Semio: A meta-model for formalizing reusable parametric semantic architectural concepts for non-standard collaborative testable designing

Saluz U., Geyer P.
Leibniz University Hannover, Germany
u.saluz@iek.uni-hannover.de

Abstract. Every architectural design is the result of many decisions. The more a design can be organized into loose coupled, robust, and reusable parts, the faster, flexible, and reliable is the design and decision-making process. The earliest organization of a design is an architectural concept. The objects of such are high-level, non-standard and change quickly. Various platforms for parametric definition of objects exist but they are not generally interoperable as they share no interfaces. This paper proposes a meta-model that allows to organize a design into non-standard objects by capturing semantics as a graph and compute it. Thus, enabling descriptive instead of imperative parameterization of architectural concepts. Such formalization offers new possibilities regarding authoring, testing, collaboration, and reuse. Further, computational tools (like version control, unit and integration testing, graph algorithms, graph rewriting, recommender systems, graph neural networks) become applicable. An interoperable prototype has been implemented and a use-case has been provided.

1. Introduction

Early decisions in planning of construction projects have the largest impact on the ecological, economical and social performance (Bragança et al., 2014). Despite the effectiveness of rule of thumbs in early design, exploring different architectural designs (AD) helps to find better solutions (Iano and Allen, 2022). An AD is mostly about organization of virtual objects for initiating processes in reality. It is expressed in a conservative sense through *iconic* models such as physical models, pictures and drawings, but in a wider sense it also includes *analogue* models such as a simulation model and *symbolic* models such as mathematical models (Roberts, P.H. et al., 2019). While finding form is the result of *concrete* modelling, it is intrinsically linked to *cognitive* modelling (Akalin and Sezal, 2009). Such a cognitive model can be both abstract through concepts and virtually concrete in shape. An AD is both geometric and semantic.

Developing an AD, designing, happens in multitude of ways (Smith and Smith, 2014). Yet, the process rarely consists only of individual, unlinked and unstructured decisions. Often architects design with a guiding mental structure: an architectural concept (AC). Despite major architectural theories which provide mostly analytic and proactive methods for design space exploration (DSE) (Johnson, 1994), there is unsurprisingly no general definition of what an AC is, but instead there are many personal definitions (Eilouti, 2018; Vesnic, 2017). The development of ACs is normally a fast, minimal rule based and inaccurate process that is supported through formalization free tools such as sketching (Goldschmidt, 2017). This is a large distinction to a special purpose computation such as energy simulation, where standardized processes (U.S. Green Building Council, 1998) and data models exist ("gbXML," 2009).

In contrast to an AC, an AD has a lot of requirements. Designing is an *ill-defined* problem solving where it is not about finding an optimal solution to a fixed question but finding an acceptable solution to changing questions (Purcell and Gero, 1996). In consequence, it is a non-linear process that not only iteratively increases the requirement levels but can jump back at any time to an earlier state and change decisions from before (Grobman et al., 2010). Such

rollbacks mostly lead to repetitive tasks with high complexity. Assuming two inclined rectangular buildings | __ are merged into a L-shape building after the floorplans have already been separately developed, the chance of needing further adjustments at the intersection corner of the L shaped building is high. On the other hand, it is less likely that changes are needed in the end of the building. Most design knowledge about AD is documented in an implicit way through plans or a Building Information Modelling (BIM) model which does not allow an automatic transfer of partial design knowledge. This requires even small adjustments where processes stay identical but only a change of values can trigger the entire pipeline again. Assuming a façade is moved 10cm more towards the outside in a late planning phase, this most likely won't change the computational method of structural or fire safety calculations but instead it will change certain numbers in the same process. Despite all workflows being almost identical, the whole process must be restarted, containing a lot of manual repetitive work and communication efforts.

The concretization of an AC into an AD is therefore a time-intensive task. Despite BIM having shown to have the potential to increase productivity in planning (Teo et al., 2015), overall productivity in the construction sector has stagnated (*Reinventing construction through a productivity revolution - Full report*, 2017) and the level of digitalization is among the lowest of all sectors (*Digital America: A tale of the haves and have-mores - Full report*, 2015). The purpose of this paper is not to aid in AC development, and it assumes that it has been already found, instead it aims to aid in the concretization of an AC into an AD. Examples of such concretizations are refinement of dimensions of spaces and objects, checking requirements such as programmatic compliance, computing preliminary performance indicators through simulation, creating drawings of lower scale, improving parallel design among different planners.

Computers have a tremendous potential in computing repetitive processes where the structure of a process is not changing. In AD, this requires efforts of formalization of what AD knowledge is and how it can be transferred.

1.1 Related work

Depending on the domain, focuses are more knowledge-oriented about how to structure an AC or more formalization-oriented on how to compute an AD. Both are equally important and need to be brought together if a proposed method is not to be used analytically but proactively for designing. A promising approach for knowledge capturing among different domains are graphs, either with knowledge graphs, such as provided with ("Resource Description Framework (RDF)," 2014), or labelled property graphs (LPG), such as provided with ("Neo4j," 2010). For this purpose, work is related even when not directly concerned with representing ADs but when methods for knowledge formalization are used in the context of graphs.

Graph transformation

When knowledge is represented by a large graph, then partial knowledge is directly represented as a partial graph. The formalism that helps in computing such relationships or even construct a large graph from partial graphs is called graph transformation/rewriting. The application possibilities to use this formalism for domain specific knowledge has already been recognized since long (Rozenberg, 1997; Agrawal et al., 2003). The shortcomings of adoption in the engineering domain remains due to the difficulty of abstraction of knowledge representation in general with the combination of geometric and semantic knowledge in specific, limited software implementations and even fewer learning resources (Kolbeck et al., 2022).

Architecture theory

Architectural theories are mostly not intended to be formal enough to be directly translated into a formal structure. Yet, methods for designing complex things, which require high organization, that were developed by architects, have proven to be effective in providing methods to identify actors and to assign them concepts as also identifying relationships between them. This solves one of the major adoption problems. For this reason, some architectural theories have found wide applications outside of the domain of architecture (Molly Wright Steenson, 2022). A famous example is Alexander, with design patterns (Alexander et al., 1977) that have been very successful in computer science through the adoption of the Gang of Four (GoF) (Gamma, 1995). Patterns are to be understood as a meta-model of an AC. The main intent of patterns is to capture knowledge about modelling of the past and make it accessible for future projects. Despite their success for solving design problems in software architecture, their high level makes them hard to formalize and have so far only found little support inside software (Radermacher, 2000).

Design theory

In design theory, Gero has formalized design and processes with Function-Behaviour-Structure (FBS) (Gero, 1990; Gero and Kannengiesser, 2006). A very important concept is *design prototypes* which are prototypes that a designer can use to adapt in a new design without having to derive a completely new design from scratch. This allows designers to reuse experience from the past and accumulate with time. In the terminology of this paper, they can be seen as a map of AC with an associated AD from which the designer can reuse the AC in a new situation. However, despite the clear formal structure of FBS, when applying it to specific design through the need to conceptualize it, these lines can blur through the suggestive nature of language and the philosophical problem of differences between what is *structural* and what is *intentional* (Dorst and Vermaas, 2005). So far, FBS has not found its way into major software design tools. The idea of a prototype has been integrated in the proposed meta-model.

Procedural modelling

In the family of procedural modelling different computational methods for designing such as shape grammars (Stiny, 1975), L-Systems (Shih, 2020), cellular automaton (Krawczyk, 2002), agent-based systems (Rosenman and Wang, 1999) and visual programming (Burnett and McIntyre, 1995) have been developed, changed and/or applied.

In shape grammars, knowledge is encoded in geometry and decisions are hence rules based on geometry. The major problem is that they are generally not computable due to the problem on how to generally represent geometry (Wortmann and Stouffs, 2018; Stiny, 2006). Further, these rules are often hard to control (Lipp et al., 2019) which is why they have been simplified with set grammars (Wonka et al., 2003). The simplification had great success in Computer Generated Imagery (CGI) in order to model large scale models where exterior is more important than interior (Müller et al., 2006; Parish and Müller, 2001). It has been implemented into major modelling environments ("Esri CityEngine | ArcGIS Desktop," 2008). Despite the possibility to maintain manual modification by identifying rule derivation trees, it has been difficult keep the user out of the loop (Lienhard et al., 2017). To make adoption into AD more approachable, investigation into layout modelling using constraint graphs have started (Para et al., 2021).

L-systems rely on the design being encoded as a text sequence. Partial text sequence are recursively replaced by other sequences. To make a connection between text sequence and AD

required an incredible high level of abstraction. L-Systems is a good formalism for modelling self-similar objects such as plants, which AD is mostly not.

A cellular automaton consists of a set of cells where each cell can have a finite number of states and neighbours. After every generation, a rule set describes the state of next generation of cells. By varying the initial state and evaluating the cellular automaton after a finite amount of time, computations such as computing an AD can be achieved. Cellular automations are good at simulating dynamic systems, whereas ADs are mostly not.

All methods are very expressive in the sense that every AD can be decomposed by them in theory but knowledge but from a knowledge representation perspective, the abstraction needed can be very unintuitive, bulky, and not reusable. Challenges are how to evolve ADs, especially how to handle exceptions and how these can be preserved among variants, reuse partial knowledge, and collaborate with others.

An exception in the family of procedural modelling are visual programming environments such as ("Grasshopper," 2007; "Dynamo," 2023; nortikin, 2023), that are regular programming environments inside of conventional modelling software which offer full control over the underlying modelling environment. They have gained increasing popularity in AD and proven to be powerful to parametrize self-contained objects (Hirschberg et al., 2020). The lack of structure in these tools make them evidently suffer from low scalability, poor interoperability, little reusability and bad collaboration possibilities (Davis and Burry Mark, 2011; Harding and Shepherd, 2017).

BIM authoring tools

More focused on later planning stages, BIM tools try to limit the number of objects with the goal to offer stable features with rich semantics. Abstraction mechanisms for customization like Families in Revit or GDL (Geometric Description Language) in ArchiCAD allow for full customization, controlled nesting, and strong coupling of custom objects. Although the goal of these interfaces is also to produce parametric objects, these interfaces follow a black box philosophy where designers mainly tweak instance parameters and rarely logic. This makes them unsuitable for generative design (Ma et al., 2021). Semantic expressibility through interoperable data-standards is a foundation for the proposed meta-model.

Mechanical engineering

Due to the high amount of functional requirements and high degree of standardization of individual parts, research in mechanical engineers has had a lot of interest in computational design synthesis (CDS) (Chakrabarti et al., 2011; Königseder et al., 2016). A wide range of methods regarding constraint formalization, functional aspects (Ma et al., 2013), assembly representation (Bahubalendruni et al., 2015) through AND/OR-trees (Homem de Mello and Sanderson, 1990), connectivity graphs, bond graphs or other vector based representations, or as port-based design (Liang and Paredis, 2004; Singh and Bettig, 2004) have been developed. The idea of connectivity graphs and assembly through trees which is based on port-based design is later used in the proposed meta-model.

Aerospace engineering

In aerospace engineering the development of new variants is expensive, however, in fields such as satellite design it is a requirement (Schaefer and Rudolph, 2005). The completeness of such approaches show the challenges that grammar-based have but also that a multi-disciplinary integration of knowledge is possible such as in (Vogel, 2016).

Biology

Port graphs (Andrei, 2008) originally coming from modelling molecular connections and more general Attributed Port Graphs (APT) (Ene et al., 2018) are a special case of the later proposed *layout* (graph) where the ports are not predetermined but instead defined through a function that potentially returns an unlimited amount of ports.

Physics

The highest abstraction with the most powerful definition are hypergraph rewriting systems (HGS) explored such as in ("The Wolfram Physics Project," n.d.). They allow to capture possible multivalent relations between objects.

Artificial Intelligence (AI)

HGS also formalizes chatbots in AI with questions such as finding questions from answers (Vepštas, n.d.). When representing designs as hyper graphs, the same type of AI is applicable.

1.2 Overview

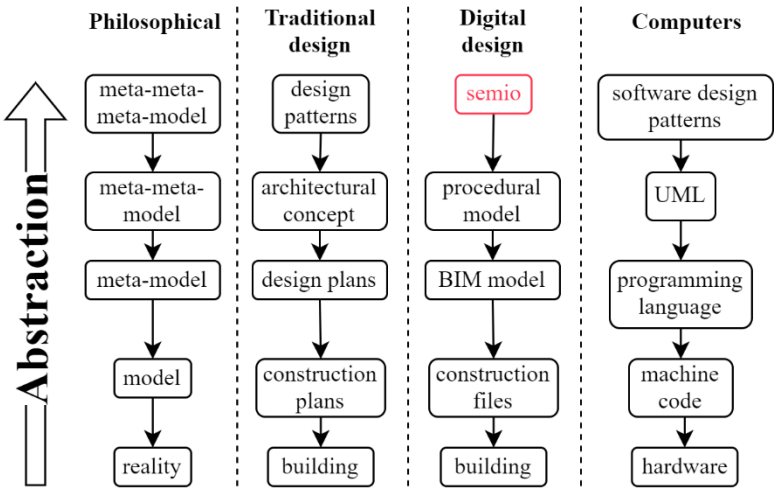

Figure 1: Context

The goal of this paper is to allow parametric ACs to be formalized in an office, project, or personal level. It extends formalization-free architectural design tools while trying to introduce the least amount of formalism and to offer the possibility of enriching them with semantic data in a platform-independent way. The goal is to unify the rapid prototyping and reusability of formal grammars that exploit the customization capabilities of parametric modelling environments and the advantages of BIM's semantics while trying to have a knowledge representation that is close to the design domain. The purpose of *semio* is not to replace any existing modelling software or data-standard but act as additional interoperability layer on top of them.

2. Meta-model

2.1 Concepts

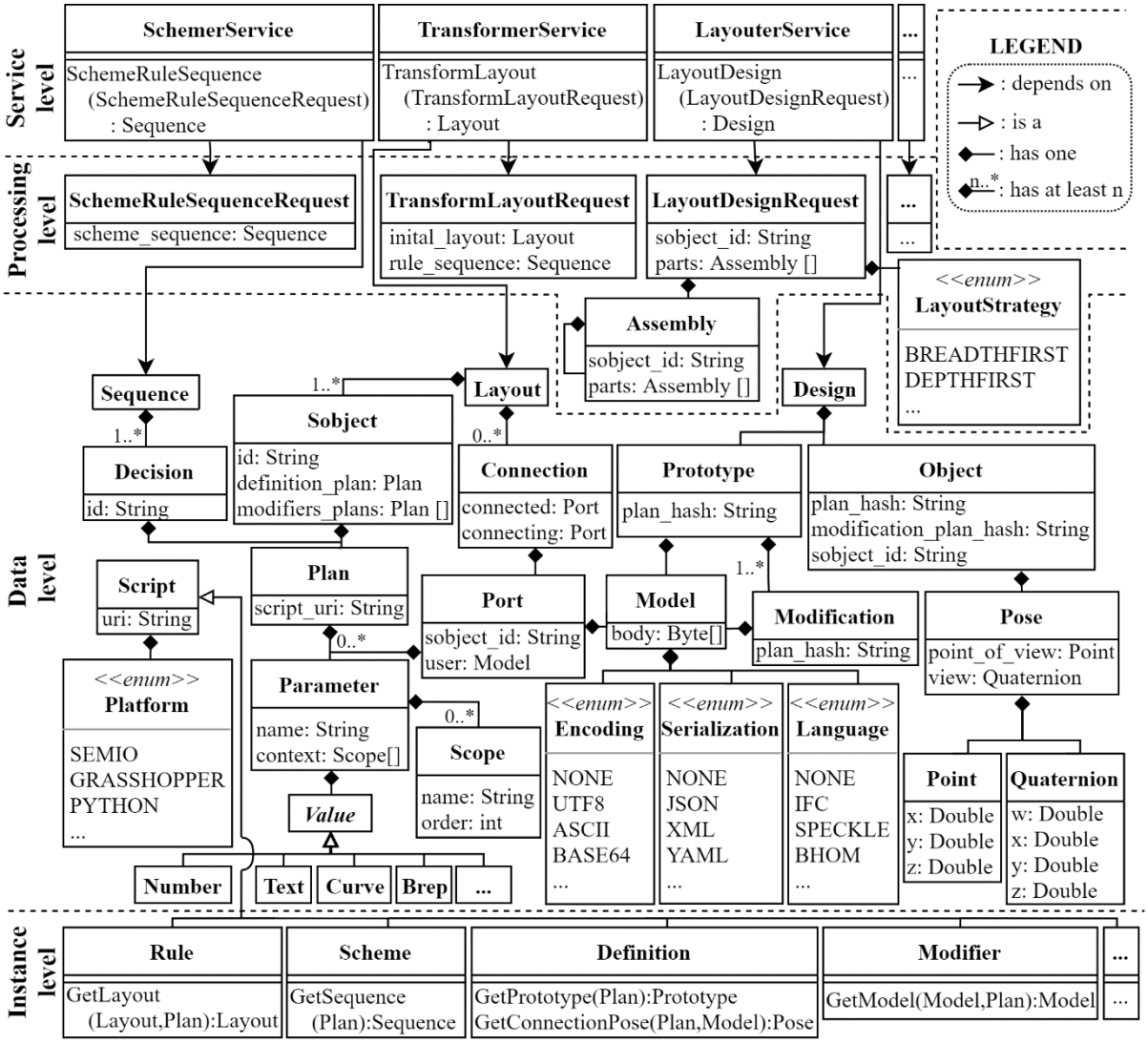

Figure 2: Meta-model UML diagram

A *parameter* is a key-value pair with meta-data such as the context of the *parameter*. The *context* helps to build more complex data structures like lists, dictionaries, trees, or graphs which are native to *platforms*. A *parameter* can be numerical (integer, natural number, …), textual (string, enum) or geometrical (point, vector, line, curve, brep, …). *Parameters* together with a uniform resource identifier (uri) of a *script* form a *plan*. *Plans* are used to decouple specification of *objects* from *object* creation. They can be seen as a descriptive blueprint or a contract. The specification is bundled in a virtual semantical object, so called *sobject*, which is assumend to be parameteric and can return *models* of itself and interact with other *sobjects* through *ports*. A *port* is a description of a *connection* that returns a *pose*. A *pose* is a reference coordinate system for an *object*. To *connect* two *objects*, each *object* specifies a *port* and a representation of itself in form of a *model* that the other *object* will see during the *connection* process inside of a *connection*. *Sobjects* and *connections* form a *layout*. The *layout* can be interpreted as a property graph where the *sobjects* are the nodes and the *connections* are undirected edges. It can be seen as the DNA of a *design*. A *design* is a container for *objects* which are spatial occurrences of *prototypes*. A *prototype* is a collection of *models* which were

all produced by the same *plans* from the same *definition* and the same *modifiers*. A *definition* is a *script* for a *sobject* that returns *prototypes* and implements the *port* mechanism. All standard BIM objects in BIM authoring tools like wall and slab are in a wider sense *definitions*. While a *definition* is for creation of specific *objects*, a *modifier* is a general-purpose *script* that modifies *models* from the same *language*. A *language* is the schema on how to interpret the *body* of a *model*.

A *layout* is further decomposed into *rules* and *decisions*. As a *layout* can be interpreted as a graph, graph rewriting becomes applicable. A rule is a pattern where the input is a *layout* along with *parameters* and the output is a modified *layout*.

A *script* is a document that when given to a *platform* along with *parameters*, it provides outputs. A Python function, a Grasshopper script, or a 3D model with different design variants on different layers could all be *scripts*. A *script* therefore doesn't necessarily need to be executable in a traditional sense like a conventional program, but it can be as simple as a lookup table that is interpretable by a *platform* which is normally a software that can process the *script*. It can be seen as a function. In order that these *scripts* work seamlessly together and remain swapable, they need a common understanding of inputs (*parameters*) and outputs. Due to the high-level abstraction of what a *script* is, it is possible to *parametrize* a *layout* and resolve it inside of *script*. Such a script can be used e.g., as a new *definition* which can be used for a *sobject* in a higher-level *layout*. Recursion can be further used to decouple individual *scripts*.

3. Use-case: capsule tower

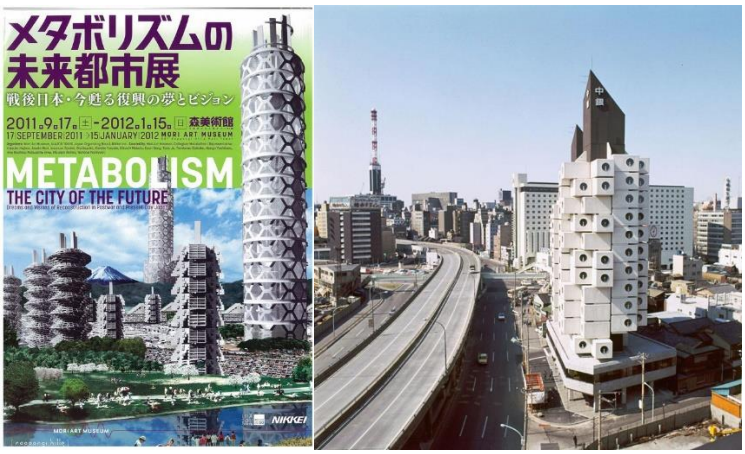

Figure 3: metabolism exhibition flyer (left)[1], Nakagin Capsule Tower (right)[2]

To exemplify how the meta-model works, an example based on the Nakagin Capsule Tower and ideas of the Metabolism movement was modelled. A fictive and simplified design process is illustrated. The design team consist of *Enzo*, *Sho*, *Ken*, *Yono* and *Masa*. Enzo is the head of the group and had an idea of a high-density city that can organically adapt to the changing needs of its habitants. When Enzo presents his ideas to the group, Sho has an idea of a modular system consisting of towers as vertical infrastructure and capsules as providers of function. The team has agreed to pursue with the ideas and develop a prototypical system serving the first main function of the city: providing a home to people. To keep things simple, they start with single

[1] https://www.mori.art.museum/english/contents/metabolism/about/ (Accessed: 15.02.2023)
[2] https://www.tokyotimes.org/archives/nakagin12.jpg (Accessed: 16.02.2023)

room type capsule system which they plan to extend in the future with larger capsules which are interconnected to form larger apartments. Ken proposes to develop the capsule system further and Yono wants to focus on the towers.

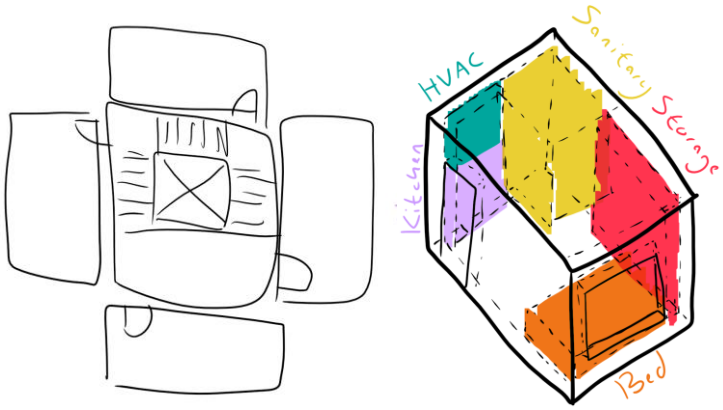

Figure 4: Shos tower-capsule AC (left), Kens capsule AC (right)

After fifteen minutes, Sho has the first *definition* ready which doesn't contain any internal logic yet, but has already the shape, so Ken can use it for testing preliminary designs. Ken comes up with a general rule that stacks capsules over each other. Ken has advised Sho and Yono that their definitions must meet on the door. After some time, the shaft is ready and version one is explored.

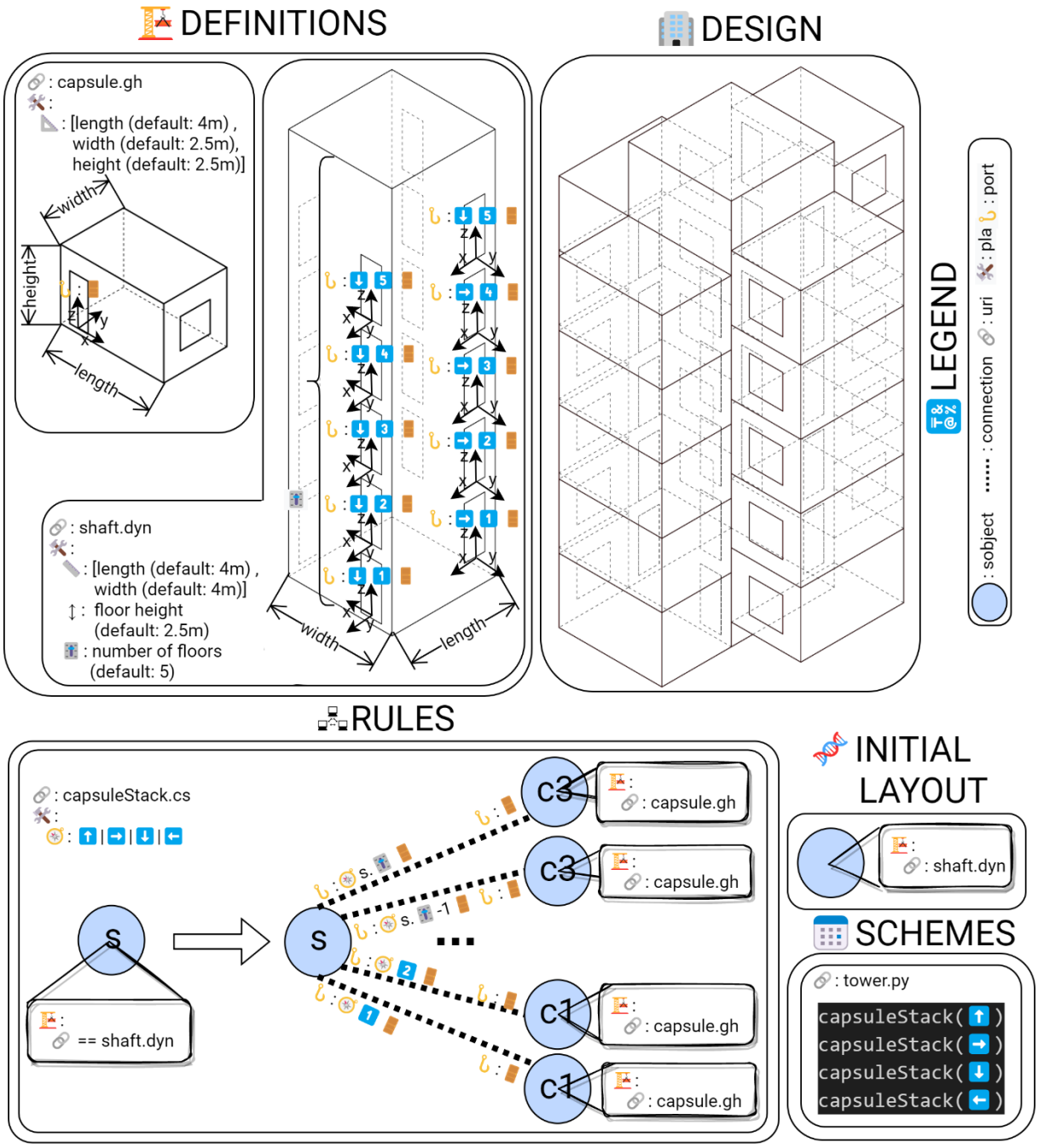

Figure 5: Version one of capsule tower

When they show it to Enzo, he want's a higher density and suggest doubling the number of capsules by expanding the capsule system to have a second type with the entrance in the back. Sho starts to work on expanding the capsule, Ken adds the second door per floor and adjusts the default values of the shaft.

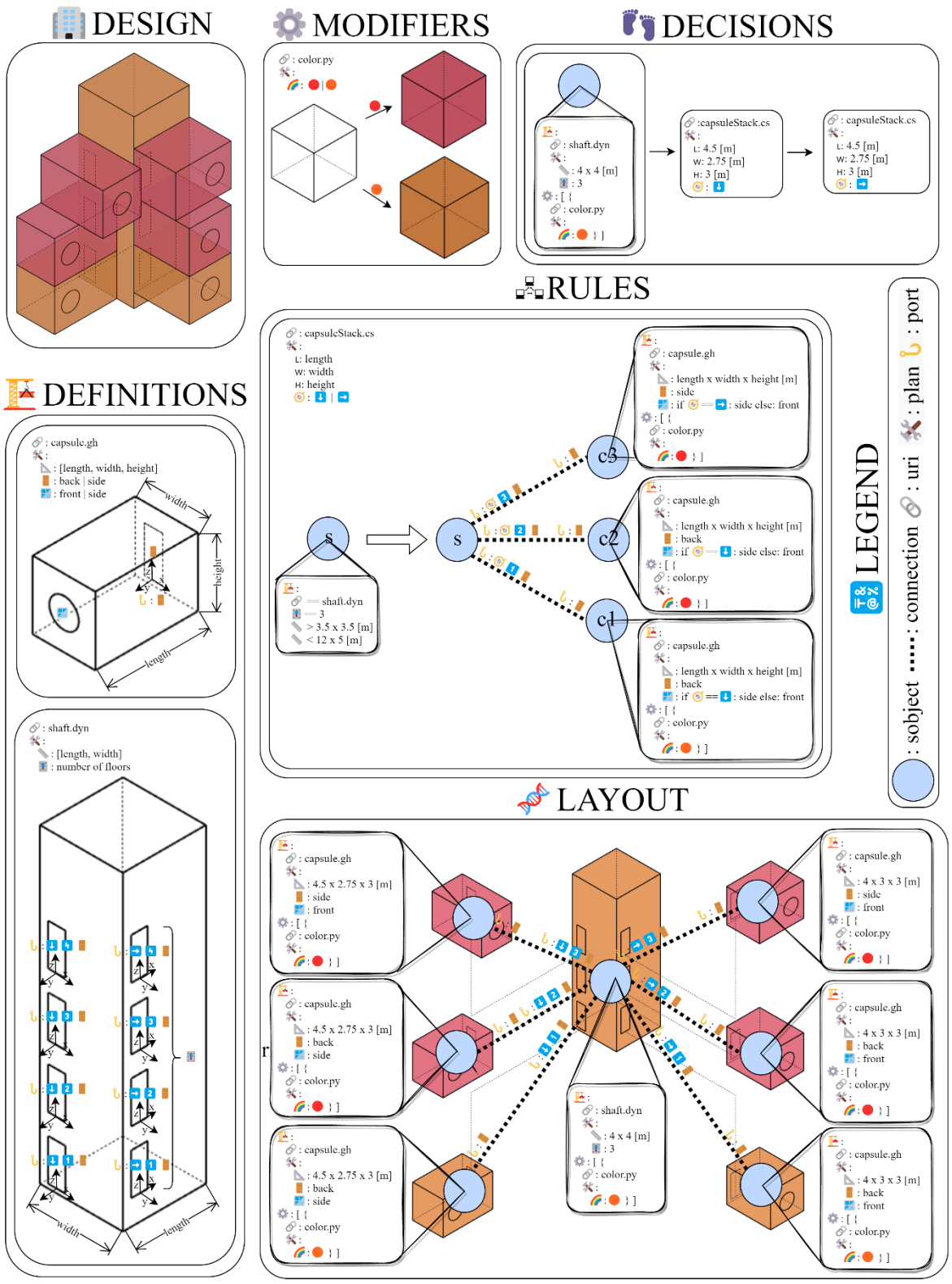

Figure 6: Version with updated capsules

Figure 7: Version three

4. Software prototype

To demonstrate the meta-model an open-source prototype has been implemented[3]. A backend for the computation of *layouts*, a user-interface (UI) to view and request *designs* and an extension for Grasshopper to adapt *scripts* have been written. The backend is a microservice architecture with currently five services. The meta-model has been defined over the programming-language independent Interface Definition Language (IDF) protobuf and service clients and servers communication was realized over gPRC. The prototype is architecturally designed for extensibility and interoperability.

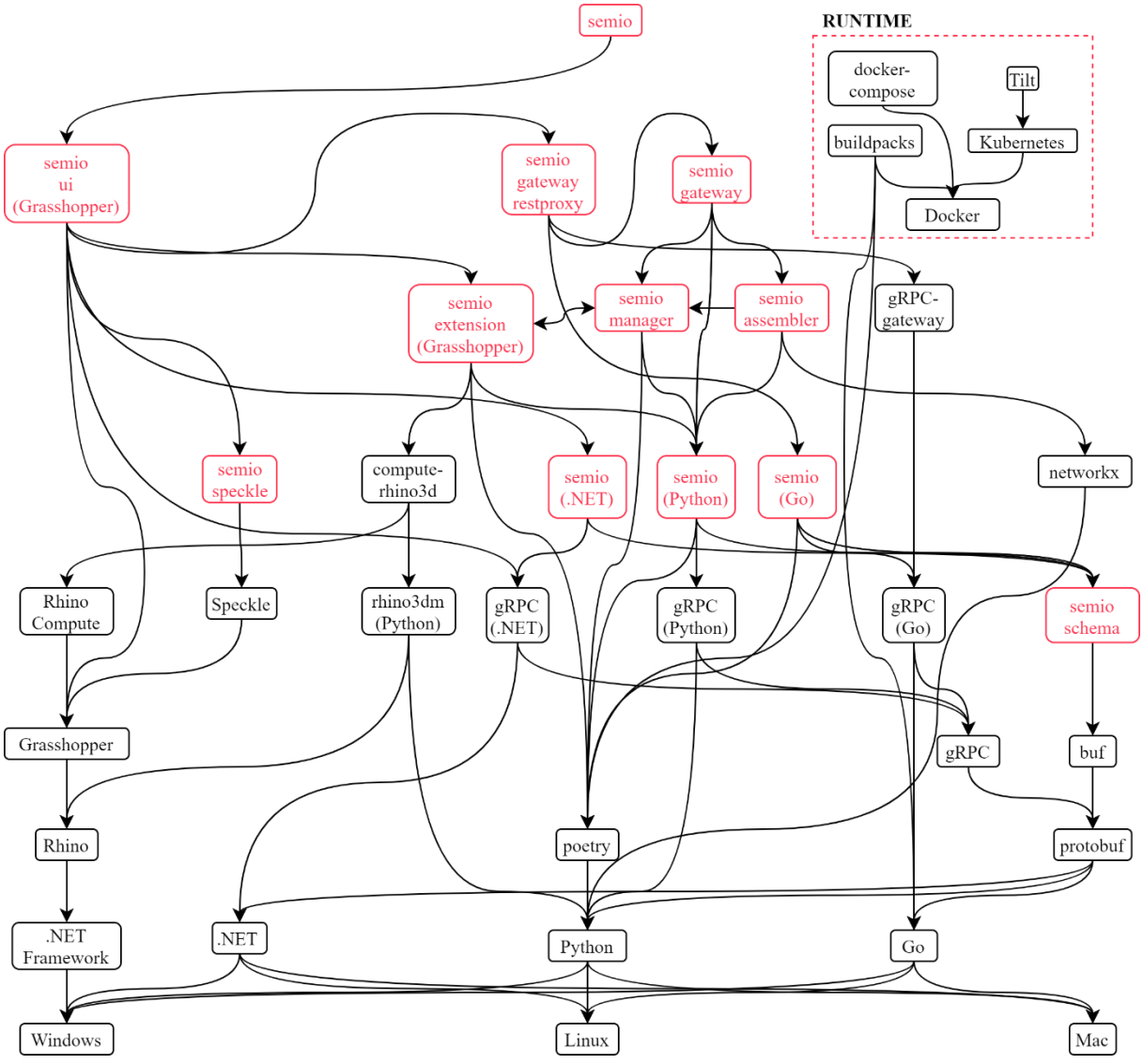

Figure 8: Implemented components

[3] https://github.com/usalu/semio

5. Discussion

5.1 New possibilities

The complexity and size of a BIM model are high. A Common Data Environment (CDE) only manages the infrastructural complexity, but the intellectual complexity is still given. A common approach to solve this issue is to decompose the model into disciplines and recompose it on *data drops*. With such an approach the structural engineer has liability for all standard objects that have a structural function. This brings a couple of coordination problems e.g. clash detection (Akponeware and Adamu, 2017). Most of the times rules and fuzziness are agreements between disciplines to avoid clashes and optimally a result of a clash detection is a detection of which rule was broken instead of simply delivering the objects that collide. With *semio* on the other hand, the complexity is decomposed according to project specific *objects* and the recomposition is gradually happening through *rules*, *decisions, definitions,* and *modifiers*. These additional levels offer various new possibilities.

Collaboration

Every *script* can be created, updated by a separate person. This allows to parallelize processes and offers possibilities for shorter development cycles. Versioning of partial *scripts* helps in understanding overall changes and increases explainability.

Testing

If a *rule* produces a partial *layout* where objects collide, then this can be resolved much earlier. Further a new *layout* can be tested for graph morphisms and if all local matches satisfy a pre-tested *layout*, then the chances that the total *layout* will be collision free are higher. Besides collision testing, functional testing according to discipline e.g. load requirements for certain structural *objects*, loop temperature for hvac-systems for *connections* or sound insulation for *objects*.

Authoring

Once the design has been architecturally decomposed and first scripts have been writing, it is easy to modify *parameters* and change *rules*. This allows for great design space exploration possibilities that can be combined with a What You See Is What You Get (WYSIWYG) UI such as in an Augemented /Virtual Reality (AR/VR). Further Artificial Intelligence (AI) are possible e.g., recommendation systems based on similarities of *layouts* become or Graph Neural Networks (GNN) can be predicting *rules* or *layouts*. Other Machine Learning (ML) methods can be used to predict *parameters*, find patterns in *layouts,* or be used as a surrogate model for a *script* to reduce computing times.

Reusability

The highest level of abstraction with the highest generality are *modifiers*. They offer the greatest level of reuse. The second level of abstraction are *definitions*. They are self-contained and can easily be reused in a new project. To reuse a *layout* all *definitions* and *modifier* must be present. They can be replaced with new *scripts,* but the *layout* is most likely dependent on their functionality. The largest impact of reducing effort is achieved when you can reuse *rules*. They can with small adjustments have a great effect on the *design*. Overall, the general reusability depends on how well the overall *design* is decomposed and how well the interfaces are cut.

5.2 Open research questions

Designs are by nature hard to decompose because it is a system of a lot of objects with a lot of interlinked functionalities e.g., two rooms share only one wall which exists on its own. When a design is prefabricated then objects are more and more visible, but they are not always equal to the mental objects that architects used during planning. One way to solve such issues is by setting *definitions* for virtual objects which are not necessarily visible in the end. A void can be modelled as an individual *definition* or as a *modifier* if it only affects one other *object*.

Further can rules not only add new *sobjects* as in set grammars but also remove some. This is a powerful mechanism but from a complexity standpoint this is dangerous and can fast lead to infinite recursion when misconfigured. Advantage and disadvantages must be examined on test cases and patterns should be developed as best practices.

The views on what an AD is differ depending on the domain. A *layout* that can derive e.g. a structural and energetical model needs to provide a mechanism for attaching domain specific knowledge not only to *sobjects* and *connections* but also allow to map *layouts* between each other. Data-models and methods for assisting such knowledge representation need to be developed.

6. Conclusion

This paper shows that a decomposition and recomposition of a design through the proposed meta-model into non-standard objects while keeping interoperability is possible. A use-case could exemplify that process. New possibilities that follow from applying this new method were anticipated but need still need to be proven. The new dynamic and problems that appear with such a design method need to be case-based compared with conventional approaches to evaluate the use that the increased effort of formalization brings.

References

Agrawal, A., Karsai, G., Shi, F., 2003. Graph transformations on domain-specific models. Journal on Software and Systems Modeling 37, 399.

Akalin, A., Sezal, I., 2009. The Importance of Conceptual and Concrete Modelling in Architectural Design Education. International Journal of Art & Design Education 28, 14–24. https://doi.org/10.1111/j.1476-8070.2009.01589.x

Akponeware, A.O., Adamu, Z.A., 2017. Clash Detection or Clash Avoidance? An Investigation into Coordination Problems in 3D BIM. Buildings 7, 75. https://doi.org/10.3390/buildings7030075

Alexander, C., Ishikawa, S., Silverstein, M., 1977. A pattern language: towns, buildings, construction. Oxford University Press, New York.

Andrei, O., 2008. A Rewriting Calculus for Graphs: Applications to Biology and Autonomous Systems. (Un calcul de réécriture de graphes : applications à la biologie et aux systèmes autonomes).

Bahubalendruni, M.V.A., Biswal, B.B., Khanolkar, G.R., 2015. A review on graphical assembly sequence representation methods and their advancements. Journal of Mechatronics and Automation 1, 16–26.

Bragança, L., Vieira, S.M., Andrade, J.B., 2014. Early Stage Design Decisions: The Way to Achieve Sustainable Buildings at Lower Costs. The Scientific World Journal 2014, e365364. https://doi.org/10.1155/2014/365364

Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Hernandez, N.V., Wood, K.L., 2011. Computer-Based Design Synthesis Research: An Overview. Journal of Computing and Information Science in Engineering 11. https://doi.org/10.1115/1.3593409

Davis, D., Burry Mark, B.J., 2011. Untangling parametric schemata: enhancing collaboration through modular programming.

Digital America: A tale of the haves and have-mores - Full report, 2015. . McKinsey Global Institute.

Dorst, K., Vermaas, P.E., 2005. John Gero's Function-Behaviour-Structure model of designing: a critical analysis. Res Eng Design 16, 17–26. https://doi.org/10.1007/s00163-005-0058-z

Dynamo, 2023.

Eilouti, B., 2018. Concept evolution in architectural design: an octonary framework. Frontiers of Architectural Research 7, 180–196. https://doi.org/10.1016/j.foar.2018.01.003

Ene, N.C., Fernández, M., Pinaud, B., 2018. Attributed Hierarchical Port Graphs and Applications. Electron. Proc. Theor. Comput. Sci. 265, 2–19. https://doi.org/10.4204/EPTCS.265.2

Esri CityEngine | ArcGIS Desktop, 2008.

Gamma, E. (Ed.), 1995. Design patterns: elements of reusable object-oriented software, Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass.

gbXML, 2009.

Gero, J.S., 1990. Design Prototypes: A Knowledge Representation Schema for Design. AI Magazine 11, 26–26. https://doi.org/10.1609/aimag.v11i4.854

Gero, J.S., Kannengiesser, U., 2006. A Function-Behaviour-Structure Ontology of Processes, in: Gero, J.S. (Ed.), Design Computing and Cognition '06. Springer Netherlands, Dordrecht, pp. 407–422. https://doi.org/10.1007/978-1-4020-5131-9_21

Goldschmidt, G., 2017. Manual sketching: Why is it still relevant? The active image: architecture and engineering in the age of modeling 77–97.

Grasshopper, 2007.

Grobman, Y.J., Yezioro, A., Capeluto, I.G., 2010. Non-Linear Architectural Design Process. International Journal of Architectural Computing 8, 41–53. https://doi.org/10.1260/1478-0771.8.1.41

Harding, J.E., Shepherd, P., 2017. Meta-Parametric Design. Design Studies, Parametric Design Thinking 52, 73–95. https://doi.org/10.1016/j.destud.2016.09.005

Hirschberg, U., Hovestadt, L., Fritz, O. (Eds.), 2020. Atlas of digital architecture: terminology, concepts, methods, tools, examples, phenomena. Birkhauser, Boston.

Homem de Mello, L.S., Sanderson, A.C., 1990. AND/OR graph representation of assembly plans. IEEE Transactions on Robotics and Automation 6, 188–199. https://doi.org/10.1109/70.54734

Iano, J., Allen, E., 2022. The Architect's Studio Companion: Rules of Thumb for Preliminary Design. John Wiley & Sons.

Johnson, P.-A., 1994. The Theory of Architecture: Concepts Themes & Practices. John Wiley & Sons.

Kolbeck, L., Vilgertshofer, S., Abualdenien, J., Borrmann, A., 2022. Graph Rewriting Techniques in Engineering Design. Frontiers in Built Environment 7.

Königseder, C., Stanković, T., Shea, K., 2016. Improving design grammar development and application through network-based analysis of transition graphs. Design Science 2, e5. https://doi.org/10.1017/dsj.2016.5

Liang, V.-C., Paredis, C.J.J., 2004. A Port Ontology for Conceptual Design of Systems. Journal of Computing and Information Science in Engineering 4, 206–217. https://doi.org/10.1115/1.1778191

Lienhard, S., Lau, C., Müller, P., Wonka, P., Pauly, M., 2017. Design Transformations for Rule-based Procedural Modeling, in: Computer Graphics Forum. Wiley Online Library, pp. 39–48.

Lipp, M., Specht, M., Lau, C., Wonka, P., Müller, P., 2019. Local editing of procedural models, in: Computer Graphics Forum. Wiley Online Library, pp. 13–25.

Ma, J., Hu, J., Zheng, K., Peng, Y., 2013. Knowledge-based functional conceptual design: Model, representation, and implementation. Concurrent Engineering Research and Applications 21, 103–120. https://doi.org/10.1177/1063293x13487358

Ma, W., Wang, X., Wang, J., Xiang, X., Sun, J., 2021. Generative Design in Building Information Modelling (BIM): Approaches and Requirements. Sensors 21, 5439. https://doi.org/10.3390/s21165439

Molly Wright Steenson, 2022. Architectural Intelligence. The MIT Press.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van Gool, L., 2006. Procedural modeling of buildings, in: ACM SIGGRAPH 2006 Papers, SIGGRAPH '06. Association for Computing Machinery, New York, NY, USA, pp. 614–623. https://doi.org/10.1145/1179352.1141931

Neo4j, 2010.

nortikin, 2023. Sverchok.

Para, W., Guerrero, P., Kelly, T., Guibas, L.J., Wonka, P., 2021. Generative Layout Modeling Using Constraint Graphs. Presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 6690–6700.

Parish, Y.I.H., Müller, P., 2001. Procedural modeling of cities, in: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01. Association for Computing Machinery, New York, NY, USA, pp. 301–308. https://doi.org/10.1145/383259.383292

Purcell, A.T., Gero, J.S., 1996. Design and other types of fixation. Design studies 17, 363–383.

Radermacher, A., 2000. Support for Design Patterns through Graph Transformation Tools, in: Nagl, M., Schürr, A., Münch, M. (Eds.), Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 111–126. https://doi.org/10.1007/3-540-45104-8_9

Reinventing construction through a productivity revolution - Full report, 2017. . McKinsey Global Institute.

Resource Description Framework (RDF), 2014.

Roberts, P.H., Bruce Archer, Ken Baynes, 2019. Modelling: the language of designing. Loughborough University.

Rozenberg, G., 1997. Handbook of Graph Grammars and Computing by Graph Transformation: Volume 1: Foundations. WORLD SCIENTIFIC. https://doi.org/10.1142/3303

Schaefer, J., Rudolph, S., 2005. Satellite design by design grammars. Aerospace Science and Technology 9, 81–91. https://doi.org/10.1016/j.ast.2004.08.003

Singh, P., Bettig, B., 2004. Port-Compatibility and Connectability Based Assembly Design. Journal of Computing and Information Science in Engineering 4, 197–205. https://doi.org/10.1115/1.1779659

Smith, A.C., Smith, K.S., 2014. Developing Your Design Process: Six Key Concepts for Studio. Routledge.

Stiny, G., 2006. Shape: talking about seeing and doing. MIT Press, Cambridge, Mass.

Teo, E.A.L., Ofori, G., Tjandra, I.K., Kim, H., 2015. The potential of Building Information Modelling (BIM) for improving productivity in Singapore construction, in: Symposium Conducted at the Meeting of the 31st Annual ARCOM Conference. pp. 7–9.

The Wolfram Physics Project: Finding the Fundamental Theory of Physics [WWW Document], n.d. URL https://www.wolframphysics.org/ (accessed 4.8.23).

U.S. Green Building Council, 1998. LEED.

Vepštas, L., n.d. Graphs, Metagraphs, RAM, CPU.

Vesnic, S., 2017. What is an architectural concept? The "concept" of Deleuze and "project" of Eisenman. Filoz drus 28, 1122–1135. https://doi.org/10.2298/FID1704122V

Vogel, S., 2016. Über Ordnungsmechanismen im wissensbasierten Entwurf von SCR-Systemen (doctoralThesis). Stuttgart : Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart.

Wonka, P., Wimmer, M., Sillion, F., Ribarsky, W., 2003. Instant architecture. ACM Trans. Graph. 22, 669–677. https://doi.org/10.1145/882262.882324

Wortmann, T., Stouffs, R., 2018. Algorithmic complexity of shape grammar implementation. AI EDAM 32, 138–146. https://doi.org/10.1017/S0890060417000440