Revision Control and Automatic Documentation for the Development Numerical Models for Scientific Applications

Martin Zinner*, Karsten Rink†, René Jäkel*, Kim Feldhoff*, Richard Grunzke*,
Thomas Fischer†, Rui Song‡, Marc Walther†§, Thomas Jejkal¶, Olaf Kolditz†‖, Wolfgang E. Nagel*

* Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden
Dresden, Germany
E-mail: {martin.zinner1, rene.jaekel, kim.feldhoff}@tu-dresden.de,
{richard.grunzke, wolfgang.nagel}@tu-dresden.de

† Department of Environmental Informatics
Helmholtz Centre for Environmental Research (UFZ)
Leipzig, Germany
E-mail: {karsten.rink, thomas.fischer, marc.walther, olaf.kolditz}@ufz.de

‡ Technical Information Systems
Technische Universität Dresden
Dresden, Germany
E-mail: rui.song@tu-dresden.de

§ Professorship of Contaminant Hydrology
Technische Universität Dresden
Dresden, Germany
E-mail: marc.walther@tu-dresden.de

¶ Institute for Data Processing and Electronics
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
E-mail: thomas.jejkal@kit.edu

‖ Professorship for Applied Environmental System Analysis
Technische Universität Dresden
Dresden, Germany
E-mail: olaf.kolditz@ufz.de

***Abstract*—As software becomes increasingly complex, automatic documentation of the development is becoming ever more important. In this paper, we present a novel, general strategy to build a revision control system for the development of numerical models for scientific applications. We set up a formal methodology of the strategy and show the consistency, correctness, and usefulness of the presented strategy to automatically generate a documentation for the evolution of the model. As a use case, the proposed system is employed for managing the development of hydrogeological models for simulating environmental phenomena within a research environment.**

Keywords–Software development; Automatic generation of documentation; Revision control; Backup and Restore; Metadata; Improvement of research environment; Support of research process.

I. INTRODUCTION

In scientific applications, dedicated software packages are used to create numerical models for the simulation of physical phenomena [1]. In the scope of this paper we will focus on environmental phenomena, such as the simulation of flooding, groundwater recharge or reactive transport using innovative numerical methods. Such simulations are crucial for solving major challenges in coming years, including the prediction of possible effects of climate change [2] [3], the development of water management schemes for (semi) arid regions [4] [5] or the reduction of groundwater contamination [6] [7].

The modeling process is usually a complete workflow, consisting of a number of recurring steps. To better understand the need for documentation and storing multiple versions of the same model, we would like to roughly outline the process:

1) *Data acquisition:* Relevant data sets required for setting up a model and parameterizing a process simulation are collected. For hydrogeological processes, this includes the digital elevation model (DEM), stratigraphic information from boreholes or other sources, production rates from wells, precipitation rates from climate stations, in- and outflow rates for the region of interest, etc.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

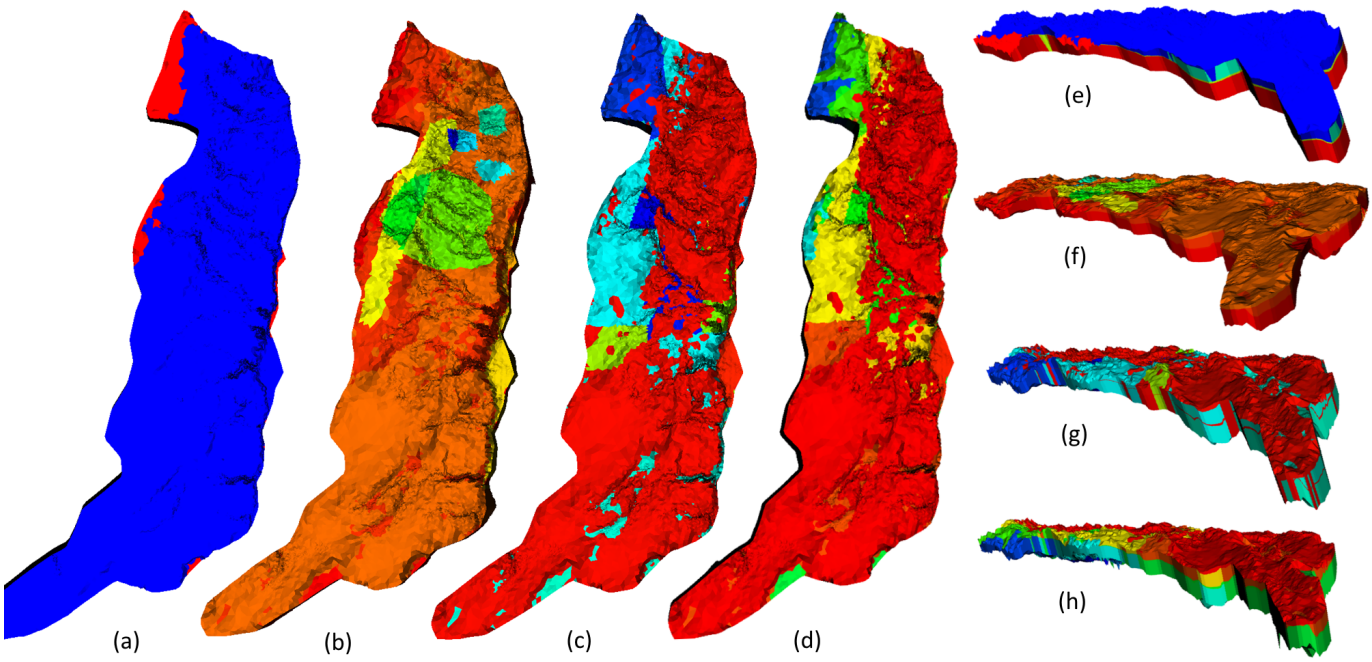

Figure 1. Example for the development of a numerical model over multiple iterations. All figures depict a numerical groundwater model for a region in the Middle East [8]. Figure (a) and (e) show the initial model from January 2011. Figure (b) and (f) depict that state of the model in August 2011 when more information on the surface had been added. Figure (c) and (g) show the complex state at December 2012 when also additional subsurface information had been integrated. Finally, Figure (d) and (h) show the final state of the model in April 2013 after a number of simplifications. Figure (a)–(d) show the top of region of interest, while Figure (e)–(h) show an isometric view such that stratigraphic information is visible.

2) *Data integration:* Information is usually collected from different sources and data sets have been acquired using various measurement devices (e.g., remote sensing data from satellites, sensor data, manually created logs) When aggregating the information, artifacts in data and inconsistencies between data sets need to be removed. This includes fairly simple tasks, such as projecting all data sets into the same geographic coordinate system, but also challenging work, such as dealing with missing or conflicting data [9].
3) *Model Generation:* Information from the input data sets is used to create a numerical model. For instance, DEM and borehole information are used to create a finite element mesh of the subsurface region of interest, source- and sink terms have to be integrated into the mesh, precipitation and outflow information are used to create boundary conditions, and mesh elements need to be parameterized based geohydrological parameters [10].
4) *Process Simulation:* Time stepping scheme, non-linear solver type, pre-conditioner and linear solver for the numerical schemes need to be selected and parameterized. For HPC-Applications, a domain decomposition needs to be performed for the the subsurface mesh. At this point, one or multiple runs of the actual simulation are done [11]
5) *Validation / Visualization:* Simulation results are visualized and checked for plausibility. Results are compared to actual numbers, for example from observation wells, outflow measurements or using other means of validation [12].

In reality, the above workflow will not be executed as linear as suggested above. Often, it is not obvious exactly, which data sets are necessary to run a meaningful simulation. Precise data sets are often hard to come by and need to be requested from state offices or bought from commercial services. The best practice is to set up one or more initial models using data that is available at the time to get a first prototype and see potential problems during simulation or when comparing results to actual validation data. In addition, researchers usually want to create a model that is as complex as necessary but as simple as possible. Complex models tend to be more precise but have more degrees of freedom: boundary conditions and (coupled) processes are hard to parameterize and numerically challenging, the run time is usually (much) longer and problems occurring due to the structure of the model are harder to track down. In contrast, simple models might not be able to represent the region of interest or the simulated process adequately and the correctness or precision of simulation results may be insufficient. Unfortunately, the modeling process itself in general is not transparent and traceable and often poorly documented. A typical model – consisting of a set of parameter files – is developed over many weeks or months (see Figure 1). Usually a large number of revisions are necessary to update and refine the model, such that the simulation represents the natural process as realistically and plausibly as possible. Examples for reasons to adjust the finite element mesh representing a subsurface model domain include changes to element size used to either allow for an adequate representation of the processes of interest (e.g., groundwater flow, heat conduction, dispersion of chemical compounds), integration of additional datasets (e.g., river / stream networks, wells, distribution of soil types) or availability of more precise measurements for data already integrated, re-meshing due to numerically difficult configurations, or

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

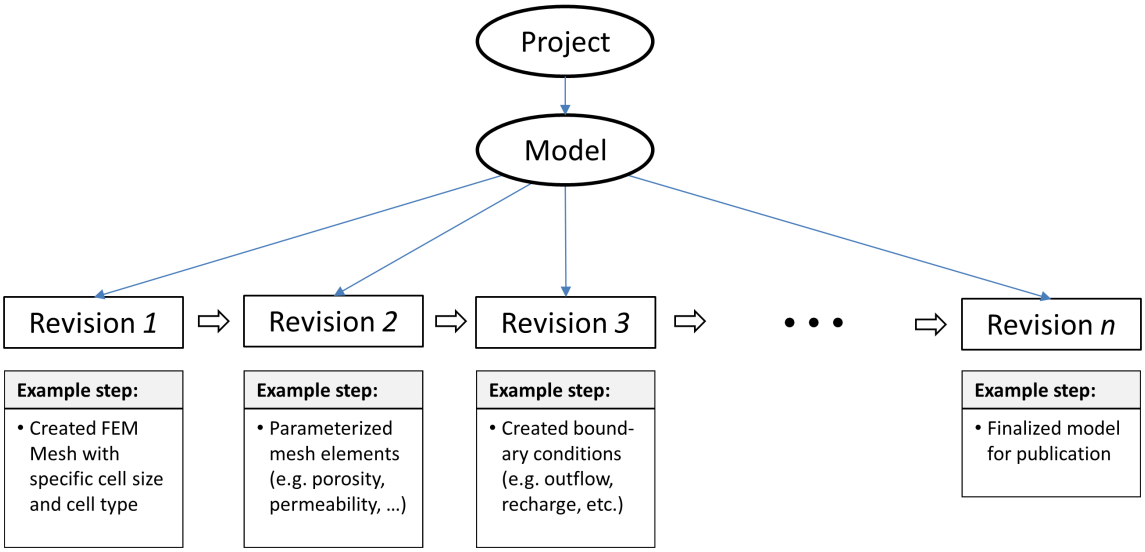

Figure 2. Model development over several revisions

information reduction when the model has become too complex. Other parts of the model, such as boundary conditions or the numerical configuration might also change for different reasons. Especially with multiple researchers working on one model, it is not hard to imagine that it can become difficult to keep track of changes and the reasons why certain aspects of a model have been adjusted; especially when models are developed or maintained over a period of multiple years. It can also become difficult to restore a certain previous state of a model after a series of changes by one or more scientists over a given period of time.

In this paper, we address these particular challenges. Specifically, we present a revision control system, which in addition to the backup/restore functionality tracks the changes in each modeling step, thus generating an internal documentation of the evolution of the model.

A. *Technical Challenges and Objectives*

The basic concept for the simulation of a certain phenomenon in a given region is a model. As shown in the previous section, this model is developed over several iterative steps. In the scope of this paper, we will refer to these steps as "revisions", compare Figure 2. As mentioned, the first setup of the model is often used to get an overview over existing data and to get familiar both with the region of interest as well as the basic behavior of phenomena within that region. At this stage, potential problems, missing data or specific numerical requirements might already become apparent. After creating the first revision (as well as after each subsequent modeling step) a simulation is run, using the model. Depending on the result of the process simulation, further revisions will try to solve these issues by addressing the shortcomings of the simulation result. Typical follow-up steps include adding refined or previously missing data, adjusting or refining the finite element mesh, changing process parameterizations or numerical schemes, etc. (see Figure 3).

The framework for revision control for environmental model development is being implemented at the Helmholtz-Centre for Environmental Research (UFZ) [13] using the Karlsruhe Institute of Technology Data Manager (KIT DM) [14] as a software framework for creating and maintaining repositories for research data.

The Metadata Management for Applied Sciences (MASi) [15] research data management service is currently being prepared for production at the Center for Information Services and High Performance Computing (ZIH) at Technische Universität Dresden. It utilizes the advanced KIT DM framework to provide a service that enables the metadata-driven management of data from arbitrary research communities. This includes automating as many processes as possible including metadata generation and data pre-processing.

The current solution, utilized at UFZ, is completely file based and it is usually stored locally on the laptop of each scientist.

Depending on the complexity of the model and the phenomena that need to be simulated, the number of parameter files varies between three and several hundred, with each file up to several megabytes. The minimum configuration for the OpenGeoSys simulation software [11] requires a finite element mesh, geometric information to specify spatial conditions, as well as a project file containing all process-based information and numerical parameterizations. However, for complex case studies, additional files may become part of the model, for instance to represent boundary conditions. Examples include weather radar data (typically one file per timestep), data from observation wells (typically one time series per well), geometries of changing conditions in the model domain such as advancing/receding coastlines during floods/droughts (one file per timestep). The changes from one revision of the model to the next can be very small, e.g., when one parameter value changed in an input file. However, the changes can also become major, for example when a geometric constraint is updated, which in turn requires re-meshing the model domain and adjusting associated boundary conditions and possibly even the precise type of process that is used because the domain that

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

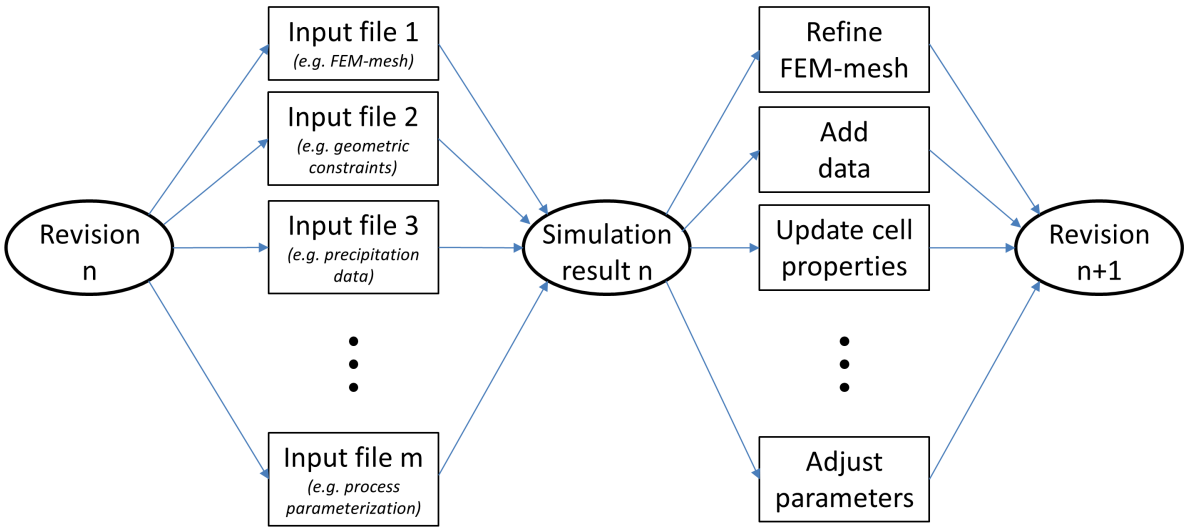

Figure 3. File-based view of revisions and simulation as part of the model development process. Here, a revisions is represented by a collection of input files. Running a simulation will give a set of output files. If simulation results are not satisfactory given existing validation measurements or the researcher's experience, or if the simulation does not converge at all, changes to the model will be made based on the result. Examples include refining the mesh (e.g., if the simulation started to oscillate), adding data (such as a river geometry to use as an additional boundary condition), update cell properties (e.g., adjust permeability of stratigraphic layers so groundwater flow will behave differently) or adjust the parameterization of the numerical process (e.g., choose smaller time steps if the simulation did not converge).

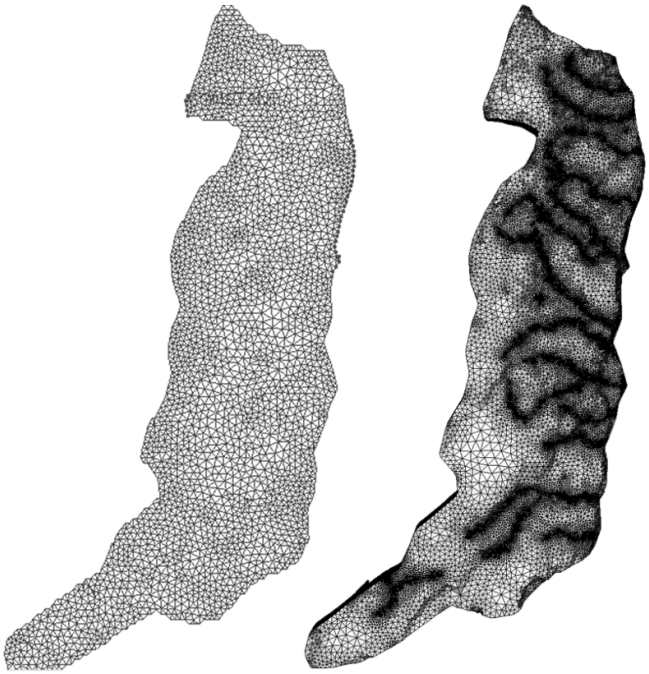

Figure 4. Example for changes of a finite element mesh: The left-hand side depicts the homogenous mesh created for the first iteration of the modeling process, the right hand side the final adaptive mesh, which is refined towards rivers and wells [8].

had previously been considered saturated is now unsaturated. Figure 4 shows an example of changes made to a finite element mesh during the development of a model.

The main deficiencies for researchers working with the current file-based solution are:

1) No overview of the development of the model (especially after handling the model for a long time)
2) Difficulty to trace parameter changes and the reasons for those changes
3) No implicit or explicit documentation of the changes
4) Each user stores the data on his laptop at his own discretion
5) Data is lost if hard disk crashes and there is no backup
6) Joint working on the same model is cumbersome

The benefits of the new framework will include the advantages of a classical revision control system (like Git [16], or Apache Subversion [17]), in particular:

1) Uniform, central, and consistent storage of the individual modeling steps a) each scientist will be able to view the simulation data he is entitled to b) backup functionality if the data is lost,
2) possibility to track and analyze / evaluate the changes,
3) data is still available if the PhD student leaves the company,
4) shared access of the latest development of the model.

A revision is defined as the state of the components already persisted and accessible by a unique identifier. Thus, the content of the components of a revision cannot be altered any more. The current set of the components, which can be actively changed is called the *working set*.

The main objectives we focus on, to achieve our scope, are:

1) Central persistent storage of the model to include all the modeling steps and the management of the revisions.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

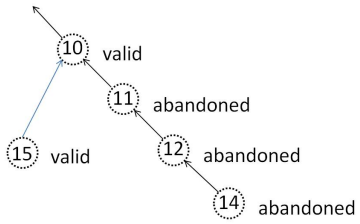

Figure 5. Tree structure of the development of the model

2) Design and development of a metadata repository regarding a) revision control and b) the changes of the parameter files between subsequent revisions. Additionally, information regarding parameter values, simulation software, etc. can be persisted.

3) An efficient and disk space saving strategy, such that a specific parameter file is stored only if its content has been modified.

4) Generation of an internal documentation of the model development, such that it can be easily understood and reconstructed.

It is out of scope of this research to persistently store the results of the simulation. If necessary, it can be generated again or a direct storage strategy could be used. Storing the results of the simulation together with the parameter files would leverage our sophisticated storage strategy, since the size of the parameter files is in the range of megabytes, where the size of the simulation files is in the range of gigabytes. The storage of the simulation results is only meaningful if it takes too long to newly generate the results. The model development is stored in a tree structure, such that each node (revision) has a unique link to its predecessor, see Figure 5. The tree structure is necessary to be able to identify modeling steps where the results of the simulation are not promising, and thus this revision is not pursued further (termed *abandoned)*. In this case, the development of the model is continued from a previous revision (termed *active*), thus performing a rollback on the evolution of the model and creating a new branch in the version control tree structure.

Usually, metadata is defined as data about data. Metadata files can be generated automatically or they can be set up manually. The *flow configuration file* is the main metadata file and it is generated automatically during the evolution process of the model and contains the basic information regarding the revision control system, which is necessary to generate the internal documentation of the evolution of the model, i.e.,

a) model name;

b) predecessors and the current revision;

c) cryptographic hash value and status of the parameter files;

d) parameter change information in condensed form, etc.

The main metadata file sustains the possibility to automatically capture, track, analyze, and evaluate the changes in each modeling step.

Additionally, users can define their own metadata files, which can be created for the whole case study or for a specific revision and could contain additional information regarding a) name of the project; b) model area; c) modeled process; d) software used, including the version; e) contact person; f) source reference regarding the applied methods and data used; g) utilization rights, etc.

Besides documentation, metadata also allows for easy identification of the uploaded data. Search for specific values (e.g., model name, author, etc.) over all metadata elements can be performed for example by using ElasticSearch [18].

B. Outline

The structure of the paper is as follows: In Section II, we give a short overview over the state of art and detail some differences of our approach, both in concept and realization; in Section III, we demonstrate our novel strategy, which is used for the revision control system to generate the implicit documentation of the evolution of the model; in Section IV, we augment the classical pseudo code presentation of the algorithms to a formal, mathematical description of our selective backup strategy and show the consistency and correctness of the backup and restore functionalities. We present the software implementing the formal description and its application to a use case of the UFZ in Section V. Finally, we conclude our work and give an outlook for future research and development in Section VI.

II. Related Work

The concept of revision control systems (RCS) is not new (see Tichy [19]). The task of the RCS as defined by Tichy is version control, i.e., keeping software systems consisting of many versions and configurations well organized. The concept of a revision is similar to our approach, an ancestral tree is used for storing revisions. The major difference is that – as set up by Tichy – each object (like a file) has his own revision tree, whereas we follow an overarching concept, such that files may remain unchanged between revisions. Furthermore, the evolvement of the revision is linear, but it can use *side branches*, for example one for the productive version and one for the development [19].

Löh et al. [20] present a formal model to reason about version control, in particular modeling repositories as a multiset of patches. Patches abstract over the data on which they operate, making the framework equally suited for version control from highly-structured XML to blobs of bits. The mathematical definition of patches and repositories enable Löh et al. to reason about complicated issues, such as conflicts and conflicts resolution. The main application field that Löh et al. targets is the distributed (software) development with its challenges regarding the complex operations on the repositories, such as merging branches or resolving conflicts. They introduce a precise, mathematical description of the version control system to accurately predict when conflicts may arise and how they may be resolved.

Our mathematical model is not based on the work of Löh et al., it has been developed from scratch to enable the characterization of the selective backup strategy.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

The possibility to use metadata, such as the patch's author, time of creation, or some form of documentation is shortly discussed in [20]. Details are left to the designers of a specific revision control system. Also the concept of reverting changes, i.e., the ability to return to a previous version by undoing a modification that later turns out to be undesired, is discussed from a theoretical point of view.

As stated in [21] there are some basic goals of a versioning system, such that:

1) People are able to work simultaneously, not serially.
2) When people are working at the same time, their changes do not conflict with each other.

These two goals do not apply in our case. Formally, users can work simultaneously, making changes independently, but for a simulation they need all the parameter files. The classical use case, such that a programmer changes the internal specification of a module without changing the external interface is not applicable in our case, each change in a parameter file leads to different simulation results. Unfortunately, the usual versioning systems do not support our advanced requirements regarding usage of metadata and enhanced automatic documentation generation of the evolution of the model.

The automatic generation of documentation has also been the scope of intense academic and industrial research. It has been recognized that the importance of good documentation is critical for user acceptance [22]. Jesus describes in [23] a paradigm for automatic documentation generation based on a set of rules that, applied to the models obtained as result of the analysis and design phases, gives an hypertext network describing those models. On the contrary, our approach has the advantage that the algorithm that is used for the selective backup strategy also delivers the data for the automatic documentation. PLANDoc [24] documents the activity – of planning engineers – by taking as input a trace of the engineer's interaction with a network planning tool. Similarly, in [25] Alida, an approach for fully automatic documentation of data analysis procedures, is presented. During analysis, all operations on data are registered. Subsequently, these data are made explicit in XML graph representation, yielding a suitable base for visual and analytic inspection. The high level approach in Alida – using the information generated during the production process to automatically create the documentation – is similar to ours.

As a final note, we looked very carefully at the existing revision control systems, both at the academic and the commercial ones, and found no adequate revision control system, worth to adapt to fulfill our needs towards a system, which automatically supports and tracks the evolution of the model during its development process.

III. Selective Backup Strategy

The aim of our revision control system is to provide an enhanced backup strategy, termed *selective backup strategy,* such that only the components of the working set that have been modified are considered for backup, see Figure 6. This is an enhancement of the usual incremental backup strategies, storing a particular modification of a file only once, in order to provide the framework to generate the metadata regarding the modifications and accordingly to generate the implicit documentation of the model.

A correspondent *selective restore strategy* is used, i.e., the latest versions of all components are downloaded, such that at the end the recent version of the model is assembled out of the historical backups.

A. Backup

Only part of the current working set is uploaded into the data repository, and the uploaded information cannot be altered or removed later. According to our selective backup strategy a) for the first revision: all components are uploaded (see Figure 6 left side); b) for the subsequent revisions: only the components that have been modified are uploaded (see Figure 6 right side).

B. Restore

It will be possible to download all relevant information regarding a specific revision (including parameter files and metadata files). This requires identifying and downloading the full set of components necessary to run a simulation. The required information is stored in the main metadata file during the backup process. Hence, the main metadata file stores information regarding all files that have been uploaded including the unique identification of the uploaded object and the cryptographic hash values of the respective files.

1) Full Restore: The full restore should be applied if files have been lost, or the development of the model is intended to be pursued by other users, etc. The full restore retrieves the whole set of parameter files, such that a simulation can be done on the restored system.

2) Revision Restore: This functionality restores the files corresponding to a (previous) revision and permits to continue the simulation corresponding from that revision. This method enables the tree structure of the revision history. The corresponding information is retrieved from/written to the main metadata file.

C. Flow Configuration

We present now some implementation details. The relevant information for the functioning of the selective backup and the corresponding restore strategy is stored / updated automatically – using XML – in the flow configuration file.

This file stores general information as: a) short name of the model; b) the number of the last revision; c) the object id under which the files belonging to that revision were uploaded; d) additional information in order to identify the project, the revision, etc. A simple example is given in Figure 7. Additionally, general information regarding the revision history such as: a) the revision number; b) the object id of the uploaded object; c) the predecessor (parent revision); the total number of files versus the number of files, which have been changed and in consequence uploaded, etc., as given in Figure 8. File and revision specific information are also tracked, such that for each file the revision where the file has been changed and the corresponding cryptographic values are tracked. This way, it is ensured that a specific state of a file is stored only once and that the revision under which this file has been stored can unambiguously be identified, see Figure 9 for an example.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

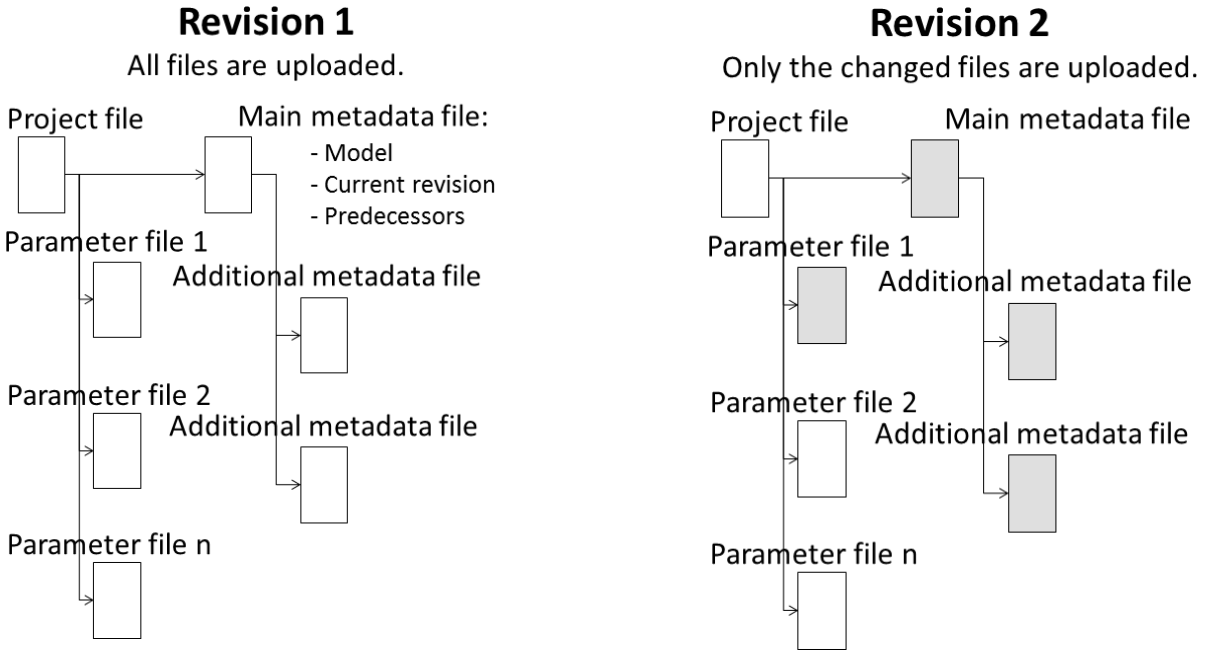

Figure 6. Selective backup strategy. Uploaded files at first (left) and second revision (right).

```
<GeneralConfiguration>
   <ModelShortName>cube_1e0_neumann</ModelShortName>
   <LastRevisionNr>25</LastRevisionNr>
   <LastDigitalObjectID>3ccafed6-cf0d-486d-bbe3-
        edff4159f6c5</LastDigitalObjectID>
   <LastNotes>cube_1e0_neumann / Incr. / It.Nr.: 25 /
        Standard Incremental Upload</LastNotes>
</GeneralConfiguration>
```

Figure 7. Excerpt 1 of example configuration file

```
<Revision>
<RevisionNr>7</RevisionNr>
<DigitalObjectID>8ce20b42-c15f-427c-ac1a-d575295f5412</
    DigitalObjectID>
<ParentRevisionNr>3</ParentRevisionNr>
<TotalNrFiles>20</TotalNrFiles>
<NrFilesUploaded>1</NrFilesUploaded>
<Notes>cube_1e0_neumann / Incr. / It.Nr.: 2 / For Testing
    Upload and Download</Notes>
</Revision>
```

Figure 8. Excerpt 2 of example configuration file

```
<FileCharacteristics>
   <FileName>cube_1x1x1.gml</FileName>
   <FileLastRevision>43</FileLastRevision>
   <FileHistory>
      <FileRevision>
         <StorageRevisionNr>1</StorageRevisionNr>
         <StorageDigitalObjectID>8aac5970-a366-49ce-a052
              -6c8f82ced85c</Storage\-Digital\-ObjectID>
         <FileSize>1622</FileSize>
         <FileCreationTime>2016-08-18T08:25:33Z</
              FileCreationTime>
         <FileLastAccessTime>2016-08-23T15:59:55Z</
              FileLastAccessTime>
         <FileLastModifiedTime>2016-08-18T08:25:33Z</
              FileLastModifiedTime>
         <FileCryptoIDMD5>66
              a9800e8b02d85001cdd13930b85ea3</
              FileCryptoIDMD5>
         <FileCryptoIDSHA1>5
              c942ba146b41e38262f23ed350e214c757d8803</
              FileCrypto\-IDSHA1>
      </FileRevision>
```

Figure 9. Excerpt 3 of example configuration file

D. Accurate versioning

Based on the architecture of the system, the selective backup strategy corresponds to a centralized revision control system, i.e., there is a central revision number – in our case the modeling step –, such that the version of each file is tied to this central revision number. In contrast, revision control systems like Git [16] are decentralized, i.e., generally, users maintain the versioning of their part, without affecting the overall release number. In our case, this centralized approach is of crucial importance, since small changes in one parameter file can substantially affect the outcome of the simulation. The selective backup strategy enables a paradigm change in the theory and practice of (centralized) revision control systems, it enables an accurate tracking of the changes during each revision on file level including the identification of the effective version of each file. This means especially that in contrast to Git and SVN [17], during revision change, each file is compared to previous versions and either assigned a new version number or – if possible – reassigned a previous one. Such a distinction is not absolutely necessary, for example during software development (main application field for Git and SVN), but it is of crucial importance for pursuing the exact model development.

We illustrate now the selective backup strategy by means of the example as delineated in Table I. Let $\{F_1, F_2, F_3, \ldots, F_n\}$ be files comprising a numerical model and $\{R_1, R_2, R_3, \ldots, R_m\}$ the revisions to adjust the model for a successful simulation of a process. According to the architecture we use, the number of revisions is greater than the number of the files involved, i.e.,

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

$n < m$. We number the versions of a specific file continuously in the order they were generated, starting with 1. We notate by an upper index the revision during which the version was generated, i.e., for the file F_1 the version $V_4^{R_6}$ represents the fourth version generated during revision R_6. Not all files are necessarily updated with a revision. For instance, file F_1 is unchanged during revisions R_4, R_5, thus the model keeps using the file version $V_3^{R_3}$. In contrast, file F_2 is modified in each revision. However, during revision R_5, the file is reverted to a previous state such that its version is equal to $V_2^{R_2}$ and, instead of storing a duplicate, the previous copy of the file is used.

Using revision systems like Git or SVN, a new version of the file F_2 would be created. Instead, our algorithm, using the selective backup strategy, verifies if a version with new content has been created or the respective content has already been used before.

The use of the selective backup strategy is not restricted to the model development for scientific application, but can be applied everywhere where a centralized revision control system is used. Its intended target are applications, which need to track the effective version of the files, potentially related to revision numbers.

IV. The Formal Model

We introduce a mathematical model in order to use the advantages of the rigor of a formal approach over the inaccuracy and the incompleteness of natural languages. We augment the classical pseudo code presentation of the algorithms to a formal, mathematical description and show the consistency and correctness of the backup and restore functionalities.

A. Notations

We use a calligraphic font to denote the index sets. We denote by $\mathscr{C} := \{C_i \mid i \in \mathcal{C} \text{ and } C_i \text{ is a component}\}$ the finite set of the components, i.e., the disjunct union of the parameter files and the metadata files. Let S be an arbitrary set. We notate by $\mathscr{P}(S)$ the power set of S, i.e., the set of all subsets of S, including the empty set and S itself. By $card(S)$ we notate the cardinality of S. Let $n \in \mathbb{N}$ and let $f : X \to X$ be a function. Finally, we denote by $f^n : X \to X$ the function obtained by composing f with itself n times, i.e., $f^0 := id_X$ and $f^{n+1} := f^n \circ f$.

B. Introducing Components and Revisions

Some components – at least one, but not necessary all – are modified, then a simulation is performed. We call this state of the components a *revision*. Each revision is backed up to a persistent storage. We have in a natural way a total ordering $<$ on the set of the revisions considering the order they were generated. We denote by $\mathscr{R}$ the ordered set of the revisions, i.e., $\mathscr{R} := \{R_i \mid i \in \mathcal{R} \text{ and } R_i \text{ is a revision}\}$. Let $m := card(\mathscr{R})$ be the number of revisions. In order to keep the notations straightforward we set $\mathcal{R} := \{1,2,3,...,m\}$, such that $\forall i \in \mathcal{R} \setminus \{m\} : R_i < R_{i+1}$. We denote by $C_k^{(i)}$ the component C_k having the state at revision R_i, therefore, we denote by $\mathfrak{R}_i := \{C_k^{(i)} \mid k \in \mathcal{C}\}$ the set of the components having the state corresponding to revision R_i.

We denote by $\mathfrak{R}$ the matrix of the evolution of the model, hence $\mathfrak{R} := \{C_k^{(i)} \mid k \in \mathcal{C} \text{ and } i \in \mathcal{R}\}$. Therefore, $\mathfrak{R}$ contains the history of the content changes of the components during the evolution process of the model.

Let *HASH* be the set of all the hash values. We define the content of a component $C_k \in \mathscr{C}$ corresponding to a revision R_i formally as the function:

Definition IV.1 (Content of components) *We set*

$$CONT\colon \mathfrak{R} \to HASH,$$
$$C_k^{(i)} \mapsto CONT(C_k^{(i)}) := \textit{hash value of } C_k^{(i)}.$$

Remark IV.1 *Let $k \in \mathcal{C}$, let $i, j \in \mathcal{R}$, such that $i \neq j$. In order to track the change process during the evolution process of the model, we are interested only in comparing the content of the same component at different revisions (i.e., the content of $C_k^{(i)}$ versus the content of $C_k^{(j)}$).* ■

Definition IV.2 (Origin of a revision) *Let $R, Q \in \mathscr{R}$. We say that Q is the origin of R, notated by $Q = ORIGIN(R)$, if and only if the revision R has been obtained by direct modification of the content of the components having the state at revision Q. For formal reasons we define $ORIGIN(R_1) := R_1$.*

Remark IV.2 *Let $R \in \mathscr{R}$ arbitrarily chosen. Then there exists a unique $Q \in \mathscr{R}$, such that $Q = ORIGIN(R)$. This is a direct consequence of the definition above (see Definition IV.2).* ■

Let $m = card(\mathscr{R})$, such that $m \geq 1$. We define the predecessor and the successor of a revision formally as:

Definition IV.3 (Predecessor of a revision) *We set*

$$PRED\colon \mathscr{R} \to \mathscr{R},$$
$$R \mapsto PRED(R) := ORIGIN(R).$$

Definition IV.4 (Successor of a revision) *We set*

$$SUCC\colon \mathscr{R} \to \mathscr{P}(\mathscr{R}),$$
$$R \mapsto SUCC(R) := \{Q \in \mathscr{R} \mid R = ORIGIN(Q)\}.$$

Remark IV.3 *$\forall R \in \mathscr{R} \Rightarrow PRED(R)$ is unequivocally determined (see Remark IV.2), in contrast, there exists to a revision $R \in \mathscr{R}$ a subset J of $\mathcal{R}$, such that $SUCC(R) = \{R_j \mid j \in J\}$.* ■

Unfortunately, the structure of the evolution of the model is not linear. If for any reason the evolution of the model is in impasse, then the development of the model is not continued from the latest revision, but a previous revision is taken as a starting point. The revision, which led to the impasse is not pursued any more (i.e., it is *abandoned*). On the other side, a revision is *active* if it is part of the successful completion of the model. Formally, we define the status of a revision as follows:

Definition IV.5 (Status of a revision) *Let $m := card(\mathscr{R})$ the number of revisions. We set*

$$STATUS\colon \mathscr{R} \to \{active, abandoned\},$$
$$R \mapsto STATUS(R) := \begin{cases} active & \text{if } \exists n \geq 0 : \\ & R = PRED^n(R_m), \\ abandoned & \text{otherwise.} \end{cases}$$

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Table I. Example for the selective backup strategy.

	R_1	R_2	R_3	R_4	R_5	R_6	...	R_m
F_1	$V_1^{R_1}$	$V_2^{R_2}$	$V_3^{R_3}$	$V_3^{R_3}$	$V_3^{R_3}$	$V_4^{R_6}$	...	$V_{m_1}^{R_{m_{k1}}}$
F_2	$V_1^{R_1}$	$V_2^{R_2}$	$V_3^{R_3}$	$V_4^{R_4}$	$V_2^{R_2}$	$V_5^{R_6}$	...	$V_{m_2}^{R_{m_{k2}}}$
F_3	$V_1^{R_1}$	$V_1^{R_1}$	$V_2^{R_3}$	$V_3^{R_3}$	$V_1^{R_1}$	$V_4^{R_6}$	...	$V_{m_3}^{R_{m_{k3}}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
F_n	$V_1^{R_1}$	$V_1^{R_1}$	$V_1^{R_1}$	$V_1^{R_1}$	$V_2^{R_5}$	$V_2^{R_5}$	...	$V_{m_n}^{R_{m_{kn}}}$

Informally, the predecessor of a component $C_k^{(i)}$ is the component $C_k^{(j)}$, such that R_j was the latest revision where the component C_k has been changed. Formally, we model the successor and predecessor of a component C_k during the revision process as a function.

Let $i \in \mathcal{R}$ and $k \in \mathcal{C}$ arbitrarily chosen. Set $A(i,k) := max\{l \in \mathcal{R} : l < i \text{ and } CONT(C_k^{(i)}) \neq CONT(C_k^{(l)})\}$ then

Definition IV.6 (Predecessor of a component) *We set*

$$PRED: \mathfrak{R} \to \mathfrak{R},$$

$$C_k^{(i)} \mapsto PRED(C_k^{(i)}) := \begin{cases} C_k^{(i)} & \text{if } (i=1), \\ C_k^{(A(i,k))} & \text{if } (i>1) \\ & \quad \text{and } A(i,k) \text{ exists}, \\ C_k^{(1)} & \text{otherwise}. \end{cases}$$

Definition IV.7 (Successor of a component) *We set*

$$SUCC: \mathfrak{R} \to \mathscr{P}(\mathfrak{R}),$$
$$C_k^{(i)} \mapsto SUCC(C_k^{(i)}) := \{C_k^{(j)} \in \mathfrak{R} \mid C_k^{(i)} = PRED(C_k^{(j)})\}.$$

Remark IV.4 *The predecessor of $C_k^{(i)}$ is uniquely determined. This follows directly from the definition above. In contrast, the successor of $C_k^{(i)}$ is not necessary unique, but there exists a unique $R_j \in \mathscr{R}$, such that $C_k^{(j)} \in SUCC(R_k^{(i)})$ and $STATUS(R_j)$ is active. Similar considerations also hold for revisions.* ■

Proposition IV.1 (Existence and uniqueness) *Let $R \in \mathscr{R}$, such that $STATUS(R) = active$. If $SUCC(R) \neq \varnothing$ then there exists a unique $Q \in \mathscr{R}$, such that $Q \in SUCC(R)$ and $STATUS(Q) = active$.*

Hint *The existence and the uniqueness follows directly from the definition of the status of a revision (see Definition IV.5) and the uniqueness of the predecessor (see Remark IV.3).* ■

We are now able to formulate our strategy to generate the successive revisions.

Lemma IV.1 (Linearity) *Let $m = card(\mathscr{R})$. Then there exists a unique subset $\mathscr{R}'$ of $\mathscr{R}$ with $\mathscr{R}' = \{R_1, R_{i_1}, R_{i_2}, R_{i_3}, ..., R_{i_l}, R_m\}$, such that $R_{i_1} \in SUCC(R_1)$ and $\forall i_k : i_1 \leq i_k < i_l \Rightarrow R_{i_{(k+1)}} \in SUCC(R_{i_k})$ and $R_m \in SUCC(R_{i_l})$ and $\forall R \in \mathscr{R}' : STATUS(R) = active$ and $\forall R \in \mathscr{R} \setminus \mathscr{R}' : STATUS(R) = abandoned$.*

Hint *It is a direct consequence of the uniqueness of active successors (see Proposition IV.1).* ■

Corollary IV.1 *The sequence of the active revisions is linear.* ■

Let $i \in \mathcal{R}$ arbitrarily chosen, a component can have at the revision R_i two statuses *modified* and *preserved*, the value *modified* means that the component has been modified during the revision R_i, in contrast *preserved* means that the component remained unchanged at revision R_i. More formally, we define the function:

Definition IV.8 (Status of a component) *We set*

$$STATUS : \mathfrak{R} \to \{modified,\ preserved\},$$

$$C_k^{(i)} \mapsto STATUS(C_k^{(i)}) := \begin{cases} modified & \text{if } (i=1), \\ modified & \text{if condition 2 holds,} \\ preserved & \text{otherwise}. \end{cases}$$

with condition 2: $\forall j \in \mathcal{R}$ with $1 \leq j < i : CONT(C_k^{(i)}) \neq CONT(C_k^{(j)})$.

Remark IV.5 *From a formal point of view, all components corresponding to the first revision are considered modified. For the subsequent revisions only the components whose content has been altered are considered modified.* ■

C. Backing up a revision

Based on the values of the *STATUS* function, we define the upload strategy. We are interested to upload only those components, which have been modified since the latest revision and the current state has not been uploaded previously.

Definition IV.9 (Upload of a component) *We set*

$$UPLOAD: \mathfrak{R} \to \{yes,\ no\},$$

$$C_k^{(i)} \mapsto UPLOAD(C_k^{(i)}) := \begin{cases} yes \text{ if condition 1 holds,} \\ no \text{ otherwise}. \end{cases}$$

with condition 1: $STATUS(C_k^{(i)}) = modified$.

Remark IV.6 *This means especially that $C_k^{(i)}$ will be uploaded if and only if it has been modified and its content is different from the content of all its predecessors.* ■

We will define now a function in order to model the backup process of an entire revision. As we will see, only those components will be backed up at a specific revision R_i, which have been modified at this revision.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Definition IV.10 (Backup of a revision) *We set*

$$BACKUP\colon \mathscr{R} \to \mathfrak{R},$$
$$R_i \mapsto BACKUP(R_i) := \{C_k^{(i)} \mid k \in \mathcal{C}, \text{ such that } UPLOAD(C_k^{(i)}) = yes\}.$$

Remark IV.7 *This means especially that the components that have not been changed at revision R_i are not included in the backup of the revision R_i, this is the quintessence of the selective backup strategy.* ■

In order to be able to model the download and restore process, we need to do some additional analysis. Those opposite functions cannot be defined straightforwardly as the reverse function of *BACKUP* and *UPLOAD*, since after the restore is fulfilled, all the relevant components must be available, not only those persisted at the corresponding revision.

In order to have all the relevant information for the restore process, we build during the evolution of the model a matrix $(INF_k^{(i)})_{k\in\mathcal{C},i\in\mathcal{R}}$, such that this matrix contains the information relevant for the download and restore operations. This information contains the content of the components, such that comparisons can be done and relate it to the previous backups.

Hence, the matrix $(INF_k^{(i)})_{k\in\mathcal{C},i\in\mathcal{R}}$ contains at least the information regarding the revision at which the component was physically stored, such that it can be retrieved from there and additional information regarding the content (hash value) of the components.

Formally, we define $(INF_k^{(i)})_{k\in\mathcal{C},i\in\mathcal{R}}$ as a function:

Definition IV.11 (Component upload meta inf) *We set*

$$INF\colon \mathcal{R}\times\mathcal{C} \to \mathcal{R}\times HASH,$$
$$(i,k) \mapsto INF(i,k) := (j,v) \text{ if } \left(C_k^{(i)} \in BACKUP(R_j) \text{ and } CONT(C_k^{(i)}) = v\right).$$

Remark IV.8 *This means especially that a component $C_k^{(i)}$ having the hash value = v has been backed up at revision R_j and $j \in \mathcal{R}$ is the lowest index number, such that $CONT(C_k^{(i)}) = CONT(C_k^{(j)})$.* ■

D. Restoring a revision

We define now the opposite function to *UPLOAD* as follows:

Definition IV.12 (Download of a component) *We set*

$$DOWNLOAD\colon \mathfrak{R} \to \mathfrak{R},$$
$$C_k^{(i)} \mapsto DOWNLOAD(C_k^{(i)}) := C_k^{(j)} \text{ if } (UPLOAD(C_k^{(j)}) = yes \text{ and } CONT(C_k^{(j)}) = CONT(C_k^{(i)})).$$

Remark IV.9 *$DOWNLOAD(C_k^{(i)}) = C_k^{(j)}$ means especially that the component C_k was uploaded at the revision R_j.* ■

We define the restore function having the opposite functionality to the backup function. The main difference to the usual restore strategy is that restoring the components backed up at revision R_i is not enough, since usually only a subset of the components are backed up at a specific revision. To circumvent this impediment, the revisions at which those components have been physically uploaded are identified and are restored from those locations. When the restore operation is completed, then, the complete set of components necessary for a simulation is available.

Formally as a function:

Definition IV.13 (Restore of a revision) *We set*

$$RESTORE\colon \mathscr{R} \to \mathscr{P}(\mathfrak{R}),$$
$$R_i \mapsto RESTORE(R_i) := \{C_k^{(j)} \mid k \in \mathcal{C}, \text{ such that } DOWNLOAD(C_k^{(i)}) = C_k^{(j)}\}.$$

Remark IV.10 *This means especially that the latest version of the components are restored.* ■

We are now able to formulate the Lemma regarding the uniqueness of the upload, i.e., a new state of a component C_k during the revision process is backed up only once.

Lemma IV.2 (Uniqueness of the upload) *Let $k \in \mathcal{C}$ and $v \in HASH$ be arbitrarily chosen. If $\exists j \in \mathcal{R} : CONT(C_k^{(j)}) = v$ then there exists a unique $i \in \mathcal{R}$, such that $UPLOAD(C_k^{(i)}) = yes$ and $CONT(C_k^{(i)}) = v$.*

Hint *Set $i := min\{l \in \mathcal{R} \mid CONT(C_k^{(l)}) = v\}$. Then according to the definition of the status of a component (see Definition IV.8) $STATUS(C_k^{(l)}) = modified$ holds true. The result follows from the definition of the upload of the components (see Definition IV.9).* ■

We can formulate now the main Lemma, which states that we have no spurious downloads.

Lemma IV.3 (Accuracy and completeness) *Let $k \in \mathcal{C}$ be arbitrarily chosen. We have:*

$$a)\forall i,j \in \mathcal{R} : DOWNLOAD(C_k^{(i)}) = C_k^{(j)} \Rightarrow \left(UPLOAD(C_k^{(j)}) = yes \text{ and } CONT(C_k^{(j)}) = CONT(C_k^{(i)})\right),$$
$$b)\forall i \in \mathcal{R}\ \exists j \in \mathcal{R} : (j \le i), \text{ such that } C_k^{(j)} \in DOWNLOAD(C_k^{(i)}).$$

Remark IV.11 *The property in a) means that we have no false downloads, i.e., each downloaded component has been uploaded some time ago. It follows from the definition of the download of a component (see Definition IV.12). The property in b) means especially that the download is complete, i.e., the latest version of each component is downloaded. It is a direct consequence of the uniqueness of the upload (see Lemma IV.2).* ■

Corollary IV.2 (Complementarity) *The two functions RESTORE and BACKUP are complementary, i.e.,*

$$\forall k \in \mathcal{C}, \forall i \in \mathcal{R} : \left(C_k^{(i)} \in RESTORE(R_i)\right) \Leftrightarrow \left(\exists j \in \mathcal{R} : C_k^{(j)} \in BACKUP(R_j) \text{ and } CONT(C_k^{(i)}) = CONT(C_k^{(j)})\right).$$

■

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

```
<OpenGeoSysProject>
   <mesh>cube_1x1x1_hex_1e0.vtu</mesh>
   <geometry>cube_1x1x1.gml</geometry>

   <processes>
      <process>
         <name>GW23</name>
         <type>GROUNDWATER_FLOW</type>
         <process_variable>pressure</process_variable>
         <hydraulic_conductivity>K</hydraulic_conductivity
             >
         <linear_solver>
            <lis>-i cg -p jacobi -tol 1e-16 -maxiter 10000
                </lis>
            <eigen>
               <solver_type>CG</solver_type>
               <precon_type>jacobi</precon_type>
               <max_iteration_step>10000</
                   max_iteration_step>
               <error_tolerance>1e-16</error_tolerance>
```

Figure 10. Excerpt of example project file `cube_1e0_neumann.prj`

```
    <points>
        <point id="0" x="0" y="0" z="0"/>
        <point id="1" x="0" y="0" z="1"/>
        <point id="2" x="0" y="1" z="1"/>
        <point id="3" x="0" y="1" z="0"/>
    </points>

    <surfaces>
        <surface id="0" name="left">
            <element p1="0" p2="1" p3="2"/>
            <element p1="0" p2="3" p3="2"/>
        </surface>
        <surface id="1" name="right">
            <element p1="4" p2="6" p3="5"/>
            <element p1="4" p2="6" p3="7"/>
        </surface>
```

Figure 11. Excerpt of example geometry file `cube_1x1x1.gml`

V. Implementation and Application to Specific Use Case

We developed and tested AGEDRE (**A**utomatic **GE**neration of **D**ocumentation using **RE**vision control) as a prototype at UFZ and validated our theoretical concepts. We used a client / server environment at the ZIH of the Technische Universität Dresden, implementing our client in Java from scratch. On the server side, we used KIT DM [14] as the repository.

AGEDRE is a command line utility, offering the basic functionality required for a revision control system and a sophisticated error handling to deal with the complexity of the selective backup strategy on the client side and of KIT DM on the server side. In order to persist the data, AGEDRE offers two primitives, *FullUpload* as a primitive for a complete upload of all data related to a project and *Upload* as a selective backup strategy to store only modified files. Accordingly, the opposite primitives to retrieve the data are *Download*, *DownloadRevision* and *DownloadFile*. The primitive *Download* is only formally the counterpart of *Upload*, it retrieves the latest version of each file, which has been uploaded, i.e., the files of the latest revision. As mentioned, the user needs the latest version of all parameter and metadata files in order to be able to perform the simulation. In contrast, the primitive *DownloadRevision* is used to continue the modeling process from an older revision, it restores the files into the working directory, thus overwriting the latest revision. The latest revision is backed up to the file system, in order to avoid loss of data in case of inadvertent use of this primitive. The primitive *DownloadFile* has been introduced in order to restore the latest backed up version of a file, if it has been accidentally deleted or has been corrupted.

The project file (see Figure 10 for an example) is the leading file regarding the configuration of a simulation. It contains the names of the additional parameter files (namely `cube_1x1x1_hex_1e0.vtu` and `cube_1x1x1.gml`) and the configuration parameters for the simulation. Hence, each project file corresponds to a model and accordingly, the development of the model comprises modifications of the project file and the corresponding parameter files.

When the primitive *Upload* is called for a project file for the first time, the corresponding dynamic flow configuration file is initialized. For an example of a flow configuration file, see the excerpts in Figures 7–9. The revision number is set to one and the cryptographic MD5 and SHA-1 hashes of each file are calculated and stored in the dynamic flow configuration file. While SHA-1 is practically collision free and it is also used by Git for integrity purposes [26], alternatively, SHA-512 could be used for enhanced security [27] [28]. For subsequent uses of the *Upload*-primitive, the cryptographic values of the parameter and metadata files are compared to the respective values stored – for previous revisions – in the flow configuration file. If the cryptographic values of file F is different of all the previous cryptographical values of file F, then the content of F is considered modified and it is backed up within the current revision. Otherwise, F is not part of the current revision. A corresponding entry is made in the dynamic flow configuration file regarding the revision under which the file – having the given content – has been backed up. Hence, the file `cube_1x1x1.gml` has the cryptographic values stored at revision 1 (see Figure 9). If the file `cube_1x1x1.gml` has, for example, not been altered at revision 2 then no similar entry is performed in the dynamic flow configuration file.

When starting the primitive *Download* – i.e., downloading the files corresponding to the latest revision – then the relevant information regarding the physical storage place of the latest version of each file – i.e., <StorageDigitalObjectID> – is retrieved from the dynamic flow configuration file, by considering the latest entry for each file (see Figure 9). Hence, all the related files can be accessed on the repository and retrieved accordingly.

The use case at UFZ is not designed for concurrent use. In a software development environment, users of version control systems are confronted with merge conflicts and their resolution. In contrast, simultaneous work is in the scope of our application strictly related to the model development process. The model is developed iteratively, small changes in a few parameter files can have tremendous impact on the model. Hence, members of a team can download the latest revision and can work simultaneously improving the next step. They can compare the changes of parameter files and the simulation results. Conflicts can theoretically occur but are much more unlikely due to a smaller number of users editing files and each user usually having a specific task to solve. Also, a (semi-)automatic handling of conflicts is near impossible in this scenario since

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

fixing conflicts requires a contextual understanding of what a specific change means in the context of a given model. Members of the team have to agree on the best outcome for the next step before uploading it as the next revision. Alternatively, they can agree on abandoning the current revision by continuing the model development from an older revision. All team members have to download the revision they agreed upon, and continue development from that point.

In addition to the dynamic flow configuration file – upon whose content they do not have direct influence – users can define and set up their own metadata files. These metadata files can contain additional – high-level or aggregated – information regarding the model development and can be used for additional documentation or for identifying model or revision characteristics. The metadata files are also very important to enable the differentiated security policy at UFZ, such that users can access the metadata files – for example by using ElasticSearch [18] – if they have the appropriate rights on the file system. In contrast, users can access simulation data (parameter files) according to their rights on the repository system KIT DM. Thus, metadata for projects is accessible for all members within the UFZ, while sensible data can only be accessed by a small number of researchers related to the project. All files of the examples can be found at [29].

VI. Conclusion and Future Work

Irrespective of the fact that creating documentation is a very challenging task and that writing documentation is considered by most of the developers as an extra-effort rather than a commendation, it is rather impossible providing precise, exact, accurate, uniform and consistent documentation for developers and users requirements. Therefore, formalized automated documentation methods are necessary to develop.

The main advantage of the automatic generation of the documentation of the modeling process is the accuracy of the documentation, since there is no discrepancy between the actual generation of the model (developer's perspective) and the corresponding documentation (user's perspective). This way, we have circumvented the dilemma of writing exact manual documentation and have contributed to the paradigm change towards design and implementation of automatic documentation assuring accuracy and exactness.

Since our formal model is independent of the use case at UFZ, our approach to automatically generate a documentation for the evolution of the model during model development has a generic character and it can be applied to all domains where numerical models are developed. Moreover, the formal model is generalizable beyond the use case presented in this paper.

We have based our solution at UFZ exclusively on common techniques, which are not dependent on specific file formats or specific applications. Examples include the utilization of the XML format for the documentation, hash code based file comparison techniques as well as methods which are independent of the syntax or semantic of the underlying data. With one exception, i.e., the specification of the list of file types which should be monitored, we use only assumptions specific to the open source project OpenGeoSys [11], to applications in the earth modeling domain or to numeric simulations. The system could version and document images, texts, or any other data in the same way.

We have opted for a revision control system using the KIT DM framework based on MASi among others, to have a completely decoupled access to the information surrounding the model development, i.e., metadata easily accessible from various locations and the model development itself, accessible only to the members of the developing team. Alternatively, using modern methods for concurrency control applied to modern database systems is a viable and future-oriented approach. This way, the difficulties on joint and concurrent development could be diminished.

To summarize, we have described in detail to what extent our versioning system facilitates simplifies and makes fully transparent the activity of application scientist. We have presented the main challenges in Section I-A, then the challenges are substantiated and concrete solutions are provided to them. Moreover, in Section IV we have set up a formal model in which the given solution is validated. This gives us hints regarding the generalization spectrum beyond the use case at UFZ.

Currently, there are no convenience features for users employing the framework. In the current implementation prototype, the executable is started in the command line, also the flow configuration file, which contains the documentation for tracking the evolution of the model is in XML format. Accordingly, additional user tests are necessary to define and implement a corresponding GUI to assure the expected readability of the document by extracting and visualizing the appropriate information. In order to build meaningful user interfaces, an intense dialog between developers and users is essential [30].

Additionally, further research is necessary to generate a high level form documentation of the changes in the parameter files. For example, when running stochastic simulations (e.g., using the Monte Carlo approach [31]) and parameters are simultaneously changed in many places, then an appropriate mechanism should be set up to assure the consistency of the changes and the creation of correct documentation.

We believe that by studying the automatically generated documentation regarding the development of the modeling workflows – especially those steps, which did not lead to a successful completion of the simulation – there is an increased possibility of knowledge extraction (by using machine learning strategies or similar techniques), such that the generation of the modeling workflows can be dramatically improved and the number of modeling steps can be considerably reduced.

Acknowledgment

This work was supported in parts by the German Federal Ministry of Education and Research (BMBF, 01IS14014A-D) by funding the competence center for Big Data "ScaDS Dresden/Leipzig". This work was also supported in parts by the German Research Foundation (DFG) via the MASi project (NA 711/9-1, STO 397/4-1). We are also thankful to Dr. Nico Hoffmann (Technische Universität Dresden) for his valuable advices and comments during the developing and writing process and Dr. Agnes Sachse (né Gräbe) for her data on her hydrogeological case study.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

REFERENCES

[1] M. Zinner et al., "Automatic documentation of the development of numerical models for scientific applications using specific revision control," in ICSEA 2017, The Twelfth International Conference on Software Engineering Advances, L. Lavazza, R. Oberhauser, R. Koci, and S. Clyde, Eds., Oct. 2017, pp. 18–27, IARIA Conference. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=icsea_2017_1_30_10110

[2] Intergovernmental Panel on Climate Change, Climate Change 2014 – Impacts, Adaptation and Vulnerability: Regional Aspects. Cambridge University Press, 2014.

[3] C. J. Vörösmarty et al., "Global threats to human water security and river biodiversity," Nature, vol. 467, 2010, pp. 555–561.

[4] J. Grundmann, N. Schütze, G. H. Schmitz, and S. Al-Shaqsi, "Towards an integrated arid zone water management using simulation-based optimisation," Environ Earth Sci, vol. 65, no. 5, 2012, pp. 1381–1394.

[5] H. Hötzl, P. Möller, and E. Rosenthal, The Water of the Jordan Valley. Springer, 2009.

[6] M. Walther, J.-O. Delfs, J. Grundmann, O. Kolditz, and R. Liedl, "Saltwater intrusion modeling: Verification and application to an agricultural coastal arid region in Oman," Journal of Computational and Applied Mathematics, vol. 236, no. 18, 2012, pp. 4798–4809, fEMTEC 2011: 3rd International Conference on Computational Methods in Engineering and Science, May 9–13, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042712000659

[7] C. Liu, Q. Wang, C. Zou, Y. Hayashi, and T. Yasunari, "Recent trends in nitrogen flows with urbanization in the shanghai megacity and the effects on the water environment," Environmental Science and Pollution Research, vol. 22, no. 5, Mar 2015, pp. 3431–3440. [Online]. Available: https://doi.org/10.1007/s11356-014-3825-4

[8] A. Gräbe et al., "Numerical analysis of the groundwater regime in the western Dead Sea Escarpment, Israel + West Bank," Environ Earth Sci, vol. 69, no. 2, 2013, pp. 571–585.

[9] K. Rink, L. Bilke, and O. Kolditz, "Visualisation Strategies for Environmental Modelling Data," Env Earth Sci, vol. 72, no. 10, 2014, pp. 3857–3868.

[10] T. Fischer, D. Naumov, S. Sattler, O. Kolditz, and M. Walther, "GO2OGS 1.0: a versatile workflow to integrate complex geological information with fault data into numerical simulation models," Geoscientific Model Development, vol. 8, 2015, pp. 3681–3694. [Online]. Available: http://www.geosci-model-dev.net/8/3681/2015/

[11] O. Kolditz et al., "OpenGeoSys: An open source initiative for numerical simulation of thermo-hydro-mechanical/chemical (THM/C) processes in porous media," Environ Earth Sci, vol. 67, no. 2, 2012, pp. 589–599.

[12] K. Rink et al., "Virtual geographic environments for water pollution control," Int J Dig Earth, vol. 11, no. 4, 2018, pp. 397–407.

[13] Helmholtz Centre for Environmental Research – UFZ, "Homepage of Helmholtz Centre for Environmental Research," https://www.ufz.de/index.php?en=34216, retrieved: November 2018.

[14] Karlsruhe Institute of Technology – KIT, "KIT Data Manager," http://datamanager.kit.edu/index.php/kit-data-manager, retrieved: November 2018.

[15] R. Grunzke et al., "The MASi repository service - comprehensive, metadata-driven and multi-community research data management," Future Generation Computer Systems, 2018. [Online]. Available: https://doi.org/10.1016/j.future.2017.12.023

[16] S. Chacon and B. Straub, Git and Other Systems. Berkeley, CA: Apress, 2014, pp. 307–356. [Online]. Available: http://dx.doi.org/10.1007/978-1-4842-0076-6_9

[17] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick, Version control with subversion - the standard in open source version control. O'Reilly, 2008, retrieved: November 2018. [Online]. Available: http://www.oreilly.de/catalog/9780596510336/index.html

[18] C. Gormley and Z. Tong, Elasticsearch: The Definitive Guide, 1st ed. O'Reilly Media, Inc., 2015.

[19] W. F. Tichy, "Rcs – a system for version control," Software: Practice and Experience, vol. 15, no. 7, 1985, pp. 637–654. [Online]. Available: http://dx.doi.org/10.1002/spe.4380150703

[20] A. Löh, W. Swierstra, and D. Leijen, "A Principled Approach to Version Control," http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.8649, retrieved: November 2018.

[21] E. Sink, Version Control by Example, 1st ed. PYOW Sports Marketing, 2011.

[22] C. L. Paris, Automatic documentation generation: Including examples. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 12–25. [Online]. Available: http://dx.doi.org/10.1007/BFb0034794

[23] R. Swan and J. Allan, "Automatic generation of overview timelines," in Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR '00. New York, NY, USA: ACM, 2000, pp. 49–56. [Online]. Available: http://doi.acm.org/10.1145/345508.345546

[24] K. McKeown, K. Kukich, and J. Shaw, "Practical issues in automatic documentation generation," in Proceedings of the Fourth Conference on Applied Natural Language Processing, ser. ANLC '94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, pp. 7–14. [Online]. Available: http://dx.doi.org/10.3115/974358.974361

[25] B. Möller, O. Greß, and S. Posch, "Knowing what happened - automatic documentation of image analysis processes," in Computer Vision Systems - 8th International Conference, ICVS 2011, Sophia Antipolis, France, September 20-22, 2011. Proceedings, 2011, pp. 1–10. [Online]. Available: https://doi.org/10.1007/978-3-642-23968-7_1

[26] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," Cryptology ePrint Archive, Report 2017/190, 2017, http://eprint.iacr.org/2017/190.

[27] C. Dobraunig, M. Eichlseder, and F. Mendel, Analysis of SHA-512/224 and SHA-512/256. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 612–630. [Online]. Available: https://doi.org/10.1007/978-3-662-48800-3_25

[28] M. Szydlo and Y. L. Yin, Collision-Resistant Usage of MD5 and SHA-1 Via Message Preprocessing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 99–114. [Online]. Available: https://doi.org/10.1007/11605805_7

[29] D. Y. Naumov et al., "ufz/ogs-data: Initial zenodo release," Aug. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.840660

[30] C. Helbig, L. Bilke, H.-S. Bauer, M. Böttinger, and O. Kolditz, "Meva - an interactive visualization application for validation of multifaceted meteorological data with multiple 3d devices," PLOS ONE, vol. 10, no. 4, 04 2015, pp. 1–24. [Online]. Available: https://doi.org/10.1371/journal.pone.0123811

[31] E. Jang et al., "Identifying the influential aquifer heterogeneity factor on nitrate reduction processes by numerical simulation," Advances in Water Resources, vol. 99, Jan. 2017, pp. 38–52.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org