1 INTRODUCTION

1.1 The non-scalable nature of Directed Acyclic Graphs (DAG)

Within the realm of highly complex and large-scale architectural projects, quitting the tender phase does not necessarily implicate that all digital objects have been actually “frozen”, meaning that the geometry is fully generated and accessible. Those will be further design-processed during the post-tender phase, once the materials used and available machines are perfectly known and the fabricator actually needs specific data (before and during construction). This data is generally very expensive, since it takes time to generate, organize and share amongst all trades.

The usually tight time-line framing such complex architectures has an important influence on how the team manages, handles and delivers the latter. Hard deadlines require high productivity preventing therefore the possibility to organize all computational processes within a neatly designed Directed Acyclic Graph (DAG) based workflow (the underlying logic on which are based many existing generative design tools, such as Grasshopper and Dynamo, for example). Indeed, trying to encapsulate the full complexity of the project’s database within one single pipeline would most probably take more time than completing the project itself, especially when we know that the AEC industry is fragmented through many different software platforms used by the various stakeholders.

Therefore, the different trades need to come up with their own custom methods, interoperability tools and quick "shortcuts" in the design process (such as exported sub-models and email exchange processes), producing redundant models and files in order to communicate with others and across different software environments. Consequently, computational design specialists and consultancy practices have been developing their own internal methods in order obtain the best control on the generated data (and corresponding metadata) through all stages.

We will illustrate such practices by describing the workflows developed both at Front and Design-to-Production, both being well known practices specialized in rationalizing very complex geometries at late stages in the design process, specialists in rationalizing geometrically complex architectural projects for further delivery of precise fabrication data to construction companies.

2 STATE OF THE ART

2.1 Front

During the realization of the City of Dreams Casino Hotel in Macau (conceived by Zaha Hadid Architects), the consultancy practice Front developed a modelling strategy (“Building Information Generation”) enabling parallel generation of information and attributes (through their own custom Rhino3d Plug-in Elefront) necessary for further fabrication. The whole modelling process consisted of a strategic alternation between the generation of objects in Grasshopper and their subsequent storage and classification within Rhino3d models (in which the geometry was
“locked”, or “dead”) from which was generated further information through a next iteration of Grasshopper sessions (within which parametric linkage was kept), and so on. The computational design process was therefore scattered into several parts, allowing manual interventions, proof checking and potential corrections from the expert user, before generating a new set of data based on the previous one, and thus through all scales until the highest modelling resolution (Van Der Heijden, 2015).

2.2 Design-To-Production

The consultancy practice Design-to-Production organizes its objects through specific layer tables within Rhino3d, enabling both its storage and classification. Similar to a directory tree structure, a layer table is organized into different levels (or depths) that communicate from a root layer to all its successor sublayers. Generally, both the root-layer (containing the component’s label) and the leaf-layers (containing its geometries) are defined through the protocol generating the information. The intermediate layer levels are identified by the component’s name and mainly serve to structure the information in order to make it more human-readable, defined by the expert user him/herself. Therefore, the layer tree is independent from the “semantics” of the component and just a UI-feature. Depending on the nature of the component (being a beam, an opening, a facade element, etc.), it will be stored within a specific submodel (or worksession) that refers to a larger master file from which data communication with other submodels becomes possible (Scheurer, 2012).

Both modelling processes developed at Front and Design-to-Production carefully curate and fragment the design process into separate parts in order to allow manual interventions, classifying the generated objects and passing important information between the different (sub-)models. However, this segregation of the design process and its current curation could be improved and better supported by the development of specific tools and interfaces, enabling the user to experience a more seamless workflow at late stages in the project. While section 3 will focus on inter-scalar search interfaces, section 4 will describe different methods to build and queries custom schemas. Finally, section 5 will tackle the problem of interoperability and schema-based workflows.

3 INTER-SCALAR SEARCH INTERFACES

3.1 Dataviz tools: Sunburst Diagrams

In order to get a better understanding of Design-to-Production’s data structure, the layer table has been extracted and displayed through the form of a Sunburst Diagram (Fig. 1). Each level is represented by a circle, the latter being divided into smaller arcs corresponding to the sibling layers, the offset or each arc corresponding to the number of objects contained within each layer. Such diagram allows the user to obtain a better insight into the general structure and intricacy of the layer table, and quickly spotlight the specific layers that contain a high number of objects.

3.2 LayerStalker: an Inter-Scalar Search Interface

The functionality of the corresponding Sunburst Diagrams (acting so far as pure data visualization tools)
has been extended by introducing interaction features through “LayerStalker” (Fig. 2), a developed embedded interface within Rhino3d from which the user is able to call multiple layers based on generic strings or keywords (e.g. “detailed volume”, “dowel”, “drill”, “axis”, “connector”, etc.). This is exemplified in Figure 2, where the value “drill” is called, displaying all objects whose respective layer names contain the exact same tag, and hiding all the others. The multi-scalar model on the right displays the selected layers and highlights the gathered centralized information within the Rhino3d viewport, on the left. Such example highlights the fact that implicit relationships exist between objects that are situated within unrelated leaves of the data structure. Indeed, different ways of organizing information exist, and one classification strategy might actually hide another. Therefore, objects need to be properly referenced by meaningful attributes, user strings or User Dictionaries, in order to be able to operate efficient queries later on during the design process. The interface described here has been developed within Rhino3d, relying specifically on the respective software’s geometrical database. However, one might speculate to generalize such approach by querying objects from a more holistic database which could gather as well data coming from external software platforms. Therefore, section 4 describes how to assemble and query cross-platform custom schemas based on existing tools, workflows and technologies.

3.3 LayerExplorer: A Zoomable Search Interface

Because the layer hierarchy might be too intricate and complex to be grasped and comprehended as a whole, I would also suggest that a zoomable interface would be required, letting the user navigate between the different levels of resolution that he or she defined upstream. “LayerExplorer” (Fig. 3) has been developed and built within the Processing environment platform, based on an existing sketch (http://www.generative-gestaltung.de/1/M_5_5_01_TOOL) and adapted for this specific scenario. It takes as input the existing Rhino3d’s LayerTable data structure and translates it within the Processing’s interactive environment.
4 BUILDING AND QUERYING CUSTOM SCHEMAS

4.1 Initial experiment with Speckle

The next experiment (Fig. 4) has been built on top of Speckle, a main data communication protocol platform which offers at its core a database generic enough to host geometrical objects coming from various software platforms. The minimum amount of information necessary for the future reconstruction of those objects is stored along with customized user dictionaries containing parallel information defined by the user. This enables the construction of custom schemas embodying a neat hierarchy of objects and sub-objects, allowing external stakeholders to operate meaningful queries from the database at later stages.

In order to exemplify this process, we take here as a case study a “MainBeam” object (Fig. 3, left), created by Design-to-Production. This “MainBeam” is defined by a group of sub-objects stored across many different layers and sub-layers from the software’s layer table, the latter acting as UI feature embodying the object’s schema itself. This nested layer hierarchy has been translated into a Rhino3d’s UserDictionary (Fig. 3, middle) and further converted into a JSON format that has been sent to the Speckle’s database, currently relying on MongoDB (Fig. 3, right), from which specific targeted queries can be operated upon, both structured and unstructured. In the first case, the user can enter a specific path corresponding to the object’s custom schema, and retrieve its corresponding geometry (Fig. 5). In the second case, the user can enter a specific type or attribute to query all concerned objects. For example, Figure 6 represents the data transfer of a Design-to-Production’s “MainBeam” that is composed from many different data-rich object
types (placed on different layers and grouped together), from its original model environment (left) to a new one (right) within which can be retrieved filtered information. In this case, all objects of type “Curve” are called and re-generated along with their respective attributes (Layer, Group, Color, etc.).

4.2 SchemaBuilder

Based on the previous experiments, a more generic UI/UX interface, called “SchemaBuilder” (Fig. 7), has been developed, allowing to seamlessly build custom nested hierarchies of geometries, ready to be sent to the Speckle’s database (along with their respective schemas and properties/attributes). It would prove to be extremely useful for the many consultancy practices that interact with data-rich models at late stages in the design process. Ideally, this interface should be able to process both software specific object schemas (e.g. a RhinoObject with its respective properties and attributes) as well as the user-defined customized schemas (e.g. the D2P’s “MainBeam”) that usually host a multitude of different software specific objects. The next sections describe the current UI/UX flowchart as well as its tree interface and speculates on different features that could improve the current version of SchemaBuilder.

4.2.1 SchemaBuilder’s UI/UX Flowchart

First, the user selects an object located within the software environment. Then, after clicking on the “Attach Properties” button, the corresponding object’s schema unfolds as a checklist from which it is possible to manually check/uncheck the properties/attributes that need (or not) to be kept and attached to the object. The user might encounter a situation where the attributes that he or she wishes to attach are not directly accessible from the software object’s schema. Therefore, the latter is still editable after the first generation its inherent properties, so the user can manually add other custom ones. The interface could also
embed some small pipelines helping the user to automatize such workflow, if attributes non-accessible from the software object’s schema reveal to be important ones. Because specific sets of attributes might need to be re-applied onto a multitude of different objects, the user should be able to save different schema’s versions embedding all previously checked/unchecked properties/attributes in order to re-apply them later onto other objects. Therefore, the UI/UX should also contain a “Skeleton” section where could be saved the different produced skeleton versions and load the saved ones to SchemaBuilder. Finally, we must also consider the scenario where the user wants to embed its own custom schemas, classes and subclasses, containing at the leaves of the data structure geometrical, software specific objects. Therefore, the SchemaBuilder interface should also allow the user to load its own custom objects. The SpeckleAbstract class is allowing to do so (along with the SpeckleCore Converter), by serializing and deserializing standard .NET classes. Once the user has finished to build his object with SchemaBuilder, he or she could choose between two options: either sending it directly to the database or adding it to a queuing system, another part of the UI where nested objects would be fully saved. This would allow the user to save “Object A” on the side while building “Object B”, before nesting “Object A” within “Object B”. Finally, the SchemaBuilder’s interface constantly serializes the customized object into a UserDictionary or a JSON string that can be interpreted by the Speckle’s local converter and database, from which can be retrieved any stored custom object by external trades.

4.2.2 SchemaBuilder’s Tree Interface

To navigate across the schema, the user can use the tree interface that keeps track of the parent-child relationships between the objects. UI-UX features have been prototyped, such as highlighting the object’s geometry while hovering its location within the tree, folding/unfolding branches, and pairing the objects

Figure 8. A Grasshopper definition demonstrating a digital design workflow between data generated by Design-to-Production within Rhino3d (right) and shared through a common schema via Speckle to Buro Happold, from which can be retrieved just the necessary information to run a structural analysis within Robot (right) using their BHoM (Bureau Habitat Object Model) interface.
5 SHARING SCHEMAS THROUGH COMMON DESCRIPTIONS

We have described in the previous sections different prototypes, methods and workflows to visualize, cross-reference, build, explore and share schemas online. All those methods could enable more transparency in the design process and therefore improve interoperability between the different trades working together on a same architectural project. In collaboration with Design-to-Production and Buro Happold (an international engineering practice partly based in London), a last experiment has been developed in order to assess how far we can enhance communication processes between different companies through schema-based workflows, using as an example a specific timber assembly. A common schema has been shared between the two practices that are working towards different ends, Design-to-Production generating full geometrical description of each architectural component and Buro Happold focusing on obtaining precise structural analysis results. The retained common schema has been discussed by both trades, which agreed on the minimum information necessary to generate both of their respective data. This scenario enables a bi-directional workflow from which data can be transferred seamlessly between the respective working environments of the two companies (Fig. 8).

6 CONCLUSION

This paper presented both a state of the art in managing modelling processes of large scale and complex architectural projects, as well as different methods aiming at improving contemporary workflows in the AEC industry. Even though the approach described here only focused on the hierarchical relationships between objects, we believe that parallel solutions could also be developed and applied in the future on the time scale and versioning aspect of the project (e.g. keeping track of data transfer between the trades throughout the whole timeline). All the experiments and workflows described in this paper do not constitute a single piece of software, but aim at presenting a holistic approach, or paradigm, within which it could be possible to both build, query, visualize and explore models in more meaningful ways.

7 ACKNOWLEDGEMENT

This research is undertaken as part of InnoChain. This project has received funding from the European Union’s Horizon 2020 research and innovation program under the Marie-Sklodowska-Curie grant agreement No 642877. We would also like to acknowledge the support of Design-to-Production, Buro Happold and McNeel, and thank all of their involved consultants for their insightful comments and advices for the current and future development of this research.

8 REFERENCES

Scheurer, F. 2012. From Thinking to Modeling to Building, Digital Workflows in Architecture: Design-Assembly-Industry, 110-131. 12.
Van Der Heijden, R., E. Levelle, and M. Reise. 2015. Parametric Building Information Generation for Design and Construction.
Speckle (https://speckle.works/) is an extensible Design & AEC data communication protocol and platform, initiated and developed by Dimitrie Stefanescu, Marie Curie Fellow within the international Innochain Training Network.