VERSIONING IN EARLY-STAGE HOSPITAL SIMULATION

Gabriel Wurzer

Vienna University of Technology

gabriel.wurzer@tuwien.ac.at

ABSTRACT
We present a framework for augmenting early-stage hospital simulation by versioning. The benefits are twofold: On the one hand, we give planners the opportunity to quickly *derive alternative models and variations*, which is in line with the mindset of early architectural design. On the other hand, the simulation can be combined with the revision history to form a *design tool*: For each revision, we take the simulation results as performance indicator, in order to determine what design paths to follow and what to abandon.

Keywords: Architecture, Hospital Planning, Early Design, Versioning, Framework

1. INTRODUCTION
In architectural terms, hospital design is the progression from a preliminary building concept to the implementation of a built form that takes up the envisioned clinical processes. In that context, we have continuously argued that simulation should be applied *as design tool* at the very beginning of a project, in addition to being an *optimization tool* for later stages (cf. e.g. Wurzer, Lorenz and Popov 2012): Early-stage planning determines most of the factors that define a hospital - spatial structure, operational routines and so on (see Background: Early Hospital Design Model). In contrast, later-stage planning sees these as requirements, subject to implementation and optimization - which is where simulation usually comes into play. However, simulation applied at this stage can easily get in its own way, if it suggests a change to any of the mentioned factors would be needed: In reality, it is often too late to incorporate far-reaching changes once a project is well underway.

1.1. Problem
Applying simulation early promises to be a possibility for informing the design team over the performance of a planned concept; however, this also comes at a price: In early planning, there exists not one single “fixed” building concept that acts as a model, but rather several alternatives and variations thereof in parallel. Clearly, investing effort in modeling each one of these would be too much of an effort, considering the timeframe available at this stage (typically 2-3 months, source: own survey of 55920 architectural competitions contained in the online database of the Austrian Chamber of Architects and Engineering Consultants, www.architekturwettbewerb.at).

1.2. Contribution
We present a conceptual framework that can keep track of revisions of a design, such that modeling efforts are kept to a manageable minimum. For that, we use a versioning system as being commonly used in source code management, albeit adapted to the problem domain at hand. In more detail,

- We first give a background on early hospital simulation (Section 2.1) and raise the issue that there is lack of versioning in that area (Section 2.2). After describing our model (Section 2.3), we give a background on versioning (Section 2.4), which forms the basis and terminology that is used throughout our paper.
- The core of our ideas lies in the twofold application of versioning to hospital simulation, first in the context of **model building** (see Section 3) and secondly as means for **tracking its evolution** (see Sections 4 and 5). The description of these aspects is given in some length, for which we make no apology: The details of how we can handle revisions intelligently were far from obvious for us, and we believe this to be also a point where others can draw a benefit out of our work. With the same purpose, we give a summary of our prototype implementation under Section 6.
- We conclude our paper with a discussion outlining applicability, alternative versioning mechanisms and techniques and future work (Section 7).

2. BACKGROUND
2.1. Early Design Simulation in Hospital Planning
Early hospital planning is a twofold activity (refer to Figure 1): It begins on the side of the client (who has to prepare the design brief), and continues at the side of the planning teams who takes part in a competition. There are two domains in which early-stage simulation can take place: *Site planning* and *functional design*. White (2004) characterizes these as *“[…] forces which*

locate building spaces and activities on the site. Function tends to locate building spaces in an introverted way in that they are primarily looking inward to each other for the rationale behind their positions in the scheme. Context, on the other hand, wants the spaces to migrate to different positions on the site in response to conditions outside the building. In function, the attraction is between spaces. In context, the attraction is between spaces and external site condition." -White 2004, p.22.

In that sense, simulation for site planning tries to look at external factors that affect a building, such as wind, noise, shading and so on. Functional simulation then tries to simulate its operation, e.g. the flow of people and material. Regardless of whether simulating interior or exterior factors, it is common that there are several alternative designs which need to be assessed. Doing so leads iteratively to new variations (basically a spiral process, although some phase models in hospital planning also bring forward the idea of "planning once, at the correct spot within the project" - see e.g. Lohfert 1973). Meetings in which alternatives and variations are scrutinized are called *design reviews*, a term that will be often recur in this paper. The key goal in that setting is to exchange views on the so-far produced preliminary designs, resulting in *design decisions* over which paths to follow, which to abandon or which to merge. Argumentation can rely on any means - formal (e.g. spreadsheets giving areas, operational diagrams showing patient flow, the results of a simulation, etc.) or informal (mostly narrative). The documentation of the decisions taken, together with the data on which they were based, is one of the reasons why we have undertaken this work. But even without this embedding into the context of the design process, simulation can benefit from knowing the evolvement of a model. However, for this to happen, we need to bridge the gap to versioning within simulation, which will examined closely in the next sections.

2.2. Lack of Versioning in Hospital Simulation

Hospital simulation has so far not embraced the topic of versioning beyond "document control" (i.e. uploading a model file as new version, as means for sharing content within a project team). To fortify that point, let us consider Arena, FlexSim HC, MedModel, AnyLogic and Simulink as representants of a broader field of software packages being used in the field. All of these have the ability to use a versioning system, either through direct integration within the program, or externally (via a client program). However, only AnyLogic and Simulink offer the possibility to revision control also the constituents of its models (internally being Java classes in the first case and code blocks in the second case - see Walker, Friedman and Aberg 2007), which goes into the right direction. What we envision, however, is more like Product Data Management (PDM), in which the creation and change to parts of a product are tracked through an entire lifecycle. Computer Aided Design (CAD) packages for

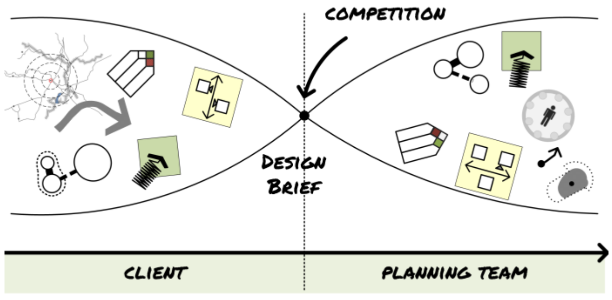


Figure 1: Early-stage planning occurring both at the client side and the side of a planning team.

engineering (e.g. CATIA, SolidWorks), for example, offer such an approach. However, they require an architect to specify an exact form (plus extensive metadata) for each part, which does not fit early design all too well: At this stage, planners rely on "sketchy" data - schematics, relationship diagrams and so on. Furthermore, there are hardly any Building Information Models (BIMs) for early stages of design, on top of which a PDM would sit. A literature survey shows that there are some authors who have brought forward an early-stage BIM with added simulation (e.g. energy, see Al-Sallal and Degelman 1994; environmental aspects, see Hartog, Koutamanis and Luscuere 1998). However, versioning is not considered or used.

Without being especially picky, we may note that the notion of "early planning" being used in literature differs significantly from what we would see as "preliminary" or "sketchy": the form of spaces is taken quite literally, and not as placeholder for later design work that will formulate them. Yet, representations used in early planning are very delicate in the sense of what is to be understood literary and what is only meant to be read as intermediate stage (and thus left open). A summary of such a sketchy representational model, which acts as the basis for our paper, is presented in due course.

2.3. Early Hospital Design Model

We have conducted an intensive analysis of early planning, by comparing four standard textbooks on functional design (White 1986; Schönfeld 1992; Neufert and Neufert 2002; Lohfert 2005) and furthermore adding information on site planning (White 2004) to find out what the commonly "agreed-upon" form of an early representation is. We have specifically targeted *schemata*, since these seem to be the style that features most prominently in all of these works when we come to hospital planning:

- A schema is a preliminary floor plan composed of *spaces* given as rectangles (see Figure 2a). There are two caveats to be mentioned right away: (1.) A space is not meant as a room. It is rather meant as a spatial containment, which might be physically enclosed (e.g. "hospital") or open (e.g. "waiting zone"). (2.) A space is not meant as form, even though being rectangular. It is merely a placeholder that might or might not be replaced with geometry later on during the design process. This, however, is done well after early design, in a phase called “form finding”. Furthermore, and especially for hospitals, spaces are color coded according to their role within the hospital (e.g. in German-speaking countries according to DIN 13080-2003-07).

- Spaces may contain other spaces, thus forming a spatial *hierarchy* - either a tree (strict containment) or a semi-lattice (spaces can overlap).
- Spaces have attributed *functions*, i.e. names of activities that may be performed in that space (see Figure 2c). Such functions are inscribed into spaces and possess a certain capacity (when considered as resource which can be acquired). However, they may be used in the whole area of a space and not only at the specific location where they appear.
- The *circulation* (i.e. the “way network” on the inside and outside of a building) is inscribed into the spaces in the form of lines (see Figure 2b).
- *Activities* are usually not depicted as single entities (see Figure 2d). Rather, chains of activities (*processes*) are depicted on top of the schema, as directed path *along the circulation* (see Figure). Processes are furthermore actor-centric, as they contain only activities of one specific group (patients, staff, visitors, but also material carts, etc.). This distinction is very important when defining staff-only areas and other more private areas, and thus, a color coding is usually employed for easy differentiation.

In principle, this description of schemas applies to all forms of early work in architectural planning. The connection to the context “hospital planning” is nevertheless important, since it narrows down the model to the subset actually used in practice. To further stress this point, other types of buildings are not as *process-driven*: There is, for example, not an explicit definition of work processes that precedes design, or, in the more extreme cases, there are no processes but behavioral rules that come into play (e.g. in shop design). Separation of traffic is not as strict (compare with public/private, staff/patients, sterile/non-sterile, etc. used in a hospital).

A further distinction is in the simulation employed on top of the basic model. Every function becomes a resource of limited capacity, which is used by activities of a process. The usage over time can be used to tailor their capacity to the expected patient volume, among many other possibilities for verifying a design (check adjacency, path lengths, etc.). An elaborate description of this was given in Wurzer, Lorenz and Pferzinger 2012, and is omitted here for the sake of brevity. In this paper, we rather wish to concentrate not on the actual simulation, but on the mechanism that enables us to model variations of models quickly, and to compare the results as means to inform the design process.

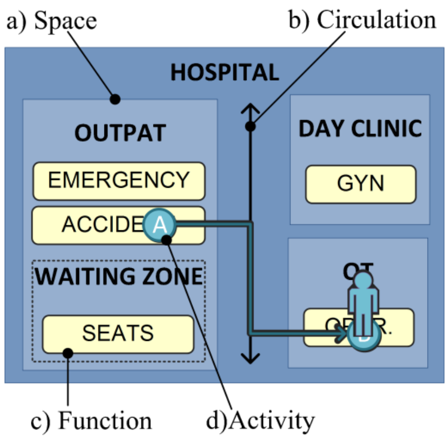


Figure 2: Elements of the architectural schema

2.4. Revisions within the Design Model

In early design, process model and space evolve in a side-by-side fashion, with changes happening in parallel. In the simplest case, revisions are handled *by convention* - each document is given a revision number. Exchange happens via a centralized platform (e.g. file-server) or decentralized (e.g. emails). However, handling large projects in this way is cumbersome and error-prone. Furthermore, on most centralized exchange platforms, there is the possibility to utilize a *versioning system* which automatically tracks revisions to files and also provides ways of ensuring that users may work in parallel. What follows is a brief description of centralized versioning, on which we have concentrated our efforts.

The basic unit for which a revision is stored is a file. The collection of all files and their respective revisions is called the central *repository*, typically stored on a file server. In contrast, each team member has only a local collection of files with exactly one version on his computer, which is called the *working copy*. Changes (creation, update or delete operations) to the files or their content are not registered by the versioning system as a revision until the user actively triggers a *commit* operation, which copies the files to the central repository. Other users may receive the updated versions of files by *updating* their working copy, either to the latest revision (*trunk*) or one that was explicitly specified.

In the latter aspect, *tagging* comes into play: A whole set of files can be attributed a label (e.g. “release 1.0”), which can later be utilized to fetch a defined snapshot of the repository that existed at a certain point in time. A closely coupled concept is that of defect tracking: If there is a problem occurring in a certain (tagged) snapshot of the repository, patches to files will be made at exactly this level. In practice, this involves opening a new line of development (*branch*), to which all commits concerning that version will go. Tags in a branch are typically given a subordinate label (e.g. “release 1.0 service pack 1”).

Apart from defect tracking, branching is also used to isolate experimental work from the main line of development. If it turns out that an experimental branch should be reintegrated with into the trunk, i.e. all conducted updates should be made available there, a *merge* operation comes into play: Each file from the branch is compared with the most recent version in the trunk. Several conflicts need to be resolved: For example, a file may have been deleted or renamed in either two locations. Furthermore, the actual file contents may conflict (when work has proceeded in parallel, updating or deleting parts that both versions have in common). There is no other way of conflict resolution than to fix these issues by hand: In the simplest case, the user conducting the merge must decide which change is promoted - that in the branch or the trunk version. More complicated cases involve blending both changes, which can be a quite laborious task.

The actual mechanism for handling parallel work differs among versioning systems: **Lock/Modify/Write** is a pessimistic strategy that restricts access to a file or folder (including its contents) to exactly one user. On the level of the local working copy, all files but those for which a lock has been acquired are read-only. After editing, a user returns the lock, either committing his changes or discarding them (thereby receiving the latest version of the files in question from the repository). **Copy**/**Modify**/**Merge** is an optimistic strategy that allows all users to simultaneously change files. As changes are committed, however, conflicts such as the ones mentioned earlier have to be resolved.

Note that these mechanisms are not the main focus of the paper, which argues for the utility of versioning for model building and simulation. Either method - Lock/Modify/Write or Copy/Modify/Merge, does not spare us from the troubles of conflict resolution that occur during merging (see Section 3.1 for an elaboration). Because locking is simpler to implement, however, we have chosen it as the basis for this paper. Some further thoughts on Copy/Modify/Merge as well as two alternative types of versioning (*decentralized versioning* and *object versioning*) will be given in the course of the discussion (see Section 7).

3. VERSIONING APPLIED TO MODEL

We can see the design process itself as form of versioning, in which multiple planners collaborate to produce a building concept sequentially through revisions, and parallel through variations and alternative models. Mapping this to the presented concepts found in revision control, we can say that:

- Each committed set of changes creates a revision (see Figure 3a).
- Alternative models are independent lines of development (see Figure 3b). Technically, this may be achieved through multiple repositories or a single repository containing two independent trunks.
- Variations are derivations of a previous model, internally represented by a branch (see Figure 2c). They are very similar to revisions, in the sense that alterations and extensions of a model can be performed. The difference, however, is that multiple variations can exist side-by-side for the sake of the design review (see Figure 3d). In the latter, one might also find alternative models. The difference between the two concepts is that variations refer to the same trunk and share a revision history (“ancestor models”, if you will). On the contrary, alternative models live in different trunks and have no common ancestor.
- “Merging” may occur in two distinct cases: Either a merge is made in the same trunk (i.e. between two variations), or in two different trunks. Only the first case is called a (true) *merge*, and follows the semantics described under Section 3.1. In the second case, however, the two merged revisions have no common ancestor, as they live in two different trunks. Thus, we can only incorporate the model from one branch into the other, which we will call *import*.
- Baselining a certain snapshot of the repository (e.g. to reflect that a revision represents an “agreed upon” deliverable) can be achieved by means of tagging (see Figure 3e).

These descriptions have just scratched the surface of how we can map concepts from hospital design to the technical means provided by a versioning system. Coming to more detail, especially regarding the use of simulation during design, we note that our model does not distinguish between structural changes and parameter changes in what it regards as “revisions”: For example, changing the size of a space is equally seen as

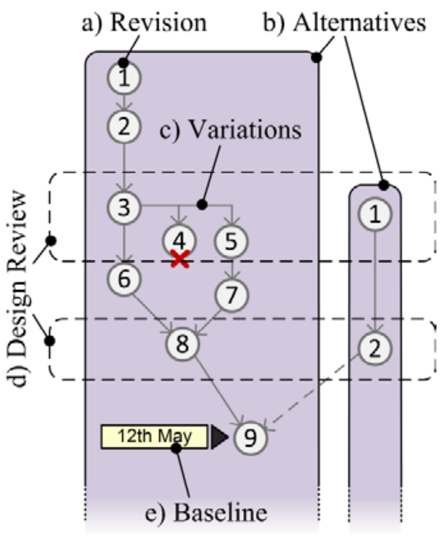
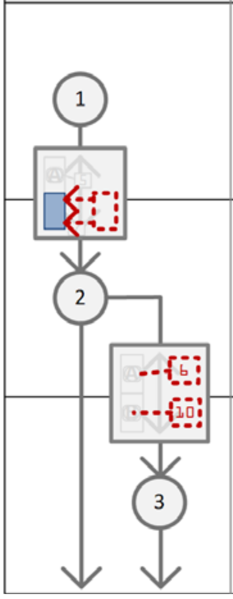
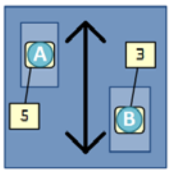
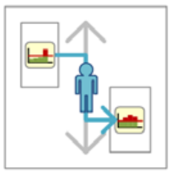
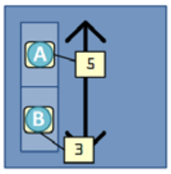
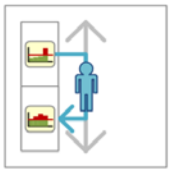
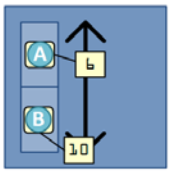
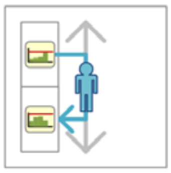


Figure 3: Design terms mapped to versioning

change as modifying the capacity of a function. We argue that it is up to the designer's intent whether such a revision is considered a modification (i.e. regular revision) or a variation (revision in a new branch). This missing differentiation concerns especially simulation experiments, in which we regard the alteration of the parameter space or the structure as own revision, for which we store the results in a 1:0-1 manner (every revision can have results attached, this is). Note that in the case of nondeterministic simulation, we expect that multiple simulation runs are already aggregated so that this holds.

An illustrative example in this sense is given in Table 1. The first revision is the initial commit, which defines two spaces each containing one function with bounded capacity (5 and 3). Furthermore, the model contains a process given by two activities A and B, which reference the underlying functions. The simulation run of revision 1 computes the utilization of both functions (time graph) and flow along the circulation (depicted as arrow). After examining these results, the designer opts to change the location of one space (e.g. because of proximity considerations), which requires to commit the new revision 2. In the simulation run, he notices that there is a constant resource shortage concerning both functions. The according measure would be to increase their capacity. However, the planning team has already allocated all available resources, and arguing for more might cause some troubles with the building client. Therefore, the planner creates a variation of revision 2, rather than a modification on the original branch (Footnote: what sounds like a peculiarity in this case, namely whether to commit a revision or extend to a new branch, is really useful when having a multitude of experiments, each contained in the latter). During a meeting with the building client, revisions 2 and 3 could then be brought forward and compared.

Table 1: Simplified Example Model with Versioning

Versioning System	Model	Simulation
		
		
		

3.1. Resolving merge conflicts

A merge incorporates two revisions, resulting in a third one, which is added to the target branch (the other is the source). In the simplest form, the two branches are alternatives (i.e. no common ancestor); in this case, the user has the choice to import the whole or parts of the hierarchy of the source. Note that constraints still apply - one must for example check if spaces overlap, and, if so, let the user resolve these either directly in the merge or afterwards (warning icon depicted on inconsistent space).

In case we have two variants (i.e. branches with a common revision) being merged, the user has to choose what changes he wants to contribute to the merged model. Figure 4 shows a simplified example in which we regard two revisions being merged. In each revision, we can see the spatial hierarchy in which A-E denote spaces, and ~, - and + respectively denote change, deletion and addition since the last common revision.

In the first case (see Figure 4a), a change was made to a space (or its properties) in either revision. We may automatically propose to use that version for the merged result, although this is not prescriptive: It may well occur that the planner wants to review what the changes are, before choosing to apply them.

The second and third case involve tree conflicts in which the user has to decide which of two sub-trees are chosen: In Figure 4b, space B has been changed in the left revision and deleted in the right one. In Figure 4c, this is reversed (space C_{left} is deleted, C_{right} updated), however, there is additionally a new space Eright under C_{right} that will be affected by the choice.

The fourth case (Figure 4d) shows the addition of a space E_{left} that does not affect the other revision. Here, E_{left} has nothing to do with E_{right}, it is just an additional space under A_{left} that is imported into the merged model as described for alternatives.

Under the hood, we have to use a tree walk algorithm on both trees that marks sub-trees as conflicting and gathers options for merging such as in Figure 4b and c. Depending on the type of conflict encountered, the algorithm has to either stop (deletion encountered) or continue further down into the hierarchy (change encountered).

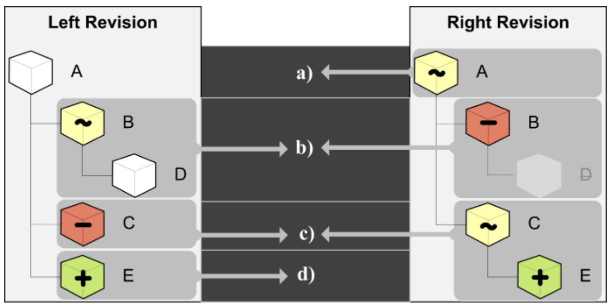


Figure 4: Merge conflicts (~ change, + add, - delete).

4. INFLUENCE ON AND BY SIMULATION

There is an additional aspect that is worth spending some time on: The simulation itself. By correlating the deltas between the revisions and the simulation results, one can get an insight into the changed performance. Other advantages, such as a comparison of the changed or completely alternative building structure, are beyond the context of this paper (considering that the intended community is in simulation), however, a good starting point for doing such a comparison would be work by Grasl and Economou (2007).

On the other hand, the simulation might be a good indicator for compacting the revision history; during the buildup of a building layout, it happens often that committed revisions are not a reflection of significant work that has been carried out, but a mere technically motivated coincidence. When the hospital project transitions into a stable phase, it might be advantageous to merge revisions that do not carry significance (because the concept at this stage was intermediary or the simulation results proved to be non-satisfactory) into a single one. The flag that signifies a "significant" revision can be based on specific tags, which were introduced earlier in the paper (e.g. "Baseline", "Significant", etc.). This capability might seem trivial at first, but it affect the performance of the overall versioning system and the planning team alike: The more revisions have to be retrieved, the greater are the performance impacts (as CAD data is generally large and involves a lot of metadata to process, in case of computing deltas); the mental ability to sift through the revision history is also a point to consider in this case (i.e. usability).

5. SIMULATION AS DESIGN TOOL

When we regard the design process as generator for variations and alternatives, we may augment the work that is done in early hospital planning through comparison of the results. The details over how this may be done on the level of the results have been pulled out into a parallel paper (Wurzer and Lorenz 2013), which utilizes Fuzzy techniques to compare crisp results to linguistic requirements (especially adjacency and process interaction constraints). In the context of this paper, we deal with the implications of such a comparison - either the rejection or acceptance of a revision being present in a branch.

As Rittel and Webber (1973) note, design is a *wicked problem* that needs to be understood not merely by its deliverables but by the argumentation chain that leads a planning team to accept either one or another solution. A system capable of providing insight into the latter was coined as 'Issue Based Information System' (IBIS), which is exactly the direction into which our approach goes. When we regard the versioning system as such a chain (see Figure 5) and the simulation outcomes connected to each result as performance indicator, it is clear that the process of developing a hospital concept become targeted at this aspect; however, *a hospital is a process-driven building type* that requires a 24/7 operation, so this view is quite sane. Furthermore, in the light of the time constraints for developing an initial concept (see Introduction), there is need for an approach that can guide the conceptual phase such that the design can evolve rapidly, but at the same time in a coordinated fashion. Embedding simulation in this phase can combine both needs, but even further: It can anticipate decisions that would otherwise be deferred to later phases in which changes are harder to implement.

6. PROTOTYPE IMPLEMENTATION

What we present herein is a *framework*, or put differently: A structured concept which allows us to utilize versioning as means for rapid generation of variations and alternatives in early hospital planning. Our prototype implementation, which is currently under development, is implemented in Java and operates on a centralized Subversion repository and the corresponding local working copy. The choice of Subversion is quite arbitrary, it must be noted: We do not require a specific system, as long as the operations which we utilize (and these are comparatively little) are present. The translation between our (higher-level) versioning operations and the corresponding subversion commands is carried out via a plug-in implementation, which runs under Rhino for Java (a JavaScript engine).

We store our model as xml files in folders. Every folder stands for a space. In there, we have a single xml file which contains the properties and nested elements that belong to this space. We utilize locking as given in Subversion, in order to allow only one user access to a folder. However, as subversion can only lock files but not folders, we handle a lock to the space xml file as "lock for the folder", which requires some logic embedded into our software. The same is true for merging: Subversion would normally compare xml files to get their textual differences; however, this is not accurate since they are really objects with references, and this process becomes thus a little bit more complicated. Therefore, we especially deal with merging under the next section.

Simulation outcomes are stored at the top level of the file tree, at the moment utilizing a simple report-type depiction (see right part of Figure 5) which clearly

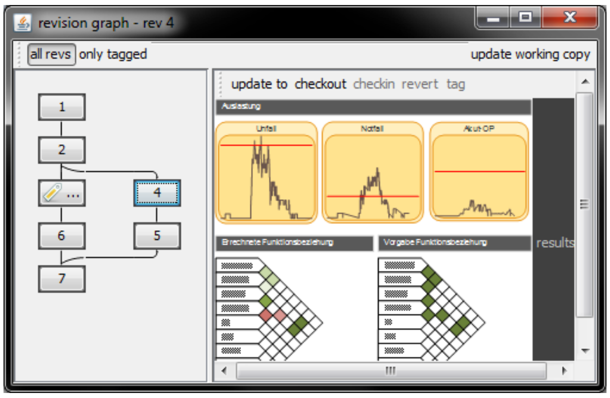


Figure 5: Revision History as Argumentation Chain, Simulation as Performance Indicator

should be improved to semantically dig through the revision deltas (future work). Also in the concept, but not implemented yet, is the explicit declaration which branches are active and which ones discontinued (see red "x" depicted in Figure 3), which could easily be done with the help of special tags.

7. DISCUSSION AND FUTURE WORK

The preceding elaboration has shown that merging revisions can raise inconsistencies (see Section 3.1, "Resolving Merge Conflicts"). The model must be able to cope with these, in order to let the planner resolve them later on. Thus, we may note as implementation notes for authors wishing to implement such a model that: (1.) Even a tree hierarchy must be implemented as a semi-lattice, in order to accommodate overlaps. (2.) References must be handled with care, as delete operations could have removed the original entities.

The aspect of references strongly relates to *object databases*, which are the usual means for persistence in a PDM: In contrast to the file-based approach presented herein, versioned entities and references among them are stored. More than that, concepts such as inheritance between entities and explicit definition of relations between these (also acting as constraints) allow one to migrate much of the logic that must otherwise be implemented in the application to the persistence layer. Thus, we may conclude that this would clearly be a valuable field for future research.

The same applies to the locking mechanism: It must be investigated during actual field studies with the implementation if our merge algorithm is suited for parallel work using a *Copy/Modify/Merge* cycle, and, even more importantly, if such a continuous merge is seen as beneficial by the users.

Decentralized versioning is also on our future work list: Each user's has a local repository, from which others can pull (if allowed, they can also push their own content into it). As consequence, there is no need to be online when commiting, which is a huge advantage for offline work; it further means that users will commit very often, rather than "once" when everything has stabilized. The pull and push mechanism may, however, require a bit of rethinking on the side of the users.

ACKNOWLEDGEMENTS

This work has been partly funded by The Technology Agency of the City of Vienna (ZIT) as part of the ((MODYPLAN)) Project, as part of the Call Smart Vienna 2012.

REFERENCES

Al-Sallal, K.A. and Degelman, L.0., 1994, A Hypermedia Model for Supporting Energy Design in Buildings, In: *Proceedings of the 14th Conference of the Association for Computer Aided Design In Architecture*, pp. 39-50, St. Louis, Missouri.

DIN 13080-2003-07, Division of hospitals into functional areas and functional sections, Norm, Beuth, Berlin.

Grasl, T. and Economou, A., 2007, Spatial Similarity Metrics: Graph theoretic distance measurement and floor plan abstraction. In: *Proceedings of the 12th International CAAD Futures Conference*, pp. 251-263, 11-13 July, Sydney.

Härder, T. and Reuter, A., 1983, Principles of Transaction-Oriented Database Recovery, *Computing Surveys*, 15 (4), 287-317.

Hartog, J.P. den, Koutamanis, A. and Luscuere, P.G., 1998, Simulation and evaluation of environmental aspects throughout the designprocess [sic!], In: *4th Conference on Design and Decision Support Systems in Architecture and Urban Planning*, ISBN 90-6814-081-7, Paper ID 25, 21 pages. Available from: http://repository.tudelft.nl/assets/uuid:f4654ac4-0ba7-4e0e-a0fe-24d1084e8a71/Koutamanis_1998_1.content.pdf [accessed 7th March 2013]

Lohfert, P., 1973, *Zur Methodik der Krankenhausplanung*, Werner-Verlag, Düsseldorf.

Rittel, H. and Webber, M. 1973, *Dilemmas in a General Theory of Planning*, Policy Sciences, 4, 155-169.

Walker, G., Friedman, J. and Aberg, R., 2007, Configuration Management of the Model-Based Design Process, SAE Paper 2007-01-1775. Available from: http://www.mathworks.com/tagteam/40503_SAE-2007-01-1775-Configuration-Management-MathWorks.pdf.pdf [accessed 7th March 2013]

White, E.T., 2004, *Site Analysis - Diagramming Information for Architectural Design*, Architectural Media, Tallahassee.

Wurzer, G., 2010, Schematic Systems – Constraining Functions Through Processes (and Vice Versa), In: *International Journal of Architectural Computing*, 08 (02), 197 – 213.

Wurzer, G., Lorenz, W.E. and Pferzinger M., 2012, Pre-Tender Hospital Simulation Using Naive Diagrams As Models, In: *Proceedings of the International Workshop on Innovative Simulation for Health Care 2012*, ISBN: 978-88-97999-13-3, Dime Università Di Genova, Paper ID 35, 6 pages. Available from: http://publik.tuwien.ac.at/files/PubDat_209962.pdf [accessed 7th March 2013]

Wurzer, G., Lorenz, W.E. and Popov, N., 2012, Meeting Simulation Needs of Early-Stage Design through Agent-Based Simulation, In: *Proceedings of the 30th International Conference on Education and Research in Computer Aided Architectural Design in Europe*, pp. 613-620, September 12-14, Prague.

Wurzer, G. and Lorenz, W.E., 2013, From Quantities to Qualities in Early-Stage Hospital Simulation, Submitted to the 2nd *International Workshop on Innovative Simulation in Healthcare*, pending review.