Model-Driven Conception for Planning and Implementation of Software Configuration Management

Arturs Bartusevics, Leonids Novickis

***Abstract*—** Software Configuration Management controls evolution of software development process to include only valid configuration items in the final product. To establish this control, a set of tasks should be implemented in software development project: version control, build and deploy management, source code management etc. Usually companies already have tools and solutions to implement mentioned tasks. The main challenge is an implementation of software configuration management tasks in new projects. This implementation should be done with minimum additional efforts and customization using existing solutions and tools where it is possible. The study offers novel model-driven approach for planning and implementation of software configuration management using models with different level of abstraction. Firstly, meta-model for general model-driven approach is provided. Using this meta-model, three different models are developed for planning and implementation of software configuration management. Finally, simplified use case provided to describe designed models from practical side and direction of further works are underlined.

***Keywords*—** Software Configuration Management, Model-Driven Approach, Environment Model, Model Transformation.

I. Introduction

NOWADAYS software configuration management is not only challenge to choose an optimal version control system and branching strategy for a particular software development project [1, 4, 15]. Many different tasks should be solved to support the overall process, for example, identification of software configuration items, version control, configuration item status accounting, build management, release management, etc. The set of tools should be installed and integrated between themselves to support full configuration management process [1, 4]. Usually software development companies already have tools such as bug tracking systems, version control systems, build and deployment frameworks, release building tools, etc. The mentioned tools are implemented for software configuration management in the existing projects. The main challenges are implementation of all configuration management tasks for new projects as soon as possible, and achievement of fully-automated level to reduce manual efforts. To achieve this, existing solutions, scripts, frameworks should be reusable as much as possible. In case new solutions are to be developed, they should also be reusable. All of the mentioned solutions should be decomposed to independent units and parameterized. It means that a particular function, script, framework or some other unit should receive set of parameters and return expected result or error message. No any details about other solutions or provenance of received parameters should be included in the body of the current function, script or framework. Solutions organized by this way should be reusable and particular components should be used in other projects [18, 19]. In practice it means that, for example, function for compilation JAVA project from source should receive parameters and return an executable JAR file. A function body should not contains any details about version control repository, continuous integration servers, a bug tracking system or any hardcodes. All information should be provided by parameters. Only parameterized and independent functions could be used in other projects without additional efforts for customization. In practice very often solutions for version control, branching, building, installation and release management are mixed and very specific for a particular project. Some script, called “refresh_test_environment” could be imagined as an example. The script contains import of source code from particular repository, compilation and building details, hardcodes etc. All specific values of the current project, absolute paths of directories, addresses of servers are hard-coded in the script. It is not possible to use the same script in other project without additional customization. There is necessity to design framework for independent solution units to solve particular tasks of configuration management process. Firstly, the picture of general software configuration management process should be created. After that all needed solution units should be selected from the mentioned framework to apply general process that have been designed. It will reduce efforts for manual customizations and save resources during setup of configuration management in new projects.

This paper provides a new model-driven approach for implementation of software configuration management. Different models of the new approach allows getting general flow of configuration management process, select actions needed to implementation of mentioned flow and choose specific solution from special database for each action. The main scopes of the new model-driven approach are increase reusability of existing solutions for configuration management

and assist to design parameterized and reusable solutions to save up resources. Unlike other approaches, the new solution is not oriented to particular tool or platform and provides relations between abstract process and concrete solutions for a particular project. Other novelty of this research is a set of models designed using new approach. These models help to establish dynamic, documented planning of software configuration management, and allow reusing existing solutions instead of developing new one.

The current paper, firstly, provides overview of studies related to software configuration management to identify trends of improvement and the main problems of existing solutions. As a result, a new model-driven conception is provided for software configuration management. The main part of the article describes implementation of new model-driven approach for configuration management. Use case for designed models is given to illustrate practical aspects of them. Finally, directions of further works are provided.

II. Related Works

As far back as 1992, there was published an article [16] introduced to five future challenges in configuration management area. One of the main ideas is related to configuration management service model, which could be implemented with special tools. Many things have changed since then; more standards have been developed in software development area and tools for specific configuration management tasks have been designed. In a recent interview with a long-term expert in configuration management area [17] was mentioned the year 1998 when there was an attempt to create a “super tool” to integrate all solutions of configuration management in one place. The attempt was failed because solutions was too complicated in practice. Configuration managers and programmers were afraid of "majesty and mysticism" of such a tool. As the future trend, the configuration management expert [17] emphasizes challenge to enhance trust between configuration management and programmers. The main requirement for this is a clear procedure, which could be trusted. Other configuration management experts [1, 4] note that it is necessary to plan the process and only then apply tools for implementation otherwise solutions will be ineffective and will require additional resources. Modern solutions require reusable approaches that allow coming efficiently from the process general requirements to technical implementation.

During analyzing studies dedicated to approaches of reuse oriented solutions, more ideas from MDA have been found. The most important task in configuration management is the version control and the significant part of model-driven researches is devoted to solve this task [18, 19, 20]. New approaches try to improve version control and management of source code [20]. Abstract models designed to improve development of version control systems [18, 19]. There are also solutions offering an abstract model for overall configuration management process based on software quality standards and specifics of development methodologies [21, 22, 23]. Usually the proposed approaches are not supported by tools which could allow doing experiments and evaluating benefits. But the common thing for all solutions is MDA (Model-Driven Architecture) idea. According to the main principles of MDA approach, all models should have sources and transformation rules should be defined to transform one model to other. Solutions also should be supported by tools, otherwise practical implementation is not possible.

The following solutions [12, 7, 8] consider a configuration management process as a whole, not just a specific task. Solution in article [12] has been developed configuration management and model-driven development unification concept, meta-model, which allows creating an abstract model of software configuration. The solution is focused on projects where development is based on a model-driven approach, but there are no recommendations how this approach can be used in projects with classical development methodologies.

Configuration management principles for solution [8] were taken from the ITIL (Information Technology Infrastructure Library) [17] standards and later abstract models were created. With this models configuration, the management process could be created and later the model could be transformed into a platform specific model. Although that solution also includes an implementation for model-driven configuration management, it is focused on a single technology (JAVA).

Study [7] focuses on various configuration management tools mutual integration. In order to maintain a full configuration management process, a number of tools are required: version control systems, bug tracking systems, build servers, continuous integration servers and other tools. As practical experience indicates, all tools work separately from each other. The main scope of solution is to integrate different tools to solve all tasks for configuration management. However, in order to integrate various configuration management tools together, it is necessary to define a general concept of each integrated tool [7]. The study offers an ontology for configuration management process. This ontology is used as a configuration management model that shows how various configuration management tools should be integrated. The study does not have any specific instructions how the ontology can be used for a specific project configuration management. It is not clear what kind of ontology editors are advised to use and how to determine the moment when the changes have to be made.

III. Model-Driven Approach for Software Configuration Management

The main idea of new approach is based on MDA principles. To increase reuse of existing solutions in a field of software configuration management, process should be represented by models with different level of abstraction. Using transformation rules and reducing abstraction level in software configuration management model, it is possible to select concrete script or tool from general repository. Planning and implementation of software configuration management could be decomposed to three levels:

- Creating of computing independent model of

software configuration management process.

- Getting platform independent model using transformation rules.
- Getting platform specific model and selecting existing tools and scripts for all particular parts of modelled process.

New model-driven approach is abstract. It allows defining different domain specific languages to represent software configuration management from different sides. The approach is based on the following components:

- User – works with models, manage transformations where it is necessary.
- Metamodel – domain specific language for representing software configuration management from particular side.
- Model – representation of software configuration management in particular project. Model could be created or generated only from particular Metamodel.
- TransformationAlgorithm – an algorithm that works with particular kind of models and using transformation rules or manual interactions from User, generates a model with different level of abstraction.
- Element – additional part of model-driven approach that devoted to help to particular TransformationAlgorithm. For example, transformation algorithm could use special database where different scripts, tools or libraries are stored. Algorithm could use this database and could allow User to select some particular solution for concrete part of process.

A meta-model for model-driven software configuration management provided in Fig. 1.

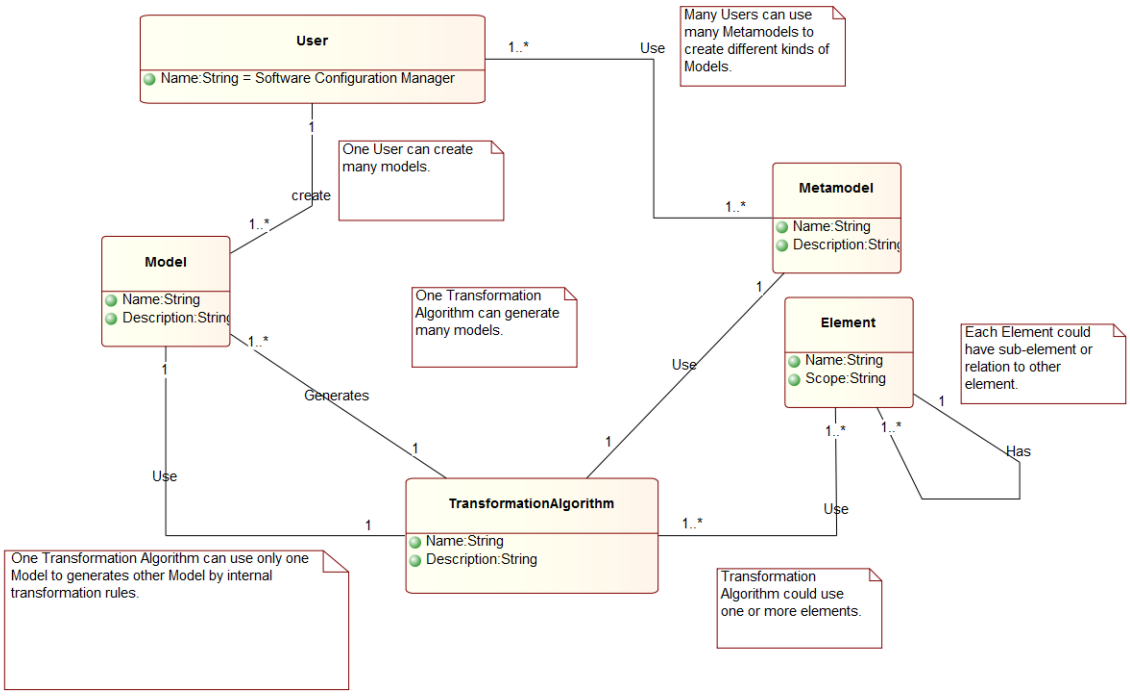

Fig. 1 Meta-Model for Model-Driven Software Configuration Management

As represented in Fig. 1., User can generate different Models using special Metamodels or defined domain specific languages. Conception also could contains transformation algorithms to transform one kind of models to other. Depends on situation and type of model, transformation algorithm could use Metamodel or some additional Elements, for example, warehouse of scripts, tools or frameworks.

IV. Implementation of New Model-Driven Approach

Implementation of models for software configuration management is based on conception provided in previous section. New graphical domain-specific language is designed to represent all new models provided in this study. New domain-specific language developed using MetaEdit+ tool. This tool allows developing new graphical modelling languages and changing own language during modelling process [22]. During development of new software configuration models, ideas from [20, 21] paper have been taken. Mentioned studies underline that in 21-century script for support of software configuration management could not be static. Only dynamic and model-driven scripts and tools could increase its reuse and save up resources during implementation of software configuration management process in new

projects.

Novelty, presented in current article, is improvement of researches described in papers [23, 24].

There are three levels of models in provided approach:

- *Platform Independent Environment Model (PIEM)* – provides a model of all instances included in a software development project. A model also contains all flows of software changes between different environments. This model provides overview of general infrastructure of the project in context of instances. In additional, this model contains all actions needed to transfer software changes between different environments. The actions are abstract and do not contain any specific details for a particular platform. For example, action "Compile" should be used to compile software from source code, but in this model, any details about software technology, compilation algorithm, and platform are not known.
- Platform Specific Action Model (PSAM) – provides an extended variant of Platform Independent Environment Model because actions are fulfilled with details about platform, technology, specific scripts, etc. In this model, action "Compile" already have information about details of technology, compilation algorithms, platform, etc. It means that in this model all details are known, for example, it could be ANT build script for JAVA projects.
- Code Model (CM) – provides a set of files and scripts generated according to PSAM model. Scripts are executable from continuous integration server to implement all transfers of software changes between environments described in PIEM model.

General picture of a new model-driven framework provided in Fig.2.

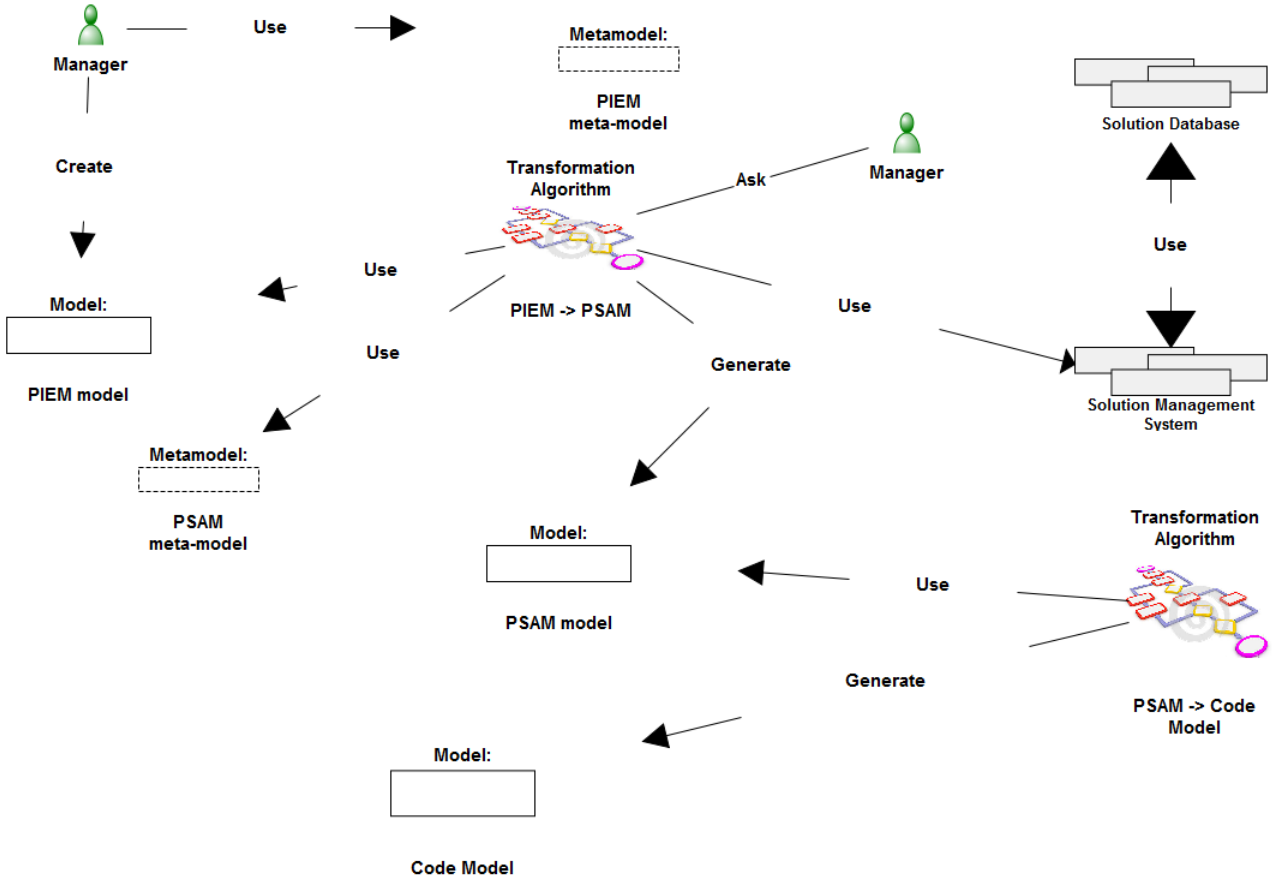

Fig. 2 General Overview of Implemented Model-Driven Approach

Designed model-driven framework starts with creation of Platform Independent Environment Model by Manager. To make PIEM model (Fig. 2.) manager works with PIEM meta-model – special domain-specific language that designed by MetaEdit+ to describe all environments in project, flows of software changes between them and actions needed to implement these transfers.

During the next step, transformation algorithm "PIEM -> PSAM" works with PIEM model that is created by Manager. Using meta-model of PSAM model and Solution Management System, the transformation algorithm generates PSAM model – platform specific action model. As it could be seen in Fig. 2., "PIEM -> PSAM" transformation algorithm asks Manager to choose platform and tools during generation of PSAM. Solution Management System is additional element of

designed model-driven framework, which provides graphical user interface for Manager to choose solutions for particular tasks from Solution Database. The structure of Solution Database provided in Fig. 3.

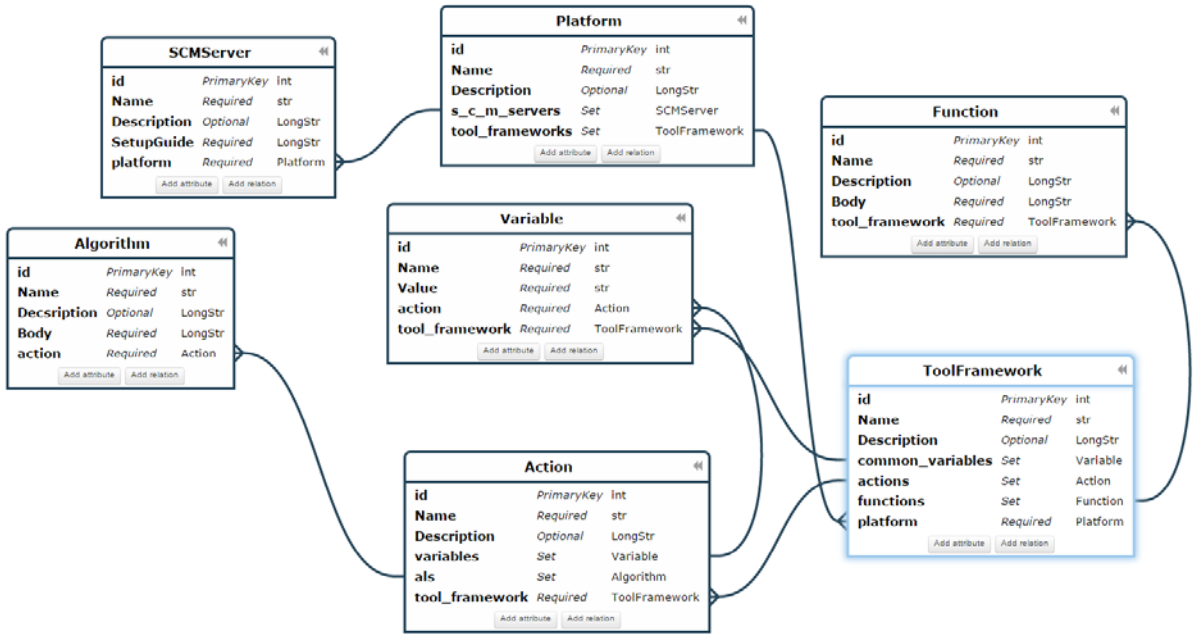

Fig. 3 Structure of Solutions Database

Solutions Database in context of provided solution is a warehouse of all experience, tools, scripts and solutions in particular company. Database contains the following tables:

- Platform – contains information about all platforms where solutions of software configuration management implemented, for example: Linux, Windows etc.
- SCMServer – continuous integration server devoted to manage all actions of software configuration management. Examples: Hudson, Jenkins, Bamboo, CruiseControl etc.
- ToolFramework – contains information about all tools and frameworks needed for implementation of actions. For example, to merge software changes from one source code branch to other, at least one tool of version control is required. This tool could be, for example Subversion. In this case, ToolFramework table contains a row “Subversion”. This row has relations to other database tables: Action, Variable, and Function. The table Function contains all functions that could be executed from particular platform using this tool. In case of Subversion, functions could be “svn_merge”, “svn_commit”, “svn_update” etc. ToolFramework rows also have relations to actions that could be implemented and variables that should be defined to execute particular actions.

Transformation algorithm PIEM ->PSAM (Fig. 2.) prepares structure from two parts:

- Information about project, SCM server jobs, environments. This part of PSAM should be taken from PIEM model.
- Information about platform, SCMServer, actions, tools or frameworks, variables. This part of PSAM model should be fulfilled by selecting particular solutions from database, provided in Fig. 3. PIEM -> PSAM transformation algorithm allows Manager to choose solutions from database using Solution Management System.

Meta-model of PSAM provided in Fig. 4.

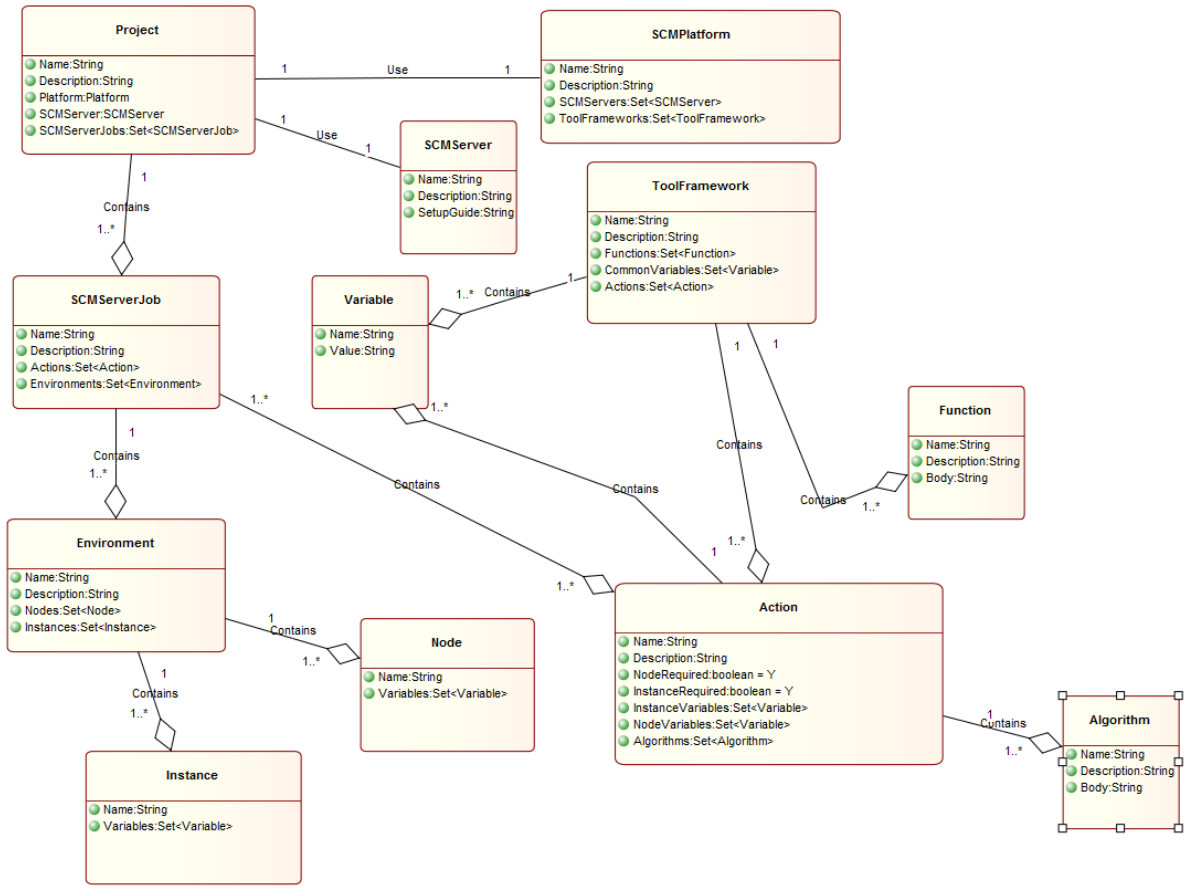

Fig. 4 PSAM meta-model

Transformation algorithm “PSAM -> Code Model” provided in Fig. 2. generates a set of source code files. This source code is executable from SCM Server, for example, Jenkins.

Implementation of software configuration management process by new model-driven framework, using PIEM, PSAM and CM models allows using existing libraries, frameworks, tools, scripts and functions that are already implemented in other projects. It could save up resources and reduce risks of unexpected errors. Additionally, implementation process contains strongly defined steps and such steps are documented and supported by graphical modelling tools developed using MetaEdit+.

V. Use Case for Models of Software Configuration Management

Current section presents implementation of simplified configuration management process to illustrate practical application of designed models: PIEM, PSAM and CM.

There are two environments: DEV and TEST. Current example introduce a software “X”. This software has developing by programmers in development environment, called DEV. To track problems and changes in mentioned software, programmers use bug-tracking system JIRA. Each change in software should be related to particular issue in JIRA. Source code of software “X” controlled by version control system Subversion. After changes, related to particular issues, are done, it should be transferred to TEST environment for testing process. Jenkins server will be used to manage all task in current process.

In current test case, only two steps of process will be implemented:

- Extract from JIRA all issues ready for testing,
- Find particular changes in Subversion system for all issues detected during previous step.

PIEM model provided in Fig. 5. defines the following components:

- Two environments: DEV and TEST.
- View for continuous integration server “test_delivery” which will contains all jobs related to transfer of software changes between mentioned environments.
- Particular job name “DEV_TO_TEST”.
- Set of actions. Action ‘getIssues’ should extract issues from JIRA, but action ‘getRevisions’ should find particular revisions in Subversion repository.
- Script ‘dev_to_test.sh’ contains a source code for implementation of actions mentioned before. Code of this script will be available only in Code Model.

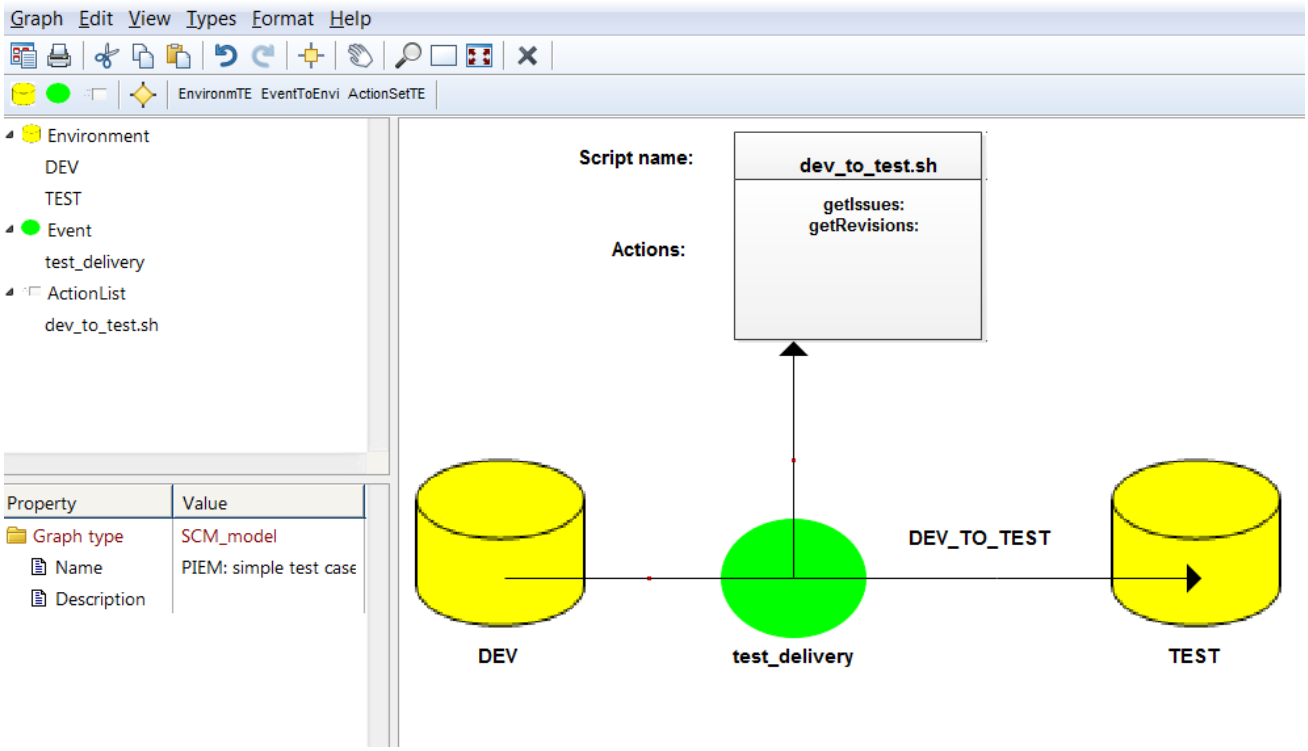

Fig. 5 Platform Independent Environment Model

Ready PIEM model provided in Fig. 5. has been transformed to PSAM model by transformation algorithm "PIEM->PSAM". Transformation process has the following steps:

- Using MetaEdit+ standart functionality, ready PIEM model exported to XML format.
- PIEM model in XML format has been parsed by "PIEM->PSAM" algorithm. During this step, initial structure of PSAM has been prepared: information about project, environments and jobs of continuous integration server.
- User selects platform, kind of SCM Server, tools for implementation of each action in PIEM.
- According to items selected at previous step, algorithm asks user to define nodes and instances introduced by PSAM. To execute particular actions by tool or framework, depends on action, node or instance should be created. The first action 'getIssues' should extract issues from JIRA. However, process could have many JIRA projects. So, algorithm asks to specify all instances of JIRA. Action 'getRevisions' should get revisions from Subversion repository. However, software could contains more than one component and locations of components should be different. So, each subversion repository should be specified as part of independent node.
- Finally, ready PSAM model has been stored in XML format and it is ready to be transformed to Code Model.

In context of this study Code Model for Linux platform and transformation algorithm "PSAM -> Code Model" have been designed. Transformation algorithm works with PSAM model in XML format and prepare structure of Linux Shell scripts ready to be executed from Jenkins continuous integration server. The structure of Jenkins jobs generated according to PIEM model. Fig. 6. represents structure of Jenkins server for experiment. Name of view "test_delivery" and name of job "DEV_TO_TEST" have been taken from PIEM model.

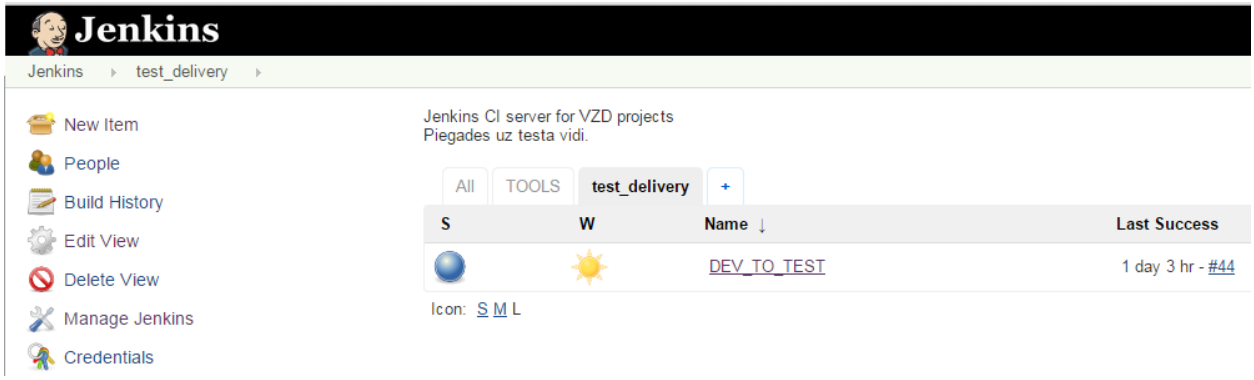

Fig. 6 Jenkins Server

Code Model for current experiment provided in Fig. 7.

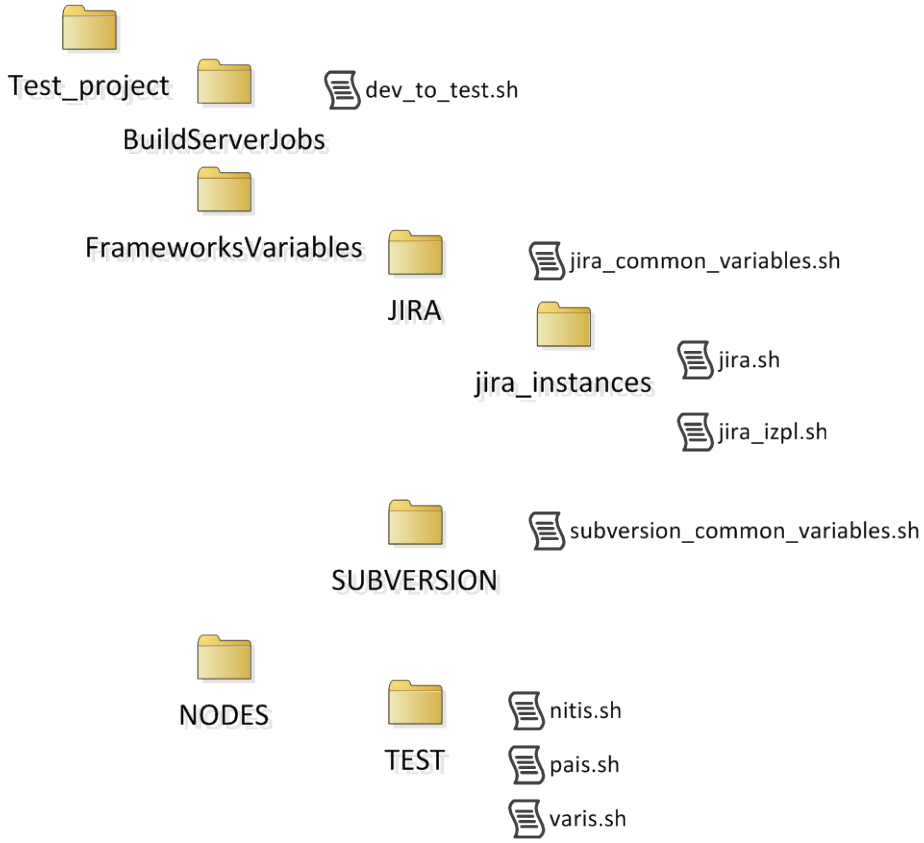

Fig. 7 Code Model

Scripts located in directory "NODES/TEST" represents variables of different parts of software "X" mentioned before. These scripts are generated automatically using values entered by user during creation of PSAM model. Three shell scripts nitis.sh, pais.sh and varis.sh means that software "X" contains three components and each component stored in different Subversion repository.

All shell scripts in directory "FrameworksVariables" represents all necessary actions with special tools, in our case: JIRA and Subversion. Scripts located in folder "FrameworksVariables/JIRA/jira_instances" represents two instances or two different JIRA projects from what issues should be extracted.

Scripts jira_common_variables.sh and subversion_common_variables.sh contains common variables needed to call particular functions of SUBVERSION and JIRA frameworks. Values of variables defined during creation of PSAM model, but in Code Model, variables are only represented in Linux Shell format.

Finally, script BuildServerJobs/dev_to_test.sh is a main script of current experiment. This script will be called from Jenkins job "DEV_TO_TEST" provided in Fig. 6. Script contains the following parts:

- Script name – value is taken from PSAM model.
- Reference to functions of each tool or framework. Any framework contains a set of functions. Each function devoted to execute single action. For example: commit, update, getIssues, merge etc.
- Reference to common variables of each tool or framework. In our case, there are script jira_common_variables.sh and subversion_common_variables.sh.
- References to home directories of all available instances and nodes. In our case, there are scripts in folders "NODES/TEST" and "FrameworksVariables/JIRA/jira_instances".
- Algorithms for each action from PSAM and PIEM models. Algorithm in cycle goes through each node and instance depend on action type and execute necessary actions for all nodes or instances. It is important that algorithms are only recommended but not necessary for real-live solution. After Code Model is ready, configuration manager could modify default algorithm generated by framework. However, functions of frameworks or tools should not be modified in normal case. All changes in functions of frameworks should be tracked and should be under classic procedure of software configuration management.

Fragment of shell scrip "dev_to_test.sh" provided at Fig. 8. This fragment contains an algorithm for action "getRevisions" and references to variables mentioned before.

```
#!/bin/sh

#This script collect and install delivery from DEV to TEST

#Konstantes
C_SCRIPT_NAME="DEV_TO_TEST"

#Tools and Frameworks

#================JIRA=============================================================#
. ./Platforms/Linux_shell/ToolsFrameworks/JIRA/jira_functions.sh
. ./Projects/Test_project/FrameworksVariables/JIRA/jira_common_variables.sh
export JIRA_INSTANCE_HOME="./Projects/Test_project/FrameworksVariables/JIRA/jira_instances"

#================SUBVERION========================================================#
. ./Platforms/Linux_shell/ToolsFrameworks/SUBVERSION/subversion_functions.sh
. ./Projects/Test_project/FrameworksVariables/SUBVERSION/subversion_common_variables.sh
export NODES_HOME="./Projects/Test_project/NODES/TEST"
#=================================================================================================================================================

#Action getRevisions
#=================================================================================================================================================

#Iet cauri visām nodem, nolasa attiecīgus parametrus un nosaka uz atbilstošās vides nepiegādātas revīzijas
for nodes in `ls ${NODES_HOME}`
do
    #Inicializē konkrētas dalas mainīgus
    . ${NODES_HOME}/${nodes}
    #nosaka nepiegādātas revīzijas
    SUBVERSION_GET_UNDELIVERED_REVISIONS ${SUBVERSION_URL} ${SUBVERSION_USER} ${SUBVERSION_PASSW} ${SUBVERSION_START_REVISION} ${SUBVERSION_ATRRIBUTE} ${SUBVERSION_TEXT_FILE_FOR_REVISIONS}

    echo "Nepiegādātas SVN revīzijas ir:"
    cat ${SUBVERSION_TEXT_FILE_FOR_REVISIONS}

done
# END Action getRevisions
#=================================================================================================================================================
```

Fig. 8 Fragment of script for Jenkins job

VI. Differences of Model-Driven Approach from Other Related Works

This section devoted to underline differences of provided approach from other solutions for software configuration management. Authors introduce the following differences:

- Model-Driven approach for implementation of software configuration management represented as a meta-model with abstract components. This fact allows designing new domain specific language to represent software configuration management by different sides. Conception also allows using different transformation algorithm with special transformation rules and together with some external elements. In this study, external elements are Solution Management System and Solution Database.
- Approach provides strongly defined, documented and supported by tools steps for implementation of software configuration management. Initially, approach shows only steps and meaning of different models and do not impose to use concrete tool that "will solve any problems".
- Approach provides a way to organize existing solutions to increase its reuse. After solutions are stored in Solution Database, each new project will use existing functions, scripts or frameworks, as it is possible. As experiment from previous section shows, using provided approach, code-writing actions are minimal, because significant part of all

code generated automatically and represented in Code Model.

- Provided approach uses nodes and instances. It allows supporting dynamic scripts because adding or removing instances/nodes do not requires modifications in main scripts.

VII. Lessons Learned using Platform Independent Environment Model

Platform Independent Environment Model has been used in different software development projects and the following lessons are learned:

- Only tested configuration could be trusted and successfully moving from one environment to other. For example, if in TEST environment five changes are tested, all of them should be moved to the next environment (QA – quality accepting). However, technically, it is possible to move only particular changes from one environment to other, and such release could be unstable because of functional dependencies between different items. It is very important to explain all risks during moving changes between environments partially, so customers should see it on PIEM.
- PIEM could be perfected by new elements to improve simulation of single environment. For example, in customer's opinion, it could be better if element "Environment" will contain supplements to describe infrastructure better. It could be achieved by entering such elements as servers, connections between them, firewalls, size of storages, etc.

VIII. Conclusions and Further Works

The study provides new model-driven approach for implementation of software configuration management. The main scope is to increase reuse of existing solutions and reduce efforts to implement the process in other projects. Meta-models for Platform Independent Environment Model, Platform Specific Action Model and for Code Model designed as implementation example of the provided approach. In addition, use case for designed models is given. Finally, differences from other approach are underlined.

In order to continue research, it is necessary to carry out the following activities:

- With the help of experiment, develop criteria that evaluate models benefits in software development projects,
- Based on developed criteria, evaluate benefits of designed models,
- Develop criteria to assess whether the developed model-driven approach for configuration management implementation corresponds to guidelines of ISO/IEC 15504, ITIL, CMMI standards.
- Design Code Models and transformation algorithms for other platforms.
- Add and improve tools and frameworks in existing platforms.

The approach provided in this article is abstract and only general stages, kinds of models and basic elements are defined. The authors hope that the new approach will generate new ideas because many useful lessons could be learned from different implementations of this model-driven approach.

Acknowledgment

The research has been partly supported by the project eINTERASIA "ICT Transfer Concept for Adaptation, Dissemination and Local Exploitation of European Research Results in Central Asia's Countries", grant agreement No. 600680 of Seventh Framework Program Theme ICT-9.10.3: International Partnership Building and Support to Dialogues for Specific International Cooperation Actions – CP-SICA-INFSO.

References

[1] Aiello, R., Configuration Management Best Practices: Practical Methods that Work in the Real World (1st ed.). Addison-Wesley, 2010.

[2] Berczuk, A., Software Configuration Management Patterns: Effective TeamWork, Practical Integration (1st ed.). Addison-Wesley, 2003.

[3] Calhau R., Falbo R., A Configuration Management Task Ontology for Semantic Integration. Proceedings of the 27th Annual ACM Symposium on Applied Computing Pages 348-353 ACM New York, NY, USA, 2012.

[4] Giese H., Seibel A., Vogel T., A Model-Driven Configuration Management System for Advanced IT Service Management. Available at: http://www.hpi.unipotsdam.de/giese/gforge/publications/pdf/GSV-MRT09_paper_7.pdf, 2009.

[5] Nikiforova O., Pavlova N., Gusarovs K., Gorbiks O., Vorotilovs J., Zaharovs A., Umanovskis D., Sejans J. Development of the Tool for Transformation of The Two-Hemisphere Model to The UML Class Diagram: Tehnical Solutions and Lessons Learned. Proceedings of the 5-th International Scientific Conference „Applied Information and Communication Tehnologies", 2012, Jelgava, Latvia, pp. 11-19.

[6] Osis J., Asnina E., Model-Driven Domain Analysis and Software Development: Architectures and Functions. IGI Global, Hershey - New York, 2011, 514 p.

[7] Pindhofer W., Model Driven Configuration Management. Master work of Wien University, Wien, 2009.

[8] Удовиченко, Ю. Управление изменениями и кессонная болезнь проектов. Available at: http://experience.openquality.ru/software-configuration-management/, 2011.

[9] Dart, S., The Past, Present, and Future of Configuration Management. CMU/SEI-92-TR-8, 1, 25., 1992.

[10] CMCrossroads. 2014. How Configuration Management Is Changing: An Interview with Joe Townsend. [ONLINE] Available at: http://www.cmcrossroads.com/interview/how-configuration-management-changing-interview-joe-townsend?page=0%2C0. [Accessed 02 September 2014].

[11] Yongchang, R., Fuzzy Decision Analysis of the Software Configuration Management Tools Selection. In ISCA 2010. France, 19-23 June, 2010. Information Science and Engineering (ISISE): ACM. 295 - 297., 2010.

[12] de Almeida Monte-Mor, J., GALO: A Semantic Method for Software Configuration Management. In Information Technology: New Generations (ITNG), 2014. USA, 7-9 April, 2014. ITNG: IOT360. 33 - 39., 2014.

[13] Toth, Z., Using Version Control History to Follow the Changes of Source Code Elements. In Software Maintenance and Reengineering (CSMR), 2013. Italy, March 5–8, 2013. IEEE Digital Library. 319 - 322., 2013.

[14] Estublier, J., Software configuration management: a roadmap. In ICSE '00 Conference on The Future of Software Engineering. USA, June 4-11, 2000. IEEE Digital Library: ACM. 279 - 289., 2000.
[15] Ruan, Li., A new configuration management model for software based on distributed components and layered architecture. In Parallel and Distributed Computing, Applications and Technologies, 2003. China, August 27-29, 2003. IEEE Digital Library: IEEE. 665 - 669., 2003.
[16] Mingzhi, M., A New Component-Based Configuration Management 3C Model and its Realization. In Information Science and Engineering, 2008. China, December 20-22. 2008. IEEE Digital Library: IEEE. 258 - 262., 2008.
[17] Galup, S. D., Dattero, R., Quan, J.J., Conger, S., An Overview of IT Service Management. Commun. ACM, 2009, vol. 52, no. 5, pp. 124-127.
[18] Szenssi, S., Vamossy, Z., Kozlovszky, M., Evaluation and comparison of cell nuclei detection algorithms. 16th International Conference on Intelligent Engineering Systems (INES 2012), Lisbon, 2012, pp. 469-475.
[19] Kerekes, Z., Toth, Z., Szenasi S., Vamossy, Z., Sergyan, Sz., Colon Cancer Diagnosis on Digital Tissue Images. Proceedings of IEEE 9th International Conference on Computational Cybernetics. Tihany, 2013, pp. 159-163.
[20] Ragan, T., 21st-Century DevOps--an End to the 20th-Century Practice of Writing Static Build and Deploy Scripts, Linux Journal, 230, pp. 116-120, Computers & Applied Sciences Complete, EBSCOhost, viewed 22 October 2014.
[21] Azoff, R., DevOps: Advances in Release Management and Automation. [ONLINE] Available at: http://electric-cloud.com/wp-content/uploads/2014/06/EC-IAR_Ovum-DevOps.pdf, 2014.
[22] Vasiljevics, I., Milosavljevics, G., Dejanovics, I., Filipovics, M., COMPARISON OF GRAFICAL DSL EDITORS. The 6th PSU-UNS International Conference on Engineering and Technology (ICET-2013), Novi Sad, Serbia, May 15-17, 2013.
[23] Novickis, L., Bartusevics, A. Model-Driven Software Configuration Management and Environment Model. In: Recent Advances in Electrical and Electronic Engineering: Proceedings of the 3rd International Conference on Systems, Communications, Computers and Applications (CSCCA"14), Italy, Florence, 22-24 November, 2014. Florence: WSEAS Press, 2014, pp.132-140. ISBN 978-960-474-399-5. ISSN 1790-5117.
[24] Novickis, L., Bartusevičs, A., Lesovskis, A. Model-Driven Software Configuration Management and Semantic Web in Applied Software Development. Proceedings of the 13th International Conference on Telecommunications and Informatics (TELE-INFO '14), IIstanbul, Turkey December 15-17, 2014.

Arturs Bartusevics currently is a Doctoral Student at Riga Technical University, the Faculty of Computer Science and Information Technology, the Institute of Applied Computer Systems. He obtained BSc (2008) and MSc (2011) degrees in Computer Science and Information Technology, respectively, from Riga Technical University. His research areas are software configuration management, release building and management process and its optimization. He works at Ltd. Tieto Latvia as a Software Configuration Manager.
E-mail: arturik16@inbox.lv

Leonids Novickis is a Head of the Division of Software Engineering at Riga Technical University. He obtained Dr.sc.ing. degree in 1980 and Dr.habil.sc.ing. degree in 1990 from the Latvian Academy of Sciences. He is the author of 180 publications. Since 1994, he is regularly involved in different EU-funded projects: AMCAI (INCO COPERNICUS, 1995-1997) – WP leader; DAMAC-HP (INCO2, 1998-2000), BALTPORTS-IT (FP5, 2001-2003), eLOGMAR-M (FP6, 2004-2006) – scientific coordinator; IST4Balt (FP6, 2004-2007), UNITE (FP6, 2006-2008) and BONITA (INTERREG, 2008-2012) – RTU coordinator; LOGIS, LOGIS-Mobile and SocSimNet (Leonardo da Vinci) – partner, eINTERASIA (FP7, 2013-2015)- project coordinator. He was an independent expert of IST and Research for SMEs in FP6 and FP7. He is a corresponding member of the Latvian Academy of Sciences and an elected expert of the Latvian Council of Science. His research fields include Web-based applied software system development, business process modeling, e-learning and e-logistics.
E-mail: lnovickis@gmail.com