TRANSPARENCY OF DESIGN AUTOMATION SYSTEMS USING VISUAL PROGRAMMING – WITHIN THE MECHANICAL MANUFACTURING INDUSTRY

Heikkinen, Tim

Jönköping University

ABSTRACT

One challenge with design automation is system transparency with adjustable granularity because of the many different forms of representation from multiple disciplines. Previous research has focused on visualization through the generation of graphs, packaging into electronic books, and model highlighting. The research presented in this paper focuses instead on a visual programming approach, commonly applied in the building industry, where design assets and external references are wrapped into visual components and managed on a canvas with information input/output relations displayed. This entails additional documentation efforts, but the visualization is arguably more useful as groups and levels of granularity are adjusted by the engineers themselves as a part of the development work. To explore visual programming and its potential benefits as a way of enabling transparency with adjustable granularity of DA systems within mechanical manufacturing industry, an existing textual design automation system was transformed into a visual one using Grasshopper® (a visual programming environment) and discussed with respect to DA system transparency, feature-based CAD, and DA system development.

Keywords: Design automation, Visual programming, Knowledge management, Visualisation, Design for Additive Manufacturing (DfAM)

Contact:
Heikkinen, Tim
Jönköping University
Industrial Product Development, Production and Design
Sweden
tim.heikkinen@ju.se

Cite this article: Heikkinen, T. (2021) 'Transparency of Design Automation Systems Using Visual Programming – within the Mechanical Manufacturing Industry', in *Proceedings of the International Conference on Engineering Design (ICED21),* Gothenburg, Sweden, 16-20 August 2021. DOI:10.1017/pds.2021.586

1 INTRODUCTION

Design Automation (DA) can be seen as a knowledge-management approach within engineering design where knowledge is codified and stored in order to promote its utilization (McMahon et al., 2004) [1]. DA is, in other words, mainly concerned with representing the 'how' of design tasks within design processes in general ways so that it can be re-used. This includes simple information handling (e.g. moving information from one system to another) as well as knowledge processing (e.g. using information to answer a question using a set of rules) (Cederfeldt and Elgh, 2005).

The specific problem studied here is one of transparency with adjustable granularity of DA systems to support developers, users, and managers. Historically, DA systems have been developed as 'black-box' solutions where meaning, context, and design intent are hidden or parts of non-associative formal and informal models (Verhagen et al., 2012). Transparency is an important aspect for managing DA systems in general (Hjertberg et al., 2018), as well as during development and use with respect to system relations (Heikkinen, et al., 2020). DA systems can be comprised of several different representation formats, be it spreadsheets, CAD models, scripts, databases, text-files, etc. and especially when multiple disciplines are combined it becomes challenging to get an overview of the system constituents and their relations.

Prior research within mechanical engineering has looked into generating graphs, using both manually added tags and an understanding of the underlying semantics (Hjertberg et al., 2018; Johansson et al., 2018), integration into electronic books and Howtomation® (Johansson and Poorkiany, 2019), and model highlighting (Poorkiany et al., 2016). In the study presented here another approach is examined, where visual programming is utilized to enable transparency with respect to information input/output-relations on adjustable levels of detail. In terms of a research question:

RQ: How can visual programming be used to improve DA-system transparency?

This is examined by adopting a visual programming approach within an existing textual DA system and comparing the differences. The existing textual DA system enabled so called Lattice-based Structural Topology Optimization (LSTO) which included modelling lattice structures with a wire-frame representation in a feature-based CAD software, performing Topology Optimization (TO) in an external analysis tool, and FE-model representation in a CAD-integrated pre-processor (further details below).

The rest of this paper starts by introducing related work, followed by a more detailed description of the textual DA system, its transformation using visual programming, and discussion of the results. Finally, the research question is addressed in conclusions and future work are presented.

2 RELATED WORK

Transparency within DA systems have been discussed and enabled in different ways within mechanical manufacturing and building industries. The related work which this study builds upon and further develops is therefore separately introduced below and concludes with a clarification of how the concepts introduced relate to each other.

2.1 Mechanical manufacturing industry

Hjertberg et. al. (Hjertberg et al., 2018) studied DA system transparency with adjustable granularity to support system maintenance and suggested an approach where dependencies are manually added or automatically retrieved and saved externally in XML-files. Using the XML-files, graphs can then be generated and used to support in searching and understanding how changes can propagate. Each documented dependency should also include information about its author, purpose, how it works, demands, etc. to further support maintenance. An important contribution is adjustable granularity, where dependencies between system constituents can be visualized on different levels, both between documents as well as within. The same approach is later extended and evaluated further by incorporating dependencies between more types of documents within a platform context Stolt et. al. (Stolt and Elgh, 2019).

[1] Considering Knowledge-based Engineering as a specific type of DA, where knowledge-based systems theory is applied.

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

Johansson et. al. (Johansson et al., 2018) also generates graphs from DA systems, or engineering knowledge represented by spreadsheets, CAD-models, and knowledge bases, with the specific purpose of supporting a connectivism work approach. By exploring the underlying semantics within each model type, nine relations within and between CAD, spreadsheets (configurations specification), and knowledge bases can be generated automatically. These graphs can then be used to e.g., find input-parameters, extract product variant masters (kind-of and part-of relations are extracted from the CAD models), and better understand the system logic. Further Poorkiany et. al. present an approach of representing relations to design rationale, tested on a automation system (Poorkiany et al., 2016). In contrast to essentially generating a map (as a graph for instance) the approach relies on listening to selection-events which then show the related design rationale within the specific software context operated. So, if an element in a spreadsheet is selected which has documented design rationale then this is shown or even highlighted in the design rationale model (e.g., geometric element in CAD). Finally, Johansson et. al. present an approach where Knowledge Objects (KO's), defined as "bundles of human comprehensible knowledge representation and computer routines for the automated application of the represented knowledge" (Johansson and Elgh, 2019), are integrated into electronic books (EPUB's) and how different stakeholder benefits from this. A tool which utilizes a visual representation called Howtomation® works as an "interactive process" view where KO's and its parameters are tested, set-up, and documented. The electronic books and Howtomation® add transparency on different levels of granularity but also enable rationale behind different elements to be integrated. Electronic books make the DA system transparent to anyone, including managers and users, and Howtomation® makes it transparent to developers, KO users and knowledge acquisitionists. Sandberg et. al. (Sandberg et al., 2017) present a knowledge-based master-model approach where multiple disciplines work with one common product definition which can be used to generate product variants and its associate analyses models, enabling multidisciplinary optimization. It is introduced here because it was first introduced within the mechanical manufacturing industry but has since then also been studied within the building industry, making use of a visual programming approach (discussed further in the next sub-section).

2.2 Building industry

Within the building industry, Visual Programming Languages (VPL's) have been used as the middleware or 'glue' between different models and tools (Negendahl, 2015; Sandberg et al., 2019) to enable a so called distributed integration approach (Negendahl, 2015).

Negendahl (Negendahl, 2015) discusses the integration between geometric modelling and analysis from both the human and model domains. Three model integration approaches are presented: combined model method (one main tool controlling sub-tools), central model method (each tool shares one common model), and distributed model method (tools are integrated using a middleware). The distributed model method is described as a response to the centralized model method where a top-down control and one-way operation is advocated. In a distributed model method, there is a middleware component which instead enables communication between tools. If a VPL is used as middleware it is called an integrated dynamic model which is becoming a prevailing method for integrating geometric modelling and analysis. The usability and encouraging community associated with VPL, in contrast to lower-level programming languages, is presented as an important aspect of enabling customization which also supports multi-disciplinary work (both practitioners within geometric modelling and analysis can operate the middleware).

Sandberg et. al. (Sandberg et al., 2019) introduce a BIM-based master-model approach where product definitions constituents are integrated using VPL wherever the commercial tools lack functionality. In this sense it is an example of an integrated dynamic model where standardized IFC is capitalized on and multidisciplinary optimization enabled. The master-model approach is described as similar to BIM in the sense that there should be one central model, BIM has one central model for created information and the master-model for "the definition of the product" (Sandberg et al., 2016). Kubicki et. al. (Kubicki et al., 2018) present and discuss VPL as a way of supporting usability of a master-model approach.

Singer et. al. (Singer and Borrmann, 2015) present a knowledge-based engineering approach for infrastructure design utilizing a multi-model generator approach (as described in (La Rocca and Van Tooren, 2010)) in combination with visual programming and an abstraction layer concept. The multi-model generator is a master-model where the product definition is composed of high-level primitives and the middleware is composed of capability modules, together they define the building model and

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

actions during the synthesis and analysis of infrastructures. Both high-level primitives and capability modules are implemented as visual components within a VPL, motivated by its ease of use, explanatory power, decision transparency and dependency tracking. The abstraction layer concept is synonymous to the adjustable granularity discussed here, access control on layers is also introduced.
Runberger et. al. (Runberger and Magnusson, 2015) introduce, among other things, a graphic standard to support legibility and reuse of VPL, specifically within Grasshopper®. It is comprised of a modularization strategy where input, control, monitor, visibility, bake, and design are grouped, annotated, and coloured.

2.3 Clarification of concepts

As a clarification, some of the concepts introduced in the related work section are related to each other here in combination with a new one called design asset (see Figure 1). Design asset (or resource) is a general concept for an object which can be used to support a design process, whether that process includes automated operations or not. It is a common denominator between Master model, Multi model generator, and Knowledge object. Master models, such as the Multi model generator, are a type of Design asset which include a Middleware and its associate Product definition. Each Product definition consists of additional Design asset's which could be of type Knowledge object.

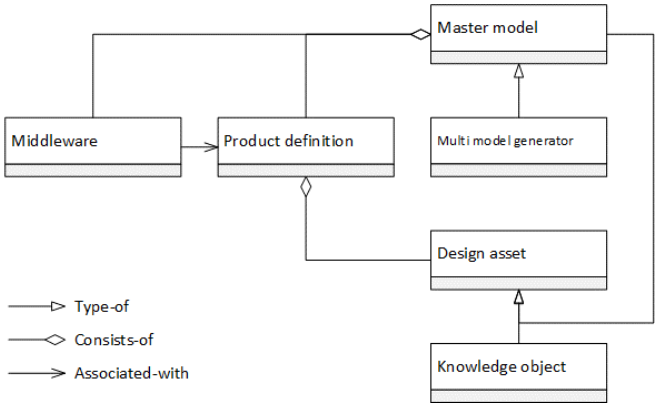

Figure 1. Relations between some of the concepts introduced (UML class notation).

3 TEXTUAL DA SYSTEM - CONTEXT AND SYSTEM CONSTITUENTS

The textual DA system was developed as a part of a case-study (presented here (Heikkinen, et al., 2020)) where the initial goal was to develop an ability to integrate TO within an existing multidisciplinary DA system used to explore design spaces and construct meta-models to learn about trends and trade-offs. Exploring design spaces involve the fully automatic synthesis of variant designs, pre-processing, and post-processing which means that no degree of manual work on individual variants can be allowed (it would be too time-consuming). Since the utilization of traditional TO usually requires some interpretation and manual modelling before getting a proper B-Rep CAD model which all downstream disciplines required, LSTO was researched and developed. Generally, these multidisciplinary automation systems start by defining the design space to explore in terms of design requirements and parameters, including their limits. These are used to generate a design of experiment (DoE) which specifies the designs which will need to be generated and analysed to get an understanding of their relations (using different meta models). From the project specification a system instance is then set up by choosing or developing different design assets, such as scripts for CAD-model configuration and automated meshing as well as template parametric CAD-models and load-cases.
The textual DA system is a master-model where one heterogeneous set of design assets (see Table 1) are integrated using a middleware constructed by using software API's in a compiled .NET application (see activity overview in Figure 2). In the end a beam-based lattice structure is modelled and configured according to the relative density results from TO (see example in Figure 5). It starts by reading the DoE-setup from a spreadsheet either by using custom named ranges or fixed positions. Using the DoE-setup, variants are generated by retrieving references to CAD parameters of different type (e.g. numeric or string) from a baseline CAD-model. The parameter values are then altered accordingly, the model updated, and finally the geometry exported as STEP-files for downstream use.

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

Next the programmed features (see Figure 3) on the baseline CAD-model are read and together with an associate STEP-file used to configure a script which in turn is used to generate and run a TO. Once the TO is finished the resulting relative density is used to configure the beam densities which can then be used to either create a solid model (mainly for printing purposes and visual inspection, see Figure 8) or FE-model (using 1D-beam elements for efficient simulation, see Figure 3) of the lattice-structure.

Table 1. DA system design assets.

Design Asset	**Description**
Design of Experiment setup	Spreadsheet with the parameter names and value for a given study.
Baseline CAD model	Starting geometric model used to generate geometric variants.
Baseline FE model	Starting FE model used to generate analysis variants.
Programmed features	Custom CAD-features used to define the TO-specification (e.g., design space, loads, and boundary conditions) and lattice-specification (e.g., number of lattice-cells, dimension limits).
Template TO- script	A script which has been modified for easy configuration of new TO-scripts.

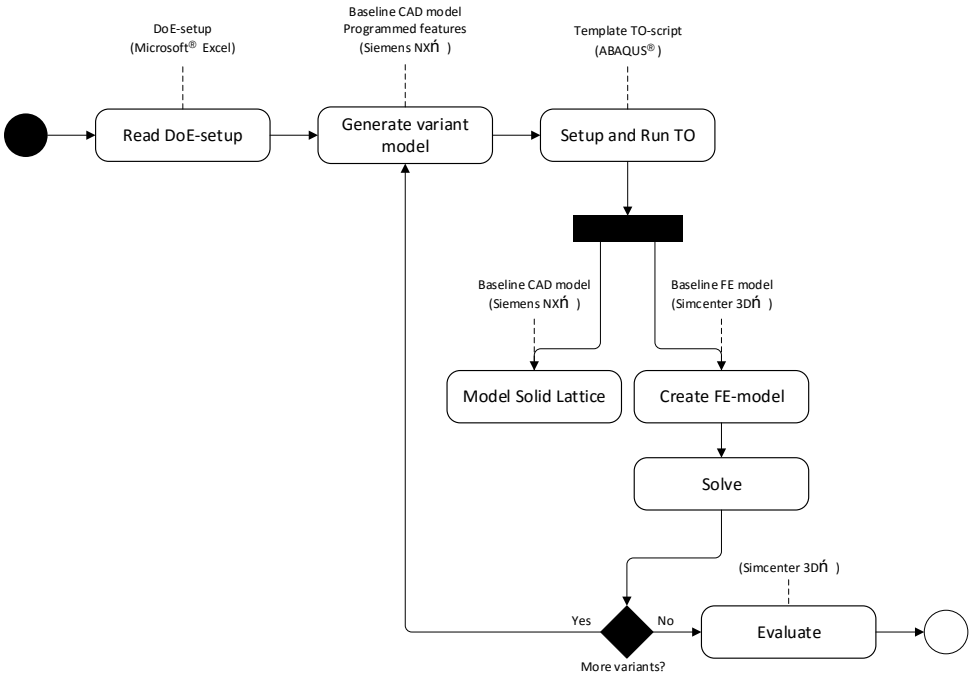

Figure 2. Middleware process, showing the main activities and supporting design assets as well as software tools in each (UML activity diagram notation).

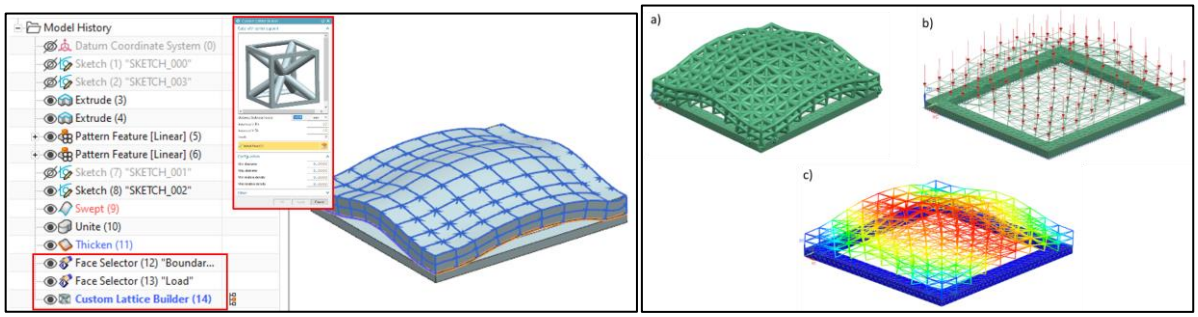

Figure 3. (Left) Example programmed features (boundary-condition, load, and lattice-structure) in feature tree (Model History) and one associated GUI. (Right) FE-model representation of lattice-structure in Simcenter 3D™: a) 1D-beams with custom sections, b) load and boundary condition, and c) displacement distribution results.

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

4 VISUAL REPRESENTATION - USING GRASSHOPPER®

To convert the textual DA system described above into a visual one the middleware (.NET application) was modularized in different ways. Figuring out the size that was suitable for the modules was difficult, but for the purpose of this study the smallest possible within the time limits given was used to really make use of the VPL as much as possible. Some of the smallest modules were wrapped into visual script components, the language support in Grasshopper® are .NET specific, so C#-script, VB-script, and IronPython, see illustration in Figure 4.

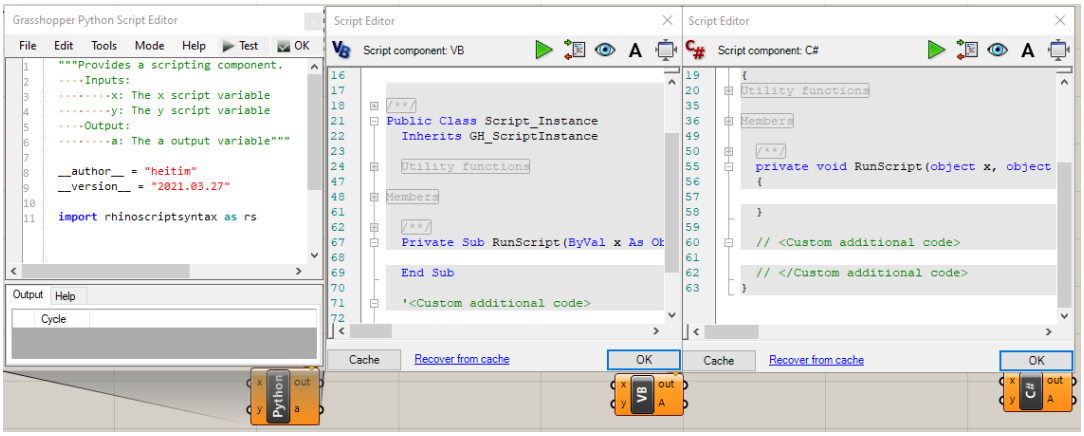

Figure 4. Wrappers (script-components) in Grasshopper®, from left to right; VB-Script, C#Script, and finally IronPython.

The design assets were wrapped as visual components (defining input and output) with references using the file-paths since the whole prototype worked on one computer and its local files. Strings can be modelled in Grasshopper® using different components, but the 'panel' was used in this case. Using the panel, one could follow the wiring and see which components had a connection, why they were used, and how they were connected. Effectively supporting system transparency on one level of granularity.

Each visual component was named properly, connected by dragging wires between the inputs and outputs, grouped into appropriate sizes, documented with text, result visualization added, and color schemes used to easily identify inputs, portions with only pre-defined visual components, portions connected to external software, and portions with textual (script components) and visual representations but with no connection to external software. See the most granular overview in Figure 5.

'Clustering' was used to enable adjustable granularity and made collections of components shareable and maintained (one definition which could be used in several places). Commonly used components could also be saved as templates and added to the toolbar for easy instantiation (see Figure 6).

To make use of object-orientation to some extent the .NET application was not removed completely. The classes for LSTO specifications which were also shared by the programmed feature, were referenced, and utilized in the process as well. This reduced the number of input/output relations required in the process but also hid some of the methods to the users.

Two changes were also made in the visual representation; a metamodel (non-linear regression using Lunchbox® add-on) was used to get relative densities at each beam location from the TO-results and the solid model representation was made possible in Rhinoceros 3D® instead of Siemens NX™. Both changes were made for efficiency reasons, the meta-model greatly reduced the search for relative densities and since Grasshopper® is tightly integrated with Rhinoceros 3D® and supports .STL-file export for printing purposes this was a more efficient process as well (see Figure 6).

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

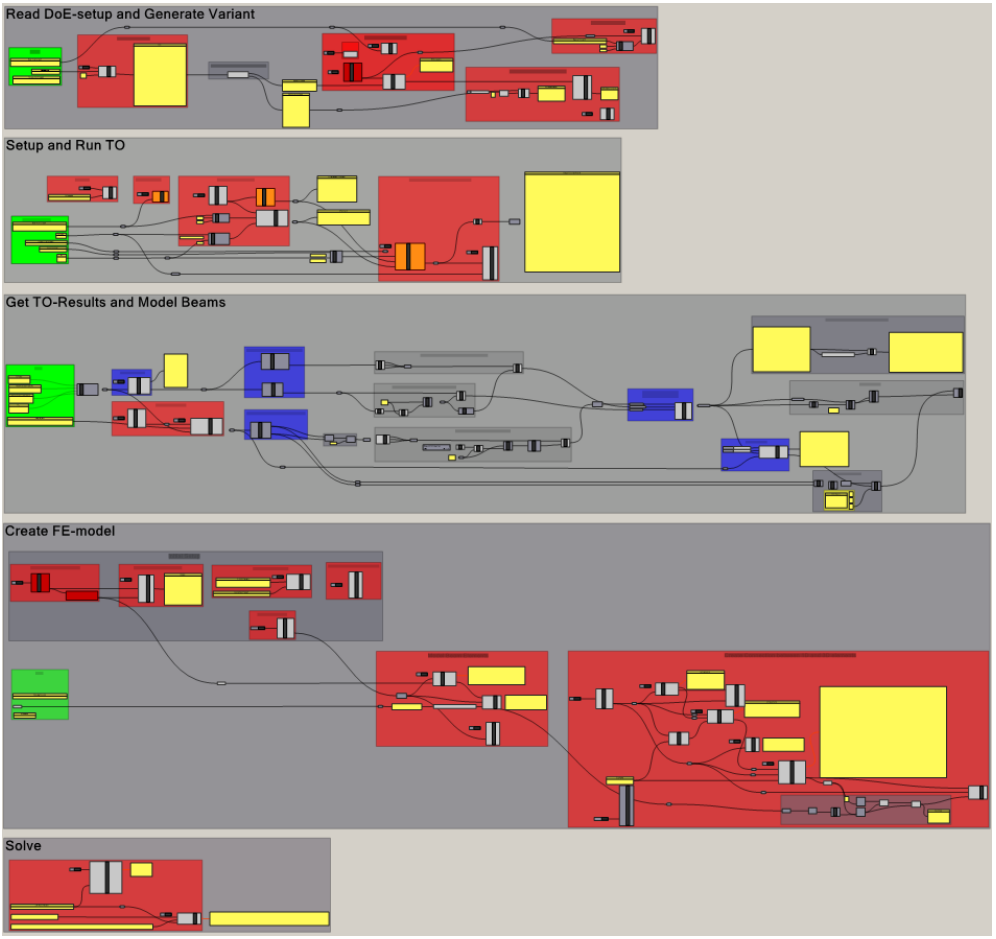

Figure 5. Showing grouped and connected visual components, colouring is used to identify groups with input (green), using external software (red), using textual representations without external software (blue), and only visual components (grey).

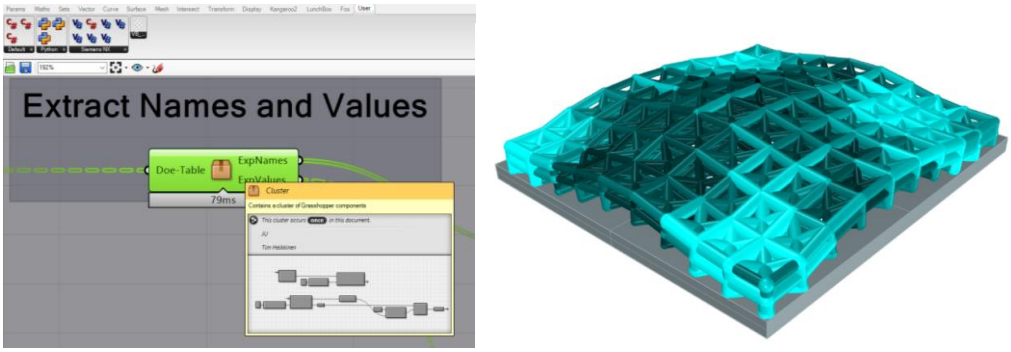

Figure 6. (Left) Enabled adjustable granularity (cluster) and commonly used components as templates in toolbar, and (right) example solid lattice-structure for printing.

5 DISCUSSION

Visual programming was explored as a way of enabling DA system transparency, this is therefore first discussed below, followed by discussions about the relations between visual programming and feature-based CAD as well as DA system development.

5.1 Visual programming and DA system transparency

Visual programming essentially gives a data-driven model which enforces at minimum a data input/output level of transparency between design assets in a specific process context. It can be used to better understand the design assets purpose within the system which could also be used to assess how changes propagate. It is possible to further divide operations between design assets, to enable

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

transparency on a more detailed level, as well as group and cluster operations, to enable transparency on a more general level (Davis et al., 2011). Documentation using colouring and strings can be used to differentiate between operations, e.g. system input/output relations as well as operations using internal and/or external software methods or as (Runberger and Magnusson, 2015) by input, control, monitor, visibility, bake, and design.

In comparison with existing approaches for DA system transparency, visual programming requires manual work in contrast to an automatic graph generation (Hjertberg et al., 2018; Johansson et al., 2018). But the outline, grouping, and documentation of visual components and relations (or nodes and edges in a graph) are arguably an important aspect of interpreting the visual representation and might be difficult in a generative approach. Relations within specific types of software models are not visualized using VPL (as in (Johansson et al., 2018)), but many software tools support the visualization of relations, e.g. feature-based CAD systems such as CATIA™ and Siemens NX™ have graph visualization capabilities and spreadsheets modelled using Microsoft® Excel can also be visualized. It is arguably the middleware with relations between software specific models, text-files, databases, code etc. which is missing but it would be interesting to integrate these within the visual programming environment as well. Finally, visual programming visualises relations within one process context (one use-case of design assets) and not non-process related relations such as documentation for instance.

Listening to selection-events and showing or highlighting related information (bi-directionally) in the software context where the engineer is working as introduced by Poorkiany et. al. (Poorkiany et al., 2016) is seen as an extension of the graph generation approach. Ones the graph (model entities and their relations) is known the design rationale system can listen to selection-events and show or highlight connections. It is an approach which uses the graph to pull information about related information to the user in his current context on a detailed level. Since it relies on a graph it needs a way of either automatically interpreting this or having someone creating it. A visual programming approach supports the capture of the dataflow through execution of the process. If visual programming is combined with tagging of nodes and edges for the graph they could be combined for improved transparency.

Using electronic books (EPUB's) and Howtomation® in combination with KO's as presented by Johansson et. al. (Johansson and Poorkiany, 2019) allows for system transparency with respect to rationale and dataflow. The Howtomation® tool resembles a visual programming environment where input and output from different visual components (KO's in Howtomation®) are modelled. Colouring is used to indicate "executed", "triggered" (ready to be executed), and "unreachable" (cannot be executed) KO instances. Adding the possibility to group and cluster visual components would enable adjustable granularity to the same level as with VPL's such as Grasshoppers®.

A multi-model generator approach with the abstraction layer concept presented by Singer et. al. (Singer and Borrmann, 2015) is the one which most resembles the final visual representation presented here. The main difference is the utilization of programmed features instead of high-level primitives. However, there might only be an implementation difference between these two concepts. High-level primitives as presented in (La Rocca and Van Tooren, 2010) are introduced as a way of enabling designers to think in terms of solutions to functional requirements (high-level design objects) in contrast to geometric primitives such as points, curves, surfaces, and solids. Features (proprietary or programmed) have similar objectives but are tightly integrated to the CAD-software (discussed in more detail below in Visual programming and Feature-based CAD).

5.2 Visual programming and Feature-based CAD

VPL's have reported to increase the use of scripting within architectural design for "geometric automation" (e.g. parametric design) and integration of analysis tools, enabling generative modelling methods and optimization (Negendahl, 2015). An interesting question is if a VPL, in combination with a feature-based CAD approach, could increase the use of automation within manufacturing industries. It arguably supports transparency with adjustable granularity (as discussed in Section Visual Programming and DA system transparency above).

Features in CAD were introduced around the 1970s and represent "the engineering meaning or significance of the geometry of a part or assembly." (Shah and Mäntylä, 1995). Feature models can be thought of as extensions to solid modelling, tackling their inability or inefficiency to (1) model non-geometric information, (2) propagate design changes on a part level (in contrast to geometry), and (3)

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

integrate with downstream applications. Feature properties include generic shape, parameters, methods (for orientation, geometric construction procedures, recognition algorithm, etc.), constraints (relating dimensions, location, or orientation), tolerances, and non-geometric attributes such as material properties, applicable machine operations, tool and fixture information, and procedures. They can increase a geometric model's meaning and support the introduction of more elaborate systems such as knowledge-based systems. Depending on the engineering task different feature views are relevant, design maybe thinks about the parts used to model the geometry, structural analysis considers contact-, load-, and boundary-surfaces and only the level of detail necessary to simulate some behaviour, and sheet-metal forming deal with bends and holes to develop tools and manufacturing procedures.
Using visual programming as a middleware between feature-based CAD and analysis tools was interesting, especially since it was tightly integrated with its own geometric modelling capabilities from Rhinoceros 3d®. This made it possible to visualize results from running the visual script components and effectively quality check and debug issues. Programmed features made it easy to read necessary analysis information, including TO-specific and FE-model representation, in an object-oriented way.
The feature-tree (or Model History in Figure 3) could be a collection of a specific type of visual components, structured according to 'part-of' relations, with many data-relations hidden (e.g., parameters, equations, and constraints).

5.3 Visual programming and DA system development

Some additional interesting preliminary outcomes from this study was (1) the system development advantages in terms of emphasizing 'function' first, (2) continuous component-level solving and caching of data, and (3) minimized cognitive load.
(1) When programming in the visual environment you start with defining data input and output explicitly. Each component needs this interface, and it is always clear how this relates to existing elements of the system. The data-driven model is forced and becomes a direct output of the development itself. This changes the dynamic of programming to a more 'functional' approach where focus is on first defining what output is sought after and then what input such a procedure would need.
(2) Another interesting thing about VPL's is the component-level (or module (Davis et al., 2011)) debugging and caching of data. This made it efficient to model the next component because ones a component has successfully run, those results can be used repeatedly. This could of course be a cause of concern when scaling this approach, however.
(3) A final preliminary outcome was the minimized cognitive load. This claim is supported by a theory of cognition called dual-coding (Paivio, 1979) which states that information can be represented and processed both verbally and visually separately. If both are used there have been studies showing different benefits with respect to information processing capacity, identifying patterns, memory, learning, problem solving, reasoning, and comprehending verbal information (Lindlöf, 2014). Lindlöf (Lindlöf, 2014) studied this within product development as an effective way of supporting communication by reducing ambiguities. The way in which this reveals itself when developing visual DA systems is a perceived reduced start-up and effective communication with colleagues (perceived, not measured in this study). The visual representation made in easier to conclude what had been done previously and what needed to be done because the functional elements (intended data processing) were there and just needed to be enabled.

6 CONCLUSIONS

Looking back at the research question it can be concluded that visual programming enforces a data-driven model where input/output relations between design assets (or knowledge representations) within implementations of DA systems are available. With grouping, clustering, textual and imagery annotation, as well as color-coding there are also possibilities of modelling more meaning, context specifics, and design intent within one integrated model. It could be a step toward integration of informal and formal models. From the study presented here it has been shown that it is possible to transform some textual DA systems with visual programming within a mechanical industry context and arguably improve its transparency. Further research looking into the integration between feature-based CAD and visual programming as well as visual programming's effect of cognitive load should be conducted to potentially increase the level of automation within the mechanical industry as it has been reported to do within the building industry.

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press

REFERENCES

Cederfeldt, M. and Elgh, F. (2005), “Design Automation in SMEs-Current State, Potential, Need and Requirements”, ICED 05: 15th International Conference on Engineering Design: Engineering Design and the Global Economy, pp. 1–15.

Davis, D., Burry, J. and Burry, M. (2011), “Understanding visual scripts: Improving collaboration through modular programming”, International Journal of Architectural Computing, Vol. 9 No. 4, pp. 361–375. https://doi.org/10.1260/1478-0771.9.4.361.

Heikkinen, T., Johansson, J. and Elgh, F. (2020), “Multidisciplinary design automation – a conceptual framework for working with product model extensions”, International Journal of Agile Systems and Management, Vol. 13 No. 1, pp. 28–47. https://doi.org/10.1504/IJASM.2020.105866.

Heikkinen, T., Stolt, R. and Elgh, F. (2020), “Incorporating Design for Additive Manufacturing in Multidisciplinary Design Automation – Challenges Identified”, Computer-Aided Design and Application, Vol. 17 No. 5, pp. 936–947. https://doi.org/10.14733/cadaps.2020.936-947.

Hjertberg, T., Stolt, R., Elgh, F. and Stolt, R. (2018), “A tool for obtaining transparency and traceability in heterogeneous design automation environments”, Computer-Aided Design and Applications, Vol. 4360 No. 14, pp. 488–500. 10.1080/16864360.2017.1419637.

Johansson, J., Contero, M., Company, P. and Elgh, F. (2018), “Supporting connectivism in knowledge based engineering with graph theory, filtering techniques and model quality assurance”, Advanced Engineering Informatics, Elsevier, Vol. 38, pp. 252–263. https://doi.org/10.1016/j.aei.2018.07.005.

Johansson, J. and Elgh, F. (2019), "Knowledge Objects Enable Mass Individualization", Computational Methods in Applied Sciences, Vol. 49. https://doi.org/10.1007/978-3-319-89890-2_5.

Johansson, J. and Poorkiany, M. (2019), “Integrating Knowledge Objects and e-Books to Support Six Roles in the Design Automation Life-Cycle”, Proceedings of the 26th ISTE International Conference on Transdisciplinary Engineering. https://doi.org/10.1039/C9PY90093D

Kubicki, S., Mukkavaara, J. and Sandberg, M. (2018), “A master model approach for design and analysis of roof trusses”, International Symposium on Automation and Robotics in Construction and International AEC/FM Hackathon: The Future of Building Things, No. December. https://doi.org/10.22260/ISARC2018/0046.

Lindlöf, L. (2014), Visual Management – on Communication in Product Development Organizations Department of Technology Management and Economics, PhD Thesis, Chalmers University of Technology.

McMahon, C., Lowe, A. and Culley, S. (2004), “Knowledge management in engineering design: Personalization and codification”, Journal of Engineering Design, Vol. 15 No. 4, pp. 307–325. https://doi.org/10.1080/09544820410001697154.

Negendahl, K. (2015), “Building performance simulation in the early design stage: An introduction to integrated dynamic models”, Automation in Construction, Elsevier B.V., Vol. 54, pp. 39–53. https://doi.org/10.1016/j.autcon.2015.03.002.

Paivio, A. (1979), Imagery and Verbal Processes.

Poorkiany, M., Johansson, J. and Elgh, F. (2016), “Capturing, structuring and accessing design rationale in integrated product design and manufacturing processes”, Advanced Engineering Informatics, Elsevier Ltd, Vol. 30 No. 3, pp. 522–536. https://doi.org/10.1016/j.aei.2016.06.004.

La Rocca, G. and Van Tooren, M.J.L. (2010), “Knowledge-based engineering to support aircraft multidisciplinary design and optimization”, Proceedings of the Institution of Mechanical Engineers, Vol. 224 No. 9, pp. 1041–1055. https://doi.org/10.1243/09544100JAERO592.

Runberger, J. and Magnusson, F. (2015), “Harnessing the Informal Processes Around the Computational Design Model”, Modelling Behaviour, pp. 329–339. https://doi.org/10.1007/978-3-319-24208-8_28.

Sandberg, M., Gerth, R., Lu, W., Jansson, G., Mukkavaara, J. and Olofsson, T. (2016), “Design automation in construction – an overview”, Proc. of the 33rd CIB W78 Conference, No. Stokes 2001, p. 9.

Sandberg, M., Mukkavaara, J., Shadram, F. and Olofsson, T. (2019), “Multidisciplinary optimization of life-cycle energy and cost using a BIM-based master model”, Sustainability, Vol. 11 No. 2. https://doi.org/10.3390/su11010286.

Sandberg, M., Tyapin, I., Kokkolaras, M., Lundbladh, A. and Isaksson, O. (2017), “A knowledge-based master model approach exemplified with jet engine structural design”, Computers in Industry, Elsevier B.V., Vol. 85, pp. 31–38. https://doi.org/10.1016/j.compind.2016.12.003.

Shah, J.J. and Mäntylä, M. (1995), Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications, Wiley, New York.

Singer, D. and Borrmann, A. (2015), “A novel knowledge-based engineering approach for infrastructure design”, Civil-Comp Proceedings, Vol. 109 No. December. https://doi.org/10.4203/ccp.109.17.

Stolt, R. and Elgh, F. (2019), “Enhancing traceability in heterogeneous design platforms”, International Journal of Product Lifecycle Management, Vol. 12 No. 1, pp. 62–80. https://doi.org/10.1504/IJPLM.2019.104382.

Verhagen, W.J.C., Bermell-Garcia, P., Van Dijk, R.E.C. and Curran, R. (2012), “A critical review of Knowledge-Based Engineering: An identification of research challenges”, Advanced Engineering Informatics, Elsevier Ltd, Vol. 26 No. 1, pp. 5–15. https://doi.org/10.1016/j.aei.2011.06.004.

https://doi.org/10.1017/pds.2021.586 Published online by Cambridge University Press