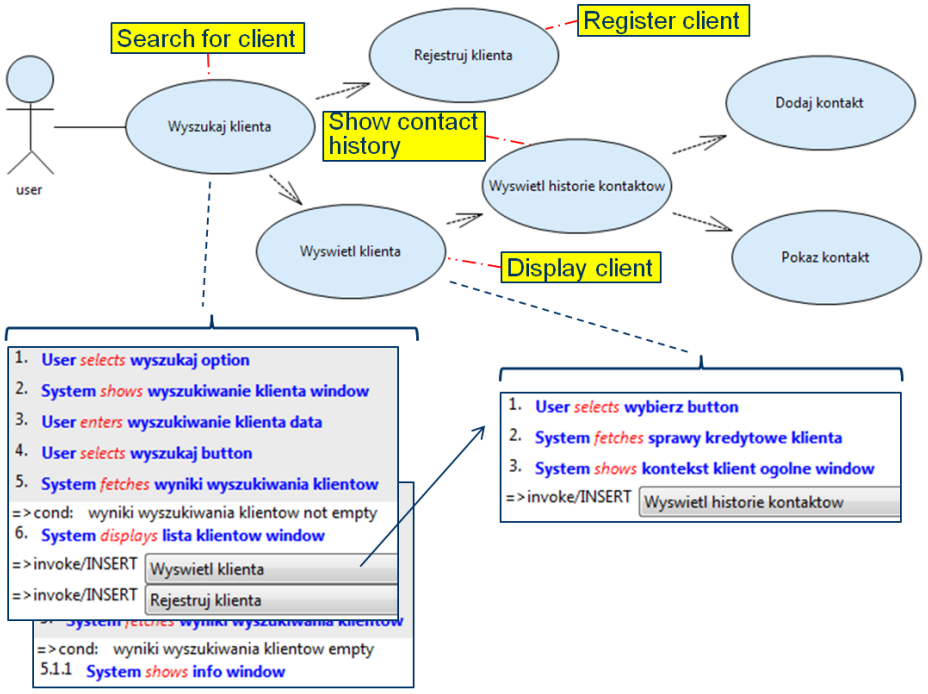

Fig. 12. Fragment of the use case model obtained during case study

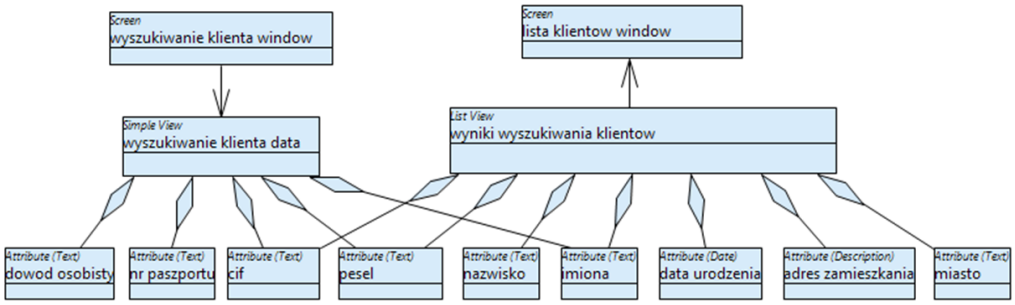

Fig. 13. Fragment of the domain model obtained during case study

simple business logic (data processing). Detailed discussion of the structure of the generated code is out of scope of this paper, and the reader is referred to other work by the authors [34, 3].

V. DISCUSSION AND CONCLUSION

The development of the TALE tool took around 20 man-months. The team consisted of the authors of this paper. The effort involved appropriate research and implementation of the RFT-to-RSL transformation algorithm and the scenario management editor. The tool is a crucial element of wider research to develop an effective method to migrate legacy systems to modern technologies. It has to be stressed that the requirements models generated and managed within the tool can be further used to generate code in a wide range of technologies [3]. The current results show that full (dynamic) code of the upper layers (view, controller/presenter in MVC/MVP architectures) can be automatically generated. The only disadvantage is that the data processing layer (model in MVC/MVP) has to be migrated using other methods.

The effort associated with migrating legacy applications using TALE is concentrated in recording and merging user-system interaction scenarios. This would involve instructing regular users of the legacy system to cover all the paths that are intended for migration. Then, the recorded paths would need to be merged into use cases. It can be noted that in contrast to typical reverse-engineering methods, the above activities do not involve workforce with advanced skills. Further generation of the new system is fully automatic. In summary, the "manual" effort to migrate a system consists of three elements: playing out scenarios + merging scenarios

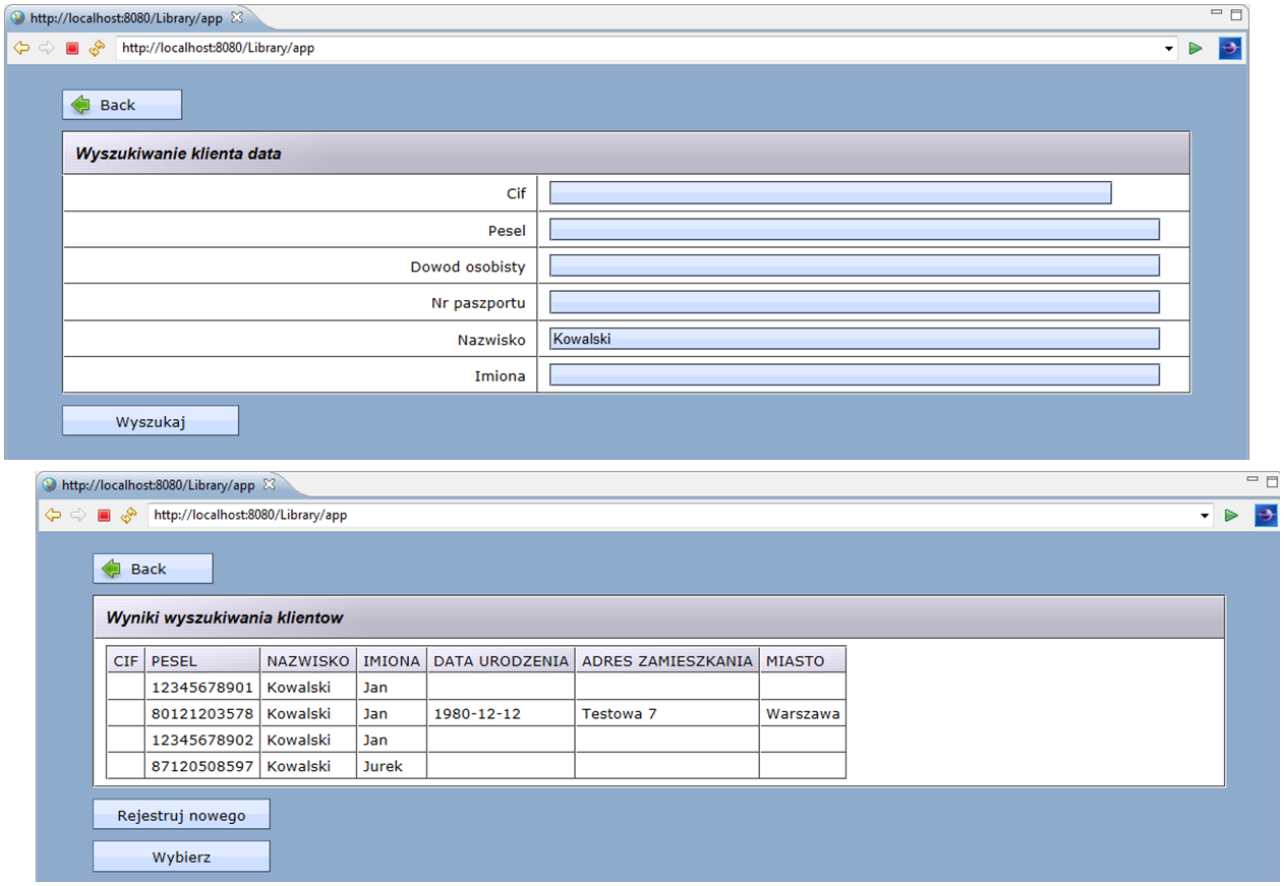

Fig. 14. Automatically generated web forms

into use cases + updating the generated system with data processing/storage algorithms. In case of lack of legacy documentation and/or lack of legacy source code, this approach seems to be the only economical solution. Even in case when the legacy source code is available, its re-engineering might often be very difficult (cf. GOTO statements in legacy code etc.) and thus not economical.

The current conclusions regarding efficiency of the approach can be only qualitative. Our initial observations show that the current capabilities of automatic retrieval of crude RSL models can improve performance by at least 20% in relation to writing same RSL models from scratch. This is mainly due to an improved process of analysing the legacy system's behaviour and generation of the bulk of the sentences with related domain elements. However, this still necessitates to be acknowledged by more objective empirical results. This is planned as the next step in research on requirements-based recovery.

In regard to implementation of TALE, several interesting observations can be emphasised. The plug-in architecture of RCP has significantly facilitated interfacing and reusing the ReDSeeDS components. Overall, the Eclipse environment provided a coherent workspace for the project, despite significant learning curve associated with its various elements. A prominent example is the GMF/EMF framework [33] used to develop the graphical model editors. It allowed us to quickly transform a metamodel (like the one shown in Fig. 5) into a rich graphical editor. Still, GMF/EMF lacks satisfactory documentation which leads to quite significant overhead associated with mastering this environment. Moreover, in the context of ReDSeeDS, we have experienced overhead due to incompatibility between the EMF and the JGraLab storage. On the other hand, JGraLab has proven to be a very efficient model repository which is also easy to apply. It provides a very rich low-level API with additional high-level wrappers for common complex tasks.

The ultimate goal for the research around TALE is certainly very practical. It can be noted that the tool can recover the logic of practically any software system in respect to its observable behaviour. This makes the tool completely independent of the legacy system's internals (often "twisted" and not recoverable by other means). It can be noted that TALE can be easily interfaced with other (G)UI ripping tools. This would necessitate changes only to the *remics.recovery.model* component that currently processes RFT scripts (XML). This gives vast possibilities for recovering application logic for various types of user interface technologies.

Future work around the TALE system will involve raising the levels of automation. This opens a vast and interesting research agenda. One of the possibilities include automating the UI interaction and screen recording. Another possibility is to develop mechanisms for analysing recorded scripts and scenarios and their automatic merging and grouping into use cases. This would need to involve the analysis of certain patterns in use case scenarios and conformance of the recovered scenarios to these patterns. A separate direction of research will be improvement of RSL's syntax and semantics for capturing the application and domain logic, and then transforming them into code. For instance, it is necessary to develop a very precise notation for UI elements and associated domain notions. With such notations, transformations from RSL models to code would necessitate very little post-transformation manual work on the generated code. This will allow to fulfil the ultimate goal of very highly automated migration of legacy systems into modern technologies.

Acknowledgment

This research has been carried out in the REMICS project (http://www.remics.eu) and partially funded by the EU (ICT-257793 under the 7th Framework Programme).

References

[1] W. Nowakowski, M. Śmiałek, A. Ambroziewicz, T. Straszak, *Requirements-Level Language and Tools for Capturing Software System Essence*, Comp. Science and Inf. Systems **10**(4), 1499-1524 (2013).

[2] H. Kaindl, M. Śmiałek, P. Wagner, et al., Requirements Specification Language Definition, Technical Report D2.4.2, ReDSeeDS Project, 2009.

[3] M. Smialek, N. Jarzebowski, W. Nowakowski, *Translation of Use Case Scenarios to Java Code*, Computer Science **13**(4), 35-52 (2012).

[4] REMICS project home page, http://remics.eu/.

[5] ReDSeeDS project home page, http://redseeds.eu/.

[6] M. Smialek, T. Straszak, *Facilitating transition from requirements to code with the ReDSeeDS tool*, [In:] *Requirements Engineering Conference (RE), 2012 20th IEEE International*, pages 321-322 IEEE, 2012.

[7] W. Nowakowski, M. Smialek, A. Ambroziewicz, N. Jarzebowski, T. Straszak, *Recovery and Migration of Application Logic from Legacy Systems*, Computer Science **13**(4), 53-70 (2012).

[8] H. Hungar, T. Margaria, B. Steffen, *Test-Based Model Generation for Legacy Systems*, [In:] *IEEE International Test Conference (ITC)*, pages 971-980, Charlotte, NC, 2003 IEEE Computer Society.

[9] R. Pérez-Castillo, I. García-Rodríguez de Guzmán, M. Piattini, *Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems*, Comput. Stand. Interfaces **33**(6), 519-532 (2011).

[10] MoDisco Project website, http://www.eclipse.org/MoDisco/.

[11] Netfective Blue Age website, http://www.bluage.com/en/.

[12] A. Ambroziewicz, M. Smialek, REMICS KDM Extension for Application Logic, Technical report, REMICS Project, 2012, Deliverable D3.6.

[13] P.H. Aiken, *Reverse engineering of data*, IBM Systems Journal **37**(2), 246-269 (1998).

[14] J.-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, *Contribution to a theory of database reverse engineering*, [In:] *Reverse Engineering, 1993., Proceedings of Working Conference on*, pages 161-170, 1993.

[15] A.M. Memon, I. Banerjee, A. Nagarajan, *GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing*, [In:] *Proceedings of the 10th Working Conference on Reverse Engineering*, pages 260-269, November 2003.

[16] E. Stroulia, M. El-Ramly, P. Iglinski, P. Sorenson, *User Interface Reverse Engineering in Support of Interface Migration to the Web*, Automated Software Engineering **10**(3), 271-301 (2003).

[17] S.A. Fahmi, H.-J. Choi, *Software Reverse Engineering to Requirements*, [In:] *Proc. Int. Conf. Convergence Inform. Technol.*, pages 2199-2204, 2007.

[18] Y. Yu, J. Mylopoulos, Y. Wang, et al., *RETR: Reverse Engineering to Requirements*, [In:] *Reverse Engineering, 12th Working Conference on*, page 234, 2005.

[19] J. Repond, P. Dugerdil, P. Descombes, *Use-case and scenario metamodeling for automated processing in a reverse engineering tool*, [In:] *4th India Software Eng. Conf.*, ISEC '11, pages 135-144, 2011.

[20] A. Bertolino, P. Inverardi, P. Pelliccione, M. Tivoli, *Automatic synthesis of behavior protocols for composable web-services*, [In:] *Proc. 7th ESEC/FSE '09*, pages 141-150, 2009.

[21] JabRef Reference Manager website, http://jabref.sourceforge.net/.

[22] C. Davis, D. Chirillo, D. Gouveia et al., *Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource*, IBM Press, 1st edition, 2009.

[23] HP Unified Functional Testing page, http://www.hp.com/go/uft.

[24] SmartBear TestComplete page, http://smartbear.com/products/qa-tools/automated-testing-tools.

[25] Selenium home page, http://docs.seleniumhq.org/.

[26] H. Astudillo, G. Génova, M. Śmiałek, J. Llorens Morillo, P. Metz, R. Prieto-Diáz, *Use Cases in Model-Driven Software Engineering*, Lecture Notes in Computer Science **3844**, 262-271 (2006).

[27] Object Management Group, *Unified Modeling Language, Part 2: Superstructure, version 2.4.1, formal/2012-05-07*, 2012.

[28] Eclipse Platform home page, http://eclipse.org/.

[29] Object Management Group, *OMG Meta Object Facility (MOF) Core Specification*, version 2.4.1, formal/2013-06-01, 2013.

[30] JGraLab project home page, https://github.com/jgralab.

[31] R.M. Karp, M.O. Rabin, *Efficient randomized pattern-matching algorithms*, IBM Journal of Research and Dev. **31**(2), 249-260 (1987).

[32] Apache Xerces project page, http://xerces.apache.org/.

[33] R.C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, Addison-Wesley, 2009.

[34] M. Smialek, W. Nowakowski, N. Jarzebowski, A. Ambroziewicz, *From Use Cases and Their Relationships to Code*, [In:] *Second IEEE International Workshop on Model-Driven Requirements Engineering, MoDRE 2012*, pages 9-18 IEEE, 2012.

Michał Śmiałek currently holds the position of a Professor. He obtained a habilitation (higher doctorate) degree in informatics from the Warsaw Military University and has graduated the Warsaw University of Technology (MSc and PhD) and the University of Sheffield (MSc). Prof. Śmialek has more than 20 years of experience in software development mainly using object-oriented methods. For several years he worked in the industry as a software developer and project manager. He teaches software modelling and requirements engineering in academia and for the major Polish companies. He published a book on UML modelling and over 70 articles in national and international refereed journals and conference proceedings. He is a member of program committees of international conferences in the area of software engineering, and did reviews for major software engineering journals. His research interests include metamodelling, model transformations, scenario-based requirements engineering and object-oriented development methods.

Kamil Rybiński is a PhD student in the Department of Theory of Electrical Engineering and Applied Informatics at Warsaw University of Technology. He received a BEng and MSc in informatics (with a specialisation in Software engineering) from Faculty of Electrical Engineering at the same University. His research interest includes requirements engineering, model-driven software development and knowledge representation. He worked as a researcher in the REMICS project.

Norbert Jarzębowski is a PhD student in the Department of Theory of Electrical Engineering and Applied Informatics at Warsaw University of Technology. He received a BEng and MSc in informatics (with a specialisation in Software engineering) from Faculty of Electrical Engineering at the same University. He worked as a system and business analyst for CUBE-CR, Roche and PZU, and as a researcher in the REMICS project.

Wiktor Nowakowski is a researcher in the Department of Theory of Electrical Engineering and Applied Informatics at Warsaw University of Technology. His main areas of research are in requirements engineering, model-driven software development and software language engineering. Wiktor also has extensive industry experience working on small- to large-scale projects, mainly as business systems analyst.

Sławomir Blatkiewicz is an IT consultant and developer. He received a BEng and MSc in informatics (with a specialisation in Software engineering) from Faculty of Electrical Engineering of Warsaw University of Technology. He worked as a researcher in the REMICS project and currently works as a Java EE developer in the banking sector.