Data-Driven Scenario Test Generation for Information Systems

P. Tanuska, *Member, IACSIT, IEEE,* and T. Skripcak

***Abstract*—This article is aimed on the data-driven scenario testing process. The first part of article starts with the general introduction into software testing. After that scenario testing is presented and also relations of scenario testing to functional requirements specifications are pointed out. Model-View-View Model architectural pattern is showed as an ideal candidate of testing friendly architecture of object oriented software system. Second part is devoted to proposal of data-driven scenario testing generation captured via UML notation. Finally, the proposal implementation is described in the context of real world object oriented system. The main goal of this article was to simplify task of testing complex functional user requirements by usage of scenarios in order to be agile enough, cut test preparation time and improve quality of the resulting software product.**

***Index Terms*—scenario testing, data-driven, UML**

I. Introduction

Software testing is one of the most important tasks in the process of software development. Companies are using testing mainly for the purpose of quality improvement. It usually starts with unit testing in projects which are guided according to Test Driven Development paradigm and continue with more complex types of tests. Scenario testing differs from the others because it is not oriented only on verification of basic functionality or fulfillment of design requirements. Its main goal is to provide the answer, whether system is usable for end users by creating complex real world scenarios.

A. Software Testing

Success of company in the information technology field is based on its ability to truly understand customer needs and according to this deliver product which has high quality attributes and offer real add value to the end user. Software testing is one of the options which, if it is used correctly, can help to accomplish these goals. It is important to say that software testing is not only tool for quality improvement. Some of testing techniques could be used at the beginning of the software development cy

Manuscript received February 4, 2011; revised July 28, 2011.

Authors would like to thank to software company MMS Softec providing technological and software framework which was used throughout whole project development.

P. Tanuska is with the Institute of Applied Informatics, Automation and Mathematic at the Slovak University of Technology, Trnava, Slovakia, (phone: +421 918 646 061; fax: +421 33 5511758; e-mail: pavol.tanuska@stuba.sk).

T. Skripcak is with the Institute of Applied Informatics, Automation and Mathematic at the Slovak University of Technology, Trnava, Slovakia, (e-mail: tomas.skripcak@stuba.sk).

understanding customer needs or at the end for the purpose of teaching users how is the system working. That is why, testing is described also as a learning process.

Software testing can by defined be many definitions, but the essentials can be reduced to idea of error preventing and quality improvement. Below are some formal interpretations of software testing:

- Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test [1].
- Software testing is the process of executing a program or system with the intent of finding errors [2].
- Software testing involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results [3].

According to [4] there are two types of professions with responsibility of test performing: testers and developers. Main difference is, that whether the test will be executed by, depends on the test type.

In general, software testing is usually classified according to three main criterions:

1) Techniques

- White-Box (Structural) Testing is representation of technique that originates from knowledge of systems internal structure. Test data is driven by examining the logic of the program or system, without concern for the program or system requirements [11]. Concrete examples are for example code analysis and code coverage inspection.
- Black-Box (Functional) Testing is representation of technique that originates only from knowledge observed from system functionality without any idea about inner working of developed product. The tester focuses on testing the program‘s functionality against the specification [11,13]. Specific case could be e.g. data driven testing.
- Grey-Box (Functional and Structural) testing is representation of technique which combine both White-box testing and Black-box testing in order to take the best from them. The tester studies the requirements specifications and communicates with developer to understand the internal structure of the system [11]. Usage of this technique is beneficial e.g. in the matter of lower test cases number.

2) Approaches

- Manual Testing is approach for human driven testing strategy. Basically it means that manual tests are not coded in computer. This type of testing is expansive and consume large amount of time and resources.
- Automated Testing is on the other hand implemented via some sort of testing framework. Software companies which applies Agile methodology likes this approach because it produce large number of

cheap and reusable test necessary for quality insurance of their products. Automate tests form bottom-up pyramid as shown in the Figure 1.

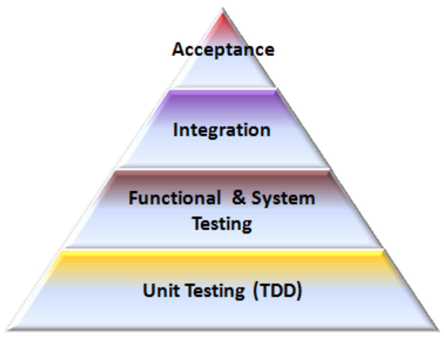

Fig. 1. Bottom-Up model for automating tests [4]

3) Types

- Unit testing is software testing technique that focuses on exercising the features of individual functions or modules in isolation [12]. Unit tests are usually written according to “Create Test First” paradigm, where phase of functionality which does not exists yet is tested to gain failure first in order to start with the refactoring processes which will results to test pass.
- Integration Testing is performed in order to test whether two or more units or modules coordinate properly. The are build on top of unit tests. There are several strategies for integration testing and usage of correct one depends on integration strategy of entire system [4].
- Functional Testing tests whether a system meets functional requirements. It does not cover internal coding of the system. Functional testing is pointed on verification whether the system behaves according to user requirements [4].
- System Testing is based on overall requirements specifications and that is why it covers all combined parts of the system [11]. It also verifies non-functional requirements like security or performance.
- Acceptance Testing is the final stage in the process of system development. Usually performed as a last step before delivery of product as an assurance that all customer needs are satisfied.

It is neither recommended nor possible to cover everything with tests. Software systems are too complex and 100% coverage would have inappropriate effect on overall price of resulting product. Finding moderate test coverage is always difficult task and it also depends on attributes of software system.

B. Scenario Testing

Scenario testing is quite specific and differs from other types of tests. The basis of scenario test is simulation of complex task lifetime in the developed system, from the end user point of view.

There is a lot of similarity between scenario and integration testing. They are related methodologies that test multiple units of a product to evaluate how well they work together. The purpose of integration tests is to verify interfaces of multiple modules, while scenario tests cover end-to-end scenarios the user cares about. Both are examples of real-world use of the product [4]. Fundamental explanations of scenario and scenario testing are as follows:

- A scenario is a hypothetical story, used to help a person think through a complex problem or system [5].
- Scenario tests are realistic, credible and motivating to stakeholders, challenging for the program and easy to evaluate for the tester. They provide meaningful combinations of functions and variables rather than the more artificial combinations you get with domain testing or combinatorial test design [6].

According to [9] good scenario test should be designed to meet these four important characteristics:

- The story is motivating. A stakeholder with influence would push to fix a program that failed this test.
- The story is credible. It not only could happen in the real world. Stakeholders would believe that something like it probably will happen.
- The story involves a complex use of the program or a complex environment or a complex set of data.
- The test results are easy to evaluate. This is valuable for all tests, but is especially important for scenarios because they are complex.

Useful feature of scenario testing is that in the real world they can morph into other types of testing. The list below shows few options of scenario test usage:

- Some people create a pool of scenario tests as regression tests [4, 8, 14].
- Test for the purpose of learning advance product usage [4, 5, 8, and 10].
- Stress testing with data that are not used in normal situations (this tests are called “Killer soaps”) [4, 8].
- Verification that key user requirements are satisfied [4].
- Surfacing design-related issues and possibly raising new design issues [4]

C. Scenarios and Functional Requirements

Software required specification (SRS) usually contains all needs of stakeholders and end users, which were discovered through the customer needs analysis process. This specification is used as part of agreement between company developing product and customer. SRS contains also functional requirements of users modeled by notation of Unified Modeling Language (UML) as Use Cases. Within the Rational Unified Process (RUP), a scenario is an instantiation of a use case - take a specific path through the model, assigning specific values to each variable. Scenarios specify actors, roles, business processes, and the goal of the actor, and events that can occur in the course of attempting to achieve the goal. More complex tests are built up by designing a test runs through a series of use case [7, 8].

D. Model-View-ViewModel (MVVM)

When testing gained enough importance and Test-Driven-Development (TDD – focused on tests) together with Behavior-Driven-Development (BDD – focused on functional requirements) stared to be used in mainstream, there occurred a need of change in the software development industry. Old paradigms where tight coupled modules were standard and code behind approach was used for developing event driven User Interfaces become obsolete. Testing required new requirements on system components and system at all. Terms like loosely coupled modules, weak references, dependency injection and composite applications are characterization of nowadays trend in software development industry.

One of most significant change is usage new architectonic patterns for system design. Model-View-View Model pattern, showed in the Figure 2, is one of them.

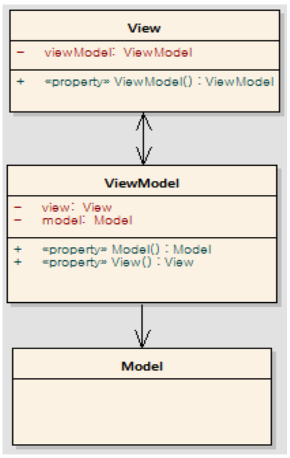
View
− viewModel: ViewModel
+ «property» ViewModel() : ViewModel
ViewModel
− view: View
− model: Model
+ «property» Model() : Model
+ «property» View() : View
Model

Fig. 2. Model-View-View Model architectonic pattern

Structure of this pattern consists from three main components:

1) View

View is the object representation of User Interface (UI). It usually contains only definition of user interface components with no or only necessary code-behind. This restriction has one useful side effect, that UI can be changed without any complication, because all the code and data are separated in View Model, which has bidirectional association with View. So when it comes to testing process, it is possible to create mock object for view and let it simulate its base functionality.

2) ViewModel

View Model is a special object which is used for encapsulation purpose of Model data. It is used for shaping data from their row form in order to present them (in literature is View Model sometimes referred as Presentation Model) to the View. Another View Model function is handling events which were propagated from View (in a loosely coupled way) and execute data driven operations in asynchronous manner. View model is usually tested without association to real View (only to mock of View object).

3) Model

Model is represented by domain structure of developed system. We know the term domain objects. In Domain-Driven-Design (DDD) it stand for pure model of system domain and encapsulate most of business behavior and functions of system. It should be copy of real world domain implemented by the notation of object oriented design. Model is usually tested via implementation of various unit tests.

II. The Proposal of Data Driven Scenario Tests Generation

As it was mentioned above scenario testing belongs to one of very important technique used in software development process. MDA (Model Driven Architecture) teach us that having a model for every aspect of developing system. However, if we want to stay agile and be able to react on changes in the project, there is necessary to connect these models together and reuse that parts which can be reused.

A. Prerequirements

Scenario testing is directly connected to functional requirements modeled in SRS, which are modeled as a group of Use Cases in UML notation. Right designed Use Case can serve as ideal starting point for the scenario, because it contains detailed description on how the function will be used by customer. Good candidate for scenario tests is Use Case that describes complex behavior, where communication with other Use Cases is also possible.

B. Overview of Data-Driven Test Generation Process

When we are talking about automated testing, there is an open question about how will the tests gain the data, which is needed for initialization and comparison. State of object-oriented system is represented by data attributes. And each test, in order to verify functionality follows from user requirements have to prepare data objects presented in it. This can be done by code, but it is the difficult way. There are several reasons why it is not very usable for the case of scenario testing:

- Data is hardcoded into the tests
- Initialization process for complex scenario can be really long
- It makes scenario less expressive
- There is a problem when we want to reuse the initial data in other scenario
- Test cannot be easily executed with different portion of starting data

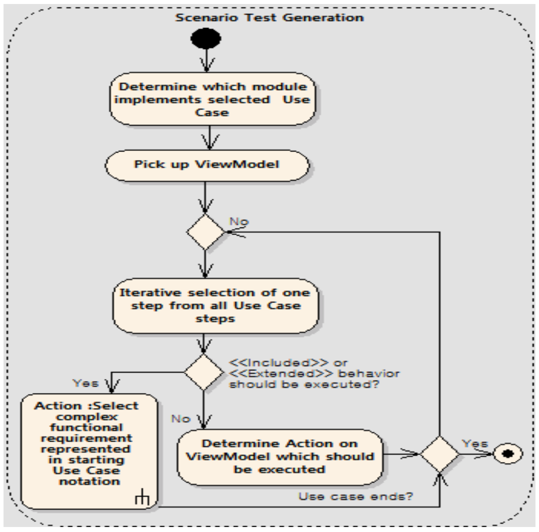
Scenario Test Generation
Determine which module implements selected Use Case
Pick up ViewModel
No
Iterative selection of one step from all Use Case steps
<<Included>> or <<Extended>> behavior should be executed?
Yes
Action :Select complex functional requirement represented in starting Use Case notation
No
Determine Action on ViewModel which should be executed
Yes
Use case ends?

Fig. 3. Detailed Activity diagram for Scenario Test Generation step

We proposed initialization of scenario tests from the structures called Data Snapshots, as the second step of activity diagram in the Figure 3 is showing.

Snapshoot stands for image, which contains clone of application state saved in specific format, so that it can be used later.

Snapshots are used also for storing valid information of scenario results. These data specifies how should state of our object-oriented system look like when the scenario will successfully end.

The last step in the process of Data-Driven scenario testing is comparison of the system state after the test ends, with referential values stored in valid Data Snapshoot.

C. *Detailed view on Scenario Test Generation*

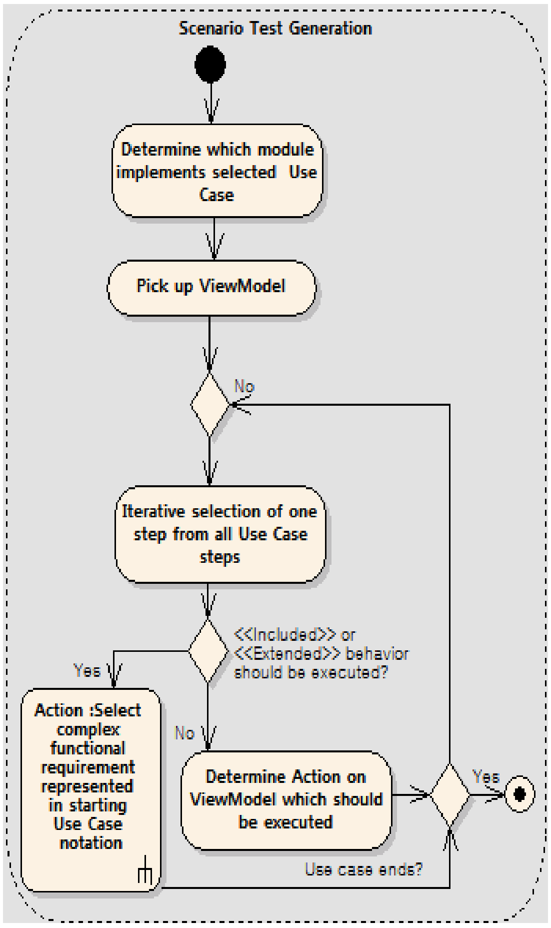
Scenario Test Generation
Determine which module implements selected Use Case
Pick up ViewModel
No
Iterative selection of one step from all Use Case steps
<<Included>> or <<Extended>> behavior should be executed?
Yes
Action :Select complex functional requirement represented in starting Use Case notation
No
Determine Action on ViewModel which should be executed
Yes
Use case ends?

Fig. 4. Detailed Activity diagram for Scenario Test Generation step

We continue our proposal explanation by detailed look at scenario test generation process described in the Figure 4. Scenario test generation starts after initialization of system state from snapshoot. Now system is ready to perform chosen complex scenario, but first it is necessary to determine which module of developed system actually implements peace of functionality required by Use Case. When the module is successfully found, extraction of View Model component is done, because in systems based on MVVM architecture View Model is the right place which encapsulated all functionality, which is provided to the end user with help of View component. Diagram continues by enumerating over all steps of the Use Case, where each step will result into one of these actions:

- If the step is Use Case representation of <<include>> or <<extend>> behavior, than action for selection of complex Use Case is recursively performed in order to generated sub scenario of our complex case.
- Else, the step belongs to this Use Case and that why View Model is responsible for executing relevant action.

According to proposed model it is possible to pass over the complex Use Case with its entire sub Use Cases and generate complex scenario.

III. Partial Implementation of Data Driven Scenario Tests Generation

Proposed model was implemented at PromanNG system developed by MMS Softec Company. It is an object oriented system for project management activities with composite architecture based on slightly altered version of MVVM design pattern. Software is programmed on top of .NET framework with C# as a default language and profit from modern approaches in software development industry. Design of system is highly impacted by MDA paradigm, but also trying to be agile enough, it means that every model is directly connected to code (auto generated) or peace of functionality, which is delivered to the end user.

A. *Application of MVVM*

1) *View*

View is implemented in notation of extensible Application Markup Language (XAML) language, with its code behind, which usually do not contain anything more than association to View Model stored as Data Context of UI View. This enables data binding and loosely coupled events (also called commands) between View and its View Model.

2) *View Model*

View Model class have access to all data and actions which user can perform through UI. It transforms the data in for the purpose of displaying them to the UI. View Model also implements necessary interfaces, which enables automatic tracking of changes on data objects for UI. Important is that View Model can be tested without initialization of real UI View, so it is ideal candidate for scenario testing.

3) *Model*

Model in the meaning of domain objects is accessible form Service class. This class publishes methods for manipulating with model and it is also extended via partial Business Rules class for the purpose of domain object validation and business rules checks.

B. *Object relational mapping (ORM)*

Domain objects in PromanNG uses object relational mapping techniques, for simplifying data access strategy of the system. ORM is because of several reasons:

- Objects are persisted to the relational (or other) data stores without any impendency problems.
- Database schema could be automatically created according to UML Class Model. So there is no need to keep two types of logical models up to date.
- It simplifies manipulation with the data and enables us to work in domain boundaries. All the queries are automatically generated.

There is one more usage of ORM technology, which really useful and time saving. Concept of Data Snapshots can be implemented in term of ORM. It is possible to take domain objects of developed system, fill them with initial values and use ORM persisting mechanism to create copy of system state. Resulting snapshoot could be used for initialization of system state before scenario test and also we can load persisted valid state in order to compare it with the one from scenario testing results. This concept can be used for creation of inputs and results which are reusable across the scenario testing.

C. Convention over Configuration

Our proposal is counting on assembly searching in order to identify correct application Module and its View Model. Good solution is to abide convention over configuration approach. Simple convections were established in order to simplify the task of determination right piece of software:

- Every application starts with name of functionality followed by key word Module
- Every View Model starts with name of Module followed by the key word View Model
- The same is applicable for View and Model classes but for now there are not needed in the process of scenario test generation

D. Example of complex Scenario generation

Data-Driven Scenario generation methodology was tested on object oriented system PromanNG. Figure 5 demonstrates example of complex functional requirement described by Use Case notation of UML.

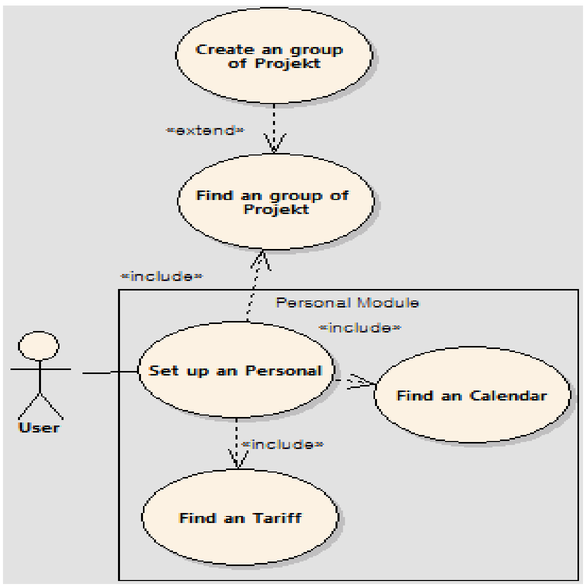

Fig. 5. Complex Use Case example for setting up an Personal

User requirement is implemented in one application module but it also uses functions from other module. For testing this specific feature, there was a scenario generated according to our proposal. UML Sequential diagram in the Figure 6 captures how the scenario is executed in the scenario testing process.

IV. Future Work

Scenario testing proposal is mainly aimed on functions verification, which is done be data comparison. In the future we would like to implement other features which could provide more information about tested system. It will be useful to capture data about system performance, memory compunction and statistically interpret them in order to move a step further and gain even better improvement of software product quality.

V. Conclusion

The aim of this article is to propose a method for data-driven scenario testing generation out of complex user requirement specification captured in Use Case notation. Method proposal is defined in UML Activity Diagram. For the purpose of loading initialization scenario data and also comparison of test results the structure called Snapshoot, which stores copy of system state, is used. Proposal is implemented and used for generate data-driven scenario tests in real world object oriented system.

References

[1] C. Kaner, “Exploratory Testing,” Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, November 2006.

[2] G. Myers, “The art of software testing, 2nd Edition” Wiley, 2004, New York, pp. 19, ISBN: 978-0-471-46912-4.

[3] W. Hetzel., “The Complete Guide to Software Testing, 2nd edition,” Wiley, 1993, New York, pp.6, ISBN: 0471565679

[4] A. Gao, I. Manolov, L. Lobo, N. Anderson, M. Hunter, R. Jaganathan, “WPF Application Quality Guide,” [Online] 2009, [cit. 2010-11-24]. Available on the Internet <: http://windowsclient.net/wpf/whitepapers/wpf-app-quality-guide.asp.

[5] C. Kaner, “The power of ‘What If…’ and nine ways to fuel your imagination: Cem Kaner on scenario testing,” Software Testing and Quality Engineering Magazine, Vol. 5, Issue 5 (Sep/Oct), p. 16-22, 2003.

[6] C. Kaner, J. Bach, „Scenario Testing“ [Online 2005], [cit. 2010-11-29]. Available on the Internet <:http://www.testingeducation.org/BBST/ScenarioTesting.html.>.

[7] R. Colard, “Test Design: Developing test cases from use cases,” STQE Vol. 1, Issue 4 (Jul/Aug), 1999.

[8] C. Kaner, „What is a good test case?,“ Software Testing Analysis & Review conference (STAR) East, Orlando, FL, 2003, Available on the

[9] Internet<:http://testingeducation.org/articles/what_is_a_good_test_case_star_2003_paper.pdf.>.

[10] C. Kaner, “Examples of Scenario Testing“, [Online 2004], [cit. 2010-12-01], Available on the Internet <:http://www.testingeducation.org/k04/ScenarioExamples.htm.>.

[11] S. Konachady, “What to expect form user scenario testing,“ [Online 2007], [cit. 2010-12-03], Available on the Internet <:http://blogs.sun.com/skonchady/entry/what_to_expect_from_user.>

[12] W. Lewis, G. Veerapillai, “Software Testing and Continuous Quality Improvement, 2nd edition,” Auerbach publications, 2005, pp. 29-39, ISBN: 0-8493-2524-2

[13] Microsoft, “Unit Testing Terms Glossary”, [Online 2010], [cit. 2010-11-25], Available on the Internet <:http://msdn.microsoft.com/en-us/library/ee413946.aspx.>

[14] A. Trnka, “Six Sigma Methodology with Fraud Detection. “ In: 9th WSEAS Interanational Conference on Data Networks, Communications, Computers (DNCOCO`10) : University of Algarve, Faro, Portugal, November 3-5, 2010. - ISSN 1792-6157. : WSEAS Press, 2010. pp. 162-165, ISBN 978-960-474-245-5.

[15] P. Vazan, O. Moravcik, “The Proposal of Procedure of Lot Size Determination in Production System. “

[16] In: DAAAM International Scientific Book. - ISSN 1726-9687. - 2008. Vol. 7. - Vienna : DAAAM International Vienna, 2008. - pp. 919-926, ISBN 978-3-901509-66-7.

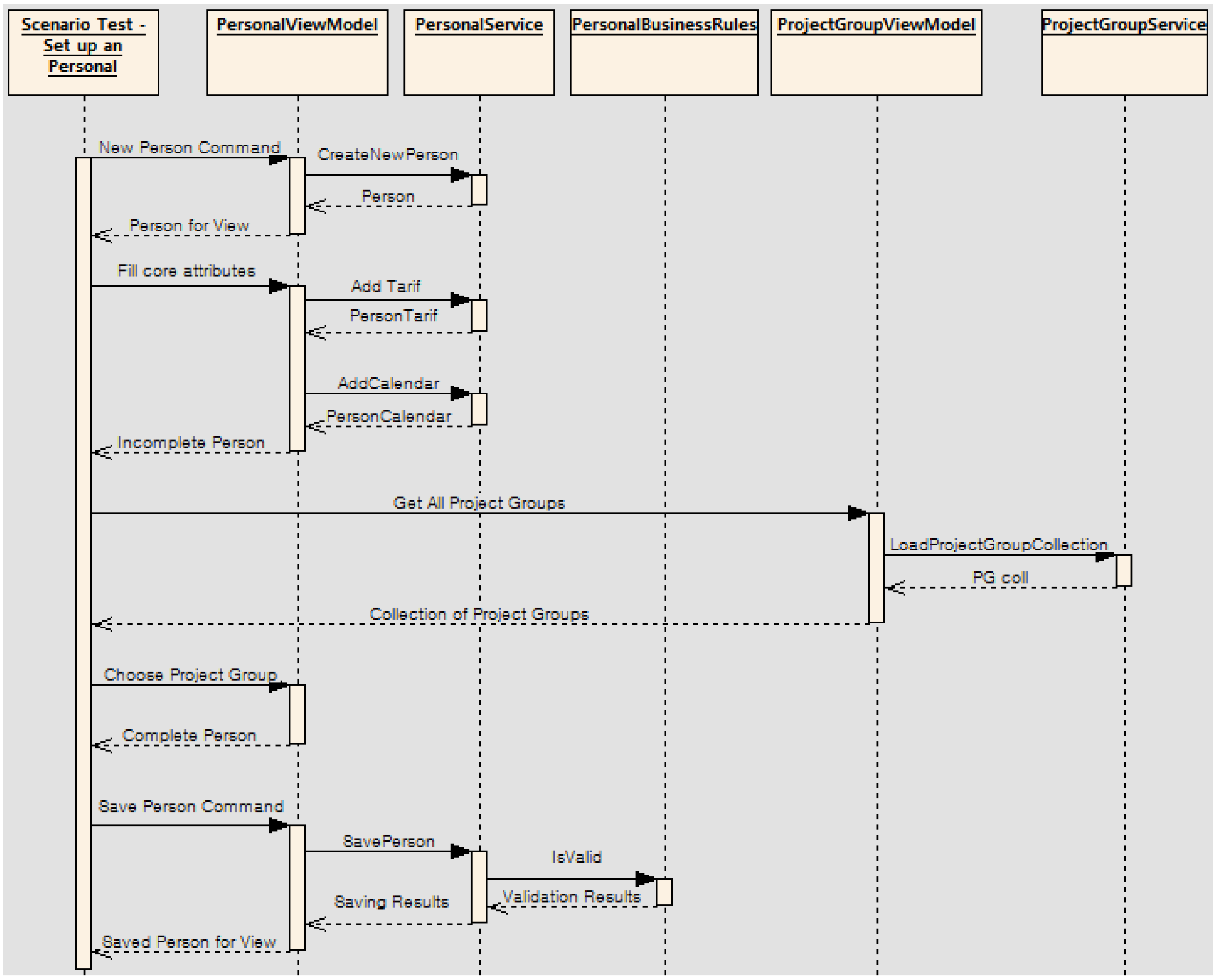

Fig. 6. Sequence diagram of generated scenario test

Pavol Tanuska (M '09 IACSIT, M '10 IEEE, M '10 IAENG) received his PhD degree (2000) in area of Applied informatics and automation from Slovak University of Technology (STU), Faculty of Material Science and Technology Trnava, Slovakia. Since 2004 he is an Associate Professor in Institute of Applied Informatics, Automation and Mathematics at the Slovak University of Technology, Trnava, Slovakia. His research and development activities include the field of Information systems development, especially problems of verification and validation and Database systems. He has published over 120 scientific and technical papers in national and international conference proceedings and journals. Email: pavol.tanuska@stuba.sk

Tomas Skripcak received his Master degree, in the field of applied informatics and automation in industry, from Slovak University of Technology in Bratislava in 2010. During studies (2005-2011), he worked as a software developer in MMS Softec Ltd. (Trnava-Slovakia), where he was responsible for design, development, documentation and testing of information systems based on .NET technology. In certain time, he is a PhD student at Slovak University of Technology in the field of process automation and informatization. Since February 2011, he is situated in Germany at Helmholtz-Zentrum Dresden-Rossendorf. He has published 2 papers in international conferences. His research area of interest includes software systems design and testing, natural user interface design, novel way of human-computer interaction.