AUTODESK UNIVERSITY

CI125300

Making 4D Accessible—Benefits of a Streamlined Workflow

Martin Holmberg
Skanska Sweden

Learning Objectives

- Understand which challenges may occur when pursuing a streamlined workflow.
- Understand the benefits of such a workflow.

Description

Discover what we as a contractor had to do to make 4D accessible as a basis for further uses. By streamlining the workflow and introducing structured information requirements regarding Building Information Modeling (BIM) models and planning, 4D becomes a natural result of the process. Having a real-time, up-to-date 4D finally provides practical uses during the production phase. From being used merely for project presentations, it's now relevant to use 4D as a basis for site planning and logistics, on-site safety introductions and briefings, and as a visual support during planning itself—providing possibilities for increased quality. We'll show our current workflow—which includes Primavera, Navisworks software, 3ds Max software—and the interfaces that we've created to streamline the process. In the end, we want you to be inspired by the possibilities and benefits of a streamlined 4D workflow.

Speaker

Martin Holmberg has been working at Skanska Sweden for several years within the civil engineering business stream. At Skanska Sweden his focus has gravitated towards utilizing his programming skills to develop innovative solutions and workflows for various BIM applications. A focus that has been with him even through his bachelor's degree in civil engineering at Chalmers University of Technology. Despite being rather young he has already acquired knowledge in a broad range of topics spanning from CAD workflows aided by Dynamo to visualization and augmented reality.

martin.holmberg@skanska.se

AUTODESK UNIVERSITY

What did we sought to accomplish?

While most of the 4D-tools are based around the artisan way of creating a simulation, we were interested in a more automated approach. This class will cover our journey in finding such a workflow.

What do we mean by 4D?

Since it's a central part of this topic let's start by specify what we mean by 4D. The following will serve as our quite informal definition of 4D in the current context.

A 4D-simulation is the resulting simulation acquired from connecting a schedule to graphical representations of the scheduled tasks so that the total representation varies with time according to the schedule. In our examples the simulation is often an illustration over how the site will evolve during the project.

Early iterations of workflows

Even though the early iterations of the workflows aren't breaking any new ground they still provided us with important insights. Therefore the different iterations and their shortcomings will be covered in this section. Navisworks is the common ground through all the different iterations of 4D-workflows covered in this section. What made us use Navisworks from the start might have been just convenience since we already had experience with it from other uses and it got the capability to create a 4D-simulation with the TimeLiner tool.

A first 4D

From the beginning we just wanted to achieve a 4D-simulation so naturally this first workflow was quite crude. By simply importing the schedule as a .csv Data Source in Navisworks and then manually attaching model items to the tasks one task at the time. After using this workflow one realizes that it leaves a lot to be desired. For instance, as soon as a model is updated it is likely that you will have to re-attach that model to the related tasks, once again task by task. In fact most changes in either the schedule or the models lead to tedious work.

An improved workflow

Navisworks provides the possibility to define search sets that filter out items from the model that meet some specified criteria. A rather obvious improvement to the first workflow is to utilize this possibility. As with the first workflow we use a .csv Data Source to get the schedule into Navisworks. This time instead of attaching model items to the tasks manually we create a search set for each task. Then using the function "Auto-Attach Using Rule" to attach the search sets to the tasks. This way when a model is updated or added the search set will automatically associate model items to tasks based on the search criteria. While this workflow has improved the original quite a bit, it still has some drawbacks. Each search set has to be done manually, and as soon as a new task is added to the schedule you will have to create a corresponding search set. By now we have also noticed that some of our scheduled activities are difficult to illustrate using only one task in the TimeLiner, as different model items of the activity needs to be represented in different ways. For example if we want simulate an excavation, and the model is composed of surfaces instead of solids, then we want the bottom surface to go from hidden to visible and the top surface to go from visible to hidden.

Excel to the rescue

The goal for further improvements was now to automate the tedious work of managing the search sets for each task while simultaneously dealing with the multiple-representations problem. A colleague of mine who knows many of the ins and outs of Navisworks knew that it was possible to import search sets as .xml. Thus, the concept that emerged was to generate an .xml with the needed search sets. Since no workflow is complete without Excel, naturally it was the go-to tool to realize this concept. The result was a VBA-script within an Excel-document that took a .csv containing the schedule as the only input. This VBA-script worked as a preprocessing step that added mirror-activities for tasks of specific types to the schedule as well as generating an .xml with search sets for each activity in the new schedule. After the preprocessing step what remained to do was simply to import the new .csv schedule as before, import the search sets .xml and run the function “Auto-Attach Using Rule”. It’s easy to see that the VBA-script must contain some kind of rules to determine how an activity’s corresponding search set should be defined. The rule we used at this stage was to have a unique identifier for each activity and then try to find a match to that identifier at a predefined property in the models. As we will see later we really trip ourselves up with this rule. By a few steps this workflow gives us a 4D-simulation that stays connected when we update the models. To update the schedule you would have to pass it through the preprocessing step with the VBA-script and then import it all again. Here’s where our rule trips us up. For the search sets to match the right model items the model items need to have a property with the identifier of the associated activities. This means that when the schedule is update in such a way that model items are associated with a different set of activities than before, then the model need to be update to reflect that change. In other words, when creating a model for the 4D-simulation one must know which activities the items of the model is associated with. This is a major drawback to an otherwise quite neat workflow.

Recent iterations of workflows

In addition to this major problem we also was experiencing a handful of other issues. Most of them was related to the fact that a whole lot of things vary from project to project. Having learned a lot from earlier iterations of the workflows, and from other hardly related projects, we decided that the next step was to setup a database with configurations for the different projects. To consume those configurations in an orderly manner the concept was to build our own plugin for Navisworks. This alone would not solve the major problem, but it would certainly open the doors to some new approaches.

Our configuration database

When we started setting up the database a whole new world of challenges dawned upon us. One of the major challenges was how to make a user interface for the projects to manage their configurations. Another challenge was how to let the planners upload their schedules to the database. Everything that was easy before became a challenge now.

All of our projects have their own site for document management etc. Since the projects where already comfortable with uploading files there, we decided to modify the sites to serve as the main interface to our database. We setup a web service to process to be the bridge between the sites and the database. After some tinkering we built a function on the document site that takes a .csv, of a schedule, from the document site and writes it to the database by calling the web service. In addition we also built an interface on the document site that lets the user edit the

data in some tables from the database, again using the web service as a bridge. This would prove itself very useful for a lot of the issues yet to be addressed.

One minor issue that we sought to fix with the database was that, in the older workflows, a task wouldn't get a task type unless a task type with that name was defined in the Navisworks file. This might not seem as a big deal, but at the time we didn't have any routines for transferring the task type definitions between files. To solve this issue we setup tables in the database directly corresponding to the TimeLiner Configure table and Appearance Definition table from Navisworks. Then we made sure the task types from the database also was present in the Navisworks file when we setup the TimeLiner. However, this would not solve the entire problem, since it was still possible to have a task type in the schedule that doesn't have a match in the database. Actually this only highlighted a bigger issue, namely that we need to validate the schedules against the configuration in the database. Since we already had the infrastructure in place all we had to do was to add a validation process in the web service and give some feedback to the user based on the outcome.

Another issue with the VBA-script approach was that the property carrying the connection to the schedule was hardcoded in the script. By adding a table in the database and allowing the data in it to be edited, using the interface on the document site, these properties could be defined individually for the different projects.

What now remains of the system is the plugin. The plugin consisted of a settings page that would let the user specify which project this Navisworks file should be associated with. Other than that there was only one function. That function lets the user pick one of the uploaded schedules and then imports it to the TimeLiner. In addition to that it also generated the search sets and attached them to the corresponding task.

As we have mentioned this workflow does now have validation of the schedules, project individual search definitions and task types. However, after building a plugin, setting up a database, launching a web service and creating some interfaces this workflow still suffers from the same major drawback as the VBA-script based workflow.

Code structure

Even though the results from the last workflow was quite underwhelming, the underlying infrastructure built up for the workflow presents an opportunity to deal with this major issue.

In most projects some kind of code structure is used to define parts of the projects. It can be set up as a combination of standard classification of building elements, geographical location or similar. These code structures tend to vary from project to project, or at least it has in the projects that I've been involved in. It seems like a code structure is exactly what we need to solve our issue, as it could provide a common language between the schedules and the models. With this insight, the concept for this improvement becomes to make sure that the schedules and the models are coded according to a common code structure for the project.

By adding two tables to the database we could provide the capability for projects to define their code structures. The first table defines the segments that in combination make up the code structure. In this table we also define some characteristics of each segment, for instance if valid values of the segments should be defined as a pattern or as a list of values. The second table

defines the valid values for those segments that are defined by a list of values. Since it's now possible that we want the search sets to match several properties some changes need to be done to reflect that. In the previous iteration we used a table to define which properties could carry the activity identifier. By changing that table so that it assigns a relation between code segments and properties we can define sets of properties that carry the entire code structure. The plugin now works in such a way that defining multiple sets of configuration will generate a search set that will find matching code structure in either of the defined properties. This means that it's possible to setup different definitions for different file formats or for different suppliers of models.

The code structure isn't only a key to solving our problem, it also presents us with a new set of problems. Before, we could set up the search sets to find model items matching a value in some specified properties. But it's quite common that these code structures are built upon a hierarchy, which means that it's not necessarily a perfect match that we seek. For instance, we might seek all model items that correspond to children of the given value in the hierarchy. What we would need is some structured description of how to build the search sets for given values of the code structure. More specific, given a value of a code segment we want to find out the search criteria to use. After some thinking we came up with an idea to setup tables that can be used to define rules that yield search criteria based on a code segment value. Associating a rule to each code segment was done simply by adding a rule column to the code segment definition table.

With code structure incorporated into the solution, the workflow now yields a 4D-simulation for which the models and the schedule can be updated independent of each other without any drawbacks.

Primavera integration

Although our workflow was quite successful, and had come a long way by now, it was still wasn't very useful for the planners, since they had to export, upload and import their schedule to get visible feedback. If the 4D-simulation was to be an aid for the planners during their work, this process needed a shortcut.

One of the planners we worked with suggested that we should have a direct integration to Primavera within our plugin. So, that's what we did. In order to be able to synchronize with Primavera, we needed some way to configure where to find each code segment inside Primavera. To setup an additional table in the database has been the go-to solution for most of the problems so far, and it didn't let us down this time either. Since we already had a function to generate a 4D-simulation based on a schedule from the database, all we had to do for the integration was to make sure that the schedule from Primavera followed the same format.

By creating this shortcut we separated the workflow into an express route for the planners to see the results of their ongoing work, and the original workflow became a workflow to publish schedules to make them accessible to the rest of the project.

Benefits

In addition to the benefits covered in this section there are some benefits that come directly from having models reviewed together, which is a consequence of creating and consuming the 4D-simulations.

Clear communication

One of the greatest benefits from having a 4D-simulation is clearer communication. It turns out that discussing a schedule using a 4D-simulation as visual support helps to avoid misconceptions, much in the same way that using a 3D-model when discussing a design helps to avoid misconceptions. This proves especially useful during the tender stages where design and schedule changes can be more frequent.

During on-site introductions and safety briefing the 4D-simulations have been greatly appreciated for improving the understanding of what is going on at the site.

Increased quality

A 4D-simulation might provide insights for the planners that might not have been so apparent otherwise. For instance it can make it clear that some activities cannot be carried out simultaneously. If such things can be found early than there is a greater chance to plan accordingly. Thus, increasing the quality of the schedule. This effect is of course greatly depending on the access of accurate models early on.

Extensions and further uses

We have seen that there are some nice benefits of having access to an up-to-date 4D-simulation. However, it still feels like there is lots of untapped potential. This section covers some of the extensions and further uses that can be accomplished with 4D as a basis.

4D-export

To truly make our 4D into a basis for further uses we need to be able to export it in some way. The schedules are already in the database so the only things that remains is to export the models and keep track of which model items that are associated with each activity.

In most CAD-formats the objects got an identifier as to separate the objects from each other. By setting up a table that defines which properties carry these identifiers, it's easy enough to store the relation between activities and identifiers in another table. Navisworks has the capability of exporting the models as FBX-files. However, when using this FBX-export it's not possible, as far as I know, to get it to include the identifiers. Which in turn means that we would not have a valid connection to the activities.

When we met this obstacle we decided to build our own FBX-exporter as to include the identifiers. While we were at it we also added some batch exporting capabilities.

Now we were able to reproduce the 4D in any software that takes the FBX format and allows some scripting.

Visualization

3ds Max happens to be a software that both provides scripting capabilities and can import FBX-files. By rewriting the 4D-setup function from the Navisworks plugin in MAXScript we was able to reproduce the 4D in 3ds Max. If we only wanted to achieve the same result as in Navisworks the export step was a waste of time. However, by taking the 4D to 3ds Max the possibilities for improving the visual appeal of the simulation are drastically increased. For instance, it's possible to alter the MAXScript in order to associate an animation scheme to the different task types.

To achieve a more interactive experience the same approach can be taken in Unity, a game engine. By reproducing the 4D in Unity we can also publish the simulation to mobile devices.

Site planning

As a 4D-simulation illustrates where work is being done at different stages of the project it is excellent as a basis for site planning.

Work planning

For work planning a 4D-simulation is a good basis to raise awareness of other ongoing activities. It's also possible to detail tasks in the 4D-simulation in order to simulate the work plan.

Conclusion

While 4D-tools are based around the artisan way of creating a simulation there are ways to achieve a more automated process. By introducing a code structure it is also possible to setup a 4D-simulation for which the schedule and the models can be updated independently. Many benefits can be acquired by having access to up-to-date 4D-simulations. It is also possible to create workflows that reproduces the 4D-simulation in other software where it can be used as a basis for further uses, such as visualization and interactive applications.