Model-Driven Software Configuration Management and Environment Model

ARTURS BARTUSEVICS, LEONIDS NOVICKIS
Faculty of Computer Science and Information Technology
RTU
Kalku Street 1, Riga LV-1658, Latvia
LATVIA
arturik16@inbox.lv, lnovickis@gmail.com

Abstract: - Software Configuration Management controls evolution of software development process to include only valid configuration items in the final product. The study offers a new model-driven approach for the implementation of software configuration management. The new approach is based on creation of configuration management models with different scopes and level of abstraction. The first model in context of new solution describes flows of software changes between different environments. The model helps to detect general risks related to software configuration management domain. The new meta-model is developed for creation of environment model as part of provided solution. This article describes an environment meta-model with examples, as well as potential benefits of the model. Finally, learned lessons and further research directions are also provided.

Key-Words: - Software Configuration Management, Model-Driven Approach, Environment Model, Model Transformation

1 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]. More 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 Environment Model developed as part of a new model-driven approach. Environment Model helps to make general and abstract overview of software configuration management process and detect general risks from the side of configuration management area.

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 point of view is provided for software configuration management. The main part of the article describes a new model-driven approach for configuration management. Environment Model, as part of a new model-driven approach described with an illustrative example. Use case for a general model-driven approach and Environment Model is given. Finally, learned lessons from using of Environment Model in practice and directions of further works are provided.

2 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, metamodel, 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.

3 Interpretation of Software Configuration Management for New Model-Based Approach

During exploring literature related to configuration management area and getting practical experience in position of software configuration management, the authors provide new interpretation of mentioned process. One of the main scopes of configuration management based on the question “How to transfer certain software changes from one instance into another at the right moment?” [1, 4, 8, 15]. Answers to this basic question requires to solve the main tasks of software configuration management process such as configuration identification, version control, status accounting, build and deploy management, release management [1, 8]. To answer this question, firstly, all instances and flows of changes should be known. Secondly, all actions should be known to apply all flows between mentioned instances. Finally, technical solutions should be defined for each action. Therefore, the interpretation of software configuration management is based on three levels. The first level provides view of all instances included to software development process and all possible changes flow between instances. The second level shows actions needed to apply all flows between instances. The last level provides particular solutions for each action defined at the previous level. Based on this interpretation, a model-driven approach have been designed.

4 Model-Driven Approach for Configuration Management Process

A new model-driven approach for implementation of software configuration management designed is based on interpretation described in the previous section and ideas of MDA [6].

There are three levels of models in provided approach:

- **Environment Model (EM)** – 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. Based on state of environments and flows of changes, general risks of configuration management could be detected.

- **Platform Independent Action Model (PIAM)** – provides set of actions needed to apply all flows from Environment Model. 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 Action Model because the same actions are fulfilled with details about platform, technology, specific scripts, etc. In this model, action “Compile” already have information provides 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.

General picture of a new model-driven approach is provided in Fig.1.

Meta-model of EM (Environment Model)  
Use Metamodel  
Configuration Manager  
1  
EM (Environment Model)  
2  
Rules of Risks, Compilation  
3  
Transformation Rules  
4  
Solutions database  
5  
Expert System  
6  
PSAM (Platform Specific Action Model)  
Configuration Management Domain  

Fig. 1. Model-Based Approach for Configuration Management Process.

Illustration of a model-driven approach in Fig. 1. is represented as flow with interactions from a configuration manager. The arrows with numbers mean particular stages of the approach. The first stage “1” provides creation of Environment Model from a special meta-model. Configuration manager builds Environment Model from the set of components from the mentioned meta-model. During the second stage “2”, created Environment Model should be compiled by special block in Expert System, called “Rules of Risks, Compilation”. Expert System in context of this work is a special warehouse for different blocks of rules and database with ready solutions of actions. After stage “2”, a configuration manager has ready and compiled Environment Model with the description of general configuration management risks if such exists. Stage “3” explores ready Environment Model by special block of Expert System called “E->P”. The main task is to detect actions needed to apply each flow between environments. During stage “4”, Platforms Independent Action Model performing from actions defined at stage “3” and from meta-model of PIAM. The stages “5” and “6” require the second interaction from configuration manager to analyze ready Platform Independent Action Model and choose solutions for each action from “Solutions Database”. Structure of “Solution Database” provided in Fig. 2.

Fig. 2. Structure of Solutions Database.

Solutions Database contains all information about all configuration management actions described in PIAM model. For example, action “Compile” could have five different solutions to compile software from source code for the following technologies: JAVA, Ruby, C++, Oracle, C#. The mandatory requirement is that all solutions are parameterized and do not have dependencies on solutions of other actions. For example, compilation script should not know any details about other actions from PIAM, hardcoded from bug tracking management, hardcoded hosts, absolute paths, etc. All necessary things should be given as parameters. Any solution stored in Solution Database has the following attributes:

ID – unique identifier in database;
• PlatformID – reference to platform;
• ActionID – reference to action. Table “Action” contains all possible actions from PIAM meta-model;
• NeededTools – set of tools to implement this solution;
• LocationsOfSolutions – information about ready scripts, frameworks, functions, including paths, locations of servers, web-pages etc.;
• Description – some notes provide additional information about implementation, specific technical details.

During the last stage “7”, ready PSAM model should be implemented. Each solution of each action should be technically applied in Configuration Management Domain according to information in Solutions Database.

5 Environment Model
The scope of Environment Model is simulation of flows of software changes between environments. During simulation, firstly all environments should be detected according to such facts as development methodology, phase of project, etc. Then, all flows of software changes should be defined. Each flow through which configuration items version comes from one environment to another belongs to a specific event. One event may have multiple flows. For example, if the event is "move configuration from DEV instance to TEST instance", the event should have two flows. One flow will be transferred to TEST1 instance to make sure that configuration is valid, but in the next flow the same configuration will be transferred to TEST instance. Environment Model shows all events involved in configuration transfer between different instances, all flows of each event and their sequence. Environment Model could be created by a configuration manager, who will be able to add, modify and delete events, flows and environments.

Environment meta-model contains the following elements:
• Set of graphical elements which are available for creating of Environment Model,
• XML interpretation of all graphical elements and hierarchy of them,
• Algorithm which converts Environment Model from graphical to XML format,
• Classes for Environment Model elements and compilation algorithm.

Graphical elements of Environment meta-model are available for a configuration manager to create Environment Model. When a configuration manager decides that the model building is complete, he runs an algorithm that converts the graphical model into XML format. Taking into account notation and hierarchy of elements, transformation algorithm converts Environment Model into XML format. Environment Model in XML format should be transferred to the block of Expert System called “Rules of Risks, Compilation”. The compilation algorithm attempts to create an object for each element using the mentioned special classes. These classes have special “add” methods that checks consistency of the element. For example, Environment Model should not contains two environments with identical names, developers should not make changes in production environment. If adding failed at some stage, the compilation algorithm returns an error message and offers the configuration manager to modify the graphic version, and once again make transformation to XML format and model compilation. At this moment, all previous objects of class models are deleted and after model modification, compilation is started again. In case of successful compilation, Environment Model is stored in computer memory, and this model is ready to be transferred to Expert System (ES) to identify risks or make transformation to Platform Independent Action Model. Environment meta-model elements are described in Fig. 3.

![Fig. 3. Elements of Environment Model.](image-url)

Element “A” in Fig.3. represents developer who makes changes in software.

Element “B” on Fig.3. represents instance for particular process of software development project, for example, DEV, TEST or PROD with set of attributes.

Element “C”, called “ConfigurationItemFlow” on Fig.3. represents the way, on which configuration items come from one environment to another or in case of development from the actor to environment.
Element “D” on Fig.3. represents all flows needed to move changes from one instance to other. One Event could have one or more flows (ConfigurationItemFlow).

Fig. 4. represents simple example to illustrate Environment Model.

The example in Fig. 4. has three environments. The scope of “DEV” environment is development of new changes for software. The Event “dev” simulates development of changes and save them to version control system. Environment called “TEST” should be used for testing new changes. To achieve this, all changes should be moved to “TEST” environment. To ensure that software build that represents new changes is valid, firstly it should be applied in environment called “Pre_TEST”. The event called “test” contains ConfigurationItemFlow with sequence “1” to simulate transfer of changes from “DEV” environment to “Pre_TEST”. If this transfer is valid, the second flow of Event “test” could be executed to apply mentioned changes to “TEST” environment. Circles in Fig. 4. represent Events of the model. The arrows represent flows of Event. The digit on the arrows is the sequence of the current flow in context of related Event.

6 Use Case for New Model-Driven Approach and Environment Model

There is a software development company with some software development projects. Implementation of software configuration management process in any of the mentioned projects should be started with some important decisions.

Firstly, a development team together with a customer should decide on how many environments should support workflow of development. One of the common workflows provides development, testing and quality accepting processes using DEV environment for development, TEST environment for testing and QA for quality accepting by a customer. Changes in software should be transferred from one environment to other, according to the mentioned workflow. It is very important to detect how many environments should be used for supporting the development process, whose branching strategy could be better for the current case, how many general risks exist from the configuration management side. In context of software configuration management, there are two general risks during moving changes from one environment to another. The first risk is related to moving only particular set of changes from one environment to another because mentioned changes could be in conflict with other changes existing in source environment. This kind of risks could be called “Integration risk” because in practice it calls merge conflicts during source code integration from one promotion branch to other. The second risk is related to deployment changes in target environment. For example, if the set of changes should be moved from DEV to TEST environment, TEST environment could be unavailable after failed deployment. To reduce mentioned risks, the new environment should be implemented between DEV and TEST. For example, it could be an environment called Pre_TEST, which should be used for testing builds of particular changes moved from DEV promotion branch to TEST promotion branch. If build is successful, it could be applied in TEST environment. On the other hand, additional environments requires additional resources: servers, administrators, etc. So, compromise should be found between the resources for environments and the importance of general configuration management risks. Environment model with detection of general configuration management risks should help to understand flows of changes between environments much better. During modelling the general view of all environments, configuration managers together with a customer could choose the best compromise between the resources for environments and the risks of configuration management.

Secondly, after preparation of Environment Model, set of technical solutions should be implemented to apply the mentioned model. Usually companies already have solutions (branching strategy, build and deploy scripts, integration with bug tracking system, etc.). The challenge is to organize this solutions by a parameterized and independent way that should increase reuse. For example, build script for JAVA project in ideal case should receive set of parameters and return an
executable file. The script should not know provenance of this parameters and further usability of the executable file, for example, in which servers it should be applied, which repository contains the source code, in which server executable file should be copied for customer use, etc. The new model-driven approach for implementation of configuration management should help organize technical solutions by a reusable way. It will allow using particular solutions (functions, scripts, utilities) in other projects and save up costs of implementation. This scope will be achieved by Platform Independent Action Model as part of a new model-driven approach. This model should detect which tasks should be implemented to apply Environment Model. Other model – Platform Specific Action Model contains all details about solution of particular action. For example, Platform Independent Model contains task “Compile” for compilation of software product. During creation of Platform Specific Action Model, particular solution for mentioned task should be chosen. A company may have more solutions (framework, scripts, functions, tools) to apply “Compile” action. All solutions should be stored in special database, defined in the new model-driven approach. In Platform Specific Model only one solution for “Compile” should be selected from the database. Refactoring of the existing technical solutions (scripts, utilities, functions, frameworks) and creating a new one according to structure provided in the new model-driven approach will make this solutions more parameterized and will increase reuse as a result.

7 Lessons Learned using Environment Model

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 Environment Model.

- Environment Model 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.

8 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-model for Environment Model has been designed as part of the provided approach. Also use case for general approach and Environment Model has been given. During using Environment Model in different software development projects, two important lessons are learned. The conclusions from these lessons could improve environment simulation process and detect further directions of works dedicated to improvement of Environment Model creation.

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

- With the help of experiment, develop criteria that evaluate environment model benefits in software development projects,
- Based on developed criteria, evaluate benefits of Environment Model,
- Develop transformation rules, which from ready environment model can create Platform Independent Action Model,
- Develop criteria to assess whether the developed model-driven approach for configuration management implementation corresponds to guidelines of ISO/IEC 15504, ITIL, CMMI standards.

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 more interesting lessons could be learned from different implementations of this model-driven approach.

Acknowledgments

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: