Smart Manufacturing Frameworks

Neythalath, Narendrakrishnan

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Smart Manufacturing Frameworks

Narendrakrishnan Neythalath

Kongens Lyngby 2021

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

The construction industry has been plagued by low productivity for quite sometime now. Automation is considered as a viable way forward to improve the situation. The industry has not witnessed proliferation of robotic technologies as seen in other fields like manufacturing due to high levels of project specificity, smaller lot sizes for production and need for customization, thus causing the sector to remain vastly under-served.

In this work, we focus on developing a software framework (Sculptor), which will help the industry to rapidly design, prototype, develop, and test robotic applications. An application is said to be understood as a cyber-physical system which contains one or more robotic manipulators equipped with necessary end-effectors and a tablet based interface to help a novice user to easily program the system.

Architects find it convenient to express such robotic processes using parametric (data flow) models. These models come with some known drawbacks related to stability, maintainability, latency and re-usability. After having developed Sculptor, we are trying to address some of the above stated problems by applying methods like software design patterns. Further, the framework was used to develop multiple commercial applications to evaluate its effectiveness.

Summary (Danish)

Den globale byggeindustri har gennem en længere årrække været plaget af lav produktivitet, og automatisering betragtes som et af de væsentligste instrumenter til at forbedre dette forhold. Modsat andre industrielle sektorer, såsom fremstillingsindustrien, har byggebranchen dog ikke oplevet en skaleret udbredelse af robotteknologier. Dette skyldes en række konceptuelle barrierer, herunder høje niveauer af projektspecificitet, samt udbredt unikaproduktion, som forhindrer anvendelse af mere konventionelle fremstillingsteknologier.

Smart Manufacturing Frameworks fokuserer på at udvikle en softwarestruktur (Sculptor), som vil hjælpe byggebranchen med effektivt at designe, prototype, udvikle og teste robotapplikationer for byggeri. En applikation forstås som et cyber-fysisk system, der indeholder en eller flere robotmanipulatorer udstyret med nødvendige end-effectorer og en tabletbaseret grænseflade til at hjælpe en uerfaren bruger med let at programmere systemet.

Arkitekter finder det praktisk at udtrykke sådanne robotprocesser ved hjælp af parametriske (grafbaserede) modeller. Disse modeller har nogle kendte ulemper relateret til stabilitet, vedligeholdelsesevne, latens og genanvendelighed. Efter at have udviklet Sculptor prøver vi at løse nogle af de ovennævnte problemer ved at anvende metoder som softwaredesignmønstre. Desuden bliver frameworket brugt til at udvikle adskillige kommercielle applikationer for at evaluere dets effektivitet.

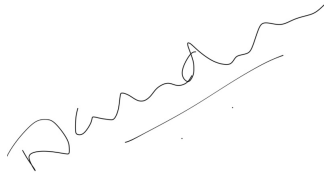
Preface

This thesis was prepared at Odico A/S and DTU Compute in fulfilment of the requirements for acquiring Doctor of Philosophy (PhD) in Computer Science and Robotics.

The thesis deals with a novel framework to design, prototype, develop and test robotic applications targeted at construction industry.

It consists of theoretical formulations and experimental evaluation of the proposed framework.

Lyngby, 31-July-2021



Narendrakrishnan Neythalath

Acknowledgements

I would like to thank my supervisors Asbjørn Søndergaard and Jakob Andreas Bærentzen for all their help and advice with this PhD. I would also like to express my gratitude to my team members - Bhavatarini Kumaravel, Nishant Chandrashekar, Deepanshu Bansal, and Aditi Chauhan. I also appreciate all the support I received from my mother (Sudha), daughter (Reva), and wife (Aruna) in this journey. It would have not been possible without them.

This work is partly funded by the Innovation Fund Denmark (IFD) under File No. 7038-00108A.

Contents

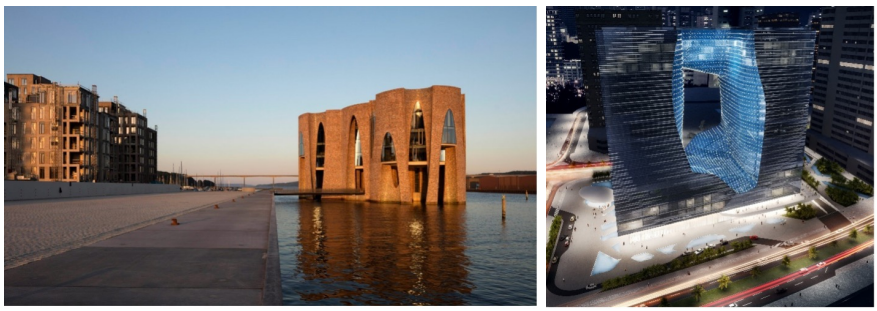
Chapter 1

Background and motivation

Odico was founded in 2012 with the ambition to transform the global construction sector through introduction of robotic technologies. The value proposition was to greatly reduce the cost of formwork manufacturing through robotic wire-cutting of casting moulds for concrete production. By taking CAD models, typically comprised of ruled surface geometries, we deduct the isocurves of the target surface and sweep a suspended wire held by a 6-axis industrial manipulator along the provided Expanded Polystyrene (EPS) block. The bespoke designs could be produced at up to 126x speed of traditional CNC-milling based method of formwork making.

This performative advantage would translate to significant reductions in formwork costs, as the long machining times lead to high costs for digitally produced, advanced moulds. In addition, the flexibility of the industrial robot setup would enable swift programming of custom geometries – a critical component within construction, in which vast majorities of cases constitute production of items in small lot sizes.

Based on this competitive edge, Odico set out – as the first company internationally – to commercialize robotic wire-cutting of EPS formwork for large scale concrete work. This was executed by the initiation of a pilot production line, which took on projects of increasing scale. In 2014, this materialized into a breakthrough project, where Odico was requested to create formwork for a load-



(a) **(b)**

Figure 1.1: (a) Fjordenhus Kirk Kapital building in Vejle Denmark, 2018. (b) Opus Dubai, UAE 2020. The Opus, designed by Zaha Hadid Architects, was listed by CNN as one of the most significant buildings in the world in 2020. Odico helped to produce EPS guidework for the production of the aluminium frames carrying the doubly curved main glass facade of the building complex.

bearing structural element for the iconic Fjordenhus Kirk Kapital HQ building (see, Fig. 1.1a), designed by the well-renowned Studio Olafur Eliasson. Encompassing a 4500 m^2 of robotically wire-cut formwork surface, the project would represent first example of the realization of a commercial construction project via the said method internationally. Following the success of the Fjordenhus project, Odico entered into a phase of organic growth, realizing in 6 years, 350 projects of various scales in 7 countries, including the United Kingdom, Dubai, Norway, Australia, Belgium and United States.

This trajectory culminated in 2018 on Odico A/S becoming the first Danish robotic company to be publicly held on Nasdaq First North Copenhagen. The strategic impetus behind the IPO was to raise financing to realize a technology vision that had materialized from the years of pilot production and R&D work: the Factory on the Fly™. It encapsulates the company's bid to provide an all-purpose general technology to bring robots to the construction sites and factories at scale (see, Fig. 1.2). This PhD focuses on realizing this vision.

From the preceding 6 years of effort to bring robotics into construction, three general, industry-wide challenges had been identified:

1. the global construction workforce is constituted by craftsmen trained in execution of manual work, with little to no exposure to robotic programming or advanced manufacturing technologies. Hence, to ensure a rapid adoption of automation, it is paramount that the Human Machine Inter-

Figure 1.2: Factory on the Fly™ at a construction site.

faces (HMIs) for these systems should have a complexity level which does not exceed that of common mobile applications;

2. as opposed to general manufacturing, which has greatly benefited from embracing a mass manufacturing paradigm in which many instances of the same product are repeatedly being produced on highly optimized specialized facilities at high throughput rate, construction is characterized by the production of either a singular piece of an item or the said items in small batches. In order to support such variability, the underlying manufacturing framework must facilitate product customization with ease;

3. ultimately, not only the shape of the product can vary, but also its typology, material composition and associated manufacturing processes. To support such constitutional variability, the entire software architecture must be structured in anticipation of the highest degree of interchangeability and modularity.

From the observation of these fundamental constraints arose the hypothesis that the above criteria may be met through integrating visual programming for the creation of parametric design/manufacturing models to drive an execution node which controls a variable robotic architecture. These two components are then procedurally expressed in a mobile application interface, which exposes only the necessary parameters for a particular, specialized manufacturing situation, in order to provide a simplistic user experience.

The problem statement for this PhD has been to a) investigate this hypothesis and its theoretical implications; b) conceptualize an optimized software architecture responding to these findings and c) prototype the architecture and appli-

cations through a plurality of commercial case studies provided by the company, to deliver real-world indications of the industrial applicability.

Of particular importance has been the emphasis on a case-based, iterative approach to the framework conceptualization, in which commercial scale implementations of various applications have informed and refined the architecture continuously through the project. As such, the impetus to go through such iterative cycles has been a requirement for ensuring the real-world validity of results.

The project's conceptual development has been instrumental to initiating and implementing the fully operational, commercial grade software framework, Sculptor, which has served as an increasingly comprehensive body for theoretical experiments. The framework has and is serving as the company's core technology which is intended to underpin a continuously growing portfolio of robotic technology solutions.

Chapter 2

Elaborating on the challenge

Despite representing the 5th largest industry segment, the global construction industry suffers from low productivity (see, Fig. 2.1), while other sectors – such as the manufacturing industries – has seen consistent increase in productivity over the years [PTT05, BHK21, BDM20]. The rising productivity in neighboring segments is widely attributed to the pervasive automation and digitization. Hence large scale adoption of robots in the construction sector is widely regarded as the primary enabler for increase in productivity [DOA+19]. However, despite large research efforts, and the availability of mature automation technologies from tangential fields such as automotive, aeronautical, naval and energy industries, ubiquitous adoption of robotic processes within the industry still remains evasive. As a result, construction is one of the least automated of the leading industrial sectors, next only to agriculture [Roh08]. Inside of a larger industrial research effort to establish a general purpose, cyber-physical technology platform (see, Fig. 2.2) for efficient automation of construction tasks, this work reports on the developments relating to the establishment of Sculptor for efficient robotic application development for the industry.

The main factors prohibiting the large scale espousal of robotic technology in construction are given below:

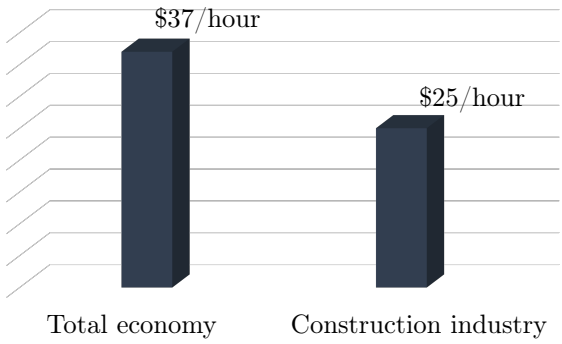


Figure 2.1: Average value added by employees per hour at a global level.[1]

1. **Cost** - A widely understood differentiating factor between construction and other manufacturing industries relying on large scale production is the circumstance, that while the latter can benefit from deploying a mass manufacturing paradigm - in which multiple instances of identical products is repeatedly produced at high volume - such repetition is not feasible in construction, in which each building project is essentially unique [DOA+19]. Since automation technologies in general manufacturing is designed to support a repetitive mode of operation, they lack the flexibility to accommodate the variability experienced in construction. To overcome this shortfall, significant research efforts [WKB+16, GK14, Men12] have been directed towards establishing parametric manufacturing workflows, in which a production system relying on one or more standard industrial manipulators with associated machining end-effectors is driven by a parametric manufacturing model, which utilizes the inherent flexibility of a manipulator with m degrees of freedom to create shape-variant instances of a customizable part design within pre-defined ranges of variables. The establishment of such workflows is complemented by additional research attention directed towards machine process innovation and mechanical end-effector engineering. Leading examples from this work entails robotic processes such as e.g. timber sawing [EGK17, KSM+14]; hot-blade and wire-cutting [FS14, Sea16]; brick assembly [Bon15]; robotic concrete printing [PZTG18, GM18, GCF18] and metal forming [TPM18]. The combined

[1] https://www.mckinsey.com/business-functions/operations/our-insights/reinventing-construction-through-a-productivity-revolution

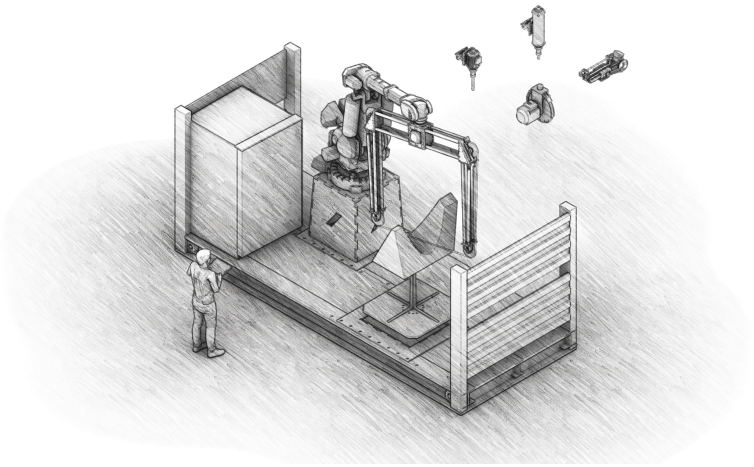


Figure 2.2: A robotic application is understood as a cyber-physical unit comprising of these hardware components: a) one or more m-axis manipulators enclosed on b) a modular frame, equipped with c) custom processing end-effectors and d) operated by a non-specialist user through a tablet or mobile application.

developments in this strand of research have recently culminated in the realization of several experimental and commercial construction projects, such as Fjordenhus Kirk Kapital HQ, Vejle, Denmark (2018) [SF17], Opus Dubai, UAE (2020) [Bho17], The Sequential Roof, Zürich, Switzerland (2016) [ABF+16], and DFAB House, Zürich, Switzerland (2019) [Dea19], signifying the principal applicability of robotic manufacturing at construction scale. While the establishment of a robotic parametric manufacturing paradigm – complemented by aforementioned advances in process innovation – have successfully demonstrated a pathway for increasing flexibility of digital production to create shape variant parts at near cost-parity to standardized production schemes – two fundamental limitations remain largely unaddressed:

(a) while parametric manufacturing workflows can be established for predefined typologies of component production, they are ill-suited to accommodate non-linear changes in fundamental parameters such as product typology, material composition or machining process. Since the variability of construction manufacturing entails exactly this across individual building projects, each non-linear change requires manual amendment of an existing or establishment of a new workflow;

(b) establishment of such workflows rely on an emerging cross-disciplinary

expertise spanning across previously isolated domains, including robotics, computational design and construction manufacturing. By contrast, the global construction workforce is overwhelmingly trained in craft based, manual processes with little to no exposure to robotic manufacturing.

Taken together, these limitations incur significant overhead in the establishment of project specific workflows, while inducing substantial inflexibility in applying them in a general regime across projects.

For clarity, aforementioned concerns may be expressed as:

$$C = C_v + C_f/n \tag{2.1}$$

where C denotes the production cost of a shape-variant part or building component, C_v is variable costs associated with producing each item unaffected by the number of produced items - typically entailing labor, material and machining time and C_f signifies the fixed, one-time costs of implementing the digital production workflow required to manufacture the shape variant family of parts, and n denotes the number of construction projects on which the particular workflow can be applied. Since global construction is generally characterized by a high number of sub-specialized trades, which operates interchangeably to construct topologically variant tasks, it stands to reason that the probability of n being one is quite high. In this case, only an extra-ordinary construction budget would be capable of sustaining the upfront cost of bespoke workflow establishment, limiting the application of such robotic schemes to high-profile building projects. To be cost affordable, C_f needs to be minimized.

2. **Knowledge gap** - There is a need to bridge the knowledge gap between crafts-trained construction workers educated in manual execution on one hand, and the deep technical expertise required for robotic programming on the other [Ela08]. In the wider community of robotic research, to which this challenge also applies, several novel paradigms have been explored, such as collaborative robotics, in which e.g. a human operator may program the robotic motion through physically translating the position of the end-effector from one point to another and recording this translation [AMAI19]; or through high-level voice commands to instruct a robot to perform certain actions [Pir05]. However, such approaches are inapt to construction manufacturing workflows, because they involve complex operations such as a) obtaining or adapting a CAD file containing geometric design; b) deriving target positions from this CAD file and c) executing the targets with high precision to produce the desired item or product through additive, subtractive or formative machining.

   Contemporary industrial software packages for handling such CAD/CAM production workflows - for instance, Catia, Solidworks, MasterCam, Fu-

sion 360, Dynamo or Grasshopper - rely on comprehensive, multi-functional interfaces, which require extensive training and often academic educational backgrounds to operate. The complexity of such systems and associated requirements for expert user knowledge stem from the need to provide versatile, general purpose tools, geared to process a plurality of manufacturing or designing tasks. By contrast, the popularization of smart devices within the domain of consumer electronics have established a wide range of common examples of advanced processes like natural language processing; image classification; or remote control of drones being accessible and operable by non-expert users within few minutes of training [IMKT21]. The fundamental enabler for this development has been the introduction of simplified mobile applications/interfaces, which rather than offering a general purpose tool set, is specialized for tasks or sub-tasks of strictly limited functional scope [YYAR02]. While such interfaces offer a strategy of knowledge encapsulation, which has proven widely successful in allowing broader audiences to access and benefit from sophisticated computing tools and processes, they do however rely on extensive usability studies, complementing the software engineering efforts required to establish the underlying processes. For these reasons, the average development time for a mobile application is comprehensive, typically quantified between 6 and 9 months for a basic application [GSMG17].

Considering the success of mobile applications in offering access to complex computational processes for broad and varied audiences, it stands to reason that such applications of comparable simplicity may enable access for non-specialist workers to operate advanced, robotic processes and on-the-fly customization of parts typically required within construction manufacturing, hereby representing a pathway to overcome the knowledge barriers prohibiting wide-spread adoption within the construction industry. However, successful mobile applications for smartphones, tablets or wearable devices may achieve millions or even billions of users, whereas for robotic applications in construction industry comparable numbers may be less than 1000 for a given robotic process, even down to 1 for an entirely project specific scenario.

Considering the significant limitation in target user group size, the normative cost of conventional mobile application development becomes prohibitive, even before considering the significant added complexity of modeling a construction robotic process (see, 2.1). Thus, even with the utilization of best software development practices, an entirely manual schema of interface implementation [MJ06] remains wholly insufficient to solve the before mentioned structural challenge of bridging the knowledge gap between construction workforce and robotic programming. However, if implementation time could be significantly reduced through either full or partial automation of the interface development process, the cost parity

threshold between a fully manual operation and development plus utilization of a project specific robotic workflow could potentially be passed, hereby enabling the pursuit of robotics within the global construction sector at a grand scale. Based on these considerations we summarize the hypotheses:

(a) Widespread acceptance of robotics in construction manufacturing may only be reached by simplifying the operation of construction robotic systems to levels accessible to crafts-trained, non-specialist users.

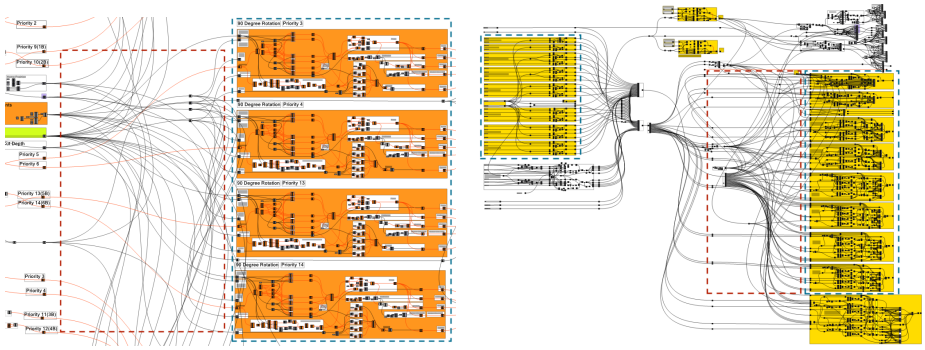
(b) The success of mobile applications within consumer electronics provide a template for such simplification, enabling operation of complex product configuration and manufacturing functions via simplistic, scope-restricted Graphical User Interfaces (GUIs).

(c) The normative development cost of such applications is however not consistent with the significantly reduced target user group sizes arising from high degrees of trade sub-specialization.

(d) Hence, to solve 2a, new software technologies are required, which can provide a substantial reduction in end-to-end interface implementation time.

After having presented two core reasons behind lack of automation in construction industry, we focus our attention on another related problem. Visual Programming Language (VPL) is utilized with increasing frequency in Architectural, Engineering and Construction (AEC) industries to harness advanced geometry and complex buildings. As a consequence, Sculptor also relies heavily on VPL. In recent developments, VPL has bifurcated into robotic research, where they are being explored as a good tool for robotic programming. While visual programming holds several key advantages justifying this attention, it comes with important limitations for developing commercial-grade software applications.

VPL based parametric modelling has since its introduction in early 2000's seen rapid adoption within the AEC industries for the design and engineering of complex buildings [Mon00]. More recently, parametric design thinking has enjoyed widespread interest within architectural and construction robotic research for the development of custom and experimental workflows [Sch13, Ras15, Gea16]. In complementary notes, core fields within general robotics have also acknowledged key advantages of VPLs in development of robotic applications, causing increasing research interest [SW16, HRSW13, PNBK13]. Some of the identified advantages in this approach are ease of programming, effortless knowledge encapsulation for less specialist developers, efficiency in establishing geometry-intensive workflows and modular architectures [PP13]. While these advantages



(a) Parametric model 1 **(b)** Parametric model 2

Figure 2.3: Examples of visual clutter (wires inside red coloured boxes) and code duplication (blocks inside teal coloured boxes) in parametric models.

are well established, the general scope of application for VPL based parametric modeling has historically been limited to custom or project specific scripts, graphs, functions or prototypical workflows within AEC organizations. As such, investigations into whether they could be applied for development of commercial scale software applications are sparse.

Rick Smith has identified the following as some of the major challenges faced in parametric modelling [Dav13]:

1. Major changes can break parametric models. As it is difficult to foresee all feature requests in the beginning of a project itself, the instability of the model raises some grave concerns when one has to amend it later on to incorporate a request from the customer.
2. It is difficult to reuse and share parametric models. Often, only the original designer is able to operate with a model, as it can be arduous for another one to comprehend the design intent making the system averse to maintenance.

Around 1960s, a large number of major software projects unexpectedly failed in the broader industry, giving rise to the software crisis. As pointed out by Turing award winner Niklaus Wirth, "systems could not be built or delivered on time, bringing some companies to the brink of collapse" [Wir08]. One of the examples is, IBM's ambitious System/360 unification project, led by F. Brooks, which in 1964 was one of the largest software projects ever undertaken. It was years late and costed millions of dollars more than budgeted [Dav13].

In 1968, NATO decided to form a group of scientists to further analyze the problems faced in software engineering. As recalled by Naur and Randell, the talk was centered around "slipped schedules, extensive rewriting, much lost effort, large number of bugs, and an inflexible and unwieldy product" [NR68]. One of the main reasons for this problem, was the fact that software programs written during those days were largely unstructured with GOTO statements littered randomly across the source code.

In many ways, one can relate these findings from NATO, to the problems mentioned by Rick Smith [Dav13]. The analysis of parametric models created by two different Computational Design Specialists (CDSs) (see, Fig. 2.3a and 2.3b), in one of our previous works, is reminiscent of the same issues which led to the software crisis in 1960s. After a careful literature study, we have concluded that the same problem has been highlighted by others as well [Dav14, Jan14]. In our work, we are looking at some of the potential solutions to improve the current situation in parametric modelling.

Chapter 3

Meeting the challenge

Prior to the development of Sculptor, one of our software engineers was tasked with building a robotic application. The result was a huge monolithic system of software which was highly inflexible to changing customer requests and needs. Moreover, it suffered from the curse that only the original developer was able to maintain it. It took 2 years of development time and in no way, it could scale up to supporting other applications. In other words, it resulted in a high C_f.

To overcome the challenges related to cost introduced in the last chapter, we propose increasing the efficiency of custom workflow implementation through deploying a model of higher order knowledge encapsulation, which enables semi-automatic application development and software architecture generalization. The purpose of this effort would be to lower the development costs of implementing bespoke, project specific applications below the cost of manual production, hereby facilitating wide spread use of robotic manufacturing in construction. The presented work addresses the software related implications of this solution.

In other words, we are presenting Sculptor that will enable someone to rapidly design, prototype and develop a robotic application. As a result, one can achieve cost parity even though the aforementioned application is not mass manufactured. This is specifically of interest in construction industry, as a building contractor usually orders only 1-2 robotic units supporting a given process. For any provider of such cells to the industry, it becomes important that a frame-

work exists to develop applications quickly so that the goal of keeping C_f to the minimum can be met.

Additionally, to bridge the knowledge gap of craftsmen, we focused on building intuitive tablet applications to interact with our robot cells so as to make it accessible for everyone. By partial automation of user interface generation, we were able to overcome the cost conundrum associated with the development of said tablet applications.

As mentioned earlier, Sculptor is based on VPL as CDS relies on that technique to build parametric models for a given process. We have seen that commercial-grade software application development using VPL is less *efficient*. For the context of our work, an *efficient* process denotes a flexible software development methodology which creates stable, reusable, performance optimal and maintainable parametric models. In this work, we demonstrate how software design patterns could contribute to further increasing this *efficiency*.

Though the term "design pattern" was an age old idea in the field of architecture, it was probably introduced by Christopher Alexander [AIS77] in a modern sense. He says, "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.". The term was later taken by the software engineering fraternity [GJV94] to describe abstract, well-established forms of program construction. A pattern is fully defined by a name, a problem statement, an abstract solution, and a discussion of implications.

The need for software design patterns in parametric models is explained in [WAK07, Qia09, Zbo15]. It is claimed that it will lead to robust and sustainable models. Following their foot steps, we have implemented six design patterns while programming graph-modelled robotic workflows. Out of six patterns discussed here, five are new to the best of our knowledge, whereas one is a well-known pattern from the Object Oriented Programming (OOP) world, applied to visual programming.

Since 1994, many design patterns have been suggested primarily for software systems based on OOP paradigm since languages based on this paradigm were on an ascendancy during that time. As we carry the concept of software design patterns to VPLs, we need to keep in mind that not all design patterns applicable for object oriented world are relevant to VPLs as these systems follow a Functional Programming (FP) paradigm which is fundamentally different from OOP.

This thesis is based on three publications and these papers are added as appen-

dices to this report. These publications are listed below for easy reference:

1. Title: Adaptive Robotic Manufacturing using Higher Order Knowledge Systems, Venue: Journal of Automation in Construction, Status: Published.
2. Title: Procedural generation of human machine interfaces from graph-modelled robotic workflows, Venue: Journal of Automation in Construction, Status: Submitted.
3. Title: Applying software design patterns for graph-modelled robotic workflows, Venue: Journal of Automation in Construction, Status: Revision.

Chapter 4

Preliminaries

There are mainly two ways of representing a geometric object (G) mathematically viz. parametric and implicit. For all our practical cases, G is either a curve (C) or a surface (S) in a 2D or 3D world.

An implicit representation takes the form $g(\mathbf{p}) = 0$ where $\mathbf{p} = (x, y, z)$ is any point lying on G described by g. With this representation, it is very easy to check whether a point lies on G. However, it doesn't give a mechanism to systematically generate a set of such points. Usually, an implicit representation is constructed so that the sign of $g(\mathbf{p})$ can be used to determine on which side of G does $\mathbf{p}$ lie.

A parametric representation, on the other hand, can be written as $\mathbf{p} = f(\mathbf{s})$ where $\mathbf{s}$ lies within a predefined domain. Let us take the example of a sphere centered around origin having radius r. It can be denoted as $x^2 + y^2 + z^2 = r^2$ implicitly whereas it takes the following parametric form:

$$\begin{aligned} x &= rsin(u)cos(v) \\ y &= rsin(u)sin(v) \\ z &= rcos(u) \end{aligned} \tag{4.1}$$

where $r > 0$, $0 <= u <= \pi$ radians and $0 <= v < 2\pi$ radians. Such a sphere is

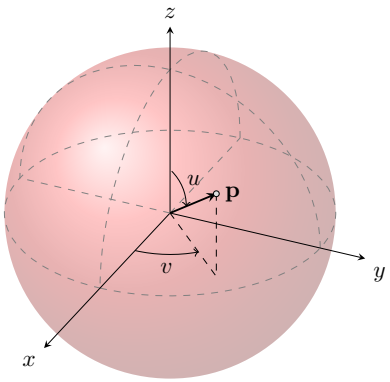


Figure 4.1: Parametric representation of a sphere

depicted in Fig. 4.1. It is worthwhile to note that f is not unique. By changing f, one can arrive at different parametric expressions for the same G.

A common procedure is to use polynomials for f. In such cases, G becomes:

$$\begin{aligned} C(u) &= \sum_{i=0}^{n} f(u)\mathbf{p}_i \\ S(u,v) &= \sum_{i=0}^{n}\sum_{j=0}^{m} f(u)h(v)\mathbf{p}_{i,j} \end{aligned} \tag{4.2}$$

where $C(u)$ and $S(u,v)$ are points on a curve/surface respectively. The parameters u and $v \in [0,1]$. There are $n+1$ and $m+1$ control points in u and v directions. $\mathbf{p}_i$ and $\mathbf{p}_{i,j}$ denote the control points for the said curve/surface.

If $f(u) = B_{i,n}(u) = \binom{n}{i}u^i(1-u)^{n-i}$ and $h(v) = B_{j,m}(v) = \binom{m}{j}v^j(1-v)^{m-j}$ respectively, 4.2 reduces to Bezier representation.

For a Basis spline (B-spline), we have:

$$\begin{aligned} f(u) = N_{i,k}(u) &= \frac{u-u_i}{u_{i+k}-u_i}N_{i,k-1}(u) + \frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}}N_{i+1,k-1}(u) \\ N_{i,0}(u) &= \begin{cases} 1, & \text{if } u_i \le u \le u_{i+1} \\ 0, & \text{otherwise} \end{cases} \\ h(v) = N_{j,l}(v) &= \frac{v-v_j}{v_{j+l}-v_j}N_{j,l-1}(v) + \frac{v_{j+l+1}-v}{v_{j+l+1}-v_{j+1}}N_{j+1,l-1}(v) \\ N_{j,0}(v) &= \begin{cases} 1, & \text{if } v_j \le v \le v_{j+1} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{4.3}$$

where k and l are degrees in u and v directions. The values of u_i and v_j are taken from a sequence called knot vector. $\mathbf{U} = (u_0, u_1, u_2, ...u_q)$ and $\mathbf{V} = (v_0, v_1, v_2, ...v_r)$ are knot vectors in u and v directions where $q = n + k + 1$ and $r = m + l + 1$.

A k degree curve and a (k, l) degree surface can be specified in Non-Uniform Rational B-spline (NURBS) form as:

$$\begin{aligned} C(u) &= \frac{\sum_{i=0}^{n} f(u) w_i \mathbf{p}_i}{\sum_{i=0}^{n} f(u) w_i} \\ S(u,v) &= \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} f(u) h(v) w_{i,j} \mathbf{p}_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{m} f(u) h(v) w_{i,j}} \end{aligned} \tag{4.4}$$

where $f(u)$ and $h(v)$ can be plugged in from 4.3.

On setting,

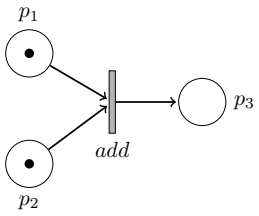
$$\begin{aligned} r_i(u) &= \frac{f(u) w_i}{\sum_{r=0}^{n} N_{r,k}(u) w_r} \\ r_{i,j}(u,v) &= \frac{f(u) h(v) w_{i,j}}{\sum_{r=0}^{n} \sum_{s=0}^{m} N_{r,k}(u) N_{s,l}(v) w_{r,s}} \end{aligned} \tag{4.5}$$

We can rewrite (4.4) in the form,

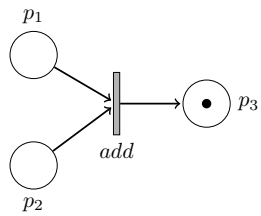
$$\begin{aligned} C(u) &= \sum_{i=0}^{n} r_i(u) \mathbf{p}_i \\ S(u,v) &= \sum_{i=0}^{n} \sum_{j=0}^{m} r_{i,j}(u,v) \mathbf{p}_{i,j} \end{aligned} \tag{4.6}$$

4.1 Petri nets

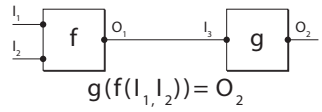
Petri net is a graphical and mathematical modelling tool applicable to many systems. The concept of Petri nets was introduced by Carl Adam Petri in his dissertation submitted in 1962. It is a bipartite directed graph consisting of two types of nodes viz. transitions and places, where edges are either from a place to a transition or vice versa. It is forbidden to connect either a place with another place or a transition with another transition using an edge. Graphically, a place is denoted using circles whereas bars or thin boxes represent transitions. A marking (state) assigns to every place in the graph a non-negative integer.



(a) A sample Petri net where p_1, p_2 and p_3 are places and *add* is a transition. *add* is in enabled state.



(b) Firing of *add* deducted 1 token each from p_1 and p_2 and added 1 token to p_3.



(c) HOKS

Figure 4.2

If a state assigns to a place (p) a non-negative integer k, we can say that p is marked with k tokens. Pictorially, if a particular p holds k tokens, we will draw k black dots (tokens) inside of p. A marking is denoted M where $M(p)$ gives the number of tokens in p. Some interpretations for transitions or places and a formal way to define Petri nets are given in Tab. 4.1 and 4.2 respectively.

In the context of our work, we have used Petri nets to model a Higher Order Knowledge System (HOKS) for parametric design/manufacturing which is a directed graph where data flows between a set of interconnected nodes (ref, Fig. 4.2c). Each node stands for a function (transition) which accepts n inputs (input places), performs some operations on these inputs to emit m outputs (output places). The *transition firing* rule for a Petri net is given by:

1. t_i is said to be enabled if each input place of t_i is marked with at least w tokens, where w is the weight of the edge from the corresponding input place to t_i (see, Fig. 4.2a).

2. A firing of an enabled transition removes k tokens from each input place of t_i and adds k tokens to each output place of t_i as shown in Fig. 4.2b.

4.2 Parametric design and manufacturing

Parametric design/manufacturing is a strategy based on algorithmic thinking that allows one to express design/manufacturing intent. Such intents define relationships between objects that any change to one will propagate to others.

Input places	Transitions	Output places
Pre-conditions	Event	Post-conditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Job or task	Resources released
Buffers	Processor	Buffers

Table 4.1: Some typical interpretations of places and transitions in Petri net.

A sample product is shown in Fig. 4.3a, the associated parametric models encoding the design and manufacturing intent are illustrated in Fig. 4.3b and 4.3c respectively.

The word parametric stems from parametric representation introduced earlier and refers to the use of certain variables that can be altered to manipulate the end result of an equation or system. One of the distinguishing features of such a paradigm is that, trace of how a particular geometry is modelled is available at any given time.

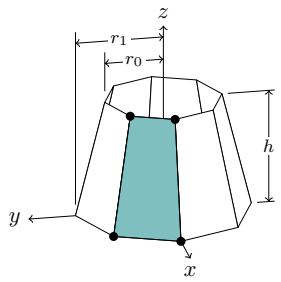
It might be a tad surprising to know that parametric design existed even before the dawn of digital computing. For example, Antoni Gaudí, a famous Spanish architect employed these techniques at the end of the 19^{th} century to design the church of Sagrada Família. To increase longevity of the building, it was important that erected columns, remain in a state of pure compression.

He traced the outline of the church on a wooden board at 1:10 scale, which he then fixed onto the ceiling of a small house next to the construction site. He hung cords from the points (anchor points) where columns were to be placed. Next, he tied small sacks filled with pellets to these strings. This model has all the characteristics of a parametric design with a set of input parameters:

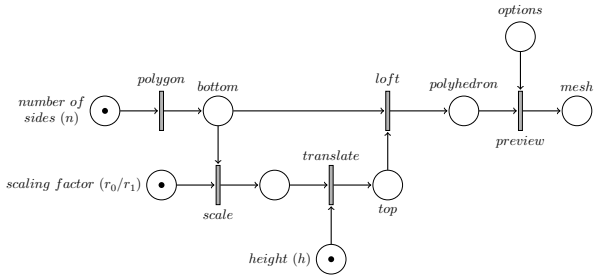
1. Length of the string

A Petri net (N) is given by (P, T, E, W, M_0):
$P = \{p_1, p_2, ..., p_m\} \mid p_i$ is a place,
$T = \{t_1, t_2, ..., t_n\} \mid t_i$ is a transition,
$E = \{e_1, e_2, ..., e_q\} \mid e_i$ is an edge between p_a and t_b,
$W = \{w_1, w_2, ..., w_q\} \mid w_i$ is the weight associated with e_i,
$M_0 = \{n_1, n_2, ..., n_m\} \mid n_i$ denotes the initial marking for p_i,
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$

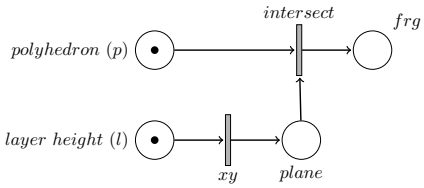
Table 4.2: Formal definition of a Petri net.



(a) Visualization of the target product which is a polyhedron (p) with a height h in the view port. r_0 and r_1 are the respective distances measured normal to the axis of symmetry (z axis).



(b) Petri net representation of the parametric design associated with the polyhedron shown on the left. It is responsible for rendering the target geometry in the view port. An user of the tablet application can interact with the geometry by selecting a surface, a curve or a dimension object. In this example, the selected surface is shown in teal.



(c) Petri net representation of parametric manufacturing which generates a tool path for additive manufacturing process by intersecting the given polyhedron (p) with a plane parallel to XY plane. p and layer height (l) are parameters to the model. As long as $0 \leq l \leq h$, the model will output a valid tool path curve via the node frg.

Figure 4.3

2. Location of the anchor points
3. Weight of the pellets

It has a set of outcomes namely the location of anchor points for the columns. It is determined by gravity and follows laws of motion from Newton. The force of gravity acted on the strings to create the shape thereby acting like an algorithm driven by physics rather than as an algorithm expressed using digital means. By analyzing the model, Gaudí was able to determine the exact location for the anchor points so that the resulting archs will experience only compression. In this report, the words Petri net and parametric model are used inter changeably.

Chapter 5

Sculptor

Sculptor provides a framework which will help to quickly develop the software required to control the robotic unit introduced in chapter 1. It comprises of 3 modules developed in-house as shown in Fig. 5.1 viz.

1. Canvas - A Unity based library which is responsible for creating enjoyable user experience (UX) for any robotic application made using Sculptor. Unity is a cross-platform game engine developed by Unity Technologies. Additionally, it also helps the CDS to contrive complex parametric models.
2. DaVinci - A computational geometry kernel created using C# programming language.

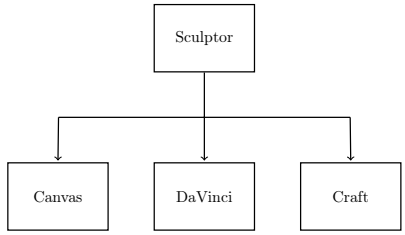


Figure 5.1: Sculptor architecture

3. Craft - A module created for robot planning and control based on Robot Operating System (ROS). ROS is a set of software libraries and tools which will help you build robot applications.

5.1 Canvas

An User Interface (UI) is the space where interactions between humans and machines occur. The objective of this transaction is to allow efficacious operation of the machine by a human operator, whilst the machine simultaneously provides feedback that aids the operator in decision-making process. As a general rule of thumb, for a good user experience, the operator should be able to provide minimal input to achieve the desired output, and also that the machine minimizes undesired outputs to the user.

In our endeavour to achieve the above stated rule, we try to present the craftsmen with intuitive tablet interfaces to operate robotic systems (refer, Fig. 5.2). Massive scale automation of construction industry will heavily depend on the simplicity of such interfaces as the industry is dominated by a workforce which is ill-suited to deal with complicated technology.

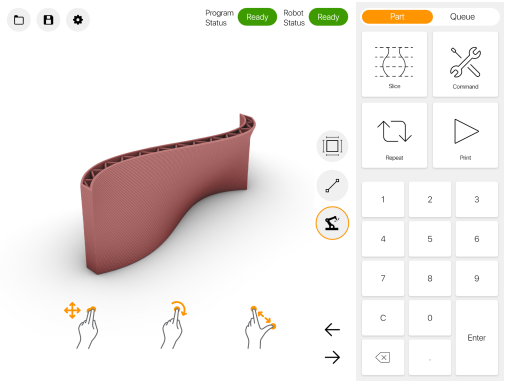


Figure 5.2: In the given interface created using Canvas, product is visualized within the view port. Touch gestures can be used to interact with the 3D model. One can do basic file operations such as opening, loading, saving of projects using the icons on the top left corner. Panel on the right hand side presents the user with actions that can be performed on the 3D parametric model or the robot. A robotic action could be as simple as asking the robot to print.

In the latest edition of Sculptor, a CDS can create parametric models using Bolt which is a visual programming interface from Unity (see, Fig. 5.3). As a result, Unity acts as an Integrated Development Environment (IDE) for developing robotic applications using Sculptor.

5.2 DaVinci

Each of the transitions used to build the parametric model, comes from DaVinci. It is largely a collection of logical, arithmetic, geometric and data manipulation operators.

Data manipulators can work on data trees, to alter their structure. For example, a merge operator can coalesce a set of data trees into a single tree. DaVinci uses polynomials to represent curves and surfaces as explained in the previous chapter. Being a computational geometry kernel, it has lots of transitions relating to geometry. We are discussing one of the algorithms from the kernel here so that reader might be able to get a better picture about DaVinci.

One of the methods by which a CDS can specify a NURBS surface is by specifying four boundary curves of the surface. Since we are going to specify a surface with these curves, two of the curves need to be in the u direction whereas the other two should be in the v direction. These curves are of the form as specified in (4.6) and are given by $C_1(u)$, $C_2(u)$, $C_3(v)$ and $C_4(v)$.

Furthermore, the curves should satisfy the following compatibility conditions:

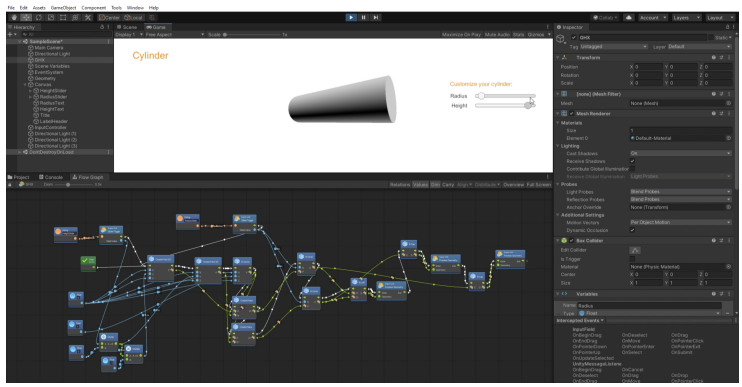


Figure 5.3: A cylinder is created using parametric modelling approach within Unity and Bolt.

1. $C_1(u)$ and $C_2(u)$ should be defined in the same $\mathbf{U}$. In a similar way, $C_3(v)$ and $C_4(v)$ must have a common $\mathbf{V}$.

2. $C_1(u)$, $C_2(u)$, $C_3(v)$ and $C_4(v)$ should have intersection points given by:

$$\begin{aligned} \mathbf{s}_{0,0} &= C_1(u=0) &&= C_3(v=0) \\ \mathbf{s}_{1,0} &= C_1(u=1) &&= C_4(v=0) \\ \mathbf{s}_{0,1} &= C_2(u=0) &&= C_3(v=1) \\ \mathbf{s}_{1,1} &= C_2(u=1) &&= C_4(v=1) \end{aligned} \tag{5.1}$$

3. $\deg(C_1(u)) = \deg(C_2(u)) = k$ and $\deg(C_3(v)) = \deg(C_2(v)) = l$

A surface can be formed by these curves by using a bi-linearly blended Coons patch defined in the following form:

$$S(u,v) = R_1(u,v) + R_2(u,v) - T(u,v) \tag{5.2}$$

where $R_1(u,v)$ is a ruled surface between $C_1(u)$ and $C_2(u)$, $R_2(u,v)$ is a ruled surface between $C_3(v)$ and $C_4(v)$ and $T(u,v)$ is a bi-linear tensor product surface.

$$T(u,v) = \begin{bmatrix} 1 & u \end{bmatrix} \begin{bmatrix} \mathbf{s}_{0,0} & \mathbf{s}_{0,1} \\ \mathbf{s}_{1,0} & \mathbf{s}_{1,1} \end{bmatrix} \begin{bmatrix} 1 \\ v \end{bmatrix} \tag{5.3}$$

Since $R_1(u,v)$ and $R_2(u,v)$ are ruled surfaces, they have a degree of $(k,1)$ and $(1,l)$ respectively.

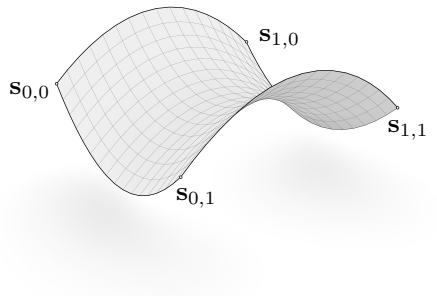


Figure 5.4: A bi-linearly blended Coons surface.

5.3 Craft

It is an assemblage of tools and libraries based out of ROS that aim to simplify the task of creating complex and robust robot behavior across a wide variety of applications. The two important services inside of Craft are motion planning and execution. From the parametric model defined by the CDS, we will be able to glean the target poses for successful execution of a given task. The motion planning service can accept these target poses along with various models and compute a collision free trajectory which can be physically realized on a robotic manipulator by the execution service. The overview of Craft module is shown in Fig. 5.5.

The motion planning service is based on MoveIt which works with planners through a plugin interface. As a result, MoveIt can communicate with and use different motion planners, making it inherently extensible. OMPL (Open Motion Planning Library) is an open source motion planning library that has implementations for randomized motion planners. MoveIt is integrated with OMPL and utilizes the planners from that library as its default set of planners.

MoveIt is primarily meant for performing "free space" motion where the goal is to move a robot from point A to point B. In these cases, we don't particularly care about how the robot reaches B from A. They form only a subset of frequently performed tasks. In manufacturing tasks like cutting or polishing, the tool has to follow a specified path to complete the assigned task. Descartes is a motion planner meant for moving a robot along some process path and is integrated with MoveIt.

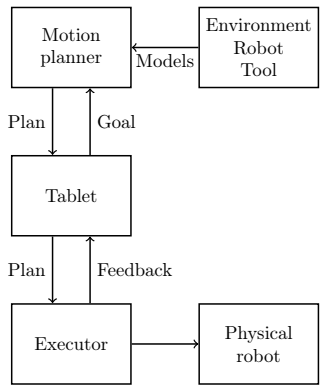


Figure 5.5: An overview of Craft.

It is also possible to provide the following constraints to MoveIt which will be considered for planning:

1. Position constraints - It can be used to restrict the position of a certain link to lie within a particular region of space.
2. Orientation constraints - It allows to specify that the orientation of a link must lie within a given range of roll, pitch or yaw.
3. Joint constraints - It sets out that a joint must be within the allowable range.

Chapter 6

Higher Order Knowledge System (HOKS)

In this chapter, we will take a deeper look at HOKS which is the primary enabler for swift robotic application development and semi-autonomous generation of tablet interfaces. The idea is elaborated on one of our papers which is attached to this thesis as an appendix (check, Appendix A).

Our knowledge encapsulation is initialized with denoting an application to entail the following components:

1. a parametric template or product model (PM), which holds a customizable design of a component, part sub-assembly or entire building design;
2. a fabrication model (FM) which, as a function of PM generates the corresponding tool path targets;
3. an execution node (EN) entailing a digital representation of the robotic system with incorporated planning module to ensure motion safety;
4. a physical robotic cell (PRC) equipped with a set of predefined end-effectors for executing one or a combination of several processes. The robotic system is containerized or mounted on an autonomously moving base for modular deployment;

5. a HMI, which exposes key parameters of PM, FM and EN to the user through a GUI.

The entirety of the application is process or product specific, allowing for a simple expression of controls. Hereby, the application enables a non-expert user, i.e. a construction worker trained in craft with no robotics experience, to (a) customize a part design on the fly or retrieve an externally customized part design and (b) safely execute production of the customized instance of that part.

The concept of such a containerized set up with a tablet interface can be seen in Fig. 2.2 where an user can be seen interacting with PRC using a HMI.

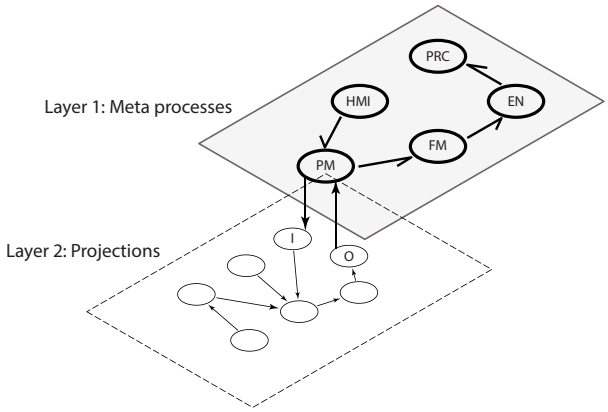
6.1 Application model

Pursuant to the proposed encapsulation strategy, the robotic application can be visualized as a bi-layered structure (refer, Fig. 6.1a) and the corresponding Petri net models are shown in Fig. 6.1b and Fig. 6.1c. Petri nets have previously been shown useful for knowledge abstraction [HSZL16, HLY16]. The first layer entails a meta-representation consisting of five components and the interactions between them.

The second layer corresponds to a topologically variable sub-system, which models the product or process specific aspects of the application (see, Fig. 6.1a). Since these sub-systems may model any variability within the constitutional parameter space including product type, material combination, machining process, location or robot system configuration, the technical challenge becomes projecting a non-finite variety of sub-system topology to a finite meta representation (layer 1) under the constraint that model should remain valid. To solve this, we implement the following method as shown in Fig. 6.2.

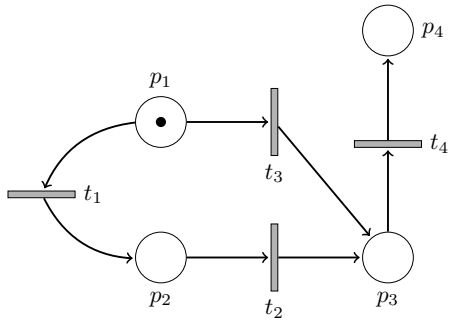
6.2 Layer 1

Each component in the knowledge encapsulation strategy viz. PM, FM, EN, PRC and HMI finds a representation in this layer to completely define various robotic applications. Associated to the components, there are transitions or projections. Depending on the current state of the application, a particular route gets active. A route is an event handling pipeline comprising of a subset



(a) Bi-layered structure

(b) Petri net representation of the bi-layered structure. t_h, t_p, t_f, t_e and t_r are transitions (projections) associated with HMI, PM, FM, EN and PRC respectively. $t_{failure}$ is the error handling routine.



(c) t_h, t_p, t_f, t_e and t_r are further represented using individual Petri nets and resides in layer 2 of our bi-layered structure. For example, for a given application, the role of a CDS is to model t_p and t_f whereas a robotic engineer will be able to build up t_e. A sample Petri net which can be used to model one of the projections is shown here. t_1, t_2, t_3 and t_4 are individual transitions which make up a function composition model.

Figure 6.1: Knowledge encapsulation strategy

of components. At any given time, there will always be only one active route. For example, when a production user is initiating a request to fabricate a piece to the robot, the active route is shown in Fig. 6.1a and Fig. 6.1b.

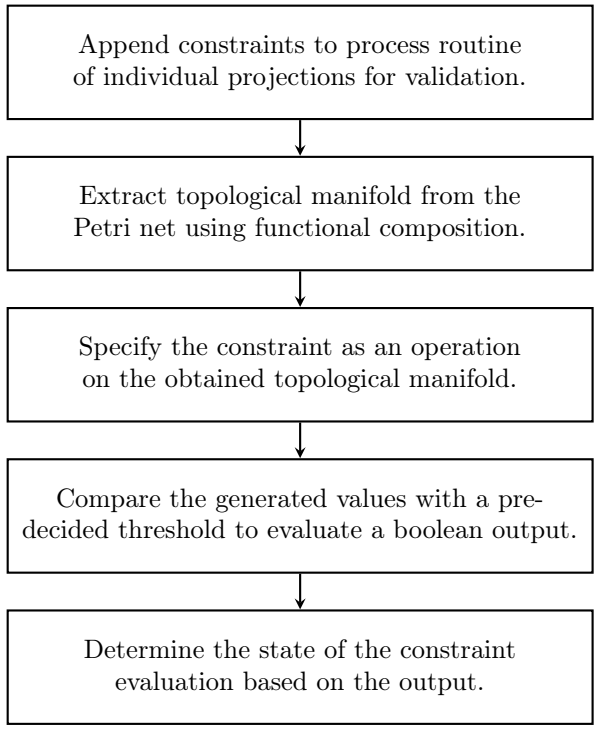


Figure 6.2: Constraint Checking Workflow

Due to the simple nature of interaction possible, it is suitable for a low skilled production user to work with this layer. He or she can tweak input parameters to achieve a specific valid product configuration and further instruct the robotic system to manufacture the product confirming to the specified parameter values.

6.3 Layer 2

As mentioned, there is a transition associated with each one of the components in layer 1. They encapsulate the complex nature of computations required to achieve the functionality of a meta node. For example, PFM (refer, Table 6.1) is primarily tasked with the computation of tool path to service the fabrication request from the user.

Expert user knowledge is required to model projections. A CDS will be able to define the transitions for PM and FM whereas a CAM operator or a robotic

Projection (transition) name	Associated meta node	Input	Output
PPM (t_p)	PM	Numeric/Boolean	Fabrication task
PFM (t_f)	FM	Fabrication Task	Robot task
PEN (t_e)	EN	Robot Task	Configuration task
PPRC (t_r)	PRC	Configuration task	Physical motion
PHMI (t_h)	HMI	Human gestures	System action

Table 6.1: Associated projections.

engineer is required for EN.

Every transition has a predefined input/output type. Since it's most of the times difficult to describe the relationship between the output and input with a single function, it is expressed using function composition which is an operation that takes two functions f and g and produces a function h such that h(x) = g(f(x)) (see, Fig. 6.3a and Fig. 6.1c).

The details of projections is provided in Table 6.1. A fabrication task contains a NURBS model, tool data and other process parameters whereas a robot task comprises of a tool path, process speed along with other specifications including Robot Model (RM), Environment Model (EM), Tool Model (TM), Work Piece (WP) and Operation Sequence (OS). The configuration task includes the position, velocities and accelerations of all the robot joints as a function of time.

Projections are further represented via Petri net. They are also composed of places and transitions just like any other Petri net. They involve a special set of transitions called constraints. An evaluation of a projection is said to be valid, if and only if all the constraints attached to the projection are satisfied.

6.4 Constraints

In Fig. 6.2, a constraint is expressed as an operation on a topological manifold otherwise referred to as a manifold. Let an arbitrary set M represent the point cloud which resides either in task or configuration space (see, Fig. 6.3b) and O be the chosen topology on M. For robotic manufacturing, it's a necessary

condition that the topological space (M, O) forms a manifold. A topological space is called a d-dimensional manifold if $\forall \mathbf{p} \in \mathrm{M}$ there exists an open set U containing $\mathbf{p}$ where the following continuous maps exist:

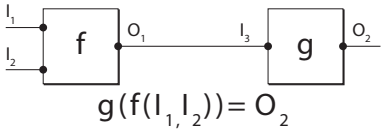
1. $x : \mathrm{U} \mapsto \mathrm{R} : \mathrm{R} \subseteq \mathbb{R}^d$
2. x^{-1}

This is called the charting condition.

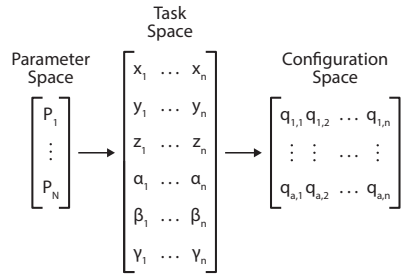
For example, consider the surface of a torus as the NURBS model included in the fabrication task, let the points in this model be M where $\mathrm{M} \subseteq \mathbb{R}^3$. We can choose standard topology as O. For any $\mathbf{p}$ in this model, we can find a U which is nothing but the open neighbourhood around the point. Since $\mathrm{U} \subseteq \mathrm{M}$, it can be represented as a NURBS surface, following which it can easily be verified that there exists a x which maps every point on U to R embedded in $\mathbb{R}^2$ in a continuous invertible manner where R forms the parameter space for U. Hence we can say that (M, O) is a manifold as it satisfies the charting condition.

There are many artifacts whose presence will convert the model into a non-manifold geometry, viz. a curve that bifurcates at a point, a set of surfaces forming an open volume or an edge shared by more than 2 surfaces, because around such artifacts the continuity of x gets broken.

A robotic arm ζ having an m dimensional configuration space C is used to process the tool path specified in the robot task. Since ζ is made of several objects connected by joints, it is subjected to kinematic constraints. These are constraints which restricts an object or a collection of objects from rotating or



(a) Function composition



(b) Topological spaces

Figure 6.3: Graph operations

translating freely in the workspace. There are two types of kinematic constraints viz. holonomic and non-holonomic.

An atlas has been defined on C and any configuration $\mathbf{q}$ of ζ is represented by a list of m coordinates $(q_1, q_2, ..., q_{m-1}, q_m)$ in some chart of the atlas. A scalar constraint of the form:

$$F(\mathbf{q}, t) = 0 \tag{6.1}$$

where F is a smooth function with non-zero derivative, is called a holonomic equality constraint. More generally, there may be k such constraints where k $\leq$ m. Typically, such equality constraints allow us to map a manifold from task space to C. Undesirable collisions as well as out of reach scenarios could be modelled as holonomic inequality constraints of the form:

$$F(\mathbf{q}, t) \leq 0 \tag{6.2}$$

and can be expressed as an operation on the manifold in task space. Constraints can be modelled either in task or C. As manifolds are inherently differentiable, constraints involving velocities or accelerations can be plugged in.

Chapter 7

Procedural generation of Graphical User Interface (GUI)

Following the discussion on knowledge encapsulation, we would like to discuss a bit on the semi-autonomous generation of user interfaces as it plays an important role in achieving our goal of keeping C_f to the minimum. Further details of the same can be found on our paper which is attached to this thesis as an appendix (see, Appendix B).

As discussed already, a robotic application is modelled as a Petri net by a CDS, which will have k inputs pre-decided by the designer of the system. The model is created by chaining transitions together to define the relationship between the input and output places. Two transitions can be chained by way of a connection. A connection is a directed link where data can flow only in one direction. It has two ends viz. origin and destination. Data can flow only from an origin to destination. The origin should be attached to the output place of a transition whereas the destination should be tied up to an input place of a transition.

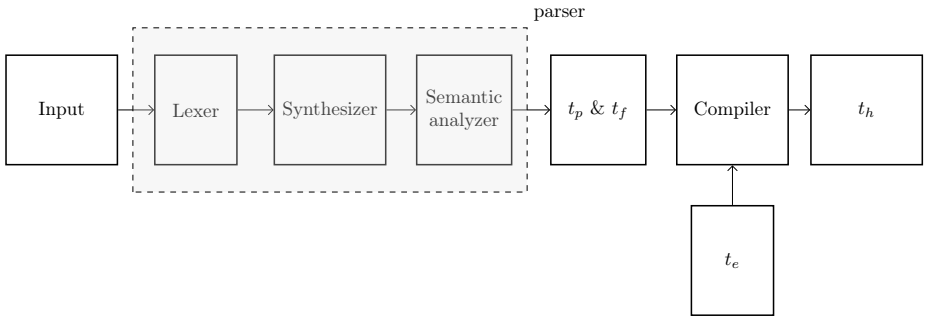


Figure 7.1: Compilation pipeline

7.1 Compilation pipeline

During the semi auto-generation phase of the GUI, the provided input file goes through a compilation pipeline (see, Fig. 7.1) which has the following steps:

1. Input - specifies the parametric design in the form of a XML/JSON file. It mentions all the transitions and places contained in t_p and t_f and their relationships with each other.

2. Parser - consumes the given input file and converts it into a process model which contains two Petri nets viz. t_p and t_f. During this phase, the file passes through 3 stages:

   (a) Lexer - the input file is split into meaningful symbols at this stage. In case any unrecognized symbols are found in the model, an error will be reported.

   (b) Synthesizer- The output of this activity will be a network of transitions and places where the connectivity information has been gleaned from the file and synthesized to a process model. In other words, each transition and place will know their predecessors and successors after this step.

   (c) Semantic analyzer - the model is analyzed for its correctness by verifying whether all the connections are valid. All the connections are type checked.

3. Compiler - accepts the synthesized process model from the previous step along with t_e to create t_h. t_e contains robot, environment and tool models required for creating collision free motion plans for execution on t_r which is the Factory on the Fly™.

t_h is a tablet based interface which holds a) a parametric CAD model of a given product with parameters accessible to customize the product within predefined boundaries; b) a fabrication menu within this interface, which holds options for the user to select pieces for production and submit that to t_f which parametrically deduct the necessary tool paths and operations within preset bounds.

All the sources inside of t_p and t_f will be exposed in t_h via UI elements. The UI element could be a text box, an active dimension object, slider etc. A functional UI can be achieved with such simplification. It is important to note that the goal is to achieve an unified user experience across Factory on the Fly™ units.

The extracted tool path is transmitted to t_e and t_r for planning and execution.

7.2 Implementation

The input XML/JSON file can be created by a standard software tools intended for working with parametric workflows. A proprietary computational geometry kernel named DaVinci was created which can be accessed via Grasshopper as well as Unity. Currently, designers use Grasshopper editor to prepare the XML/JSON file which is termed as the development workflow. We are currently exploring to migrate to Bolt which is a visual programming environment integrated with Unity. Once the input file has been prepared, it is ported to Unity which converts the provided file to Petri net models (t_p and t_f) through a custom developed parser implemented using C# programming language.

For specifying t_e, we are relying on MoveIt motion planning framework provided inside Robot Operating System (ROS). Further, Unity is used to create the iOS application in a semi-automatic manner.

Each Factory on the Fly™ unit has 3 computing devices:

1. A tablet which holds t_h, t_p and t_f.

2. A mini computer which is a PC with a small form factor. t_e resides in this device. It is powered by Ubuntu 20.04 and ROS Noetic.

3. A robot controller which receives instructions for execution on the physical robot from the above mentioned mini computer.

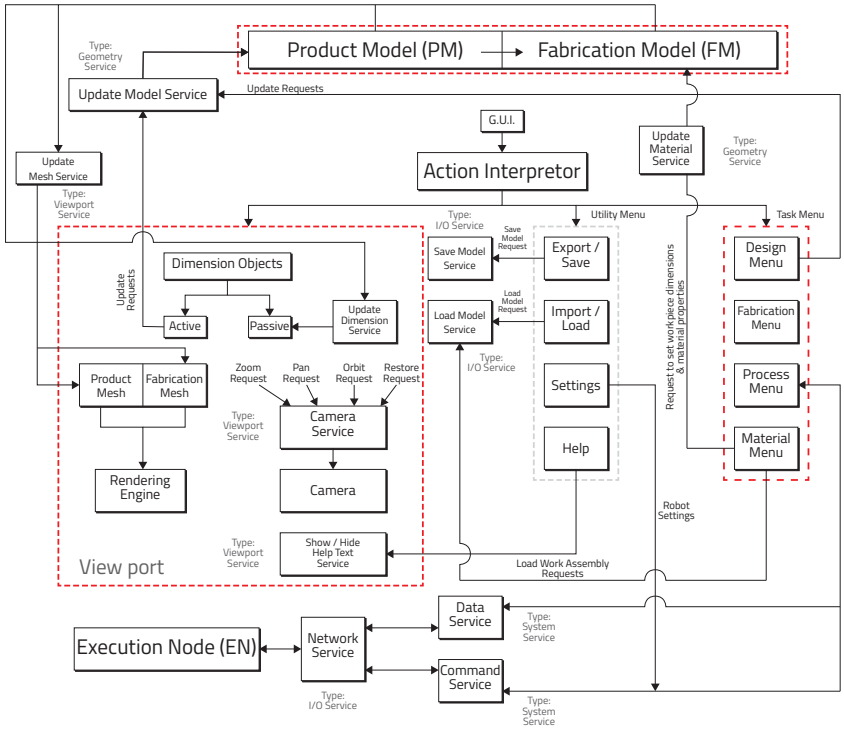


Figure 7.2: Architecture diagram for user interface

7.3 GUI architecture

The architecture of the GUI is shown in Fig. 7.2. The interface is an event based system which responds to gestures from the user. The advantage of modelling the application using the proposed method is that various functional behaviors can be achieved by performing basic operations on the graph which makes the implementation universal across applications. As a result, we will be able to go for partial automation of the generation process. Some of these behaviors are discussed below:

1. A fundamental property of the interface is that it should allow an user to customize a particular parametric model. For this purpose, sources are exposed as editable parameters to the user. On changing a value of a particular source, a series of transitions are fired resulting in the re-computation of the model. The output of this activity is stored in sinks.

The safeness can be ensured by keeping the parameters within the bounds.

2. User might want to save/load a particular state of the application. It can be easily achieved by storing the values associated with all the sources in the Petri net. On re-loading the saved state, values for all the other places in the Petri net can be computed by triggering a series of transitions as prescribed by the system specification.

3. Under some situations, the user would like to undo/redo a specific operation. In a Petri net based system, this can be accomplished by recording the values for sources so that one can move between states.

4. Monitoring of robot states.

Chapter 8

Applying software design patterns

Now let us take a brief look at some of the software design patterns proposed by us to improve parametric modelling. The complete work can be found on our third paper which is attached to this thesis as an appendix (see, Appendix C).

It is to be noted that ability to decide which design patterns can be applied to in a given situation comes with practice. In general terms, it follows the idea of factoring out the commonalities between various problems and abstracting the solution so that it can be used in other situations as well. As an example, let us look at Observer pattern popular among members from OOP community. It can be used under any situation where one process (publisher) is producing some information which many other processes (subscribers) will be interested in listening to. Some of the concrete examples of its implementations are:

1. Pick and place robot - A camera fitted to the robot arm publishes the pose of the object to be picked up by the arm as and when it enters within the reachable workspace of the robot.

2. If someone whom you are following in Twitter or Facebook publishes a new post, you will be notified about the same.

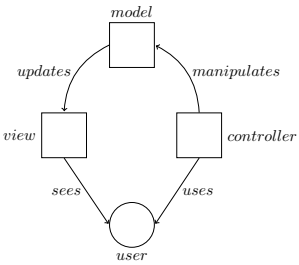


Figure 8.1: Schematic representation of MVC

In the above example, if one chooses not to see beyond the specificity of a given problem, it will be difficult to see the commonality between the two cases. After analysing parametric design models created by us, we have identified some patterns and categorized them into three groups viz. functional, relational and performa. It is not an exhaustive list and more patterns may be recognized as we continue our exercise in future.

A design pattern is introduced by a **Title**, **What**, **Use when**, **Why** and **How** as is followed in [WAK07]. The **Title** is a short name given to refer to a pattern. **What** provides a short description of the pattern. **Use when** mentions about the situations under which the said pattern is relevant to be considered and **Why** signifies the inspiration for using the pattern and sketches the associated benefits. **How** refers to the internal details on how they can be implemented within the given context.

8.1 Functional patterns

Functional patterns deal with breaking down the parametric models into simpler logical sub-units. It will help to reduce unnecessary coupling between nodes, improves readability and makes the graph more maintainable. We introduce and discuss three functional patterns namely Model-View-Controller (MVC), Design-Plan-Monitor (DPM) and adapter in this section.

8.1.1 MVC

1. **What:** Organizes the transitions and places into three collections, namely model, view and controller (refer, Fig. 8.1). A collection is simply a bevy

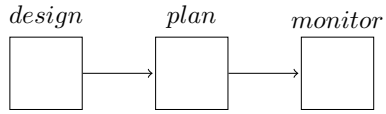


Figure 8.2: Schematic representation of DPM

of transitions and places.

2. **Use when:** You are developing a complex Petri net to model an interactive parametric process involving visual feedback. The said process can be controlled via a set of parameters.

3. **Why:** Most interactive robotic applications can be conveniently subdivided into model, view and controller so as to avoid unnecessary coupling between the various transitions. Model holds the underlying geometry data and drives the application. It can be visualized through the view. Transitions present in the controller will let the user interact with the model.

4. **How:** The model is a group of nodes which defines the geometry or tool paths for a parametric design. For instance, the model nodes in the parametric design might produce a mesh or a smooth surface defined in terms of vertices or control points. The view is formed by a cluster of nodes which determines the visualization aspects of the model, like the thickness of a curve, style or colour of a surface. A given model may have multiple associated views viz. plan, elevation, detailed etc. The controller is a set of interactive widgets that correspond to source nodes. The controller allows the user to change the parameters governing the geometry defining nodes in the model.

8.1.2 DPM

1. **What:** Organize the transitions and places into 3 collections namely design, plan and monitor as shown in Fig. 8.2.

2. **Use when:** If you are developing a robotic application which involves the design on the fly feature using Petri nets.

3. **Why:** In many of the fabrication centered robotic applications from the construction industry, it is required that the user should be able to customize the part being manufactured. A strategy is to expose some design/fabrication related parameters to the production user so that he or she can tweak the parameters to achieve the desired customization. It is

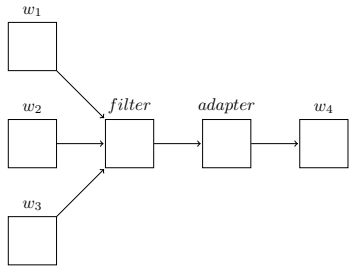


Figure 8.3: Schematic representation of adapter - w_1, w_2 and w_3 are upstream workflows whereas w_4 is the singular downstream workflow. At any given moment, $filter$ lets output from either w_1, w_2 or w_3 to pass through. *adapter* massages the outputs from w_1, w_2 or w_3 to a form which is consistent with the expected input of w_4.

our proposal that a neat way of handling such parametric workflows is to decouple the process into three sections viz. design, plan and monitor.

4. **How:** It can be implemented in the following way:
   (a) **Design** - Specifies the design of the product to be manufactured.
   (b) **Plan** - From the given design of the product, a machining strategy is determined. It involves applying lead in/out, determining the process speed depending on material properties, deciding when to turn on/off the tool and extracting tool path from the given product geometry. Further, robot motion will be planned taking into account the collision matrix. The output of this step will be a task which can be executed on the robot.
   (c) **Monitor** - Once the robot starts to execute the task, the cell has to be continuously monitored for safe operation. Human safety is of paramount importance. In case of any safety sensor breach, the robot is stopped. One can include further recovery strategies in this section to handle other error scenarios.

8.1.3 Adapter

1. **What:** Adapt the outputs of preceding workflows to match the input structure of the succeeding one as depicted in Fig. 8.3.
2. **Use when:** If there are multiple upstream workflows that converges onto a singular downstream workflow, it is important that all these upstream workflows pass on a consistent output form - one that can be acted upon by

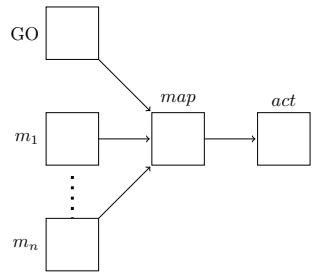


Figure 8.4: Schematic representation of mask where GO is the geometry object, $m_1, .., m_n$ are masks to be applied to GO.

the downstream workflow. In many cases, individual upstream workflows will be generating outputs in a different form.

3. **Why:** It helps to avoid duplicating the downstream workflows to suit the output form from each of the individual upstream workflows.

4. **How:** An adapter workflow is included between the said workflows, to massage the outputs of the upstream workflows such that they are synchronous with one another and match the input form of the succeeding workflow.

8.2 Relational patterns

Relational patterns aid us to better organize the data models within the parametric design workflows to represent the relations between them accurately. We are only talking about one relational pattern namely mask here.

8.2.1 Mask

1. **What:** Maintain a mask that will be used to qualify the underlying data (see, Fig. 8.4).

2. **Use when:** In most parametric workflows, each Geometry Object (GO) will have metadata associated with it. Masks can be used to store such metadata. Typically, information like colour, material so on and so forth can be stored using masks.

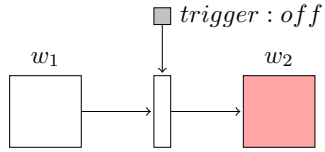


Figure 8.5: Schematic representation of isolator

3. **Why:** Usually in OOP based languages, under these scenarios, a custom class can be defined which includes the GO with all its associated metadata. However, in FP based languages, we cannot create such custom classes. Hence, the masks need to be stored as a separate data structure, so that a downstream workflow can associate the GO with a particular mask and perform some action on the GO accordingly.

4. **How:** Masks are stored using separate data structures, which have the same form as that of the ones holding GOs. They can be either constants, external inputs or be generated dynamically within a workflow. There can be multiple masks, each pertaining to one aspect of the geometry.

8.3 Performa patterns

These patterns help us to reduce latency in a given workflow to improve performance. We were able to come up with two patterns belonging to this category viz. isolator and cache. Each of these patterns is discussed below in greater detail.

8.3.1 Isolator

1. **What:** Isolate a particular workflow from the upstream workflow logically as shown in Fig. 8.5.

2. **Use when:** In many situations, we will encounter downstream workflows which shouldn't get recomputed at every instance of a change to the upstream workflow for various reasons. We can use this pattern as a bridge between two workflows to achieve the same.

3. **Why:** If there are computationally heavy downstream workflows within a parametric workflow, users will experience a latency while interacting with the parameters of the upstream workflow. Hence, designers may

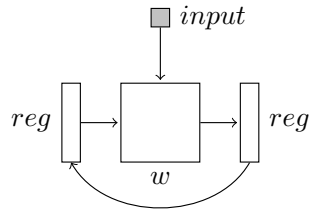


Figure 8.6: Schematic representation of cache

want to strategically defer the recalculation of such dependents so as not to hamper the user experience.

4. **How:** An isolator component can be developed which will collate and store the outputs from the upstream workflow. It will be relayed on to the downstream workflows, only on request.

8.3.2 Cache

1. **What:** Store results from a single execution of the Petri net (see, Fig. 8.6).

2. **Use when:** It can be used when one has to use the results from the previous iteration of the program during the current run.

3. **Why:** If the computation of a Petri net introduces latency due to the time complexity of the algorithm, one may be able to speed up the process by caching the output from the previous run under certain situations. By using the cached value, one can perform the current execution as an incremental operation to enhance user experience.

4. **How:** A register component can be developed which can be used to stow the outputs from the Petri net.

Chapter 9

Case studies

As mentioned before, the development of Sculptor followed a case study based approach wherein the framework was used to develop multiple robotic applications. The details of our experiments are presented here to the interest of the reader.

9.1 Application 1: Abrasive wire-cutting

In this application, the goal is to manufacture moulds for concrete casting. The results are demonstrated on staircase moulds which are one of the most advanced mould types among regularly produced concrete components. Today's method for producing a mould for concrete casting entails the following:

1. A 2D drawing of the requested design is received by the mould supplier.
2. This is translated by a CAD specialist into a production drawing.
3. The production drawing is used by the CAM specialist to program a CNC machine to manufacture the mould parts in timber.
4. The manufactured parts are assembled by a craftsman (typically a mould carpenter) to produce the moulds.

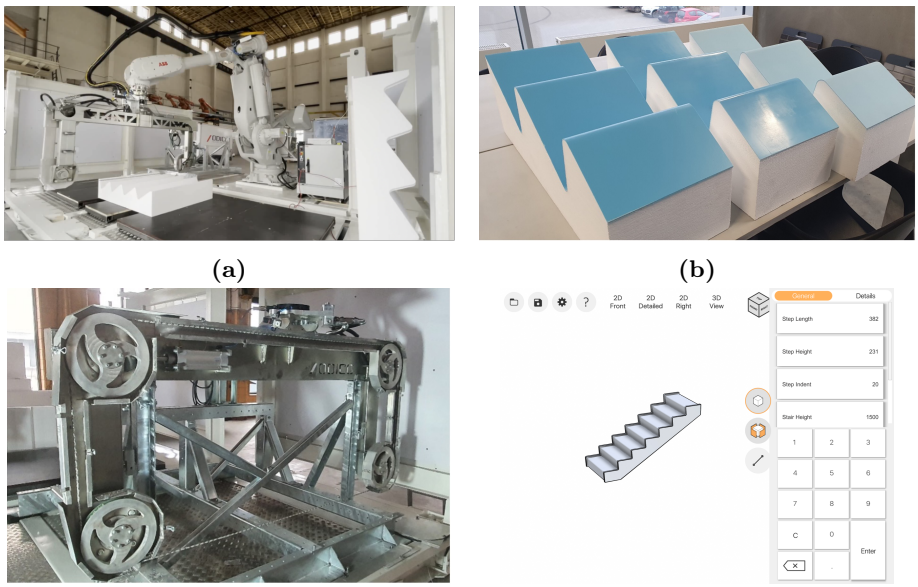
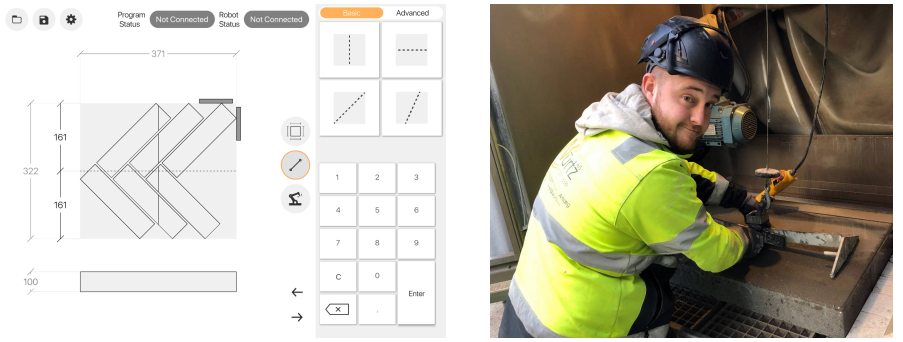


Figure 9.1: (a) Factory on the Fly™ unit; (b) samples of surface coated EPS moulds for staircase; (c) hydraulically driven abrasive wire-cutting end-effector and (d) auto generated tablet interface.

The entire operation takes typically between 1.5 - 2 working days and relies on 3 specialist competences.

We deployed the developed framework to build the following alternative: a parametric staircase model was developed, which holds the relevant variables to accommodate most common staircase dimensions and detailing. The model is represented on a tablet, which holds a design menu for altering parameter values; and a fabrication menu, which shows the mould configuration of the customized design, and enables the user to submit mould parts to fabrication. The GUI is shown in Fig. 9.1d.

The above method was implemented for an industrial, multi-process formwork production unit. The unit consists of a shipping container equipped with an IRB 8700 manipulator as depicted in Fig. 9.1a. It supports three operations viz. abrasive wire-cutting, milling (for edges) and surface coating with resin. Machining time for a staircase mould created out of a bounding box with dimensions 2400 $\times$ 1200 $\times$ 300 mm was around 30 minutes with an additional 14 minutes for surface coating. It produces a formwork piece with same level of surface smoothness, precision and durability as a timber formwork; however



(a) (b)

Figure 9.2: Auto generated tablet interface for sawing application along with the corresponding Factory on the Fly™.

at much reduced production time. The finished EPS mould (see, Fig. 9.1b) is ready for concrete casting. It is to be noted that in our current iteration, we only supported abrasive wire cutting process through the tablet interface in the interest of development time. The other processes were carried out separately. In the future, we envision exposing all processes via the tablet. Hereby, we have now simplified the workflow to involve only the following competences to use the robotic unit:

1. General know-how of operating an iOS tablet application with a complexity level comparable to other commonly available iOS applications.
2. The ability to read dimensions from a provided 2D drawing and inputting the corresponding values in the design menu. Both of these tasks can be achieved in our system with simple instructions which require no specialized training.

9.2 Application 2: Sawing

For further evaluation, framework was used to quickly develop a robotic application for tile sawing operations. In contemporary paving works, concrete and natural stone tiles are often adapted onsite through manual sawing. Adaption is typically required either because standard tiles were not available for the particular dimensions required for the project, or most commonly, to cut the tiles to fit edge conditions at roads, lamp posts or facade walls. The manual tile cutting process is both physically demanding and disturbing to the environment. It is

conducted in non-ergonomic positions with noise levels above 110 dB, and creates significant dust. For these reasons, physical wear down and early retirement among workers are common.

To address this challenge, Odico developed a mobile robotic tile cutting solution, to be operated onsite by craftsmen. By encapsulating the cutting work in a closed envelope, noise and dust generation is reduced to a minimum. To increase mobility and ease of deployment, the solution was fitted within a standard box trailer, which can be towed by any class 2 or 3 vehicle (standard passenger cars and above), and equipped with a generator for autonomous deployment. The solution is equipped with a 6 axis standard manipulator with a circular saw mounted on the flange. By manually placing or utilizing an on-board crane, tiles are lifted in place and cut by the robot. Hereby cutting speed comparable to manual cutting is achieved, but through automation of the cutting process, and reduction of noise and dust generation, physical fatigue is avoided, enabling the worker to increase the productivity rate by 1.5 – 2x over the course of a day.

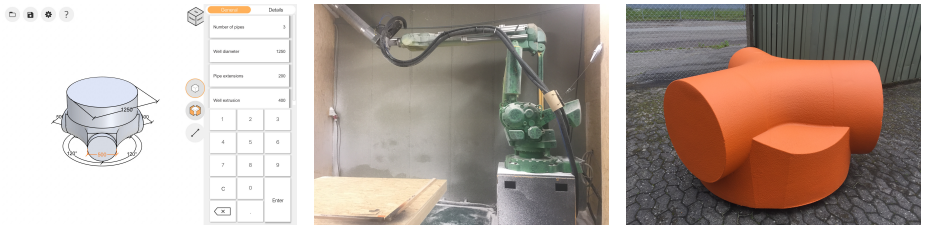
For successfully meeting the needs of this segment, the overarching requirement for the interface, was to enable unskilled craftsmen to operate the system and program unique tile cutting designs within a 5 minute training window. The preset template is a parametric model with some of the design variables exposed to the user to edit. For example, length, width and height of a tile can be edited by the user.

On deciding the dimensions of the tile, the user can choose how the tile needs to be cut by adding cut lines on the tile. With addition of each cut line, the original tile will be split into multiple pieces. User can choose one of the pieces and add it to a fabrication queue. As the queue gets processed, user can go back to the design work thereby saving valuable time.

The tablet interface is shown in Fig. 9.2a and the mobile robot station at a construction site can be seen in Fig. 9.2b. The product was launched as Odico's first commercial, off-the-shelf technology solution in August 2020. As of July 2021, it constitutes one of the main pillars for planned scaling of commercial operations.

9.3 Application 3: Milling

A third application (refer, Fig. 9.3a, 9.3b and 9.3c) was developed to target the following challenge: in today's construction for sewage and water infrastructure, a common occurring situation is that several pipes with variable diameters and



(a) (b) (c)

Figure 9.3: (a) Auto generated tablet interface for milling application; (b) along with the corresponding robotic unit and (c) end product.

approach angles need to meet in a single junction. For standard junctions, off-the-shelf precast concrete connectors can be utilized. However, in several cases, the number of pipes, the combination of diameters or the combination of approach angles is non-standard and requires a custom build solution. Today, this is handled by creating a bespoke, sub-terrain timber formwork, which is manually crafted at the site. For the test partner of the project, the duration of such work was estimated to be 2-3 days (normative average) for the formwork production.

To increase productivity, an alternative solution was proposed: a fully digitized production of EPS formwork, in which customization happens through a web application. Using an e-commerce pipeline, it is linked to a stationary, robotic CNC-milling facility which is located at the factory. The baseline can be established by comparing to current production of standard timber formworks used for precast well junctions from a factory. The process involves 3 levels of expertise: 1) a CAD specialist for expert modeling of the advanced geometry in a time span of 4-8 hours; 2) a CAM expert for programming the CNC milling of the complex piece within a time-span of 2-4 hours; and 3) a CNC operator, capable of inserting a work piece, executing a program and monitoring the process on a dedicated machining center, with on-off involvement for 3-4 hours of machining.

A GUI was made available online with sophisticated customization parameters for generating any unique combination of base diameter, number of pipes, individual pipe diameters and approach angles[1]. Within preset bounds of each parameter, an unskilled user with knowledge of the design specification for any given pipe junction can now configure the design within a 2-minute process, replacing the former need for 4-8 hours of modeling from a CAD specialist. Upon submitting the order, the design is forwarded to an associated fabrication model, which procedurally generates the corresponding tool path, hereby negat-

[1]https://odico.dk/wellmate/

(a) (b) (c)

Figure 9.4: (a) EPS form in the excavated site; (b) connection to two sewage pipes and (c) final cast of the sewage pipe junction after removal of the formwork.

ing the need for a CAM expert applying 2-4 hours of programming. Finally, this program is launched by an operator on a robotic system - a stationary 6 axis manipulator with external rotary axis. Due to the change of material from timber to lightweight EPS, we can operate the process at a higher speed, reducing machining time to 1-2 hours per piece, depending on base diameter or junction size.

9.4 Application 4: Hot wire-cutting

In the architectural history of masonry, brick arches have played a pivotal role in creating doorways, windows and portals in buildings, owing to the compressive strength of this geometry. In restoration of such buildings, as well as for new construction projects utilizing arches or other complex designs, the production of expensive timber guides is required, typically created manually or through digital means. According to an internal research conducted by Odico over the course of a 3 year period, the prohibitive cost levels of making such guides, may have contributed for the retreat of such designs from contemporary architecture.

To alleviate this restriction, a case study was developed in which an online configurator[2] dubbed Archer, would allow customers to select between 4 most common brick arch types – jack arch; roman arch; 3-centered arch; and segmental arch. Once user decides on a type, the other parameters of the support body can be specified. Upon order purchase, a rapid code file is automatically

[2]https://odico.dk/en/archer/

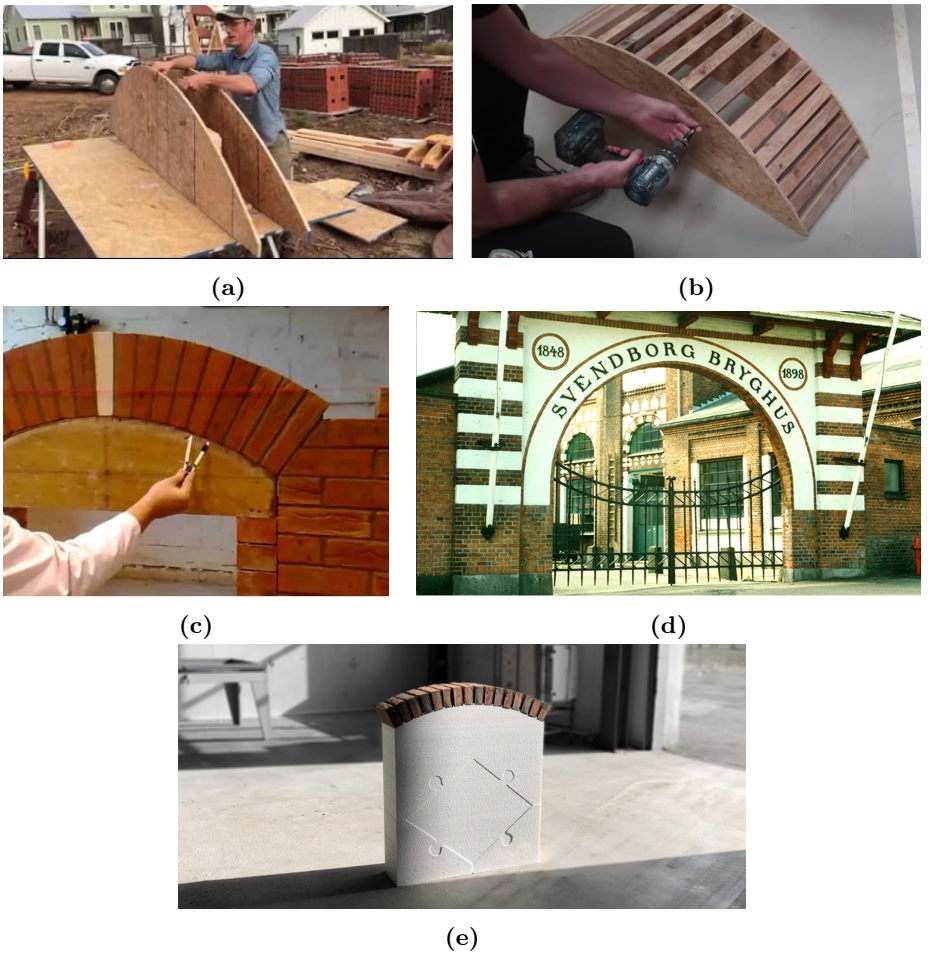


Figure 9.5: (a) and (b) Manual production of a timber brick arch guide. (c) In the construction of the brick arch, the timber guide is initially placed at the appropriate height by a supporting stack of temporary bricks. After this, bricks are incrementally placed on top of the arch with mortar applied between them. Once the arch is completed and the mortar has cured, the template is removed; (d) entrance portal at Svendborg Brewery and (e) sample of a brick arch guide produced from EPS using robotic hot wire-cutting method.

generated from the configured model, which is then executed on an associated robot station to manufacture the corresponding design out of an EPS block in less than 15 minutes. Hereby, the final cost of the guide work is reduced by

approximately 55%. In July 2021, Archer was used for the creation of a roman arch to restore the entrance portal of the preserved building of Svendborg Brewery from 1898, which was severely damaged following an accident involving a motor vehicle.

9.5 Application 5: Additive Manufacturing (AM)

3D Concrete Printing is an additive manufacturing technique to fabricate buildings or functional components for construction. In recent commercial applications, the system is based on the extrusion of cement-based material through a nozzle of a variable size/shape. Because of this, building components are produced by layered wall elements and often made of partially hollow geometries, in order to save time and material. For structural rigidity, one cannot choose a random pattern for material deposition inside the shell. In one of the pre-studies conducted by us with SDU CREATE, we were exploring the possibility of utilizing the Differential Growth (DG) [PS06] of a continuous curve to achieve a beam design of a variable density (see, Fig. 9.6), with a continuous path that can be printed without interruptions. The input to the program is a shell (simple polygon), a scalar field, and a growth factor. The growth factor controls the rate at which the curve will grow in every iteration. The scalar field is a function which associates a number to every point in space. A grayscale image was used as the field wherein the intensity lies between 0 (white) and 1 (black). In our case, we bias the growth of the curve according to the field which means that more material will be deposited in areas corresponding to darker pixels from the image. The scalar field follows the result of a structural stress analysis so that a darker pixel from the field means that more material needs to be deposited around that point.

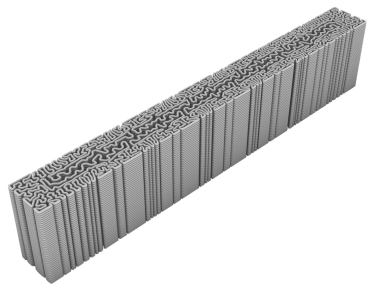


Figure 9.6: DG algorithm applied to create dynamic infills for a structural beam using scalar fields.

Chapter 10

Discussion and Further work

The goal of the project was two fold:

1. To come up with means to reduce the costs associated with the process of development of software for robotic applications targeting the construction industry.
2. To provide intuitive user experiences so as to take robotic technology to craftsmen who are ill-equipped at embracing complex systems.

For this, a novel model for second order knowledge encapsulation has been presented. The model enables a single, 5-node meta layer as an uniform control scheme for a non-finite variability of applications, hereby greatly increasing the efficiency of development of bespoke applications for construction processes.

Based on current implementation work, we project that any robotic application in the field of construction can be modelled by the same set of five nodes in the meta layer. The variation which exists between applications can be accomplished by changing the contents of the corresponding transition functions associated with each of these nodes in the meta layer, though respecting the desired input-output relationship between them.

The proposed solution contains two layers of encapsulation in which (a) complex machining processes for adaptive manufacturing of on-the-fly customized, shape-variant part designs can be operated by a non-specialist construction personnel in the form of a tablet application and (b) people from five specialist domains, viz. computational design, UI/UX development, manufacturing, robotic and process engineering are able to contribute within their domain in isolation through provision of constitutive sub-systems in a larger network.

For (a), the challenge is that the cost implications of developing a custom tablet application from scratch is prohibitive. However, general automation of software development has been proven highly complex, and remains a topic of intense research [RMAHGA$^+$15, KDA12]. We propose that by confining the scope of the problem to construction manufacturing interfaces; by relying on the closed-end solution space given by parametric models; and by leveraging the already procedural nature of workflow graphs, it is possible to establish an effective automation procedure for the given problem. In particular, the method proposal can be contrasted to general import of any random CAD file, in which the geometry generation does not happen in a closed loop system. By controlling all aspects of geometry computation within the graph, the robustness of the workflow is greatly increased, enabling a feasible automation pathway.

Further, we have presented multiple seemingly unrelated applications for robotic construction manufacturing. Each case relies on a highly customized combination of hardware configuration, tooling process, and design geometry, serving a unique need for the industry. We demonstrate that a custom interface can be procedurally generated for each case using the same, standardized model across all applications. The generated visual interface is capable of reducing the operational complexity of the underlying system to such degree that an unskilled worker within few minutes of training can effectively and safely execute the work, to produce highly customized items. Through digitization, the automation process serves to significantly increase the levels of productivity by reducing the labor time.

We further established that parametric modelling holds a significant potential for increasing the efficiency in building robotic applications in construction industry. However, there are some important open problems relating to maintainability, stability, reusability and performance of these models. We have analyzed these scenarios in greater detail and tried to solve them by applying software design patterns.

Patterns were categorized into 3 viz. functional, relational, and performa. In functional patterns, the entire workflow was decomposed into clusters comprising of nodes which are closely related to each other serving a specific purpose in the application. Relational patterns assist to manage data models within a given

parametric design in an efficient manner whereas performa patterns resulted in reducing the latency.

Further work includes on building more applications using Sculptor. Additionally, we will be focusing on exploring more design patterns suitable to the realm of parametric modelling.

Appendix A

Adaptive Robotic Manufacturing using Higher Order Knowledge Systems

A.1 Introduction

Despite representing the 5th largest industry segment, the global construction industry has for a decade seen stagnant growth in productivity, while other sectors – such as the manufacturing industries – have enjoyed nearly a quadrupling within the same period [PTT05]. Even in more recent works [BHK21, BDM20], it is mentioned that the problems related to low productivity is still prevalent in construction industry. The rising productivity in neighboring segments is widely attributed to the pervasive automation and digitization these sectors have been undergoing. Hence large scale adoption of robots in the construction sector is widely regarded as the primary enabler of potential productivity increase [DOA+19].

However, despite large research efforts, and the availability of mature automation technologies from tangential fields such as automotive, aeronautic, naval and energy industries, ubiquitous adoption of robotic processes within the industry still remains evasive. As a result, construction is one of the least automated

of the leading industrial sectors, next only to agriculture [Roh08]. Inside of a larger industrial research effort to establish a general purpose, cyber-physical technology platform for efficient automation of construction tasks, this paper reports on the developments relating to the establishment of a software framework for adaptive robotic control in construction manufacturing.

A.2 State of the art

A widely understood differentiating factor between construction and other manufacturing industries relying on large scale production is the circumstance, that while general manufacturing can benefit from deploying a mass manufacturing paradigm - in which multiple instances of identical products is repeatedly produced at high volume - such repetition is not feasible in construction, in which each building project is essentially unique [DOA+19]. Since automation technologies in general manufacturing is designed to support a repetitive mode of operation, they lack the flexibility to accommodate the variability experienced in construction.

To overcome this shortfall, significant research efforts [WKB+16, GK14, Men12] have been directed towards establishing parametric manufacturing workflows, in which a production system relying on one or more standard industrial manipulators with associated machining end-effectors is driven by a parametric manufacturing model, which utilizes the inherent flexibility of a manipulator with m degrees of freedom to create shape-variant instances of a customizable part design within pre-defined ranges of variables.

The establishment of construction scale parametric manufacturing workflows is complemented by additional research attention directed towards machine process innovation, investigating a plurality of avenues for achieving low cost, large scale production through mechanical end-effector engineering. Leading examples from this work entails robotic processes such as e.g. timber sawing [EGK17, KSM+14]; hot-blade and wire-cutting [FS14, Sea16]; brick assembly [Bon15]; robotic concrete printing [PZTG18, GM18, GCF18] and metal forming [TPM18].

The combined developments in this strand of research have recently culminated in the realization of several experimental and commercial construction projects, such as Fjordenhus Kirk Kapital HQ, Vejle, Denmark (2018) [SF17], Opus Dubai, Dubai UAE (2020) [Bho17], The Sequential Roof, Zürich, Switzerland (2016) [ABF+16], and DFAB House, Zürich, Switzerland (2019) [Dea19], signifying the principal applicability of robotic manufacturing at construction

scale.

While the establishment of a robotic parametric manufacturing paradigm – complemented by aforementioned advances in process innovation – have successfully demonstrated a pathway for increasing flexibility of digital production to create shape variant parts at near cost-parity to standardized production schemes – two fundamental limitations remain largely unaddressed:

1. while parametric manufacturing workflows can be established for predefined typologies of component production, they are ill-suited to accommodate non-linear changes in fundamental parameters such as product typology, material composition or machining process. Since the variability of construction manufacturing entails exactly this across individual building projects, each non-linear change requires manual amendment of an existing or establishment of a new workflow;

2. establishment of such workflows rely on an emerging cross-disciplinary expertise spanning across previously isolated domains, including robotics, computational design and construction manufacturing. By contrast, the global construction workforce is overwhelmingly trained in craft based, manual processes with little to no exposure to robotic manufacturing.

Taken together, these limitations incur significant overhead in the establishment of project specific workflows, while inducing substantial inflexibility in applying them in a general regime across projects.

A.3 Research challenge

For clarity, aforementioned concerns may be expressed as:

$$\mathrm{C} = \mathrm{C_v} + \mathrm{C_f}/n \tag{A.1}$$

where C denotes the production cost of a shape-variant part or building component, $\mathrm{C_v}$ is the variable costs associated with producing each item unaffected by the number of produced items - typically entailing labor, material and machining time and $\mathrm{C_f}$ signifies the fixed, one-time costs of implementing the digital production workflow required to manufacture the shape variant family of parts, and n denotes the number of construction projects on which the particular workflow can be applied. Since global construction is generally characterized by a high number of sub-specialized trades, which operates interchangeably to construct topologically variant tasks, it stands to reason that the probability of $n = 1$ is

high. In this case, only an extra-ordinary construction budget would be capable of sustaining the upfront cost of bespoke workflow establishment, limiting the application of such robotic schemes to high-profile building projects.

The presented work assumes the hypothesis that:

1. full automation of the workflow creation process is not feasible, because the number of implied variables for constitutional parameters is too high;
2. this leads to high development costs which - in combination with the small niche markets resulting from the high level of trade-specialization – makes the development of project specific workflows economically non-viable in mainstream construction, and thus leaving the majority of construction disciplines technologically under served.

To overcome this challenge, we propose increasing the efficiency of custom workflow implementation through deploying a model of second order knowledge encapsulation, which enables semi-automatic application development and framework architecture generalization. The purpose of this effort would be to lower the development costs of implementing bespoke, project specific applications below the cost of manual production, hereby facilitating wide spread adoption of robotic manufacturing in construction. The presented work addresses the software related implication of this solution.

In other words, we are presenting a software framework that will enable someone to rapidly design, prototype and develop a robotic application. As a result, one can achieve cost parity even though the aforementioned application is not mass manufactured. This is specifically of interest in construction industry, as a building contractor usually orders only 1-2 robotic units supporting a given process. For any provider of such cells to the industry, it becomes important that a framework exists to develop applications quickly.

A.4 Knowledge encapsulation strategy

Our knowledge encapsulation is initialized with denoting an application to entail the following components:

1. a parametric template or product model (PM), which holds a customizable design of a component, part sub-assembly or entire building design;

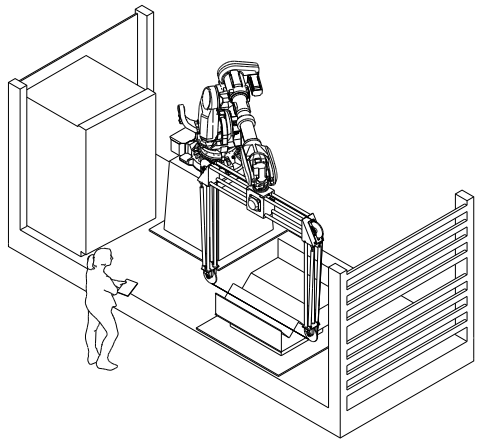
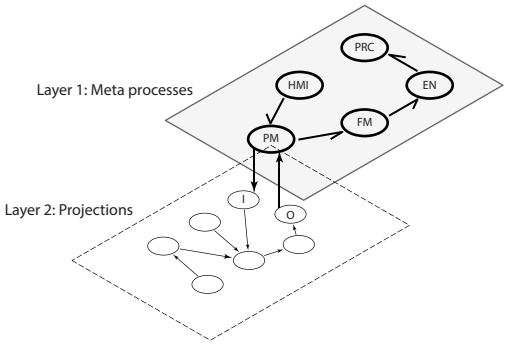


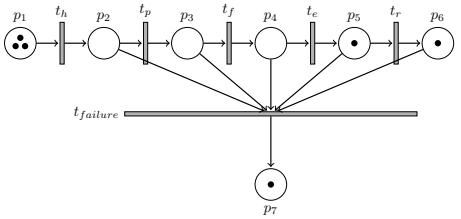
Figure A.1: An application is understood in the context of a software system operating on a cyber-physical production unit entailing a) one or more m-axis manipulators enclosed on b) a modular frame, equipped with c) custom processing end-effectors and d) operated from a tablet or mobile device by a non-specialist user.

2. a fabrication model (FM) which, as a function of PM generates the corresponding tool path targets;
3. an execution node (EN) entailing a digital representation of the robotic system with incorporated planning module to ensure motion safety;
4. a physical robotic cell (PRC) equipped with a set of predefined end-effectors for executing one or a combination of several processes. The robotic system is containerized or mounted on an autonomously moving base for modular deployment;
5. a human machine interface (HMI), which exposes key parameters of PM, FM and EN to the user through a Graphical User Interface (GUI).

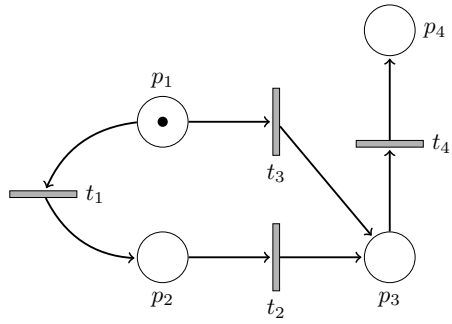
The entirety of the application is process or product specific, allowing for a simple expression of controls. Hereby the application enables a non-expert user, i.e. a construction worker trained in craft with no robotics experience, to (a) customize a part design on the fly or retrieve an externally customized part design and (b) safely execute production of the customized instance of that part. The concept of such a containerized set up with a tablet interface can be seen in Fig. A.1 where an user can be seen interacting with PRC using a HMI. The containerized set up including the tablet is referred to as Factory on the Fly™.



(a) Bi-layered structure



(b) Petri net representation of the bi-layered structure. t_h, t_p, t_f, t_e and t_r are transitions (projections) associated with HMI, PM, FM, EN and PRC respectively. $t_{failure}$ is the error handling routine.



(c) t_h, t_p, t_f, t_e and t_r are further represented using individual Petri nets and resides in layer 2 of our bi-layered structure. For example, for a given application, the role of a Computational Design Specialist (CDS) is to model t_p and t_f whereas a robotic engineer will be able to build up t_e. A sample Petri net which can be used to model one of the projections is shown here. t_1, t_2, t_3 and t_4 are individual transitions which make up a function composition model.

Figure A.2: Knowledge encapsulation strategy

A.4.1 Application model

Pursuant to the proposed encapsulation strategy, the robotic application can be visualized as a bi-layered structure (refer, Fig. A.2a) and the corresponding Petri net models are shown in Fig. A.2b and Fig. A.2c. Petri nets have previously

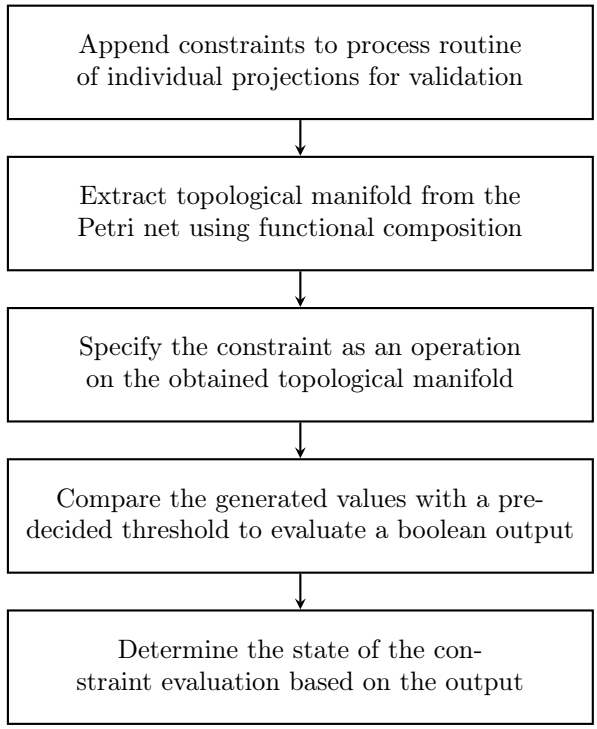


Figure A.3: Constraint Checking Workflow

been shown useful for knowledge abstraction [HSZL16, HLY16]. The first layer entails a meta-representation consisting of 5 components and the interactions between them. The second layer corresponds to a topologically variable sub-system, which models the product or process specific aspects of the application (refer, Fig. A.2a). Since these sub-systems may model any variability within the constitutional parameter space including product type, material combination, machining process, location or robot system configuration, the technical challenge becomes projecting a non-finite variety of sub-system topology to a finite meta representation (layer 1) under the constraint that model should remain valid. To solve this, we implement the following method as shown in Fig. A.3.

A.4.2 Layer 1

Each component in the knowledge encapsulation strategy viz. PM, FM, EN, PRC and HMI finds a representation in this layer to completely define various robotic applications. Associated to the components, there are transitions or

projections. Depending on the current state of the application, a particular route gets active. A route is an event handling pipeline (refer, Fig. A.2b) comprising of a subset of components. At any given time, there will always be only one active route. For example, when a production user is initiating a request to fabricate a piece to the robot, the active route is shown in Fig. A.2a and Fig. A.2b.

Due to the simple nature of interaction possible, it is suitable for a low skilled production user to work with this layer. He or she can tweak input parameters to achieve a specific valid product configuration and further instruct the robotic system to manufacture the product confirming to the specified parameter values.

A.4.3 Layer 2

As mentioned, there is a transition associated with each one of the components in layer 1. They encapsulate the complex nature of computations required to achieve the functionality of a meta node. For example, PFM (refer, Table A.1) is primarily tasked with the computation of tool path to service the fabrication request from the user.

Expert user knowledge is required to model projections. A CDS will be able to define the transitions for PM and FM whereas a CAM operator or a robotic engineer is required for EN.

Every transition has a predefined input/output type. Since it's most of the times difficult to describe the relationship between the output and input with a

Projection (transition) name	Associated meta node	Input	Output
PPM (t_p)	PM	Numeric/Boolean	Fabrication task
PFM (t_f)	FM	Fabrication Task	Robot task
PEN (t_e)	EN	Robot Task	Configuration task
PPRC (t_r)	PRC	Configuration task	Physical motion
PHMI (t_h)	HMI	Human gestures	System action

Table A.1: Associated projections.

single function, it is expressed using function composition which is an operation that takes two functions f and g and produces a function h such that h(x) = g(f(x)) (see, Fig. A.4a and Fig. A.2c).

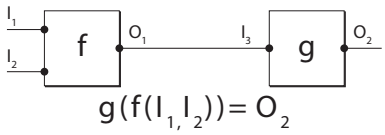
The details of projections is provided in Table A.1. A fabrication task contains a NURBS model, tool data and other process parameters whereas a robot task comprises of a tool path, process speed along with other specifications including Robot Model (RM), Environment Model (EM), Tool Model (TM), Work Piece (WP) and Operation Sequence (OS). The configuration task includes the position, velocities and accelerations of all the robot joints as a function of time.

Projections are further represented via Petri net. They are also composed of places and transitions just like any other Petri net. They involve a special set of transitions called constraints. An evaluation of a projection is said to be valid, if and only if all the constraints attached to the projection are satisfied.

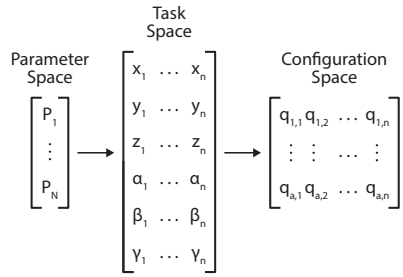
A.5 Constraints

In Fig. A.3, a constraint is expressed as an operation on a topological manifold otherwise referred to as a manifold. Let an arbitrary set M represent the point cloud which resides either in task or configuration space (refer, Fig. A.4b) and O be the chosen topology on M.

For robotic manufacturing, it's a necessary condition that the topological space (M, O) forms a manifold. A topological space is called a d-dimensional manifold if $\forall \mathbf{p} \in M$ there exists an open set U containing $\mathbf{p}$ where the following continuous



(a) Function composition



(b) Topological spaces

Figure A.4: Graph operations

maps exist:

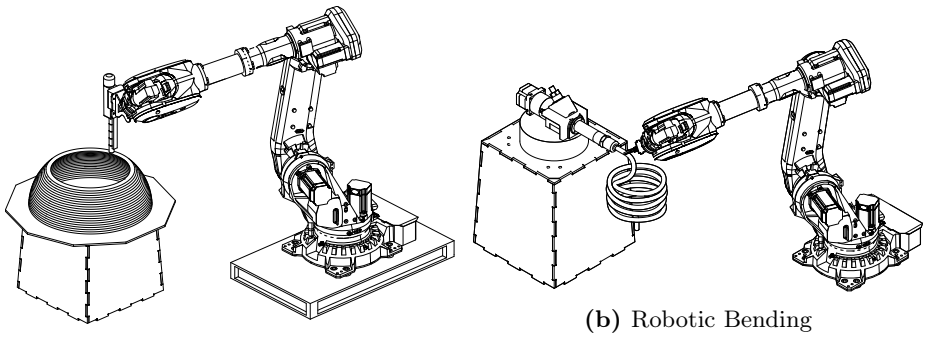
1. $x : \mathrm{U} \mapsto \mathrm{R} : \mathrm{R} \subseteq \mathbb{R}^d$
2. x^{-1}

This is called the charting condition.

For example, consider the surface of a torus as the NURBS model included in the fabrication task, let the points in this model be M where $\mathrm{M} \subseteq \mathbb{R}^3$. We can choose standard topology as O. For any $\mathbf{p}$ in this model, we can find a U which is nothing but the open neighbourhood around the point. Since $\mathrm{U} \subseteq \mathrm{M}$, it can be represented as a NURBS surface, following which it can easily be verified that there exists a x which maps every point on U to R embedded in $\mathbb{R}^2$ in a continuous invertible manner where R forms the parameter space for U. Hence we can say that (M, O) is a manifold as it satisfies the charting condition.

There are many artifacts whose presence will convert the model into a non-manifold geometry, viz. a curve that bifurcates at a point, a set of surfaces forming an open volume or an edge shared by more than 2 surfaces, because around such artifacts the continuity of x gets broken.

A robotic arm ζ having an m dimensional configuration space C is used to process the tool path specified in the robot task. Since ζ is made of several objects connected by joints, it is subjected to kinematic constraints. These are constraints which restricts an object or a collection of objects from rotating or translating freely in the workspace. There are two types of kinematic constraints viz. holonomic and non-holonomic.



(a) Additive Manufacturing

(b) Robotic Bending

Figure A.5: Robotic Processes

An atlas has been defined on C and any configuration $\mathbf{q}$ of ζ is represented by a list of m coordinates $(q_1, q_2, ..., q_{m-1}, q_m)$ in some chart of the atlas. A scalar constraint of the form:

$$F(\mathbf{q}, t) = 0 \tag{A.2}$$

where F is a smooth function with non-zero derivative, is called a holonomic equality constraint. More generally, there may be k such constraints where k $\leq$ m. Typically, such equality constraints allow us to map a manifold from task space to C. Undesirable collisions as well as out of reach scenarios could be modelled as holonomic inequality constraints of the form:

$$F(\mathbf{q}, t) \leq 0 \tag{A.3}$$

and can be expressed as an operation on the manifold in task space. Constraints can be modelled either in task or C. As manifolds are inherently differentiable, constraints involving velocities or accelerations can be plugged in.

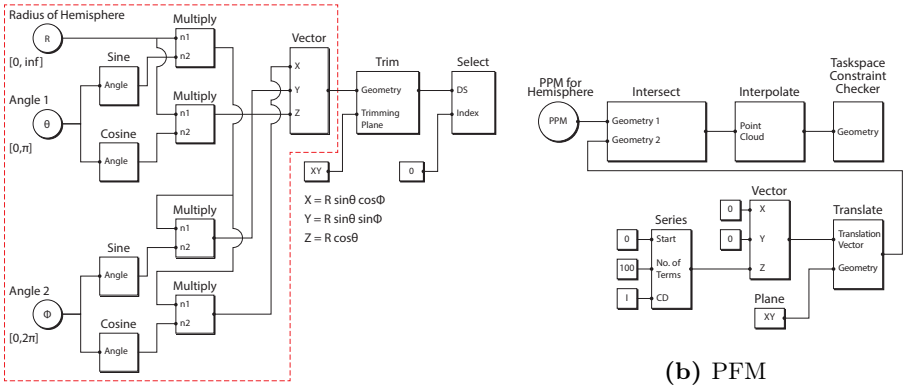
A.6 Exemplifications and industrial applications

The generality of the proposed framework has been validated using two robotic applications viz. additive manufacturing and bending. For additive manufacturing, we will use the target product as a hemispherical bowl whereas for bending, the product is in the form of a coil or spring with a rectangular cross section. A serial chain robotic arm fitted with the corresponding tool will be used for physical realization of the process (refer, A.5a and A.5b).

In both the cases, the schematic representations of corresponding PPM and PFM are shown in Fig. A.6a, Fig. A.6b, Fig. A.7a and Fig. A.7b respectively. PEN, PPRC and PHMI are not shown for brevity. The various functions shown in the representation denotes the parametric operations performed in sequential form to arrive at the target geometry or tool path. Constraint nodes are not shown for clarity but are explained briefly below.

For a better understanding of constraints explained before, we will try to apply a well known reachability analysis to check the validity of the motion. A given $\mathbf{p} = (\mathrm{X}, \mathrm{Y}, \mathrm{Z})$, in the tool path contained in the robot task needs to satisfy the holonomic inequality constraint introduced before. If R (refer, Fig. A.6a), increases beyond a threshold, $\mathbf{p}$ will breach (A.3). Similarly, any non-allowed collision between the given RM, EM, TM and WP can also be expressed in the form of (A.3).

The presented knowledge encapsulation model has been utilized in the development of a novel framework for robotic control and product customization,



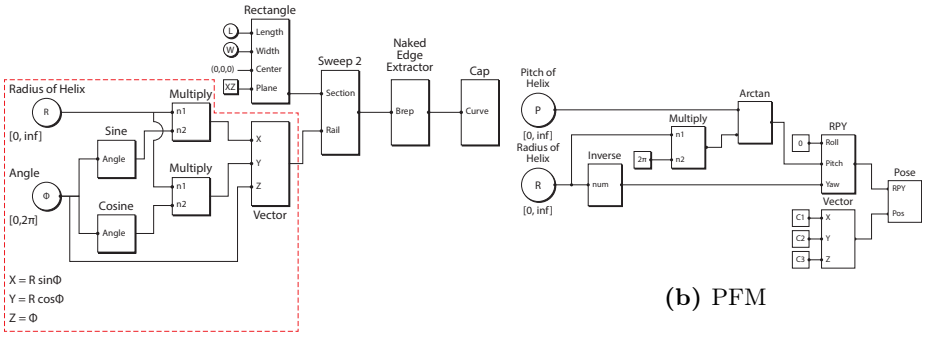
(a) PPM

(b) PFM

Figure A.6: Additive Manufacturing

Sculptor™, for high-efficiency development of new applications within the Factory on the Fly™concept for modular, containerized robotic pop-up manufacturing stations, developed by Danish technology innovator Odico Construction Robotics.

The framework has enabled CDS employed in the company with no prior training in software development to develop a plurality of functionally non-related applications such as (1) sawing of concrete tiles (refer, Fig. A.8) - objective of this portable robotic cell, is to be able to support cutting of tiles to various shapes so that they can be laid on pavements; (2) wire-cutting of custom EPS formwork (refer, Fig. A.8) - the goal is to manufacture formwork for concrete



(a) PPM

(b) PFM

Figure A.7: Robotic Bending

casting of a given product geometry. Our modular robotic cell can be shipped to the customer site and production user can operate the same using our easy to use tablet interface to customize the formwork design and generate the robot instructions for wire-cutting EPS blocks to create the bespoke formwork. In this particular instance, product geometry was a staircase; (3) CNC milling of custom sewage and water pipe junctions - a contractor had approached Odico to build a robotic application which is easy to use and assists them in creating formwork for custom pipe junctions. As opposed to the staircase application, we had to use milling since the target geometry involved doubly curved surfaces.

For application 1, a mobile product has been launched as of August 2020. Early commercial tests have demonstrated that non-specialized pavers with no prior education in robotics can operate the unit and create advanced custom tile designs within a 10 minute training window.

A.7 Discussion and conclusion

Based on current implementation work, we project that any application in the field of construction can be modelled by the same set of five nodes in the meta layer. The variation which exists between applications can be accomplished by changing the contents of the corresponding projections (see, Fig. A.2a) by

Figure A.8: (top left) Robot sawing tablet interface; (top right) Robot sawing unit and (bottom) Wire-cutting unit

respecting the desired input-output relationship between them. For example, as long as the NURBS model contained in the fabrication task is a d-dimensional manifold where $d \in (1, 2)$, PFM can accept it for further processing. Further constraints can be wired into individual projections to model various system limitations. In most cases, these constraints could be generalized to a degree that they could be used across applications. For example, the reachability or collision constraint in PEN can be applied to any robotic application.

Furthermore, in this work, we validated our framework by applying our model to examples belonging to entirely different realms as well as with some industrial applications. A closer inspection of Fig. A.6a and Fig. A.7a (refer, the components inside the red coloured region), reveals that many components could be rearranged and reused to build PPM and PFM for two applications. As a result, our original goal of keeping C_f low by distributing it over a plethora of applications has been achieved.

A novel model for second order knowledge encapsulation in the development of adaptive robotic applications for construction tasks has been presented. The model enables a single, 5-node meta layer as an uniform control scheme for a non-finite variability of applications, hereby greatly increasing the efficiency of development of bespoke applications for construction processes. It entails two layers of encapsulation in which (a) complex machining processes for adaptive manufacturing of on-the-fly customized, shape-variant part designs can be operated by non-specialist construction personnel in the form of an application and (b) people from 5 specialist domains, viz. computational design, UI/UX development, manufacturing, robotic and process engineering are able to contribute within their domain in isolation through provision of constitutive sub-systems in a larger network.

Appendix B

Procedural generation of human machine interfaces from graph-modelled robotic workflows

B.1 Introduction

Despite significant research efforts to introduce robots into construction manufacturing, wide-spread adoption of robotic workflows remain elusive within the sector [Boc07]. Among several identified challenges, a key inhibitor is the need to bridge the knowledge gap between crafts-trained construction workers educated in manual execution on the one hand, and the deep technical knowledge required for robotic programming on the other [Ela08]. In the wider community of robotic research, to which this challenge also applies, several novel paradigms have been explored, such as collaborative robotics, in which e.g. a human operator may program the robotic motion through physically translating the position of the end-effector from one point to another and recording this translation [AMAI19]; or through high-level voice commands to instruct a robot to perform certain actions [Pir05]. However, such approaches are particularly ill-suited to construction manufacturing workflows, because they involve complex operations

such as a) obtaining or adapting a geometric design CAD file; b) deriving target positions from this CAD file and c) executing the targets with high precision to produce the desired item or product through additive, subtractive or formative machining.

Contemporary industrial software packages for handling such CAD/CAM production workflows - such as for instance Catia, Solidworks, MasterCam, Fusion 360, Dynamo or Grasshopper - rely on comprehensive, multi-functional interfaces, which require extensive training and often academic educational backgrounds to operate. The complexity of such systems and associated requirements for expert user knowledge stem from the need to provide versatile, general purpose tools, geared to process a plurality of manufacturing or designing tasks. By contrast, the popularization of smart devices within the domain of consumer electronics have established a wide range of common examples of advanced processes such as natural language processing; image classification; or remote control of drones being accessible and operable by non-expert users within few minutes of training [IMKT21]. The fundamental enabler for this development has been the introduction of simplified mobile applications, which rather than offering a general purpose tool set, is specialized for tasks or subtasks of strictly limited functional scope [YYAR02]. While such applications offer a strategy of knowledge encapsulation, which has proven widely successful in allowing broader audiences to access and benefit from sophisticated computing tools and processes, they do however rely on extensive usability studies and careful calibration of the implied user interface and user experience design to achieve such accessibility, complementing the software engineering efforts required to establish the underlying processes. For these reasons, the average development time for a mobile application is comprehensive, typically quantified between 6 and 9 months for a basic application [GSMG17].

B.2 Hypothesis

Considering the success of mobile applications in offering access to complex computational processes for broad and varied audiences, it stands to reason that mobile applications of comparable user interface simplicity may enable access for non-specialist workers to operate advanced, robotic processes and on-the-fly customization of parts typically required within construction manufacturing, hereby representing a pathway to overcome the knowledge barriers prohibiting wide-spread adoption within the construction industry.

However, since global construction is characterized by very high levels of trade sub-specialization; low inter-trade exchange; inadequate data registration; and

extra-ordinarily high levels of combinatorial variability stemming from the co-existence of historical building methods (e.g. masonry, stonework or carpentry) with highly industrialized schemes of manufacturing (e.g. steel work, concrete pre-casting or facade glass production) [SB13]; the resulting size of the target groups that may benefit from such applications is limited. In other words, successful mobile applications for smartphones, tablets or wearable devices may achieve millions or even billions of users, whereas for robotic applications in construction industry comparable numbers may be less than 1000 for a given application, even down to 1 for an entirely project specific scenario.

Considering the significant limitation in target user group size, the normative cost of conventional mobile application development becomes prohibitive, even before considering the significant added complexity of modeling a construction robotic process. Thus, even with the utilization of best software development practices, an entirely manual schema of application implementation [MJ06] remains wholly insufficient to solve the before mentioned structural challenge of bridging the knowledge gap between construction workforce and robotic programming.

However, if implementation time could be significantly reduced through full or partial automation of the application development process, the cost parity threshold between a fully manual construction process and development plus utilization of a project specific robotic process could potentially be passed, hereby enabling large scale adoption of robotics within the global construction sector. Based on these considerations we summarize the hypotheses - (a) widespread adoption of robotics in construction manufacturing may only be reached by simplifying the operation of construction robotic systems to levels accessible to crafts-trained, non-specialist users; (b) the success of mobile applications within consumer electronics provide a template for such simplification, enabling operation of complex product configuration and manufacturing functions via simplistic, scope-restricted GUIs; (c) the normative development cost of such applications is however not consistent with the significantly reduced user group target sizes arising from high degrees of trade sub-specialization and (d) hence, to solve (a), new software technologies are required, which can provide a substantial reduction in end-to-end application implementation time.

This paper presents a novel method towards this target, which relies on graph based modeling of the geometric and fabrication workflows, and automatic expression of application specific control parameters within an adaptive user interface structure for tablet control of robotic systems. The work is conducted as part of a larger research effort to create an industrial software framework for development of construction robotic applications.

B.3 State of the art

Through preceding developments in the field of construction robotics, it has been established that standard industrial m axis manipulators equipped with machining end-effectors and driven through a parametric workflow, is capable of producing advanced, shape-variant designs at near mass-manufacturing throughput volumes [ABF+16, SF17, GKW14]. However, such custom workflows rely to a very high degree on highly specialized, cross-disciplinary expertise in robotics, computational design and manufacturing, which is inaccessible to the vast majority of construction professionals. As this challenge also applies within the otherwise CAD centric AEC environment, it has been proposed to create simplified interface wrappers on top of parametric graphs to ease the operation of design customization workflows [Ber16]. While this trajectory does provide a principal pathway for increasing user accessibility, current software architectures proposed require still a manual crafting - and thus time-intensive UI development. Furthermore, implementation architectures based on Windows Presentation Foundation and established parametric CAD environments such as McNeel Grasshopper or Autodesk Dynamo are highly ill-suited for development of OS agnostic tablet applications.

Overcoming this last challenge, examples of iOS based control tablet applications for architectural robotics has been proposed [DRR13, Sch13], however through architectures, which rely on a separation between interface layers on a tablet client and server side geometric operations. Hereby, instantaneous customization actions that rely on server side geometry modules becomes difficult. Additionally, these frameworks break the direct link to the underlying graph and rely on manually crafted UI, which are time-consuming to implement. Similarly, industrial HMI frameworks such as Siemens Simatic HMI Panels rely on an modularly adaptive, yet integrated, OS specific solution, but without the critical geometric functionalities required to handle CAD/CAM customization. In summary, currently proposed methods of UI encapsulation remain implementation ineffective, when considered under the pretext that sub-normative implementation times are required to reach cost feasibility in the context of construction industry.

To overcome these challenges, we propose an encapsulation architecture, which a) is based on graph-modelled parametric workflows for design and manufacturing control; b) automatically derives the corresponding, customized UI from an adaptive framework; c) is implemented for OS agnostic mobile application development. In preceding work by the authors [NSB21], a proposal was made for a knowledge abstraction model as shown in Fig. B.1. The meta layer is made up of 5 nodes which gives a logical structure to any robotic application. The details of these nodes are given below:

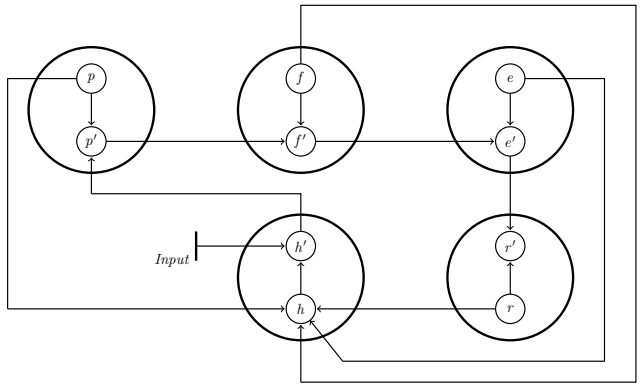


Figure B.1: Directed acyclic graph (DAG) representation of the knowledge encapsulation model. p, f, e, r and h stand for the current states of PM, FM, EN, PRC and HMI respectively. On receiving an *Input* from a human through the HMI, all nodes will propagate to their future states viz. p', f', e', r' and h' respectively.

1. a product model (PM), which holds a customizable design of a component, part sub-assembly or entire building design;
2. a fabrication model (FM), which is expressed as a function of PM and outputs corresponding tool path targets;
3. a physical robotic cell (PRC), which is equipped with one or multiple robotic arms, a set of predefined end-effectors for executing a combination of several processes. The system is containerized for modular deployment;
4. an execution node (EN), which contains a digital model of PRC with an integrated motion planning module to ensure robot safety;
5. a human machine interface (HMI), which exposes key parameters of PM, FM and EN to the production user through an intuitive user interface.

Each of these nodes is associated with a transition which is in turn a Petri net (see, Fig. B.3c). A Petri net is a directed bipartite graph that has two types of nodes viz. places and transitions used for dataflow modelling systems [Mur89]. The application is specific to a process or product, which allows a simple expression of controls. The primary role of such an application is to enable a non-expert user, i.e. a construction worker with no experience in robotics, to (a) customize a product design on the fly or import an externally customized product design and (b) safely execute and monitor production of such customized instances of a particular product.

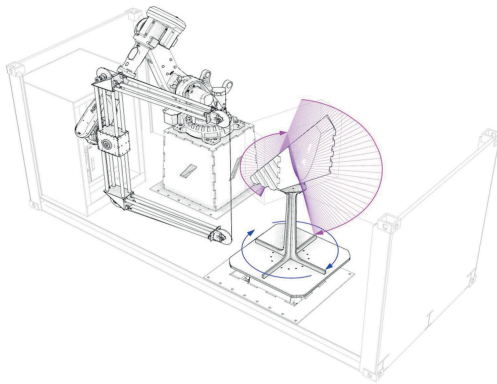


Figure B.2: Factory on the Fly™.

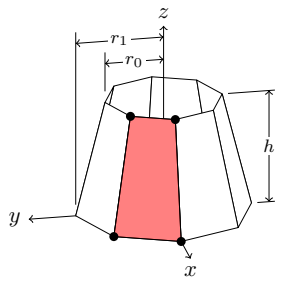
The concept of such a containerized set up referred to as Factory on the Fly™ can be seen in Fig. B.2. An user can operate it with an easy to use tablet interface.

As a result of this model, we expect a quicker deployment of new workflows due to separation of concerns and reusable nature of software units. In this paper, we would like to explain how the proposed model enables semi-automatic generation of user interfaces for a given robotic application.

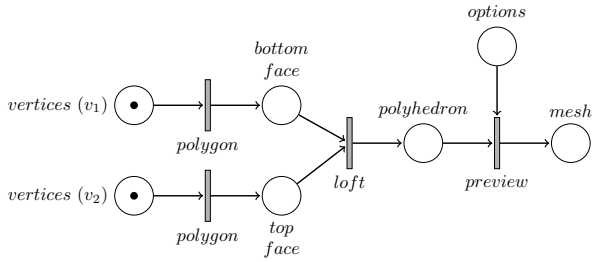
General automation of software development has been proven highly complex, and remains a topic of intense research [RMAHGA+15, KDA12]. However, we propose that by confining the scope of the problem to construction manufacturing interfaces; by relying on the closed-end solution space given by parametric models; and by leveraging the already procedural nature of workflow graphs, it is possible to establish an effective automation procedure for the given problem. In particular, the method proposal can be contrasted to general import of any random CAD file, in which the geometry generation does not happen in a closed loop system. By controlling all aspects of geometry computation within the graph, the robustness of the workflow is greatly increased, enabling a feasible automation pathway.

B.4 Robotic application modelling

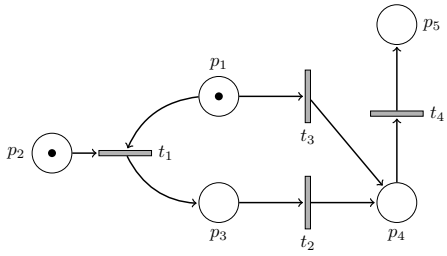
A robotic application is modelled as a Petri net by a Computational Design Specialist (CDS), which will have k inputs pre-decided by the designer of the system. The model is created by chaining transitions (see, Fig. B.3c) together



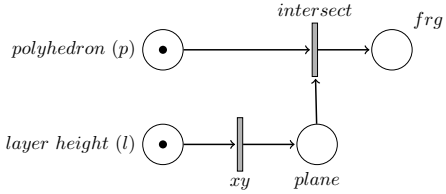
(a) Visualization of the target product which is a polyhedron (p) with a height h in the view port. r_0 and r_1 are the respective distances measured normal to the axis of symmetry (z axis).



(b) Petri net representation of the t_p associated with the polyhedron shown on the left. It is responsible for rendering the target geometry in the view port. An user of the tablet application can interact with the geometry by selecting a surface, a curve or a dimension object. In this example, the selected surface is shown in red.



(c) A transition associated with PM, FM, EN, PRC and HMI can in turn be represented using Petri nets. t_1, t_2, t_3 and t_4 are transition functions which could stand for any one of the parametric operations required to model the process.



(d) Petri net representation of a sample t_f which generates a tool path for additive manufacturing process by intersecting the given polyhedron with a plane parallel to XY plane. Polyhedron (p) and layer height (l) are parameters to the model. As long as $0 \leq l \leq h$, the following projection will output a valid tool path curve which is given out as the output from the model via the node frg.

Figure B.3: Petri net models for visualization and physical realization of a polyhedron using an additive manufacturing process.

to define the relationship between the input and output places. Two transitions can be chained by way of a connection. A connection is a directed link where data can flow only in one direction. It has two ends viz. origin and destination. Data can flow only from an origin to destination. The origin should be attached to the output place of a transition whereas the destination should be tethered

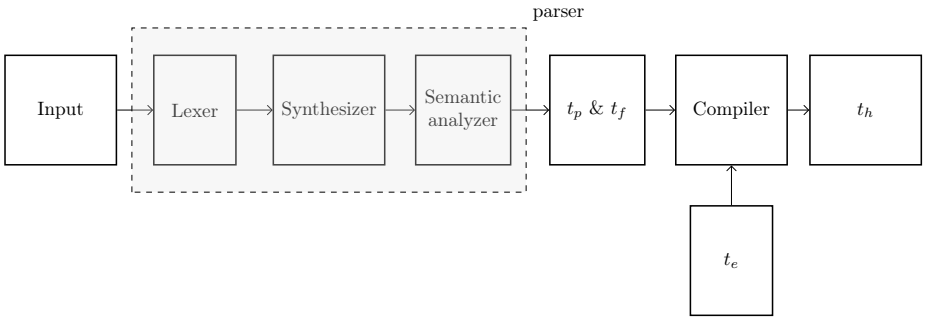


Figure B.4: Compilation pipeline

to an input place of a transition. The data types of the places should match. For example, an output place holding a double cannot be connected to an input place accepting a string.

A transition will have a single scalar or vector valued function associated with it. The parameters to the associated function is exposed as input places and the result of the function evaluation are stored in output places. Each of the places may have predecessors or successors. A place with no predecessor is termed source whereas the one with no successor is called a sink. Sources are used to provide the tokens (inputs) to the model for evaluation whereas sinks determine the outputs. Two examples of a sink are $mesh$ and frg as shown in Fig. B.3b and B.3d respectively. The former is used while the designer would like to visualize a geometry or process in the view port and latter could be used to generate robot instructions which are then passed for execution. In a given model there could be several sources as well as sinks. In Fig. B.3b and B.3d, the places marked v_1, v_2, p and l are sources.

B.5 Compilation pipeline

During the semi auto-generation phase of the GUI, the provided input file goes through a compilation pipeline (see, Fig. B.4) which has the following steps:

1. Input - specifies the parametric design in the form of a XML file. It mentions all the transitions and places contained in t_p and t_f and their relationships with each other.
2. Parser - consumes the given input file and converts it into a process model which contains two Petri nets viz. t_p and t_f. During this phase, the file

passes through 3 stages:

(a) Lexer - the input file is split into meaningful symbols at this stage. In case any unrecognized symbols are found in the model, an error will be thrown.

(b) Synthesizer- The output of this activity will be a network of transitions and places where the connectivity information has been gleaned from the file and synthesized to a process model. In other words, each transition and place will know their predecessors and successors after this step.

(c) Semantic analyzer - the model is analyzed for its correctness by verifying whether all the connections are valid. All the connections are type checked.

3. Compiler - accepts the synthesized process model from the previous step along with t_e to create t_h. t_e contains robot, environment and tool models required for creating collision free motion plans for execution on t_r.

t_r is understood as a cyber-physical production unit entailing a) one or more m axis manipulators enclosed on b) a modular frame, equipped with c) custom processing end-effectors. t_h is a tablet based interface which holds a) a parametric CAD model of a given product with parameters accessible to customize the product within predefined boundaries; b) a fabrication menu within this interface, which holds options for the user to select pieces for production and submit that to t_f which parametrically deduct the necessary tool paths and operations within preset bounds.

All the sources inside of t_p and t_f will be exposed in t_h via UI elements. The UI element could be a text box, an active dimension object, slider etc. A functional UI can be achieved with such simplification. It is important to note that the goal is to achieve an unified user experience across Factory on the Fly™ units made by Odico.

The extracted tool path is transmitted to t_e for planning safe motion for the robot manipulators present in the production unit which is designed to manufacture the selected product.

B.6 Implementation

The input XML file can be created by a standard software tool intended for working with parametric workflows. Since our designers are familiar with Grasshop-

per, we have relied on GHX file format for now. A proprietary computational geometry kernel named DaVinci was created which can be accessed via Grasshopper as well as Unity. Currently, designers use Grasshopper editor to prepare the XML file which is termed as development workflow.

We are currently exploring to migrate to Bolt which is a visual programming environment integrated with Unity. Once the XML file has been prepared, it is ported to Unity which converts the provided file to Petri net models (t_p and t_f) through a custom developed parser implemented using C# programming language.

For specifying t_e, we are relying on MoveIt motion planning framework provided inside Robot Operating System (ROS). Further, Unity is used to create the iOS application in a semi-automatic manner.

Each Factory on the Fly™ unit has 3 computing devices - (a) a tablet which

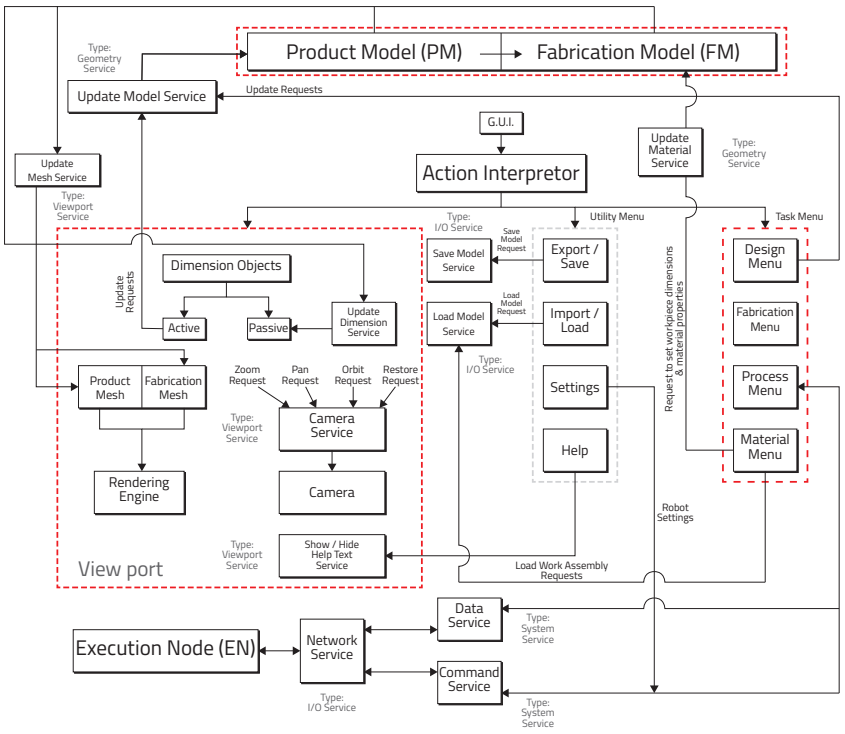


Figure B.5: Architecture diagram for user interface

holds t_h, t_p and t_f; (b) a mini computer which is a PC with a small form factor. t_e resides in this device. It is powered by Ubuntu 20.04 and ROS Noetic and (c) a robot controller which receives instructions for execution on the physical robot from the above mentioned mini computer.

B.7 GUI architecture and case studies

The architecture of the GUI is shown in Fig. B.5. The interface is an event based system which responds to gestures from the user. The advantage of modelling the application using the proposed method is that various functional behaviors can be achieved by performing basic operations on the graph which makes the implementation universal across applications. Some of these behaviors are discussed below:

1. A fundamental property of the interface is that it should allow an user to customize a particular product model. For this purpose, sources are exposed as editable parameters to the user. On changing a value of a particular source, a series of transitions are fired resulting in the re-computation of the model. The output of this activity is stored in sinks. The safeness can be ensured by bound checks on the values for the source.
2. User might want to save/load a particular state of the application. It can be easily achieved by storing the values associated with all the sources in the Petri net. On re-loading the saved state, values for all the other places in the Petri net can be computed by triggering a series of transitions as prescribed by the system specification.
3. Under some situations, the user would like to undo/redo a specific operation. In a Petri net based system, this can be accomplished by recording the values for sources so that one can move between states.
4. Monitoring of robot states.

Based on the developed system, we test the capacity of the suggested approach by developing a series of applications.

B.7.1 Application 1 - Sawing

For further demonstration, framework was used to quickly develop a robotic application for tile sawing operations. In contemporary paving works, concrete

and natural stone tiles are often adapted onsite through manual sawing. Adaption is typically required either because standard tiles were not available for the particular dimensions required for the project, or most commonly, to cut the tiles to fit edge conditions at roads, lamp posts or facade walls. The manual tile cutting process is both physically demanding and disturbing to the environment. It is conducted in non-ergonomic positions with noise levels above 110 dB, and creates significant dust. For these reasons, physical wear down and early retirement among workers is common.

To address this challenge, Odico developed a mobile robotic tile cutting solution, to be operated onsite by craftsmen. By encapsulating the cutting work in a closed envelope, noise and dust generation is reduced to a minimum. To increase mobility and ease of deployment, the solution was fitted within a standard box trailer, which can be towed by any class 2 or 3 vehicle (standard passenger cars and above), and equipped with a generator for autonomous deployment. The solution is equipped with a 6 axis standard manipulator with a circular saw mounted on the flange. By manually placing or utilizing an on-board crane, tiles are lifted in place and cut by the robot. Hereby cutting speed comparable to manual cutting is achieved, but through automation of the cutting process, and reduction of noise and dust generation, physical fatigue is avoided, enabling the worker to increase the productivity rate by 1.5 – 2x over the course of a day.

For successfully meeting the needs of this segment, the overarching requirement for the interface, was to enable unskilled craftsmen to operate the system and program unique tile cutting designs within a 5 minute training window. The preset template is a parametric model with some of the design variables exposed to the user to edit. For example, length, width and height of a tile can be edited by the user. On deciding the dimensions of the tile, the user can choose how the tile needs to be cut by adding cut lines on the tile. With addition of each cut line, the original tile will be split into multiple pieces. User can choose one of the pieces and add it to a fabrication queue. As the queue gets processed, user can go back to the design work thereby saving valuable time.

The tablet interface is shown in Fig. B.6a and the mobile robot station at a construction site and the physical product created using the process can be seen in Fig. B.6b and B.6c respectively.

B.7.2 Application 2 - Wire-cutting

In this application, the goal is to manufacture moulds for concrete casting. The results are demonstrated on staircase moulds which are one of the most advanced mould types among regularly produced concrete components. Today's method

for producing a mould for concrete casting entails the following:

1. A 2D drawing of the requested design is received by the mould supplier.
2. This is translated by a CAD specialist into a production drawing.
3. Output from STEP 2 is used by the CAM-specialist to program a CNC machine to manufacture the mould parts in timber.
4. Output from STEP 3 is used by a craft specialist (typically a mould carpenter) to assemble and produce the moulds.

The entire operation takes typically between 1.5 - 2 working days and relies on 3 specialist competences.

We deployed the developed framework to build the following alternative: a parametric staircase model was developed, which holds the relevant variables to accommodate most common staircase dimensions and detailing. The model is represented on a tablet, which holds a design menu for altering parameter values; and a fabrication menu, which shows the mould configuration of the customized design, and enables the user to submit mould parts to fabrication. The GUI is shown in Fig. B.6d. Once submitted, the required robot trajectory is transmitted to a robot station holding an Abrasive Wire Cutting (AWC) end-effector as depicted in Fig. B.6e, which cuts the corresponding mould piece from a standardized Expanded Polystyrene (EPS) material block. The EPS mould (see, B.6f) is then ready for concrete casting. Hereby, we have now simplified the process to involve only the following competences to operate a robotic unit - (a) general know-how of operating an iOS tablet application with a complexity level comparable to commonly available iOS applications and (b) the ability to read dimensions from a provided 2D drawing and inputting the corresponding values in the product design parameter menu. Both of these tasks can be achieved in our system with simple and general instructions which require no specialized training.

Machining time for a staircase mould created out of a bounding box with dimensions 2400 $\times$ 1200 $\times$ 300 mm was around 30 minutes. This is much faster than a clock time of 1.5 - 2 working days required for conventional method of mould production of a similar item.

B.7.3 Application 3 - Milling

A third application (refer, Fig. B.6g, B.6h and B.6i) was developed to target the following challenge: in today's construction for sewage and water infrastructure,

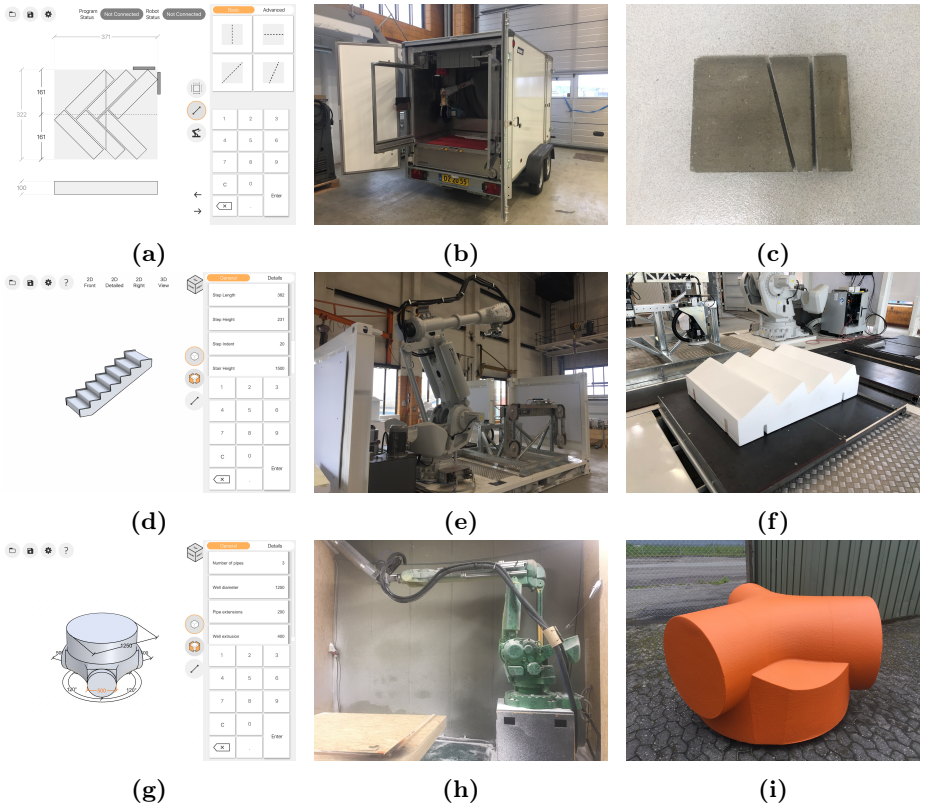


Figure B.6: Auto generated tablet interfaces for 3 robotic applications are shown here - (a) sawing; (d) wire-cutting and (g) milling. Inputs to the synthesized process model are either exposed via the right hand side panel or through editable dimension objects. User can modify them through the provided keypad. Any change in the associated inputs will fire a series of transitions within the process model. Geometry is rendered in the view port. The corresponding Factory on the Fly™ units are shown in (b), (e) and (h) and associated physical products can be seen in (c), (f) and (i) respectively.

a common occurring situation is that several pipes with variable diameters and approach angles need to meet in a single junction. For standard junctions, standard precast concrete well junctions can be utilized. However, in several cases, the number of pipes, the combination of diameters or the combination of approach angles is non-standard and requires a custom build solution. Today, this is handled by creating a bespoke, sub-terrain timber formwork, which is

manually crafted at the site. For the test partner of the project, the duration of such work was estimated to be 2-3 days (normative average) for the formwork production.

To increase productivity, an alternative solution was proposed: a fully digitized production of EPS formwork, in which customization happens through a web application. Using an e-commerce pipeline, it is linked to a stationary, robotic CNC-milling facility which is located at the factory. The baseline can be established by comparing to current production of standard timber formworks used for precast well junctions from a factory. The process involves 3 levels of expertise: 1) a CAD specialist for expert modeling of the advanced geometry in a time span of 4-8 hours; 2) a CAM expert for programming the CNC milling of the complex piece within a time-span of 2-4 hours; and 3) a CNC operator, capable of inserting a work piece, executing a program and monitoring the process on a dedicated machining center, with on-off involvement for 3-4 hours of machining.

In the implementation reported in this paper, a GUI was made available online with sophisticated customization parameters for generating any unique combination of well diameter, number of pipes, individual pipe diameters and approach angles[1]. Within preset bounds of each parameter, an unskilled user with knowledge of the design specification for any given well junction can now configure the design within a 2-minute process, replacing the former need for 4-8 hours of modeling from a CAD specialist.

Upon submitting the order, the design is forwarded to an associated fabrication model, which procedurally generates the corresponding tool path, hereby negating the need for a CAM expert applying 2-4 hours of programming. Finally, this program is launched by an operator on a robotic system - a stationary 6 axis manipulator with external rotary axis. Due to the change of material from timber to lightweight EPS, we can operate the process at a higher speed, reducing machining time to 1-2 hours per piece, depending on well diameter or junction size.

B.8 Discussion and conclusion

In this paper, we have presented 3 seemingly unrelated applications for robotic construction manufacturing. Each case relies on a highly customized combination of hardware configuration, tooling process, and design geometry, serving a unique need for the industry. We demonstrate that a custom interface can be procedurally generated for each case using the same, standardized model across

[1] https://odico.dk/wellmate/

all 3 case applications. The generated visual interface is capable of reducing the operational complexity of the underlying system to such degree that an unskilled worker within few minutes of training can effectively and safely execute the work, to produce highly customized items. Through digitization, the automation process serves to significantly increase the levels of productivity by reducing the labor time.

As a company offering robotic solutions to construction industry, we often deal with building contractors who are averse to huge investments. Mostly, we get an order for only 2-3 cells supporting a specific robotic process. It is not financially prudent to treat each application differently and develop an user interface from the scratch.

Hence we developed a software framework where a robotic application is modelled using nested Petri nets. This framework was then used to quickly develop industry grade solutions reducing the application development time by 40%. Apart from other benefits related to process modelling, this approach resulted in modular architecture and explored the commonalities between multiple seemingly different applications.

In this work, we had presented three such applications viz. abrasive wire cutting of EPS formwork, sawing of tiles for pavements and milling of EPS formwork. All of these applications were expressed by an unifying framework. As a result, we were able to support semi-automated generation of user interfaces. Such interfaces were shown to be highly effective in allowing a low-skilled production worker to easily interact and generate robotic instructions for the task. Typically, a novice user needs only a 10 minute training session to be able to productively engage with our cells.

A novel method for semi-procedural generation of user interfaces based on a parametric model provided by a CDS was presented. As a result, a non-specialist construction personal was able to carry out complex machining processes associated with adaptive manufacturing of customized parts easily and safely. Furthermore, the interface enabled the user to monitor the robot execution.

Appendix C

Applying software design patterns for graph-modelled robotic workflows

C.1 Introduction

Since second half of the 20th century, the global construction industry has witnessed stagnating productivity, while other sectors such as manufacturing, has experienced a significant increase [PTT05, BHK21, BDM20]. Automation of construction processes is generally viewed as the primary instrument to increase productivity [DOA+19]. However, due to the high levels of project specificity, smaller lot sizes for production and need for customization, conventional automation technologies as seen in other industrial fields cannot be directly adopted within construction, thus causing the sector to remain vastly underserved.

It is our hypothesis that successful adoption of automation within global construction can only be achieved through a software framework enabling rapid design, development and testing of new robotic applications. In preceding work by authors of this paper, Higher Order Knowledge System (HOKS) has been introduced to overcome this challenge [NSB21]. This framework enables a parallel

workflow within the application development process, wherein contributors with diverse skill sets can collaboratively co-develop across their knowledge domains.

In this new paper, we demonstrate how software design patterns could contribute to further increasing the *efficiency* in the development of robotic applications. For the context of this paper, an *efficient* process denotes a flexible software development methodology which creates stable, reusable, performance optimal and maintainable parametric models. As a result, a parametric design model should be described in such a generalized manner, that enables straightforward editing by multiple users at any point in time, rather than depending on the original developer. This model transferability is crucial for commercial-grade software development, as any dependency on an individual developer or software engineer, will lead to the risk of single points of failure in the event of fluctuation in the team constitution.

C.2 State of the art

VPL based parametric modelling has since its introduction in early 2000's seen rapid adoption within the AEC industries for the design and engineering of complex building design [Mon00]. In particular, it enables the establishment of control hierarchies for managing component variables which achieve a larger global effect [Sch09]. More recently, parametric design thinking has enjoyed widespread interest within architectural and construction robotic research for the development of custom and experimental workflows [Sch13, Ras15, Gea16]. In complementary notes, core fields within general robotics have also acknowledged key advantages of VPLs in development of robotic applications, causing increasing research interest [SW16, HRSW13, PNBK13].

Some of the identified advantages in this approach are ease of programming, effortless knowledge encapsulation for less specialist developers, efficiency in establishing geometry-intensive workflows and modular architectures [PP13]. While these advantages are well established, the general scope of application for VPL based parametric modeling has historically been limited to custom or project specific scripts, graphs, functions or prototypical workflows within AEC organizations. As such, investigations into whether VPLs could be applied for development of commercial scale software applications are sparse.

Rick Smith has identified the following as some of the major challenges faced in parametric modelling [Dav13]:

1. Major changes can break parametric models. As it is difficult to foresee

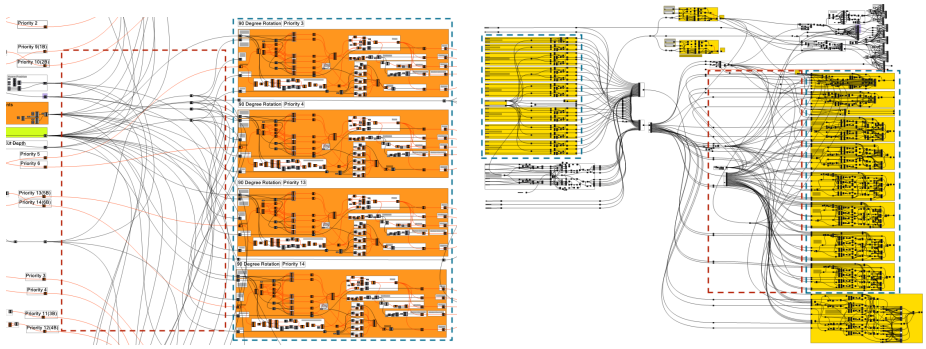
all feature requests in the beginning of a project itself, the instability of the model raises some grave concerns when one has to amend it later on to incorporate a request from the customer.

2. It is difficult to reuse and share parametric models. Often, only the original designer is able to operate with a model, as it can be arduous for another one to comprehend the design intent making the system averse to maintenance.

Around 1960s, a large number of major software projects unexpectedly failed in the broader industry, giving rise to the software crisis. As pointed out by Turing award winner Niklaus Wirth, "systems could not be built or delivered on time, bringing some companies to the brink of collapse" [Wir08]. One of the examples is, IBM's ambitious System/360 unification project, led by F. Brooks, which in 1964 was one of the largest software projects ever undertaken. It was years late and costed millions of dollars more than budgeted [Dav13].

In 1968, NATO decided to form a group of scientists to further analyze the problems faced in software engineering. As recalled by Naur and Randell, the talk was centered around "slipped schedules, extensive rewriting, much lost effort, large number of bugs, and an inflexible and unwieldy product" [NR68]. One of the main reasons for this problem, was the fact that software programs written during those days were largely unstructured with GOTO statements littered randomly across the source code.

In many ways, one can relate these findings from NATO, to the problems men-



(a) Parametric model 1 **(b)** Parametric model 2

Figure C.1: Examples of visual clutter (wires inside red coloured boxes) and code duplication (blocks inside teal coloured boxes) in parametric models.

tioned by Rick Smith [Dav13]. The analysis of parametric models created by two different Computational Design Specialists (CDSs) (see, Fig. C.1a and C.1b), in one of our previous works, is reminiscent of the same issues which led to the software crisis in 1960s. After a careful literature study, we have concluded that the same problem has been highlighted by others as well [Dav14, Jan14].

Parametric models are generally created using VPLs by a CDS. In general, a CDS is trained with a different set of skills as compared to a software developer. Current day programmers are very much acquainted with principles of software reuse, maintainability and stability. They practice these concepts in their day-to-day work as well. Some of the techniques prevalent in the industry are design patterns, modular software architectures as against monolithic systems, comments for future reference, automated module testing etc.

Though the term "design pattern" was an age old idea in the field of architecture, it was probably introduced by Christopher Alexander [AIS77] in a modern sense. He says, "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.".

The term was later adopted by the software engineering fraternity [GJV94] to describe abstract, well-established forms of program construction. A pattern is fully defined by a name, a problem statement, an abstract solution, and a discussion of implications.

The need for software design patterns in parametric models is explained in [WAK07, Qia09, Zbo15]. It is claimed that it will lead to robust and sustainable models. Following their foot steps, this paper, implements six design patterns while programming graph-modelled robotic workflows. Out of six patterns discussed here, five are new to the best of our knowledge, whereas one is a well known pattern from the Object Oriented Programming (OOP) world, applied to visual programming.

Since 1994, many design patterns have been suggested primarily for software systems based on OOP paradigm since languages based on this paradigm were on an ascendancy during that time. As we carry the concept of software design patterns to VPLs, we need to keep in mind that not all design patterns applicable for object oriented world are relevant to VPLs as these systems follow a Functional Programming (FP) paradigm. OOP is fundamentally different from FP.

C.3 Importance of VPLs

Many applications targeted at the construction industry need to be driven by parametric models. For example, in our wire-cutting Factory on the Fly unit, we need to visualize the target geometry (see, Fig. C.9a) in the view port. VPL provides a convenient way to express parametric models for the CDS.

Visual programming enables more than just software developers to program such robot applications. Modern day text based programming languages have several advanced constructs like delegates, access specifiers, and many more which might be difficult to access for non-expert programmers. A relevant analogy is using an operating system purely from command line as compared to the GUI available for Windows/Ubuntu/macOS. Unity acquiring the visual programming tool Bolt also shows how the game industry as well is looking to VPL as a way to accelerate the access to content creation to non-programmers.

The analysis will not be complete without looking into the Turing completeness associated with visual programming systems. It is important that a programming language or system is Turing complete since it means that it can then be used to solve any computational problem given enough time and memory. VPL follows the FP paradigm which is based on the Lambda calculus proposed by Alonzo Church in early 1930s. Lambda calculus is an universal model of computation that is used to simulate any Turing machine. The equivalence between Turing machine and Lambda calculus is discussed in the seminal work termed Church-Turing thesis [Jac20]. Since Lambda calculus is Turing complete, it can be argued that any given VPL can also be Turing complete in principle.

C.4 Research challenge

To summarize, this leads us to claim that visual programming is needed to widely design, develop and prototype robotic applications in construction industry. Such visual programming techniques often lead to the following problems:

1. It is generally considered that a parametric workflow is very difficult to be debugged and maintained by anyone other than the original designer. Such kind of dependencies will lead to single points of failure, wherein if the developer who has authored the workflow leaves the organization, the commercial product will remain elusive to the rest of the team.

2. In [Dav14], 2002 Grasshopper models were analyzed, researchers have reported that highly unstructured way of developing workflows with wires floating around is a big problem.

3. It has been observed that use of complex and large parametric models often results in latency. As we were using parametric models to create commercial-grade robotic applications, where users will perform on the fly design of custom components using a tablet application, latency is of big concern for us.

Now, to make this process sustainable, we will borrow some ideas from the discipline of software engineering and apply them to parametric modeling. Software design patterns are here employed to address concerns of latency, stability, maintainability and reuse of fairly complex parametric models. By this, we will be able to help avoid a situation compared to the software crisis happening to parametric design.

C.5 Petri nets

Petri net is a graphical and mathematical modelling tool applicable to many systems. In the context of our work, we have used Petri nets to represent a parametric model where data flows between a set of interconnected nodes (refer, Fig. C.2c). Each node stands for a function (transition) which accepts n inputs (input places), performs some operations on these inputs to emit m outputs (output places).

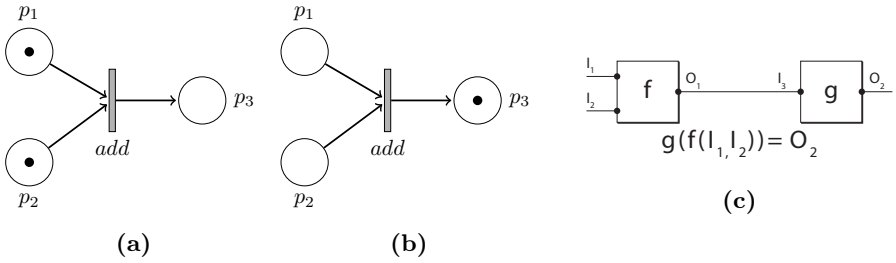


Figure C.2: (a) A sample Petri net where p_1, p_2 and p_3 are places and *add* is a transition. *add* is in enabled state. (b) Firing of *add* deducted 1 token each from p_1 and p_2 and added 1 token to p_3. (c) A simplified data flow model.

Input places	Transitions	Output places
Pre-conditions	Event	Post-conditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Job or task	Resources released
Buffers	Processor	Buffers

Table C.1: Some typical interpretations of places and transitions in Petri net

The concept of Petri nets was introduced by Carl Adam Petri in his dissertation submitted in 1962. It is a bipartite directed graph consisting of two types of nodes, namely transitions and places, where edges are either from a place to a transition or vice versa. It is forbidden to connect either a place with another place or a transition with another transition using an edge. Graphically, a place is denoted using circles whereas bars or thin boxes represent transitions. A marking (state) assigns to every place in the graph a non-negative integer. If a state assigns to a place (p) a non-negative integer k, we can say that p is marked with k tokens. Pictorially, if a particular p holds k tokens, we will draw k black dots (tokens) inside of p. A marking is denoted M where $M(p)$ gives the number of tokens in p. Some interpretations for transitions or places and a formal way to define Petri nets are given in Tab. C.1 and C.2 respectively.

The *transition firing* rule for a Petri net is given by:

1. t_i is said to be enabled if each input place of t_i is marked with at least w tokens, where w is the weight of the edge from the corresponding input place to t_i (see, Fig. C.2a).

2. A firing of an enabled transition removes k tokens from each input place of t_i and adds k tokens to each output place of t_i as shown in Fig. C.2b.

A Petri net (N) is given by (P, T, E, W, M_0):
$P = \{p_1, p_2, ..., p_m\} \mid p_i$ is a place,
$T = \{t_1, t_2, ..., t_n\} \mid t_i$ is a transition,
$E = \{e_1, e_2, ..., e_q\} \mid e_i$ is an edge between p_a and t_b,
$W = \{w_1, w_2, ..., w_q\} \mid w_i$ is the weight associated with e_i,
$M_0 = \{n_1, n_2, ..., n_m\} \mid n_i$ denotes the initial marking for p_i,
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$

Table C.2: Formal definition of a Petri net

C.6 Design patterns

It is to be noted that ability to decide which design patterns can be applied to in a given situation comes with practice. In general terms, it follows the idea of factoring out the commonalities between various problems and abstracting the solution so that it can be used under other situations as well. As an example, let us look at Observer pattern popular among members from OOP community. It can be used under any situation where one process (publisher) is producing some information which many other processes (subscribers) will be interested in listening to. Some of the concrete examples of its implementations are:

1. Pick and place robot - A camera fitted to the robot arm publishes the pose of the object to be picked up by the arm as and when it enters within the reachable workspace of the robot.
2. If someone whom you are following in Twitter or Facebook publishes a new post, you will be notified about the same.

In the above example, if one chooses not to see beyond the specificity of a given problem, it will be difficult to see the commonality between the two cases. After analysis of parametric design models created by us, we have identified some patterns and categorized them into three groups viz. functional, relational and performa in this paper. It is not an exhaustive list and more patterns may be recognized as we continue our exercise in future.

A design pattern is introduced by a **Title**, **What**, **Use when**, **Why** and **How** as is followed in [WAK07]. The **Title** is a short name given to refer to a pattern. **What** provides a short description of the pattern. **Use when** mentions about the situations under which the said pattern is relevant to be considered and **Why** signifies the inspiration for using the pattern and sketches the associated benefits. **How** refers to the internal details on how they can be implemented within the given context.

C.6.1 Functional patterns

Functional patterns deal with breaking down the parametric models into simpler logical sub units. It will help to reduce unnecessary coupling between nodes, improves readability and makes the graph more maintainable. We introduce and discuss three functional patterns namely Model-View-Controller (MVC), Design-Plan-Monitor (DPM) and adapter in this section.

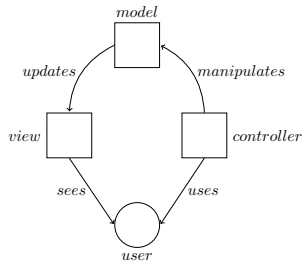


Figure C.3: Schematic representation of MVC

C.6.1.1 MVC

1. **What:** Organizes the transitions and places into three collections, namely model, view and controller (refer, Fig. C.3). A collection is simply a bevy of transitions and places.

2. **Use when:** You are developing a complex Petri net to model an interactive parametric process involving visual feedback. The said process can be controlled via a set of parameters.

3. **Why:** Most interactive robotic applications can be conveniently subdivided into model, view and controller so as to avoid unnecessary coupling between the various transitions. Model holds the underlying geometry data and drives the application. It can be visualized through the view. Transitions present in the controller will let the user interact with the model.

4. **How:** The model is a group of nodes which defines the geometry or tool paths for a parametric design. For instance, the model nodes in the parametric design might produce a mesh or smooth surface defined in terms of vertices or control points. The view is formed by a aggregate of nodes which determines the visualization aspects of the model, like the thickness of a curve, style or colour of a surface. A given model may have multiple associated views viz. plan, elevation, detailed etc. The controller is a set of interactive widgets that correspond to source nodes. The controller al-

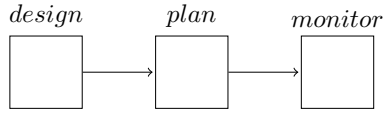


Figure C.4: Schematic representation of DPM

lows the user to change the parameters governing the geometry defining nodes in the model.

C.6.1.2 DPM

1. **What:** Organize the transitions and places into 3 collections namely design, plan and monitor as shown in Fig. C.4.
2. **Use when:** If you are developing a robotic application which involves the design on the fly feature using Petri nets.
3. **Why:** In many of the fabrication centered robotic applications from the construction industry, it is required that the user should be able to customize the part being manufactured. A strategy is to expose some design/fabrication related parameters to the production user so that he or she can tweak the parameters to achieve the desired customization. It is our proposal that a neat way of handling such parametric workflows is to decouple the process into three sections viz. design, plan and monitor.
4. **How:** It can be implemented in the following way:
   (a) **Design** - Specifies the design of the product to be manufactured.
   (b) **Plan** - From the given design of the product, a machining strategy is determined. It involves applying lead in/out, determining the process speed depending on material properties, deciding when to turn on/off the tool and extracting tool path from the given product geometry. Further, robot motion will be planned taking into account the collision matrix. The output of this step will be a task which can be executed on the robot.
   (c) **Monitor** - Once the robot starts to execute the task, the cell has to be continuously monitored for safe operation. Human safety is of paramount importance. In case of any safety sensor breach, the robot is stopped. One can include further recovery strategies in this section to handle other error scenarios.

C.6.1.3 Adapter

1. **What:** Adapt the outputs of preceding workflows to match the input structure of the succeeding one as depicted in Fig. C.5.
2. **Use when:** If there are multiple upstream workflows that converges onto a singular downstream workflow, it is important that all these upstream

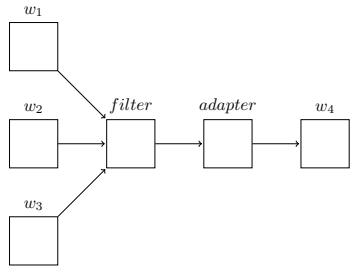


Figure C.5: Schematic representation of adapter - w_1, w_2 and w_3 are upstream workflows whereas w_4 is the singular downstream workflow. At any given moment, $filter$ lets output from either w_1, w_2 or w_3 to pass through. *adapter* massages the outputs from w_1, w_2 or w_3 to a form which is consistent with the expected input of w_4.

workflows pass on a consistent output form - one that can be acted upon by the downstream workflow. In many cases, individual upstream workflows will be generating outputs in a different form.

3. **Why:** It helps to avoid duplicating the downstream workflows to suit the output form from each of the individual upstream workflows.

4. **How:** An adapter workflow is included between the said workflows, to massage the outputs of the upstream workflows such that they are synchronous with one another and match the input form of the succeeding workflow.

C.6.2 Relational patterns

Relational patterns aid us to better organize the data models within the parametric design workflows to represent the relations between them accurately. We are only talking about one relational pattern namely mask here.

C.6.2.1 Mask

1. **What:** Maintain a mask that will be used to qualify the underlying data (see, Fig. C.6).

2. **Use when:** In most parametric workflows, each Geometry Object (GO) will have metadata associated with it. Masks can be used to store such

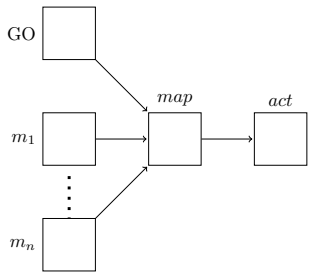


Figure C.6: Schematic representation of mask where GO is the geometry object, $m_1, .., m_n$ are masks to be applied to GO.

metadata. Typically, information like colour, material so on and so forth can be stored using masks.

3. **Why:** Usually in OOP based languages, under these scenarios, a custom class can be defined which includes the GO with all its associated metadata. However, in FP based languages, we cannot create such custom classes. Hence, the masks need to be stored as a separate data structure, so that a downstream workflow can associate the GO with a particular mask and perform some action on the GO accordingly.

4. **How:** Masks are stored using separate data structures, which have the same form as that of the ones holding GOs. They can be either constants, external inputs or be generated dynamically within a workflow. There can be multiple masks, each pertaining to one aspect of the geometry.

C.6.3 Performa patterns

These patterns help us to reduce latency in a given parametric workflow to improve performance. We were able to come up with two patterns belonging to this category viz. isolator and cache. Each of these patterns is discussed below in greater detail.

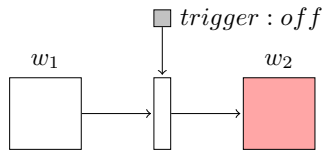


Figure C.7: Schematic representation of isolator

C.6.3.1 Isolator

1. **What:** Isolate a particular workflow from the upstream workflow logically as shown in Fig. C.7.
2. **Use when:** In many situations, we will encounter downstream workflows which shouldn't get recomputed at every instance of a change to the upstream workflow for various reasons. We can use this pattern as a bridge between two workflows to achieve the same.
3. **Why:** If there are computationally heavy downstream workflows within a parametric workflow, users will experience a latency while interacting with the parameters of the upstream workflow. Hence, designers may want to strategically defer the recalculation of such dependents so as not to hamper the user experience.
4. **How:** An isolator component can be developed which will collate and store the outputs from the upstream workflow. It will be relayed on to the downstream workflows, only on request.

C.6.3.2 Cache

1. **What:** Store results from a single execution of the Petri net (see, Fig. C.8).
2. **Use when:** It can be used when one has to use the results from the previous iteration of the program during the next run.
3. **Why:** If the computation of a Petri net introduces latency due to the time complexity of the algorithm, one may be able to speed up the process by caching the output from the previous run under certain situations. By using the cached value, one can perform the current execution as an incremental operation to enhance user experience.

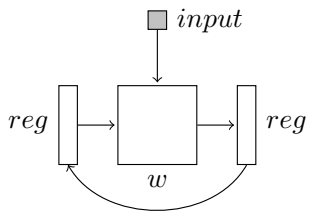


Figure C.8: Schematic representation of cache

4. **How:** A register component can be developed which can be used to stow the outputs from the Petri net.

C.7 Case studies

C.7.1 Application 1: Additive Manufacturing (AM)

3D Concrete Printing is an additive manufacturing technique to fabricate buildings or functional components for construction. In recent commercial applications, the system is based on the extrusion of cement-based material through a nozzle of a variable size/shape. Because of this, building components are produced by layered wall elements and often made of partially hollow geometries, in order to save time and material. For structural rigidity, one cannot choose a random pattern for material deposition inside the shell. In one of the pre-studies conducted by us, we were exploring the possibility of utilizing the Differential Growth (DG) [PS06] of a continuous curve to achieve a beam design of a variable density, with a continuous path that can be printed without interruptions (see, Fig. C.9b). Since DG is a well-known strategy for such problems, we will not elaborate on the algorithm in this paper. The input to the program is a shell (simple polygon), a scalar, and a growth factor. The growth factor controls the rate at which the curve will grow in every iteration. The scalar field is a function which associates a number to every point in space. A grayscale image was used as the field wherein the intensity lies between 0 (white) and 1 (black). In our case, we bias the growth of the curve according to the field which means that more material will be deposited in areas corresponding to darker pixels from the

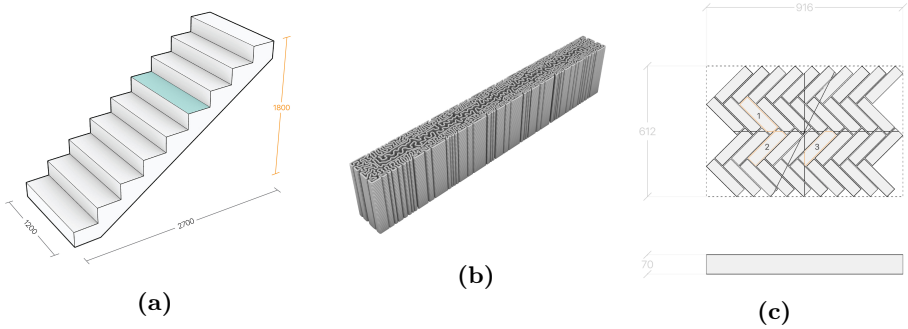


Figure C.9: MVC applied to 3 robotic applications: (a) abrasive wire cutting; (b) additive manufacturing and (c) sawing.

image. The scalar field follows the result of a structural stress analysis so that a darker pixel from the field means that more material needs to be deposited around that point.

C.7.2 Application 2: Abrasive wire-cutting

A standard 6-axis robot arm fitted with an abrasive wire-cutting tool is used to manufacture moulds for concrete casting (see, Fig. C.10a). The details can be found in [NSB]. The results are demonstrated on moulds used for casting a staircase in concrete. A tablet application was created which exposes relevant variables of the underlying parametric model for supporting product customization. On finalizing the customization, the production user can submit the mould parts for fabrication. The tool path is then extracted from parametric surfaces and relayed to a robot cell equipped with an abrasive wire-cutting tool which cuts into an Expanded Polystyrene block to create the mould.

C.7.3 Application 3: Sawing

For creating pavements to facilitate pedestrian transit, concrete and natural stone tiles are often required to be reshaped onsite because tiles of the required dimensions are not available. It mostly happens when one needs to ensure a snug fit around lamp posts or corners. We developed a mobile robotic cell, which is driven around to the site. It contains a 6-axis robotic manipulator fitted with a circular saw (see, Fig. C.10b). The tiles can be placed inside the cell, and then robot can be instructed to cut according to a design created on the fly using a tablet interface.

(a) Abrasive wire-cutting

(b) Sawing

Figure C.10

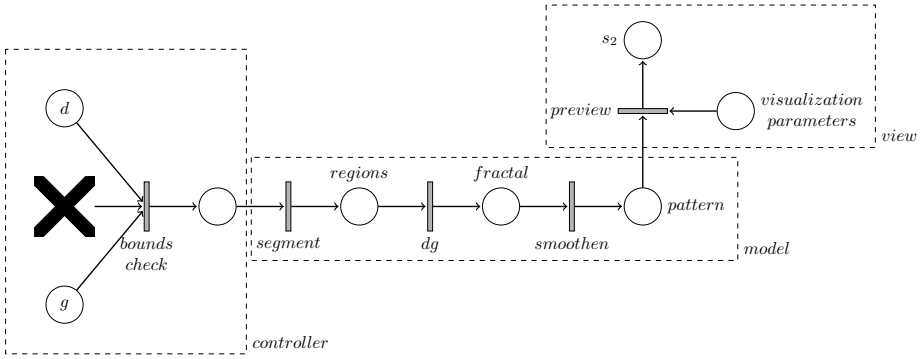


Figure C.11: MVC applied to simplified Petri net model associated with concrete additive manufacturing application with three parameters viz. diameter of the nozzle (d), scalar field and growth factor (g). The generated pattern is shown in the view port (refer, Fig. C.9b).

A digital model of the tile is exposed to the user where one can edit certain associated parameters. For example, length or width of the tile can be tweaked by the user to suit his or her needs. Post this, user can apply cuts onto the model. User can select one or more resulting pieces and instruct the robot to cut (refer, Fig. C.9c).

C.8 Discussion and conclusion

Having established that parametric modelling holds a significant potential for increasing the efficiency in building robotic applications in construction industry, we have tried to address important open problems pertaining to maintainability, stability, reusability and performance of parametric models.

If a given parametric model is maintainable, then developers other than the original one will be able to comprehend the model and be able to fix bugs or develop new features. It is our claim that functional patterns infuse this desired flexibility by a systematic organization of the parametric model into simpler Functional Units (FUs) which improves the readability of the model. Each FU is a bundle comprising of nodes which are closely related to each other serving a specific purpose in the application. To mention one of our experiences, the CDS who developed the initial parts of the sawing application, abruptly had to be pulled into another project due to business needs. However, a new CDS was able to take over the project after four hours of knowledge transfer.

As mentioned briefly before, it is often difficult to predict all the desired features at the beginning of the product life cycle itself. Quite often, this is an iterative process wherein, based on customer inputs, a decision will be made to incorporate a certain feature into the system. A stable model is amenable to such new feature requests. One should be able to incorporate such requests without breaking the existing model. Functional patterns can help to ensure only a necessary level of coupling exists between FUs. Hence a change in one section of the model, will only have a minimal impact on the other parts, thus reducing the chances of ending up with a broken model. This results in a highly efficient process for a CDS to come in and incorporate new requirements.

It has been one of our endeavors to build reusable visual functions. Many VPLs provide support for creating such functions. For example, Bolt and Grasshopper have super units and clusters respectively. Since design patterns is all about abstracting the given problem and solving the abstracted problem to maximize generality, it is our claim that by applying functional patterns, one can try to create highly reusable FUs.

In our case, MVC and DPM were employed in parametric workflows to reduce coupling between various parts of the workflow increasing the ease at which new features or change requests can be accommodated during the later phases of the project without breaking the Petri net. Fig. C.11 and C.12 depict the disintegration of wire-cutting and additive manufacturing processes according to MVC pattern. A similar approach was adopted for sawing application as well. Model is responsible to bear the skeletal structure of the product geometry under consideration whereas view guides the display aspects of the model. Controller handles how a user can interact with the view and model. If the user selects a GO (for example, a surface or a dimension object), it gets highlighted in a different colour. A mask (refer, Fig. C.12) that holds the selection state of each GO is maintained separately. If a GO gets selected, its value in the mask becomes true, which makes the *preview* render it in a different colour.

A model FU created by applying MVC pattern, can be further dissected into 3 constituent units viz. design, plan and monitor in most of the robotic applications. Fig. C.13 depicts how this was achieved in the case of sawing application. The application presents the users with multiple cut options. The number of inputs and the modelling workflows as such are subtly different for each of these options, but they are adapted using an adapter workflow and then passed onto a singular workflow involving the split operation within the design FU.

The planning and monitoring operations are downstream to design FU. In conventional parametric modelling schema, all downstream workflows will be computed in one go if any of the parameters associated with an upstream workflow changes. Introduction of isolation helped us to segregate the recomputation.

On changing any parameter of the design workflow, isolator will act as a shield so that the subsequent workflows will get recomputed only on receiving the trigger. Such a solution is not just specific to one application, but has helped us to abstract the problem and apply the same strategy on other applications as well where we had to stave off execution of dependent workflows to a more appropriate time. One may want to do that for many reasons, an example could be that the downstream workflow is computationally heavy. Another common situation is that certain workflows initiate a robotic process, due to which it should get triggered only when explicitly requested by the user as he or she wants to change the design multiple times before sending the instruction to the robot.

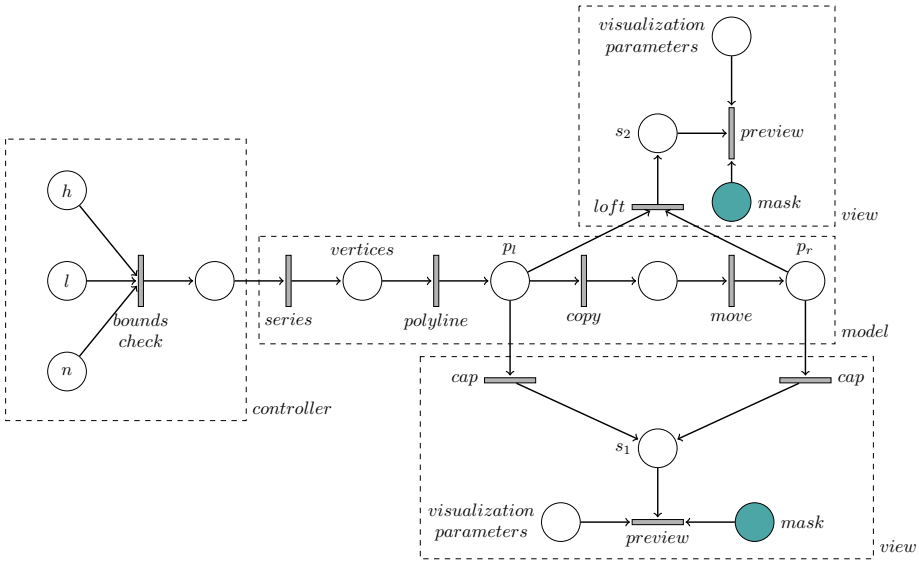


Figure C.12: MVC and mask applied to simplified Petri net model associated with product design in wire-cutting application. For brevity, we have only shown 3 parameters viz. height (h), length (l) and number of steps (n) here. Usually, the model will possess 10-15 design related parameters along with another set of parameters for fabrication. The latter set of parameters are dependent on material properties like hardness (for example, cutting speed). The design model is responsible for rendering the target geometry in the view port. A user of the tablet application can interact with the geometry by selecting a surface, a curve or a dimension object. In this example, the selected surface and dimension object are shown in teal and orange respectively in Fig. C.9a.

The concept of cache was used to optimize a specific workflow in robot sawing application. As a cut is added, computation of an associated Petri net will be triggered which will split all the tiles in the pattern affected by the cut (see, Fig. C.9c). After that, if the user chooses to add another cut, one has to either work on the output of the previous split operation or perform the operation twice

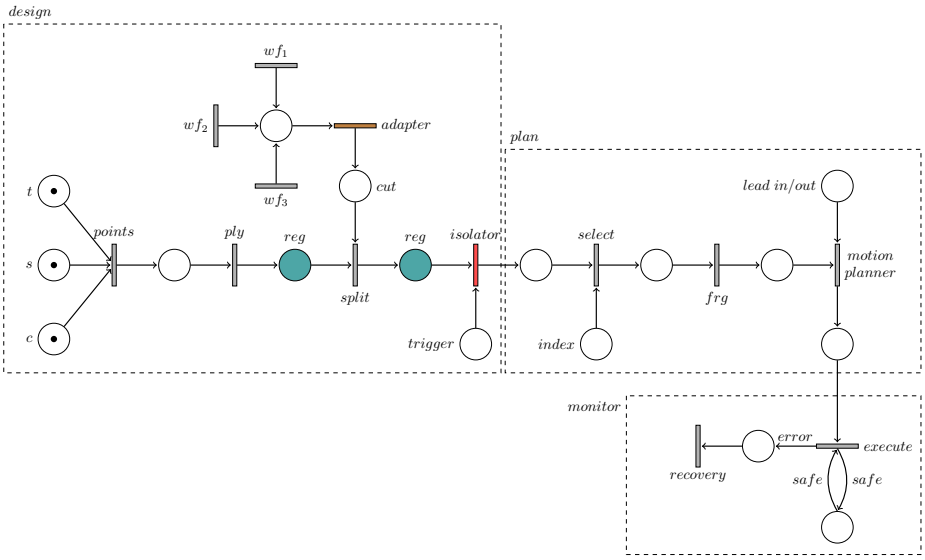


Figure C.13: Simplified Petri net representation of the robotic sawing application. *design* FU creates the tile pattern based on the parameters - type (t), size (s) and count (c). Register (reg) caches the created polygons. User can specify a particular cut by initiating one of the workflows wf_1, wf_2 or wf_3. Each of these workflows is specific to an option available to the user in the interface. For example, wf_1 might correspond to the user adding a horizontal cut onto the tile pattern. Depending on the cut chosen, either wf_1, wf_2 or wf_3 will be triggered. Outputs from these workflows differ slightly in form which is massaged using the adapter block (shown in brown). Post massage action and receiving a cut, *split* operates on the cache available in reg (shown in teal) and overwrites reg with its output. The next cut instruction operates on the current data available in reg. *isolator* (depicted in red) behaves like a normally open switch. On receiving the *trigger* signal, the switch gets closed and any downstream nodes will get recomputed. In our case, downstream nodes are the ones shown inside of *plan* and *monitor* FUs.

(once for the first cut and subsequently for the second one). It has been noted that former approach will result in an optimal performance. As stated before, parametric workflows are generally intended to be executed as a single chunk of operations where you generally don't work with the result of the previous iteration. We introduced the concept of registers (refer, Fig. C.13) to support such a procedure which resulted in up to 2x improvement in performance.

We have presented five novel software design patterns and applied it to three graph-modelled robotic workflows - out of which two are commercial products - to demonstrate that it can be used to create highly maintainable, stable, reusable and performance optimal solutions. The crisis mentioned at the beginning of the paper was created due to the unstructured way of developing software. It resulted in a non-maintainable system which never could support changing customer requirements during its life cycle in an agile manner. Recently, many practitioners have observed a similar situation developing in parametric modelling as well. Hence, by helping to develop maintainable and stable models, we were able to contribute to solving the larger crisis looming the industry.

Further, we have demonstrated the relevance of the previously reported MVC pattern in the same context. Increasing the efficiency in applications developed through visual programming may help overcoming limitations in this development paradigm, critical to furthering the use of robotics within the construction industry.

Bibliography

[ABF+16] A. Apolinarska, R. Bärtschi, R. Furrer, F. Gramazio, and M. Kohler. Mastering the sequential roof. *Advances in Architectural Geometry*, pages pp. 240–258, 2016.

[AIS77] C. Alexander, S. Ishikawa, and M. Silverstein. A pattern language: towns, buildings, construction. *Center for Environmental Structure Series*, 2:pp. 1–1171, 1977.

[AMAI19] H. Abdelfetah, A. Mustapha, M. Abderraouf, and A. Isma. Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017. *Advanced Robotics*, 33:764–799, 2019.

[BDM20] M. Bragliaa, P. Dallasegab, and L. Marrazzinia. Overall construction productivity: a new lean metric to identify construction losses and analyse their causes in engineer-to-order construction supply chains. *Production Planning and Control*, 1:pp. 1–18, 2020.

[Ber16] N. Berg. Nbbj releases human ui to bring parametric modeling to the masses. *Architect Magazine*, 1:1–1, 2016.

[BHK21] J. Berlak, S. Hafner, and V. G. Kuppelwieser. Digitalization's impacts on productivity: a model based approach and evaluation in germany's building construction industry. *Production Planning and Control*, 32:pp. 335–345, 2021.

[Bho17] S. Bhooshan. Parametric design thinking: A case-study of practice-embedded architectural research. *Design Studies*, 52:pp. 115–143, 2017.

[Boc07] T. Bock. Construction robotics. *Autonomous Robots*, 22:201–209, 2007.

[Bon15] T. Bonwetsch. Robotically assembled brickwork: Manipulating assembly processes of discrete elements. *Doctoral dissertation, ETH Zurich*, 2015.

[Dav13] D. Davis. *Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture*. PhD thesis, RMIT University, 2013.

[Dav14] D. Davis. Quantitatively analysing parametric models. *International Journal of Architectural Computing*, 12:pp. 307–319, 2014.

[Dea19] K. Dörfler and et al. Mobile robotic fabrication beyond factory conditions: Case study mesh mould wall of the dfab house. *Construction Robotics*, 3:pp. 53–67, 2019.

[DOA+19] J.M.D Delgado, L. Oyedele, A. Ajayi, L. Akanbi, O. Akinade, M. Bilal, and H. Owolabi. Robotics and automated systems in construction: Understanding industry-specific challenges for adoption. *Journal of Building Engineering*, 26:pp. 1–11, 2019.

[DRR13] K. Dörfler, F. Rist, and R. Rust. Interlacing : An experimental approach to integrating digital and physical design methods. *Robotic Fabrication in Architecture, Art, and Design*, 1:82–91, 2013.

[EGK17] P. Eversmann, F. Gramazio, and M. Kohler. Robotic prefabrication of timber structures: towards automated large-scale spatial assembly. *Construction Robotics*, 1:pp. 49–60, 2017.

[Ela08] S.M.S Elattar. Automation and robotics in construction: opportunities and challenges. *Emirates Journal for Engineering Research*, 13:21–26, 2008.

[FS14] J. Feringa and A. Søndergaard. Fabricating architectural volume: stereotomic investigations in robotic craft. *Fabricate : negotiating design and making*, 2:pp. 44–51, 2014.

[GCF18] S. Ghaffar, J. Corker, and M. Fana. Additive manufacturing technology and its implementation in construction as an eco-innovative solution. *Automation in Construction*, 93:pp. 1–11, 2018.

[Gea16] C. Gosselin and et al. Large-scale 3d printing of ultra-high performance concrete–a new processing route for architects and builders. *Materials and Design*, 100:102–109, 2016.

[GJV94] E. Gamma, R. Helm R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, volume 1. Addison-Wesley, 1994.

[GK14] F. Gramazio and M. Kohler. *Made by robots: challenging architecture at a larger scale*, volume 1. Wiley, 2014.

[GKW14] F. Gramazio, M. Kohler, and J. Willmann. Authoring robotic processes. *Architectural Design*, 84:14–21, 2014.

[GM18] S. Ghaffar and P. Mullett. Commentary: 3d printing set to transform the construction industry. *Proceedings of the Institution of Civil Engineers - Structures and Buildings*, 171:pp. 737–738, 2018.

[GSMG17] L. Ghandi, C. Silva, D. Martinez, and T. Gualotuna. Mobile application development process: A practical experience. *Iberian Conference on Information Systems and Technologies (CISTI)*, 1:1–6, 2017.

[HLY16] H. Hu, Y. Liu, and L. Yuan. Supervisor simplification in fmss: Comparative studies and new results using petri nets. *IEEE Transactions on Control Systems Technology*, 24:pp. 81–95, 2016.

[HRSW13] U. Thomas; G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann. A new skill based robot programming language using uml/p statecharts. *IEEE International Conference on Robotics and Automation (ICRA)*, 1:pp. 461–466, 2013.

[HSZL16] H. Hu, R. Su, M. Zhou, and Y. Liu. Polynomially complex synthesis of distributed supervisors for large-scale amss using petri nets. *IEEE Transactions on Control Systems Technology*, 24:pp. 1610–1622, 2016.

[IMKT21] S.H. Iqbal, H.M. Moshiul, U.M. Kafil, and A. Tawfeeq. Mobile data science and intelligent apps: concepts, ai-based modeling and research directions. *Mobile Networks and Applications*, 26:285–303, 2021.

[Jac20] C. B. Jack. The Church-Turing Thesis. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2020 edition, 2020.

[Jan14] P. Janssen. Visual dataflow modelling - some thoughts on complexity. *Education and research in Computer Aided Architectural Design in Europe Conference*, 2:pp. 547–556, 2014.

[KDA12] M. King, N. Dave, and Arvind. Automatic generation of hardware/software interfaces. *Computer Architecture News*, 40:325–336, 2012.

[KSM+14] O.D. Krieg, T. Schwinn, A. Menges, J. M. Li, J. Knippers, A. Schmitt, and V. Schwieger. Biomimetic lightweight timber plate shells: computational integration of robotic fabrication, architectural geometry and structural design. *Advances in Architectural Geometry*, pages pp. 109–125, 2014.

[Men12] A. Menges. Morphospaces of robotic fabrication. *Robotic Fabrication in Architecture, Art, and Design*, pages pp. 28–47, 2012.

[MJ06] P. Meso and R. Jain. Agile software development: Adaptive systems principles and rest practices. *Information Systems Management*, 23:19–30, 2006.

[Mon00] J. Monedero. Parametric design: a review and some experiences. *Automation in Construction*, 9:pp. 369–377, 2000.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580, 1989.

[NR68] P. Naur and B. Randell. Software engineering: A report. Technical report, NATO SCIENCE COMMITTEE, 1968.

[NSB] N. Neythalath, A. Søndergaard, and J.A Bærentzen. Procedural generation of human machine interfaces from graph-modelled robotic workflows. submitted.

[NSB21] N. Neythalath, A. Søndergaard, and J.A Bærentzen. Adaptive robotic manufacturing using higher order knowledge systems. *Automation in Construction*, 127, 2021.

[Pir05] N. Pires. Robot-by-voice: experiments on commanding an industrial robot using the human voice. *Industrial Robot*, 32:505–511, 2005.

[PNBK13] M. R. Pedersen, L. Nalpantidis, A. Bobick, and V. Krüger. On the integration of hardware-abstracted robot skills for use in industrial scenarios. *Cognitive Robotics Systems: Replicating Human Actions and Activities*, 1:pp. 1166–1171, 2013.

[PP13] T. Peters and B. Peters. *Inside Smartgeometry: expanding the architectural possibilities of computational design*, volume 1. Wiley, 2013.

[PS06] H. Pedersen and K. Singh. Organic labyrinths and mazes. *International symposium on non-photo realistic animation and rendering*, 1:pp. 79–86, 2006.

[PTT05] H.S Park, S.R Thomas, and R.L Tucker. Benchmarking of construction productivity. *Journal of Construction Engineering and Management*, 131:pp. 772–778, 2005.

[PZTG18] S.C. Paul, G.P.A.G.V. Zijl, M.J. Tan, and I. Gibson. A review of 3d concrete printing systems and materials properties: Current status and future research prospects. *Rapid Prototyping*, 24:pp. 784–798, 2018.

[Qia09] Z. C. Qian. *Design Patterns: Augmenting design practice in parametric CAD systems*. PhD thesis, Simon Fraser University, 2009.

[Ras15] F. Raspall. A procedural framework for design to fabrication. *Automation in Construction*, 51:132–139, 2015.

[RMAHGA+15] V.Y. Rosales-Morales, G. Alor-Hernández, J.L. García-Alcaráz, R. Zatarain-Cabada, and M.L. Barrón-Estrada. An analysis of tools for automatic software development and automatic code generation. *Revista Facultad De Ingeniería*, 77:75–87, 2015.

[Roh08] M. Rohana. *An investigation into the barriers to the implementation of automation and robotics technologies in the construction industry*. PhD thesis, Queensland University of Technology, 2008.

[SB13] S.M. Sepasgozar and L.E. Bernold. Factors influencing the decision of technology adoption in construction. *International Conference on Sustainable Design and Construction*, 1:654–661, 2013.

[Sch09] P. Schumacher. Parametricism: A new global style for architecture and urban design. *Architectural Design*, 79:pp. 14–23, 2009.

[Sch13] T. Schwartz. Hal. *Robotic Fabrication in Architecture, Art, and Design*, 1:92–101, 2013.

[Sea16] A. Søndergaard and et al. Robotic hot-blade cutting. *Robotic Fabrication in Architecture, Art and Design*, pages pp. 150–164, 2016.

[SF17] A. Søndergaard and J. Feringa. Scaling architectural robotics: construction of the kirk kapital headquarters. *Proceedings of Fabricate*, 1:pp. 264–271, 2017.

[SW16] F. Steinmetz and R. Weitschat. Skill parametrization approaches and skill architecture for human-robot interaction. *IEEE International Conference on Automation Science and Engineering (CASE)*, 1:pp. 280–285, 2016.

[TPM18] M. Tamke, N. Paul, and Z. Mateusz. Machine learning for architectural design: Practices and infrastructure. *International Journal of Architectural Computing*, 16:pp. 123–143, 2018.

[WAK07] R. Woodbury, R. Aish, and A. Kilian. Some patterns for parametric modeling. *Proceedings of the 27th Annual Conference of the Association for Computer Aided Design in Architecture*, 1:pp. 222–229, 2007.

[Wir08] N. Wirth. A brief history of software engineering. *IEEE Annals of the History of Computing*, 30:pp. 32–39, 2008.

[WKB+16] J. Willmann, M. Knauss, T. Bonwetsch, A. A. Apolinarska, F. Gramazio, and M. Kohler. Robotic timber construction — expanding additive fabrication to new dimensions. *Automation in Construction*, 61:pp. 16–23, 2016.

[YYAR02] A. Yariv, D. Carmel; M.S. Yoelle, S. Aya, and L. Ronny. Knowledge encapsulation for focused search from pervasive devices. *ACM Transactions on Information Systems*, 20:25–46, 2002.

[Zbo15] M. A. Zboinska. Boosting the efficiency of architectural visual scripts. *Modelling Behaviour*, 1:pp. 479–490, 2015.