

音楽土木工学を設計する——音楽のためのプログラミング言語
mimum の開発を通じて

Designing Civil Engineering of Music through the development of
mimum, a programming language for music

松浦 知也
Matsuura Tomoya

2022 年 3 月

概要

本論文で筆者は音楽のためのプログラミング言語 mimium の開発の経験を通して、音楽土工学という、音楽とテクノロジーのこれまでと異なる研究領域を提示する。

第1章では音楽土工学の概念と論文全体の構造を示す。音楽土工学とは、例えばプログラミング言語や OS、コンピューターアーキテクチャのように、必ずしも音楽のためだけに作られたわけではないが、音楽の技術環境に大きく影響を与えるものを音楽の視点から改めて再考する研究アプローチである。

今日、理論的にはあらゆることが可能である、電子計算機を介すことなしに音楽の制作や聴取を行うことはもはや難しくなっている。しかしそのような環境下にもかかわらず、表現者が自分自身でプログラムを書き自らのための道具を作るという、パーソナルコンピューティングの当初の思想、メタメディア (Kay and Goldberg 1970) としての電子計算機の姿は主流から程遠く、音楽の技術環境は常に汎用的な技術インフラストラクチャに依存している。本研究は、そうした環境の変化を促す行動である音楽土工学という学問を音楽のためのプログラミング言語 (Programming Language for Music: PLfM) という、コンピューター上で音を生成するための人工言語の設計と開発という実践と思索の中から描きだす。

第2章では、本研究が取る、道具を実際に作る中で問題を明らかにするプロセスを、デザイン実践を通じた研究 (Research through Design: Rtd) という、科学と異なる方法論で人工物を生み出すことにより学術知に貢献する方法を足がかりに、本研究の学術的方法論について説明する。1960年代以降、特にコンピューターを用いたシステムのデザインは定量的で反証可能な科学のプロセスを取り入れてきた。しかし1980年代以降、解決すべき問題の複雑化や、科学技術社会論や現代美術など、隣接する領域の影響を受けながら、単に役に立つものを作るだけでなく、定量的に効果も測定できないような、人工物の創出を通じて社会に問いを投げかけたり、議論を誘発することを目的としたデザインの意義が提示されるようになってきた。また近年では、あらかじめ完成しないことを前提としたり、長い時間をかけた社会の変革を目指すような、デザインを長い時間軸の中で捉える方法論も立ち上がりつつある。本研究は Rtd の中でも、メディア考古学と批評的デザインの接続 (Jo 2016) や、批評的奇譚づくり: Critical Fabulation (Rosner 2018) のような、実践を通じて異なるメディア技術の歴史認識を提示するアプローチを参照し、PLfM の制作を、単に新しい音楽表現を生み出すこと以外にも、その設計を通じて異なる音楽のためのテクノロジーのあり方を提示する形での貢献として提示する。

第3章では、PLfM を異なる歴史の視点から捉え直すための背景として、現代のコンピューティング環境を、メタメディアの理想が不完全な形で実現された状態として捉え、その環境下で可能な音楽家のテクノロジーに対する関わり方の1つとしての PLfM 設計を位置付ける。音楽にテクノロジーを積極的に用いる実践のあり方として、2000年代までは機能していたサーキットベンディングやグリッチのようなアマチュアリズムを伴う意図的な技術の誤用 (Cascone 2009) は今日のソフトウェア中心の、かつブラックボックス化された音楽技術文化の中ではもはや機能しない。近年の技術に焦点を当てたデザイナーやアーティストの活動を参照しつつ、今日可能なアプローチとし

て、表現者自らが技術を深く理解すること (Tweiz 2009)、技術開発そのものを表現活動の一部に位置付けること (UCNV 2020)、一度きりのパフォーマンスではなく、継続的な活動として捉えること (HandredRabbits 2018) という3つの要素が必要であることを主張する。音楽のための技術インフラストラクチャとしてのプログラミング言語という視点を導入すると、ある言語の存在意義とは、これまでコンピューター音楽のためのプログラミング環境として提示されてきた言語のように、必ずしもコンピューターを用いて新しい表現を追求することだけではない。 **PL = 著者?**

それゆえ第4章では、PLfM の枠組みをこれまでのコンピューター音楽のためのプログラミング環境とは異なるものとして捉えた上でその歴史的系譜を整理する。 **我々が提示する枠組みの中での** PLfM の最低限の定義は SuperCollider (McCartney 2002) で提示された、“コンピューター上で実行可能な音楽のための最低限の抽象化機構を備えた人工言語” である。この定義は、ある PLfM の仕様を知っていればそれより下層の汎用プログラミング言語やコンピューター自体の動作に関する知識が不要になることを意味するが、“最低限の抽象化” は ~~言語によって~~ 西洋音楽の構造に影響された Score-Orchestra-Instrument のような階層や、モジュラーシンセサイザーのメタファーを用いる Unit Generator といった概念のように、常に現実世界のメタファーに頼る状態を生み、結果的にはメタメディアとしてのコンピューターに必要な自己拡張性に限界を与えることになる。こうした傾向は結果的に PLfM を抽象化のレイヤー毎に個別の言語を開発/利用する多言語パラダイム **の** ~~状況を生み、メタファーに依らない抽象的な音楽のための計算モデルの追求は未だ不十分であることを指摘する。~~

第5章では、前章での通時的な整理に対して、共時的比較として各言語の実装方法と言語毎の特徴を表す語彙を整理する。なぜなら、PLfM をメタファーに依らない設計とその評価を可能にするためには、その言語を使って作られる音楽表現や目的とは別の観点で比較 **が** 可能であるべきだからだ。この章では音楽に限らないドメイン固有言語のデザインパターン (Spinellis 2001)、汎用プログラミング言語における評価語彙の研究 (Coblentz 2018) を参照しつつ、PLfM を使用、開発するプロセスを Human-in-the-Loop モデルとして提示し、ランタイムのブラックボックスを大きくすれば動的変更に強くなる、小さくすれば表現自体の汎用性が高まる、動的変更と汎用性を両立しようとする設計が複雑化し実装のコストが嵩むといったトレードオフが存在することを提示する。

第6章では筆者が設計した PLfM、mimium の詳細を記述する。まず第2~5章で示してきた背景を総括し、1. ブラックボックスを可能な限り減らす、2. 新しい表現を生むことを目的とし、3. インフラストラクチャとしてのプログラミング言語の必要性という3つの設計方針を示す。 **mimium** は型推論を伴う静的型付け言語であり、汎用の関数型プログラミング言語の設計や実装に、音楽のための言語仕様を最小限追加する形で実装されている。1つは関数の実行のスケジューリングを行う @ 演算子、もう1つは内部状態を伴う信号処理を通常の関数と同じよう記述可能な、状態付き関数の記法である。mimium はこの2種類の記法を持つことで、これまではブラックボックスとして与えられていた基本的処理の単位をライブラリとして (実行性能を損なわないまま) 実装可能であることを、既存の言語との比較を交えて示す。 **音?**

第7章では、mimium の現状における実装の問題点を基点にして、開発を通じて浮かび上がってきた音楽のための道具作りの異なるあり方について議論する。mimium はラムダ計算という抽象計算モデルに最低限の時間操作を備えることでメタファーに頼らない **音声** の記述を可能にはしたが、PLfM 設計においてブラックボックスの少ない言語を作る方針を突き詰めると、ホスト言語の実装とその上でのライブラリ構築という2種類の作業へ分岐してくることがわかった。結果として、音楽のための言語であるにも関わらず、その言語自体は時間操作を考慮した汎用プログラミング言語の設計に近いものになり、音楽家が自ら実装する動機が失われるジレンマが発生する。つまり、音楽家が自らのための道具を作る環境を整備しようとしても、その環境自体を作る誰かの存在はやは

り無視できず、どこかで表現者を技術者の視点へと引き込む必要性が出てくる。それこそが、PLfM 開発よりもメタ的な行動理念、例えばプログラミング言語を作る技術者に音楽の視点を提供したり、表現者に自らの表現と技術インフラが無関係でないことを伝える枠組み、すなわち音楽土木工学を要請する。

第 8 章では全体の議論のまとめを行い、今後音楽土木工学という学問において PLfM 以外に可能な研究テーマの例を今後の指針として提示する。

Abstract

This dissertation aims to develop a theoretical framework for the design of Programming Language for Music (PLfM) as an infrastructure through the implementation of minimum (minimal-musical-medium),

Today, computers are so much around us that it is inevitable that we will pass through them in order to create and receive music, but programming, which allows us to manipulate computers with the highest degree of freedom as a means of creating music, is still not the primary means in the music creation environment. In Chapter 1, the direction of the dissertation, which is to conduct research to change such a contradictory situation through the practice of designing PLfM by oneself is presented.

As a background, Chapter 2 shows that intentional misuse of technology with amateurism, such as circuit bending and glitching [Cascone, 2009], which worked until the 2000s, no longer works in today's software-centric music technology culture. As a means of countering the increasing inaccessibility of a highly advanced technology, the author compares the different approaches to open up a hidden technology with a deep understanding, presenting as a performance [Tweiz, 2009], drawing the audience into the perspective of the technologist [UCNV, 2020], creating a different infrastructure from the ground [HandredRabbits, 2018], and places the design of music programming languages as musical practices in the broadest sense.

Chapter 3 examines how Research through Design (RtD), the way of academic contribution by creating artifacts through a methodology different from science, can be applied to the research area of programming languages. Referring a value of RtD in HCI [Gaver, 2012] and Claim-Evidence-Correspondence in evaluation of programming languages [Markstrum, 2010]. The evidence part can be evaluated objectively as in engineering research, and the overall research field is positioned as a research field where one can find a theory by differentiating individual research, not by generalizing. Also, referring to Magnusson's concept of Epistemic Tools [Magnusson, 2009], the author positions programming language design as an act of designing the infrastructure of music technology over a long period of time, as an example of a situation in which RtD expands its field.

In Chapter 4, the author re-positions the history of music programming languages based on the existing literature, as an entity sandwiched between the two major histories of music infrastructure and computer science outlined in the previous chapter. The author divides the history into three major parts: the 50s, the 70s, and the 90s and beyond, each corresponding to the topics of computer music research in the lab, the era of chiptunes, and the confluence of general-purpose programming languages with theory. In this history, I will show that today's music programming languages can be regarded as music software that dares to take advantage of the language format and its characteristics among the many interfaces that exist.

In Chapter 5, the author investigates the characteristics and the implementation methodologies of PLfM. To this end, the author proposes the process of using a music programming language as a Human-in-the-Loop model, referring to the study of evaluation vocabularies in general-purpose programming languages [Coblenz, 2018], and showing that there are trade-offs: a larger runtime black box is more resistant to dynamic changes, a smaller black box increases the generality of the expression itself, and trying to achieve both dynamic changes and generality increases the complexity of the design and the cost of implementation.

In Chapter 6, the details of *mimium*, PLfM designed by the author [Matsuura, Jo, 2021] is described. First, three design principles are proposed: 1) reduce the black box as much as possible, 2) do not aim to create new expressions, and 3) the need for a programming language as an infrastructure. *mimium* is a statically typed language with type inference, and it is implemented in the form of a general-purpose functional programming language design and implementation with a minimum of additional language specifications for music. One of its features is the @ operator for scheduling function execution, and the other is the notation of functions with states, which allows signal processing with internal states to be described in the same way as ordinary functions. In this paper, we will show that *mimium* can be implemented as a library (with no loss of performance), and compare it with existing languages.

In Chapter 7, based on the problems of the current implementation of *mimium*, the author analyzes the production of *mimium* from the perspective of the trade-off choices that exist in the design of PLfM, the position of *mimium* in the history of PLfM, the design of *mimium* as RtD, and the implementation of the programming language as one of the musical practices. It is suggested that the policy of creating a language with fewer black boxes in music programming language production leads to two types of work: implementation of the host language and library construction on top of it. As a result, a contradiction was revealed: the implementation of the host language was divided into less work related to music.

In the final chapter, the ontology and history of music programming languages presented in the first half of the chapter and the implementation of *mimium* and the analysis of the implementation process presented in the second half of the chapter are further generalized. The research field of “Civil Engineering of Music” is examined, including a comparison with recent music information processing research and trends in computer architecture. As a future research topic, the author reviews the theories that relate to the fundamentals of computers, such as operating systems and computer architecture, which have become a background in the applied field of music, and show that they can provide clues to solving the problems that have become apparent with the implementation of *mimium*.

謝辭

目次

| | |
|---|----|
| 概要 | 2 |
| 謝辞 | 7 |
| 用語集 | 11 |
| 0.1 略記一覧 | 12 |
| 0.2 音楽プログラミング言語一覧 | 12 |
| 0.3 汎用プログラミング言語一覧 | 14 |
| 図目次 | 14 |
| コード例目次 | 15 |
| 第 1 章 序論 | 17 |
| 1.1 序文の前に | 17 |
| 1.2 序文 | 19 |
| 1.3 研究領域のデザイン | 22 |
| 1.4 なぜプログラミング言語なのか | 25 |
| 1.5 本論文の貢献 | 26 |
| 1.6 構成 | 27 |
| 第 2 章 歴史を記述しなおすデザインリサーチ | 30 |
| 2.1 音楽のための道具づくりを研究するとはいったいなんなのか | 30 |
| 2.2 隣接する学術領域 | 32 |
| 2.3 デザインリサーチの変遷—デザインサイエンス、デザイン思考、RtD | 34 |
| 2.3.1 デザインの科学化と科学の社会構築 | 35 |
| 2.3.2 意地悪な問題、自己反映性、デザイン思考 | 36 |
| 2.3.3 Design into/through/for Art and Design | 40 |
| 問題解決主義 (Solutionism) から問題提起へ | 41 |
| 客観的観察 (Objectivism) から共同参加/あるいは介入 | 41 |
| 2.4 音楽とコンピューティングにおける RtD の受容 | 42 |
| 2.5 歴史を作り直すデザイン—メディア考古学とスペキュラティブ・デザイン、批判的奇譚づくり | 43 |
| 2.5.1 ロスナーの「批判的奇譚づくり」 | 45 |
| 2.5.2 メディア考古学 | 46 |
| 2.6 小括 | 49 |
| 第 3 章 Why:PLfM をなぜ研究するか | 51 |

| | | |
|--------|--|----|
| 3.1 | なぜインフラストラクチャとしての音楽プログラミング言語なのか | 51 |
| 3.2 | マグヌッソンによる認識論的道具としての DMI | 52 |
| 3.2.1 | インフラストラクチャの力とシグニファイア | 53 |
| 3.3 | カスタマイズとパーソナル・ダイナミック・メディア | 57 |
| 3.4 | 万能メディア機械 vs. 不可視のコンピューター | 59 |
| 3.5 | 音楽のデジタル化と生産と消費 | 61 |
| 3.6 | サーキットベンディング、失敗の美学、グリッチ | 65 |
| 3.7 | トーマス・トウェイツ「トースター・プロジェクト」 | 65 |
| 3.8 | UCNV「The Art of PNG Glitch」 | 66 |
| 3.9 | Hundred Rabbits | 67 |
| 3.10 | 「ブラックボックス」概念自体の分解と整理 | 68 |
| 3.10.1 | 理解不可能性：知識的ブラックボックス | 69 |
| 3.10.2 | 知覚不可能性：身体機能的ブラックボックス | 70 |
| 3.10.3 | 翻訳不可能性：言語的ブラックボックス | 70 |
| 3.10.4 | アクセス不可能性：社会的ブラックボックス | 71 |
| 3.11 | アクセス不可能性の突破へ | 71 |
| 3.12 | 小括 | 72 |
| 第 4 章 | What1:PLfM の歴史 | 74 |
| 4.1 | PLfM の歴史の主な時代区分 | 74 |
| 4.2 | Computer Music in Laboratory | 75 |
| 4.3 | Unit Generator とモジュラーシンセサイザーの関係性 | 78 |
| 4.4 | ボーンの IRCAM 4X プログラミングの分析 | 78 |
| 4.5 | 90 年代 | 79 |
| 4.6 | 2000 年代 | 79 |
| 4.6.1 | より詳細な時間制御 (ChucK、LC、Gwion) | 79 |
| 4.6.2 | 低レイヤの拡張 (Faust、Extempore、Kronos、Vult、Soul) | 79 |
| 4.6.3 | 高レイヤの拡張 (TidalCycles、Sonic Pi、IXI、Gibber、Foxdot、Takt、Alda) | 79 |
| 4.7 | 小括 | 79 |
| 第 5 章 | What2:PLfM における諸用語と概念の整理 | 81 |
| 5.1 | 導入 | 81 |
| 5.2 | 実装面から見た音楽プログラミング言語の実行環境 | 81 |
| 5.2.1 | 形式的定義と実装による定義 | 82 |
| 5.3 | ドメイン固有言語のデザインパターン | 83 |
| 5.3.1 | 言語拡張と自己反映性：CoffeeCollider を例に | 84 |
| 5.3.2 | TidalCycles - ハイブリッドなアプローチ | 86 |
| 5.3.3 | DSL の中でも、音楽特有の問題 - なぜライブラリとしての DSL ではダメなのか？ | 87 |
| 5.3.4 | ビジュアル言語のシンタックスと保存フォーマット | 88 |
| 5.4 | 音楽言語設計におけるトレードオフ General, Efficient and Expressive | 88 |
| 5.5 | 音楽プログラミング行為のモデル化と評価語彙の提示 | 90 |
| 5.5.1 | User-Side | 91 |
| 5.5.2 | Computer-Side | 91 |
| | コンピューターが必要とするコスト (Runtime Efficiency/Execution Cost) | 91 |

| | | |
|-------|--|-----|
| | 実行可能な空間の広さ (Portability) | 91 |
| 5.5.3 | Edit-Execute の繰り返しやすさ | 91 |
| | ユーザー側: Learnability | 91 |
| | コンピューター側: Dynamic Modification | 91 |
| | Development 自体のしやすさ | 92 |
| 5.5.4 | それぞれのトレードオフ | 92 |
| 5.6 | Multi-Language Paradigm の | 94 |
| 5.7 | 小括 | 94 |
| 第 6 章 | 最小限の PLfM:mimium の設計と実装 | 95 |
| 6.1 | mimium の設計 | 95 |
| 6.1.1 | 既存の言語との差分 | 95 |
| 6.2 | mimium の設計 | 95 |
| 6.2.1 | 基本的な文法 | 96 |
| 6.3 | <i>mimium</i> での信号処理 | 98 |
| 6.4 | アーキテクチャ | 98 |
| 6.5 | <i>mimium</i> 固有の言語仕様 | 100 |
| 6.5.1 | @ 演算子によるスケジューリング | 100 |
| 6.5.2 | 状態付き関数を利用した信号処理 | 101 |
| | UGen に仕様されるデータ構造の比較 | 101 |
| | <i>mimium</i> での信号処理の記述 | 104 |
| | 状態付き関数のコンパイル手順 | 105 |
| 6.5.3 | 言語仕様の整理、既存の言語との比較 | 106 |
| 第 7 章 | 議論 | 108 |
| 7.1 | 序言 | 108 |
| 7.2 | 現状の実装の問題点 | 108 |
| 7.2.1 | 離散イベントの処理と信号処理の記述のミスマッチ | 108 |
| 7.2.2 | 状態付き変数のパラメトリックな複製 | 109 |
| 7.2.3 | 複数サンプルレートが混在した信号処理 | 110 |
| 7.3 | より形式的な定義のための関連研究 | 111 |
| 7.4 | 実装の方針とトレードオフの選択 | 112 |
| 7.5 | 歴史的観点 | 114 |
| 7.5.1 | 音楽プログラミング言語とは何か? | 115 |
| 7.6 | デザイン実践を通じた研究 (RtD) として | 115 |
| 7.7 | テクノロジーを用いる音楽実践として | 116 |
| 7.7.1 | 音楽プログラミング言語を作るとはどういうことか | 117 |
| 7.8 | 音楽土木工学という学問領域 | 118 |
| 第 8 章 | 結論: 音楽土木工学に向けて | 119 |
| 8.1 | Chapter ごとの振り返り | 119 |
| 8.2 | 音楽土木工学 | 119 |
| 参考文献 | | 119 |

用語集

0.1 略記一覧

- AST: Abstract Syntax Tree / 抽象構文木、プログラミング言語のソースコードをコンピューター上で表現するための（主に上流で用いられる）形式の一つ
- CST: Creativity Support Tools / 創造性の補助ツールおよびその研究領域
- DMI: Digital Musical Instrument / コンピューターをシステムを中心とした楽器
- DSL: Domain Specific Language / ドメイン固有言語、画像や音楽などある特定の領域や目的に特化したプログラミング言語
- DSP: Digital Signal Processing / デジタル信号処理
- EUP: End User Programming / ソフトウェアのユーザー自らがプログラミングを行う活動、及びそれを支援するツールの研究領域
- FFT: Fast Fourier Transform / 高速フーリエ変換、計算機上で音声信号を時間領域から周波数領域の表現に変換する離散フーリエ変換（DFT）を高速に計算するためのアルゴリズムの総称
- HCI: Human-Computer Interaction / コンピューターを使うインターフェースの研究領域の総称
- MIDI: Musical Instrument Digital Interface / 電子楽器の相互運用のための共通規格
- NIME: New Interfaces for Musical Expression / 音楽のための新しいインタフェース研究、及びその国際会議
- OSC: Open Sound Control / Wright によって提唱された音楽のための自由度の高い相互通信プロトコル
- PCM: Pulse Code Modulation / 音圧波形を一定時間ごとに標本化 (Sampling)、量子化 (Quantization) することで数値の列として音声信号を表現する方式
- PLfM: Programming Language for Music / 音楽のためのプログラミング言語（本稿で独自に定義、1章を参照のこと）
- RtD: Research through Design / デザイン実践を通じた研究
- UGen: Unit Generator / 音楽のためのプログラマブル言語で広く用いられる、フィルターやオシレーターなど DSP の基礎的な要素の総称
- VM: Virtual Machine / ソフトウェア上で定義された仮想機械

0.2 音楽プログラミング言語一覧

- ChucK : Ge Wang が開発した、サンプル単位での正確な処理を可能にする strongly-timed というコンセプトの言語。ライブコーディング演奏が可能で、miniAudicle という統合開発環境を用いることが多い。
- CSound : MUSIC N シリーズの後継として開発された音楽プログラミング環境。実装言語を Fortran

から C に移したのでこの名前になっている。MUSIC シリーズ同様、Score、Orchestra、Instrument というレイヤー分けされた内部言語を持つ構造と、Blue など複数の統合開発環境が存在することが特徴。

- Extempore : Andrew Sorensen が開発した、LISP をベースにしたライブコーディングに特化した言語。前進として Scheme をベースにした Impromptu がある。Extempore 自体は Scheme と xtlang と呼ばれる独自の拡張言語の 2 つで構成される。
- Faust : Functional Audio Stream/フランスの研究所 GRAME で開発されているブロック・ダイアグラム代数 (Block Diagram Algebra) という抽象化形式を基にした DSP に特化した言語。様々な言語で UGen として利用できたり、Web ブラウザやマイクロコントローラ上でも動作する
- Gwion: Jérémie Astor が開発する、ChuckK を発展させた言語。第一級関数を使えるなど、プログラミング言語としての抽象度を高めている。
- Kronos : Vesa Norilo が開発する、Faust をより読みやすく高度な抽象化を可能にしたような言語。Faust 同様 Web など広い環境で動作するほか、非同期的な処理も実現できる。Veneer というビジュアルプログラミング環境も備えている。
- LC : Hiroki Nishino が開発した、ChuckK を発展させた mostly-strongly-timed というコンセプトの言語。同期/非同期の処理をユーザーが明確に制御できるほか、処理が間に合わなかった際のタイムアウト処理が記述できるといった特徴がある。
- Max : IRCAM で Miller Puckette により開発され、現在デヴィッド・ジカレリによる Cycling'74 が販売する、音や映像処理のためのビジュアルプログラミング環境。
- MUSIC N: Max Mathews らによって開発された初期の非リアルタイムなコンピューター音楽プログラミング環境で、I から V、および IV の派生バージョンである 4BF、360、10、11 といった複数のシリーズを指す総称で N というワードを用いる。
- mimium: 筆者が開発している、ラムダ計算を基盤にしたミニマルな音楽のためのプログラミング言語。
- Nyquist: Roger Dannenberg が 90 年代に開発した関数型ライクに記述できる音声処理言語。現在単体ではほぼ使われていないが、オープンソースの音声編集ソフトウェア Audacity の中でスクリプティング言語として利用できる。
- Puredata : Miller Puckette が Max の後にオープンソース・ソフトウェアとして開発し続けているビジュアルプログラミング環境。
- SuperCollider: McCartney によって開発された、SmallTalk 等に影響を受けた言語。バージョン 3 以降はサーバー/クライアントがネットワークを介して連携する構造を取ることで、他の言語からプログラマブルに制御できる音声合成エンジンとしても用いられている。
- Sonic Pi: Sam Aaron によって開発された、教育を一つの主眼においたライブコーディング環境。Ruby で記述できる SuperCollider クライアント。
- Soul : 音楽制作ソフトウェア用のフレームワーク JUCE を開発した Julian Storer を中心に、JUCE を保有していた ROLI 社が開発する信号処理記述に特化した言語。2021 年 5 月で開発が停止している。
- TidalCycles: Alex Mclean によって開発された、Haskell 上でリズムや音程、グラフィックなどあらゆるものをパターンとして抽象化するライブコーディング環境。元々は Tidal という名前。SuperCollider クライアントの一つ。
- Vult: Leonardo Ragna Luiz が開発する組み込みをターゲットとした信号処理記述のための言語。OCaml で記述されている。

0.3 汎用プログラミング言語一覧

- LISP : 1960 年代にマッカーシーらによって作られた、関数やデータを含めたあらゆる表現を S 式と呼ばれる括弧で括る記法で表現する言語、およびその系列の言語。代表的なものとして Common Lisp や Scheme、Racket、Clojure などがある。

図目次

| | | |
|-----|--|-----|
| 1.1 | テュン・チェによる Handmade Computer プロジェクトのひとつ、4 BIT FSM(Finite State Machine)(Choi 2015)。 | 24 |
| 1.2 | 本書で提示する音楽プログラミングの歴史の概要。 | 27 |
| 2.1 | 音楽とコンピューティングに関わる研究領域の見取り図。 | 32 |
| 2.2 | デザインリサーチの歴史の見取り図。 | 34 |
| 2.3 | Brown による、技術発展における約束と要求の推移過程を示した図。(Brown, Rip, and Lente 2003) をもとに筆者が作成。 | 38 |
| 2.4 | (Auger 2010) より、Alternative Present と Speculative Futures の概念を表した図。 | 44 |
| 2.5 | (大久保 2021) をもとに作成した、メディア考古学の立場の広がりを示した図。 | 46 |
| 3.1 | メタメディアとしてのコンピューターの歴史の見取り図。 | 51 |
| 3.2 | パリッカとハーツによるブラックボックスを意図されない用法で用いるという意味でのサーキットベンディングの概念を表した図。 | 70 |
| 4.1 | PLfM の歴史を、リアルタイム性、可変 DSP、Lab or PC の視点で分類した概略。 | 75 |
| 4.2 | Hartley の論文における PCM の概念を表した図。 | 76 |
| 5.1 | image-20210606155655532 | 88 |
| 5.2 | Anderson と Kuibila による音楽プログラミングのモデル。 | 90 |
| 5.3 | Screen Shot 2021-06-07 at 15.12.26 | 90 |
| 5.4 | Screen Shot 2021-06-07 at 16.25.25 | 91 |
| 6.1 | Architecture of <i>mimum</i> compiler and runtime. | 99 |
| 6.2 | Example of dataflow syntax in Max | 105 |
| 7.1 | 従来の Multi-Language パラダイムに基づく言語におけるプログラミングの学習経験の差を ”いびつな階段” として概念的に表した図。 | 112 |
| 7.2 | <i>mimum</i> のようなブラックボックスを減らし、ライブラリとしての実装の役割を広げた言語における学習経験の差を ”突然急になる坂” として概念的に表した図。 | 112 |
| 7.3 | 分岐するテクノロジーに対して、テクノロジーの意図的誤用という方法論と、あり得たかもしれない現在へのけもの道を作るという本研究の立ち位置の違いを概念化した図。 | 117 |

コード例目次

| | | |
|------|--|-----|
| 6.1 | Basic syntax of <i>mimium</i> | 97 |
| 6.2 | Example of <i>dsp</i> function which merges stereo inputs and returns the same signal to each output channels. | 98 |
| 6.3 | Example of Temporal Recursion | 100 |
| 6.4 | Equivalent code to Listing 6.3 in Extempore | 101 |
| 6.5 | The code of Phasor written with object in C++. | 102 |
| 6.6 | The pseudo-code of Phasor written with Closure in Javascript. | 103 |
| 6.7 | The code of Phasor written with Faust. | 103 |
| 6.8 | The code of Phasor written with Vult. | 103 |
| 6.9 | The code of sample counter in <i>mimium</i> | 104 |
| 6.10 | The code of phasor in <i>mimium</i> | 104 |
| 6.11 | Example of Pipeleine Operator in <i>mimium</i> | 105 |
| 6.12 | Example of Sequential Composition in Faust | 105 |
| 6.13 | Example of Feedback Delay in <i>mimium</i> before state tree transformation. | 106 |
| 6.14 | Pseudo-code of Feedback Delay in <i>mimium</i> after state tree transformation. | 106 |
| 7.1 | Example of encapsulating a temporal discrete value not realized in current implementation of <i>mimium</i> | 109 |
| 7.2 | Example of parametric replication of signal processor that cannot be realized in current implementation of <i>mimium</i> | 109 |
| 7.3 | 1 サンプルずつ増加するカウンター | 111 |

第 1 章

序論

1.1 序文の前に

2021 年 9 月。北九州市の植物園でこのイントロダクションを書いている。

植物園、というか自然公園みたいなこの場所にはフリーランスの仕事で来ている。公園を大きく使ったナイトウォークのようなイルミネーションのようなイベントで、そのための音楽や SE を流すためのシステムを作る仕事である。仕事の半分くらいはスピーカーをどこに配置するかとか、どの機材を使うとかそういう話なのだが、もう半分はその沢山あるスピーカーにどう音を割り振ったり、パソコンを起動したら自動的に音が鳴り始めるようにするためのプログラムを作る作業になる。

制御のためのソフトウェアは Cycling'74 社の Max という音声処理が得意なマルチメディアプログラミング環境で構築している。長い歴史を持ち、現在音楽のためのプログラミング環境としては（知る限り）最も多く使われており、個人的には 2015 年から使い始めたこの Max というソフトウェアを使うと、たとえば展示空間にスピーカーを 10 個とか（それも、いわゆるサラウンドとかとは全然違うレイアウトで）配置してそれぞれに違う音を同期して流したりすることができる。あるいは、映像を生成するコンピュータからネットワーク経由で特定の信号を受け取ったら特定の効果音を流すようにしたり。

なんだか言葉にすると大したことがない仕事のように聞こえる。実際、プログラムを書く作業自体は主観的には大したことがないのだ（ソフトウェア特有のニッチな問題やらバグは細々あるにしても）。こんな作業でお金をもらってしまっているんだと思う時もある。

しかし実際のところ似たような仕事ができる人はそこまで多いわけでもないらしい。たとえば、仕事を受けようとしたが予定が埋まっているときに別の似たような職能を持った人を紹介しようと思っても、パッと思いつく人はそう多くない。

これは、自分がこうした仕事に慣れすぎてしまったのだろうか。半分はそうなのだろうが、この仕事をしているとやはり考えざるを得ない疑問は、コンピューターはなんでもできる装置のはずなのに、たかだか沢山のスピーカーに同時にたくさん音楽を流すぐらいのことに、どうしていちいちこんな専用の（しかも有料の）ソフトウェアを使ってカスタムツールを使ってプログラムを作るような手間がかかる作業になってしまうのだろうか、ということなのだ。

これは別に、使うスピーカーが多いから必要なコンピューター処理能力が大きい、という話でもない。たとえば、映像インスタレーション作品でどうしてもフォーマットの 3ch のスピーカーを制御したい、となったときにも、大体 10ch の時と同じくらいの面倒臭さが発生する。2ch が 3ch になったりするだけで、普通の音楽制作ソフトウェアでは扱いづらくなる。なんで macOS でファイルを選択してスペースバーを押したら音声ファイルが再生されるように、物理的なセンサーに触ったら音声ファイルを再生する程度のことに労力が必要なのか。なんでエクセルファイルをちょっと編集するぐらいの手間で 10ch のスピーカーに音声ファイルを割り振れるようにならないのか。

この疑問こそが、おそらくはインフラストラクチャやフォーマットの持つ力というものの説明なのだろう。

つまり、人々は2chのステレオ音声のフォーマットがすでにあるから2chステレオで音楽を作る。作るためのツールも2chが一番主流なのでそれが最も作りやすいように良心的な配慮をしてしまう。それに、いかに音楽や芸術表現が新しさを欲するものだとしても、とりあえずはその2chのフォーマットの中で表現が可能な新しさであれば問題にはならない。そして、別に3ch以上の音声フォーマットも表現として“不可能なわけではない”。ただ、ちょっと手間がかかるのだ。

技術決定論者/あるいはマルクス主義者風の言い方をすればインフラストラクチャ: 下部構造が上部構造——、つまり音楽表現とかを規定するということになるのだろうか。おそらくはそうではない。インフラストラクチャは表現を完全に縛りはしないが誘導する。そしてその誘導方向は既存の上部構造によって作り出されるものであり、互いに循環し合いながら徐々に路面は踏み固められていく。

こうした実感を持つようになってきたのは自分で音楽のためのプログラミング言語を作り始めようと構想しはじめた2018年の後半ごろからだ。2018年の9月から11月にかけて、筆者はSchool for Poetic Computation(SFPC)という、ニューヨークにあるアーティスト・ラン・スクールへ留学した。SFPCはテクノロジーと表現を批評的に、かつ実践的に学ぶ私学校のような場所で、年に2回、20人ほどの学生(10代から50代まで幅広い)が集まり、OpenFrameworks^{*1}を用いたグラフィックプログラミングから技術批評の文献購読までを3ヶ月間、対話を交わしながら進めていくものだった。

筆者はそれまで、物理モデリング楽器を物理的な要素で再構築するインスタレーション作品の制作や、オーディオフィードバックを主要素とする電子音響楽器の開発とそれを用いた即興演奏などを中心に活動してきた。

これらの作品制作への大元のモチベーションも詰まるところ、先述したどれだけ工夫したプログラムを作ったところで全ては2chのステレオPCMサウンドというフォーマットに収束してしまうことへの窮屈さから来る、より異なる音楽表現の可能性の追求であった。つまりインスタレーション作品も、オーディオフィードバックを用いる電子楽器も、コンピューターにスピーカーを繋げるだけでは実現不可能な表現領域を探索することに意義を見いだしていた。

しかし同時に、音楽のフォーマット(あるいは、より広範な意味合いでの“形式”)から離れたところで、インスタレーションや、展示、あるいは即興演奏のイディオムという異なる形式へと従属する対象が変わっただけのようにも感じていた。

そうした疑問を抱えながら過ごしたSFPCでの授業は全くこれまでと違う考え方を自分に与えてくれた場所であった。

SFPCの入居しているウエストベス・アーティスト・コミュニティという建物は、元々1890年代から1960年代までベル研究所の建物だった場所で、有名などころではショックレーらによる切開で初めてのトランジスタが発明された場所でもある。そして同時にこの場所はニューヨークのテクノロジー・アートの歴史の中心地と言ってもよい歴史的な経緯を持つ場所でもある。

ベル研究所のエンジニア、ピラー・クルーヴァーは美術家のロバート・ラウシェンバーグとともに、60年代にExperiments in Arts And Technology(E.A.T)というアーティストとエンジニアの共同作業のための集団を組織し、《九つのタベ》や、1970年の大阪万博ペプシ館の演出を担当するなど、のちのテクノロジーを用いた芸術制作へと大きな影響を与えている。こうしたニューヨークにおける技術者と芸術家の共同作業は、ラウシェンバーグを始め、ウエストベスにのちに入居するダンサーのマス・カニングハムや、彼らとの共同作業も行った作曲家のジョン・ケージやデイヴィッド・チューダーといったアーティストの参加したブラックマウンテン・カレッジ、またケージの参加したフルクサスのような、反制度、反形式的な思想や運動を背景としていることが特徴であり、これはたとえば人類学者のジョージナ・ボーンが「Rationalizing Culture」(「文化を合理化する」)で、フランスの電子音楽研究所において70~80年代にピエール・ブーレーズのような作曲家が電子音楽のための技術を、現代音楽という積み重ねられてきた歴史の上で正当化する過程を描いた様子とは大

^{*1} openframeworks

きく異なる (Born 1995)。

SFPC のプログラムはこうした歴史的背景をもとに、ブラックマウンテン・カレッジのような既存の教育の枠を外し、生徒が互いに教え合う機会を多く設けた”Horizontal Pedagogy” (<http://taeyoonchoi.com/2012/05/notes-on-critical-pedagogy/>) と呼ぶ学びの姿勢を重視したものであり、かつ、大阪万博を一つのピークとして捉えられる、60 年代のテクノロジーアートにおける、表現に先行して技術を無批判に用いることや、大規模化することに資本主義に迎合していってしまう傾向^{*2}への内省を踏まえたものとなっている。技術を用いる際に暗黙的に発生する政治性に自覚的になり、また技術を与えられたブラックボックスとせず、すでにライブラリやツールが存在しているものだったとしても一度自分の手で作り直すことによって体でその内容を理解することで、異なる技術のエコシステムの可能性を想像することができるようになるというわけである。

それゆえ、SFPC で教える講師達の活動形態も、必ずしも作品を作って美術館に展示することが中心なわけではない。OpenFrameworks でほぼ毎日短い CG アニメーションを作り投稿し続ける Zachaly Lieberman や、CPU の動作原理を餃子作りに見立てた”CPU Dumpling” ワークショップを行う Taeyoon Choi、ポストコロニアルスタディーズをベースに、白人中心主義的なコンピューター文化批判のインスタレーションや Zine を制作する American Artist、100 年以上ただ時を刻むだけのカウンターのようなオーバー・エンジニアリングなツールを自宅の工房で作って自分たちで売り続ける CW&T..... のように、多様な活動を見せる彼/彼女らの活動を通して（それでどうやって生計を立てられるかはともかくとしても）テクノロジーと社会の関係性への向き合い方は必ずしもアートという閉じられた空間の中だけで行うものでなくてもよいのだと実感させられた。

こうした経験が音楽のためのプログラミング言語という実に微妙な領域の制作へ筆者が本格的に入っていくきっかけとなった。

詰まるところ、“どれだけ工夫したプログラムを作ったところで全ては 2ch のステレオ PCM サウンドというフォーマットに収束してしまうことへの窮屈さ” に対して真に向き合うには、何が社会的にそのフォーマットを構築しているのかについて突き詰めて考える必要があったのだ。違う表現の形式へスライドしても結局その形式の束縛を受けるだけになってしまう。ならばやるべきは制作を無意識的に支配している形式や制度そのものの脱構築に他ならない。つまり筆者にとって音楽プログラミング言語の制作とは音楽を生み出す下部構造＝インフラストラクチャになりうる道具自体を自らの手で作ることを通じて、異なる音楽の形式や制度そのものをメタ的に制作する芸術実践でもある。

1.2 序文

本研究は音楽土木工学という、未だ存在しない学問領域を、音楽のためのプログラミング言語 mimium の設計と開発を通じて描き出す試みだ。この論文の中で筆者が提起しようとしている音楽土木工学とは、その名前の通り土木工学——つまり街における道路のようなインフラストラクチャ設計や、未来の街の姿自体を検討する都市計画、そのための構造力学や材料工学といった分野を音楽という領域に当てはめて考えてみるというアナロジーに基づくものだ。

では、音楽における土と木、インフラストラクチャとはいったいなにか？

例えばわかりやすいのは音楽を流通させるストリーミングサービスだ。まだ音楽の流通が CD や、（違法なものを含め）ファイル単位でのデータ配信が主流だった 2005 年に、クセックとレオナルドはコンピューターやインターネット技術の発展の先に音楽の姿は「水のような音楽」になると予測した (Kusek2005)。月単位で水道代を払っていただければ蛇口をひねれば水が出るように、特定のサービスと契約していただければクリックすれば好きな音楽がいつでも聴き放題である今日の音楽ストリーム（川）の姿は彼らが予想した未来そのものである。

^{*2} たとえば、(馬 2014, 58p) の坂根徹夫の引用では、70 年代はじめのオイルショックや環境問題を単に発した科学技術自体への批判との共鳴が指摘されているし、大阪万博ペシ館でのスポンサーによる会期中のプログラム中断と、E.A.T. のその後の活動の衰退をあげることができるだろう。

私たちの音楽を聴く生活のあり方は、それを運ぶサービスという名の川が整備されることによって少なからず変化した。これだけでも音楽に関わる工学的研究において、そのコンテンツにだけ焦点を当てるのではなく、ある技術そのものに音楽に関わる要素がなくとも、その技術が用いられた音楽のインフラストラクチャに目を向けることには音楽の未来を考えるにあたって価値あることだと言えるだろう。

一方で私は、音楽土木工学という言葉を実に Spotify や Apple Music のような音楽流通サービスを支える技術について考えようというだけの意味で提起しようとしているのではもちろんない。私が意図する音楽における土と木とはむしろ、音楽のために作られたわけではないが、それでも音楽のあり方に大きく影響を与えているテクノロジーやインフラストラクチャに重点をおいている。コンピューターの構造は音楽制作に使うために設計されたのではないし、インターネットは音楽配信の方法を変えるためだけに生み出されたのではないが、確実に音楽の制作と聴取の変容に影響を与えている。

2021 年現在、音楽を聴いたり、演奏したり、作ったりする上で、コンピューターが一切関与しない、という状況を考えるのは難しくなっている。作曲には Protools や Cubase に代表される DAW (Digital Audio Workstation) ソフトウェアを使用し、配信には Apple Music や Spotify のようなストリーミングサービスを通じて、デジタルデータという形で音楽は配布される。最終的に、コンピューターやスマートフォン上のソフトウェアでそのデータをデコードし、DAC(Digital-Analog Converter) によって電気信号へと変換され、その信号はスピーカーへと送られようやく空気の振動になり、私たちの耳へ届く。スピーカーの中にさえデジタル信号処理 (DSP: Digital Signal Processing) 用のチップが入っていて計算によって音質の調整をしていることも珍しくはない。2020 年以後のコロナウィルスの影響も含めれば、クラシック音楽のコンサートさえもその場で空気の振動を体感することよりも録画録音されたものをコンピューターを通じて摂取することの方が多くなってしまったかもしれない。とかく音楽文化を見れば、マーク・ワイザーの提唱したユビキタス (Ubiquitous: 遍在する)・コンピューティング (Wiser1999) の概念は字義通りには達成されたようにも見える。

それでも、そうやってコンピューターという、理論的にはなんでも可能なはずの装置を通じて生み出され届けられる音楽の聴取の形式自体は、電子計算機が作られてから 70 年が経とうというのに、2ch ステレオで 5~10 分程度の曲という、コンピューター以前の録音音楽によって形成されたフォーマットから大きく変化していない。

音楽表現を支えるための道具やインフラストラクチャは、常に進歩的に新しい表現を可能にしてきたように歴史の中で物語られるものの、実際のところその研究や道具は何を理想としていて、誰がどう方向付け形作るものなのだろうか。それは音楽家自身によって変化を促せるような対象なのだろうか？

音楽土木工学とは、こうした問題意識をもとに、特に既存の音楽に関わる工学的研究との違いにおいて、対象とする技術要素の違いにおいて特徴付けられる学問である。音楽土木工学はテクノロジーを音楽に応用するのではなく、音楽の形式の規定に大きく関わるが、必ずしも音楽のために作られたわけではない汎用的なテクノロジーを、改めて音楽という視点から作り直すことを主眼に置く。

音楽のための工学的研究は例えば録音技術の研究や音楽/音響心理学のような、必ずしもコンピューターを必要としない研究領域も当然大小存在しているものの、特に現代においては先述したユビキタス・コンピューティング的状况を考えれば、音楽のための道具やインフラストラクチャを作る研究としてもコンピューターを音楽に用いる研究は中心的研究領域となるだろう。

この中でコンピューターを用いるテクノロジー、つまり情報技術を音楽に応用する、という、音楽土木工学と対照的な立場として挙げられる代表的研究領域が、いわゆる音楽情報抽出 (Music Information Retrieval: MIR) と呼ばれる分野だ。MIR という研究分野ではコンピューターという 0/1 で音楽に関わるあらゆる種類のデータを表現することが試みられる。例えば、2ch の音声信号から録音時の個別の楽器の音声信号を抽出するような音源分離技術、Text-to-Speech、音楽の曲調の変更、といった、音楽/音声信号に含まれているリズム、ピッチ、メロディ、さらには音色や曲調のような抽象的なパラメーターを統計的手法を用いて抽出したり、逆にそ

うしたパラメーターをもとに音声情報を変換したり合成するアプローチの研究がなされている。これは言い換えれば人間の耳のような生理的器官の機能や、脳の認知的機能をモデル化することが念頭に置かれた手法である。それゆえ、近年の機械学習技術の発展とも相まって、音声や音楽に関わるコンピューティング技術の中でもその存在感をますます大きくしている。

一方で音楽土木工学が対象とする技術的要素は例えば、コンピューターアーキテクチャやオペレーティング・システム、プログラミング言語（の設計とその処理系開発）、プロトコルやデータフォーマットといったものだ。こうした技術はとくに日本では低レイヤーと呼ばれることもある、コンピューターの技術における抽象化のピラミッドにおいて比較的下層に位置するものだ。こうした機関的、まさにテクノロジーにおける土と木に相当する要素を、音楽のようなある特定の領域のために再考するというアプローチは、近年ではさほど特殊な考え方ではなくなっている。

その理由のひとつは、コンピューターの中で用いられる技術を段階的に抽象化し、各レイヤーにおける技術的最適化を試み、下層の技術はその上で何を表現するかを考えることはしなくても良いという、これまでの考え方が通用しなくなってきたことだ。

例えば画像データの処理や機械学習のような、並行して同時に計算可能なタスクはコンピューターアーキテクチャ、つまり CPU や記憶装置など、コンピューターを構成するハードウェア群自体の設計を並行処理に適したユニット（例えば、GPU）を利用することで圧倒的に処理速度を向上させることができる。この性能向上の程度は今日もはや、上層のソフトウェアのアルゴリズムだけで改善できる程度を大幅に上回っている。

コンピューターアーキテクチャ設計の分野においては、これまでムーアの法則と呼ばれる、IC チップにおけるトランジスタの集積率が年々指数関数的に増加することが見込まれてきた傾向が今後頭打ちになるだろうといった背景^{*3}をもとに、近年ではドメイン固有アーキテクチャ（Domain Specific Architecture: DSA）と呼ばれる、特定の処理のために専用のプロセッサを計算機に実装することの必要性が提起されている（Hennesy2017）。DSA の代表的な例としては Google による機械学習に特化した演算ユニットである TPU（Tensor Processing Unit）が挙げられる（Google2018）。

こうした処理性能の話に限らずとも、例えば Arduino に代表される、趣味やアーティストの作品制作のために作られた、マイクロコントローラ（AVR や PIC に代表される、OS などを用いない小規模のコンピューター）を用いた電子工作を簡便化した環境は、AVR マイコンという技術的要素と、必要最低限な標準的な入出力の規格化、ハードウェアのオープンソース化、初心者にも親しみやすい開発環境ソフトウェア（IDE）という、各種レイヤーにすでに存在していた技術をパッケージにすることで多くの非専門家に電子工作の門戸を開いてきた。同様に、プログラマブルではあるが、ある程度特定の目的に特化したミニコンピューターという系列で、音楽や楽器制作への利用を念頭においたものとして Bela のようなハードウェアも挙げられる（Bela2018）。Bela は、BeagleBone Black という、Linux（オープンソースのオペレーティングシステム）を搭載する既存のオープンソースミニコンピューターのための、音声入出力インターフェースを備えた拡張ボードだ。その特徴は単にハードウェアだけがプロダクトに含まれているのではなく、USB ケーブルで自分のラップトップなどと接続すると、Arduino と似たような形で、そのラップトップの Web ブラウザ上から簡単にプログラムを書き換えられるような開発環境がセットになっている点である。さらに、音声信号処理において特有の問題である入力データが出力に反映されるまでの遅延を極力取り除くために、オペレーティングシステムの機関部分（カーネル）に Xenomai と呼ばれる拡張が加えられていることも特徴のひとつだ。このハードウェアとカスタム OS の合わせ技を利用することで Bela は、より高速な CPU を積んだラップトップやデスクトップコンピューターでも実現できないような低遅延のシステムを構築することができる。

このように、これまで利用目的を問わずに万能に対応できるよう構築されていたハードウェアと OS のような基幹的な技術要素を（例えば音楽といった）特定の目的のために、部分的に作り替えてみたり、開発環境ソ

^{*3} 実際、デナード則という、集積率を上げて処理性能が向上しても電力消費は一定のままであるという法則は 2006 年ごろから崩れているという見解がなされている（Hennesy2017）。

ソフトウェアのような高レイヤー技術も含めた複合的な組み合わせを検討することによって特定の課題が解決される可能性は大いに増えているし、電子回路やハードウェア設計、実装自体の個人レベルでの取り組みのハードルが下がってきたこともこれらの傾向を後押ししている。

1.3 研究領域のデザイン

このように私が提起しようとしている音楽土木工学は、音楽に関わる工学の異なる関わり方の追求ではあるが、その理論的な基盤は工学や科学というよりもデザイン学にある。

デザインというとグラフィックデザインやインテリアデザインのような視覚的表象の操作や、あるいはインターフェースデザインや Web/アプリケーションのデザインのように、ユーザーの認知的特性に基づいた道具の操作方法の設計一般のような分野を想起してしまうが、デザイン学問は歴史の中でその意味を単に良い製品の創出といった側面だけではなく、社会システムのような抽象的なものを含めた、人工物の創出一般に関する方法論としての性質を持つようになってきた。デザイン学は 1960 年代以降、認知科学、記号論、科学技術社会論 (Science, Technology and Society : STS) のような様々な学術領域の知見を取り込みながら発展してきた。その中で例えば、必ずしも実用的に役に立つものを作るだけではなく、直接的には役に立たずとも、その人工物を通して未来の社会像についての議論を引き起こすことを目的としたクリティカル・デザイン/スペキュラティブ・デザインや、作られた人工物そのものを最終的な知の貢献とするよりも、それを作る過程や使われる過程で得られた知見を共有することに主眼をおいた Research through Design (RtD) のように、客観的かつ再現可能な証拠のみを材料とするのではなく、研究者やユーザー（あるいは共同参加者）の主観的表現を積極的に用いる、実証主義的な科学と異なる研究の方法論を確立しつつある。

特に Human-Computer Interaction (HCI) のような、電子計算機と人間の関わり方を探求する学問分野は、音楽のためのコンピューターを用いたシステムとインターフェースを研究する New Interfaces for Musical Expression (NIME) のような新しい研究分野を生み出してきたが、その理論的基盤は科学 (Science) のように私たちの世界の仕組みを解明、理解するための営みとは異なることもあり、RtD のようなデザインリサーチの系譜に位置付ける動きが活発になっている。本研究はそのような流れの上で、音楽のためのプログラミング言語の設計と実装の実験的経験と、音楽に関わるテクノロジーやメディアとしてのコンピューターの歴史にまつわる言説の批判的検討を通じて、音楽のテクノロジーの関わり方の異なる現在 (Alternative Presents) やありえるかもしれない未来 (Speculative Futures) (Auger 2010) を描き出すことを試みるものとなる。

本研究のアプローチは、近年のデザインリサーチの方法論の中でも 2 つの特徴的な点を持っている。まず 1 つは、アーティストや音楽家のような表現を行う主体自身が、通常所与のものとされているテクノロジー（例えば本研究では、プログラミング言語）の仕組みを自分で作ってみることで理解する、技術的なブラックボックスを開くアプローチである。この、ブラックボックスを開くという言い回しは特に科学技術社会論において 1980 年代以降とりわけ用いられてきたもので、高度に積み重ねられ分析することが不可能になっていく科学の知の体系の成立過程を、研究室におけるフィールドワーク（いわゆるラボラトリー研究）や参与観察を通じて明らかにするという意味合いで用いられてきているが、本研究におけるブラックボックスを開くとはむしろ、元々非専門家であるアーティストや音楽家が DIY 的に、通常とは異なる目的で技術の根幹の部分に分け入っていく、観察者というよりも、自らが技術者になってしまうことでその中身を理解するという意味合いで用いている。このアプローチは技術者やデザイナーの視点から見ると、デザインにおける想定されたユーザーのための道具作り、ユーザー中心的设计 (UCD: User-Centered Design) の批判的継承として位置付けられる。ユーザーフレンドリーな道具というのは、裏を返せばユーザーを受動的消費者にするデザインでもある。通常、既に使える道具として与えられているものを改めて作るという行為は、科学や工学においては車輪の再発明や、DRY (Don't Repeat Yourself) の原則のような言葉で避けられているが、学習のために自らの身体を用いて道具の仕組みを理解することは必ずしも無駄なこととは言えない。それだけでなく既に固定化されてしまった

テクノロジーの異なる姿を想像するための手がかりとなることもある。例えば城一裕らによる「車輪の再発明プロジェクト」では、レコードや写植、スピーカーのような単純な仕組みで作動するメディア技術を、デジタルファブリケーションの普及などにより、個人で利用できる技術環境が変容した中で改めて別の形で作り直すことで、紙の上にカッティングプロッターで溝を刻んだレコード、写植の文字盤を多光源プロジェクションのスクリーンとして利用する技法、耳を磁石で挟み込みコイルを近づけることで骨伝導を介して音を聴く《超超短距離電信装置》のように異なるメディア環境の可能性を想像させる道具を生み出している(城 2016)。

もうひとつの特徴的な点は、技術やメディアの歴史観の批判的再構築である。ミシェル・フーコーの「言葉と物」や「知の考古学」以降の、歴史が絶対的事実の積み重ねではなく、特定の誰かが記述することによって社会的に構築されていくことを前提とした価値観の中では、異なる技術環境を想像することはこれまでと異なる視点で技術とメディアの歴史を見つめ直すことに他ならない。こうした視点の代表的アプローチとして、フータモやパリッカのようなメディア研究者を中心に議論されているメディア考古学(Huutamo2013)と呼ばれる、淘汰されたメディア装置を足がかりに異なる技術の歴史の可能性を示唆する研究群が挙げられ、「車輪の再発明プロジェクト」はメディア考古学とスペキュラティブ・デザインのアプローチとを接続したものである。

技術を形づくる主体がその内容を深く理解し、また同時にその使われ方への批判的視座を持つということは、技術を作る個人の政治性に自覚的になることでもある。テクノロジーが社会の変化に大きな影響を与えることが当たり前になりつつある現代において、安定したパラダイムの中で政治的に中立的に見えるパズルを解くだけのような技術研究^{*4}はもはや成立しない。だからこそ、誰が誰のために技術を形作るのかという視点でのオルタナティブな工学のかたちを追求する必要がある。

ここに、音楽土木工学という学問の命名理由のもう一つの理由、土木工学という分野の英語における表現、Civil Engineering という言葉の選択がある。この Civil Engineering という語は、現在では概ね日本語における土木工学と重なる領域を指してはいるものの、元々は Engineer が軍事的な技術に関わる者という意味合いを持っていたのと対比して用いられるようになったものだ^{*5}。

つまり、音楽土木工学とは音楽とテクノロジーという都市における土と木に相当する要素を考える学問であると同時に、音楽のための市民工学という意味合いを持つ。この両面のアナロジーは、筆者が学んだ School for Poetic Computation を立ち上げた人物の1人、アーティスト/アクティビストのテュン・チェの「Open Circuit, Open City」というタイトルの文章からインスパイアされたものだ(Choi 2016)。

チェはSFPCでの授業を始め様々な場所で、自らロジック IC を組み合わせてシンプルなコンピューターをDIY的に作る「Handmade Computer」のプロジェクト(図 1.1)や、餃子を作る工程を複数人で分担し、その作業分担の方法を変えることを通じてコンピューターにおける逐次処理・並列処理の違いを体感的に学ぶような「CPU Dumpling」と呼ばれるワークショップを行ってきた。その中でテュンは次のように述べている(以下、引用は全て筆者による訳)。

コンピューターを手作りすることで、はんだ付けや配線を繰り返す中でたくさんのことを考える時間を得た。コンピューターがどのように進化してきたかを知ること、歴史に対する理解がより深まった。同時に、何が最近になってようやく可能になったのかを学ぶことで、オルタナティブな過去、そしてそれが照らし出すオルタナティブな現在とオルタナティブな未来という、テクノロジーとの異なる関係性を想像するインスピレーションを得られた。

都市はよく、コンピューターと似て、ブラックボックスの中に封じ込められ、暗号化され、抽象化される。都市の住民は、空間がどう作られ使われるかを限られた範囲でしか制御できないし、私的に保有される

^{*4} パラダイムという言葉普及させたトーマス・クーンの「科学革命の構造」では、ある安定したパラダイム(特定の科学的領域をささえる認識の共有された構造(Fukushima2021))の中での科学研究(通常科学)はパズルを解くような物と表現される(Kuhn1971)。

^{*5} Civil Engineer という語は一般的に、1750 年ごろにイギリスの工学者ジョン・スミートンが自らの専門を、相容れない軍事的な研究と区別するために名乗り始めたことが起源とされている(Florman 1989)。

公共空間もその意味でまたブラックボックスである。基礎的な部品からわたしたち自身のコンピュータをゼロから作ることで、〔自己を〕回復するためのオルタナティブな都市空間の創造を想像できはしないだろうか？

都市は、コンピュータと似て、美学的熟考のための中立的対象ではない。そうではなく、競合し合う政治性と、危うくて不安定〔*precarious*〕な生活とが置かれた場なのだ。

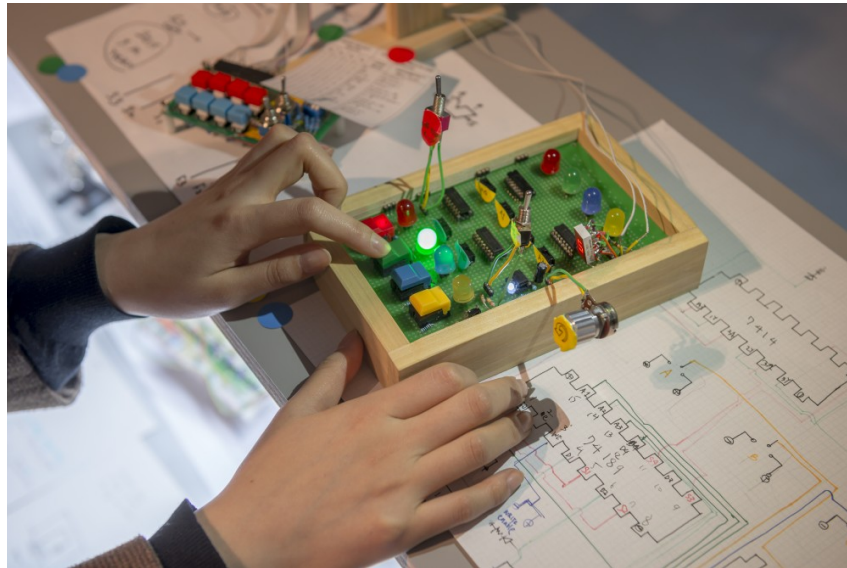


図 1.1: テュン・チェによる Handmade Computer プロジェクトのひとつ、4 BIT FSM(Finite State Machine)(Choi 2015)。

ともするとこれらのプロジェクトはいわゆるシチズンサイエンスにおける、科学実験の体験教室と同じような、馴染みのない科学的作業を身近にすることを目指したものと大差ないように思えるかもしれない。かといって、ここで作られた原始的な電子計算機はその機能性や仕組み自体がこれまでの電子計算機に対するオルタナティブなものというわけでもないし、逆に、^{批判的}芸術作品として計算機の見目に美的価値を付加しようとしているのでもない。

CRITICAL MAKING

テュンが行っているのはむしろ、自らが作ったり (DIY)、誰かと一緒に作ったりする (DIWO: Do it with Others) 過程で、そもそもテクノロジーとは誰が誰のために形作るものなのかという問いを投げかける行為だ。こうしたプロジェクトは、スペキュラティブ・デザインのような、テクノロジーのあるべき未来の姿についての議論を巻き起こすことを目的とした人工物のデザインともまた少し態度が異なる。それは、作家自らが科学的技術を実践するか否かという点である。スペキュラティブ・デザインを提唱したダンとレイビーが、デザインと科学の付き合い方の分類に関して、“Design about Science (科学研究から生じる問題や影響について、デザインを通じて考察する)” というアプローチに着目していたのに対して、テュンのようなアプローチに当てはまるのは “Design through Science(デザイナーが多少なりとも科学を実践する)” という態度である*⁶。デザイナー自身が科学技術の実践者になるという事はすなわち、技術者の身体的感覚に結び付けられており、言語かできる以前の状態に留まっている知識をデザイナー自らが身につけるということになる。この時、デザイナーや人類学者が科学技術研究者を観察したり、デザイナーと科学者がコラボレーションするだけでは取り出すことが敵わなかった、新しい伝達可能な知 (Communicable Knowledge) を見つける可能性が立ち上がった。

*⁶ (ダン and レイビー 2015, p208)。この他に “Design for Science (科学研究を伝えたり、わかりやすく説明したりするためにデザインを用いる)”, “Design with Science (デザイナーと科学の真のコラボレーション)” という 2 つの関わり方が挙げられている。

てくる*7。

本研究のアプローチは、プログラミング言語やコンピューターそのものの設計や実装のような、一般的には音楽や芸術表現と無関係なものと思われている事柄に自ら取り組み、その技術を理解することによって初めて得られる視点をデザイナー/アーティストの批評的視点と接続することによって、これまで共有されてこなかった技術者に埋め込まれた知を一度取り出し、さらに異なる方向性を見出す、いわば **Research through Design through Science(Technology)** とでも呼ぶべき態度である。



1.4 なぜプログラミング言語なのか

では、音楽におけるコンピューターの利用のされ方を再デザインするための試みとして、なぜ音楽のためのプログラミング言語を作ることが役に立つのだろうか。それは音楽のためのプログラミング言語の構造を知ることが我々の音楽に対する認識—平たく（かつ大仰に）言えば、私たちにとって今日の音楽とは一体なんなのかという疑問を理解することにつながるからだ。

音楽のためのプログラミング言語が一般的に多くのアーティストにとっては、使うことならともかく、言語自体の設計や実装に至っては技術的に高度すぎて理解することが困難であることも、ひとつ重要な点ではある。しかし、音楽プログラミング言語に限らずとも大抵の音楽制作ソフトウェアや、その中で使われるプラグイン、あるいは電子楽器などが技術的にどう作られているかはやはりほとんどの場合においてブラックボックスである。こうしたツール群と音楽のためのプログラミング言語とで何が異なるのかといえば、当然、音楽プログラミング言語が言語であることである。

プログラミング言語はコンピューターという機械に対する命令を記述するものなのだから、あたかも自然言語のように扱うのは不自然に思えるかもしれない。しかし、コンピューターに対する命令は最終的に 0/1 の羅列である機械語として記述されるのだから、もし人間が 0/1 の羅列だけでコンピューターに対する命令をスラスラと記述できるのであればはじめてからプログラミング言語などは必要ない。そういう意味では、人間が理解可能な形で記述されたテキストデータを機械語へと翻訳することで動作するプログラミング言語というシステムは、根本的には徹頭徹尾人間のための道具であり言語である。そして、プログラミング言語とは人間が思考し記述したモデルを計算機上で実際に動作させることができる道具だが、これは裏を返すと、計算機が理論上どんなに万能の装置であったとしても、人間が記述できない概念をコンピュータープログラムとして動作させることは敵わないということになる。

この万能性と言語による制約という矛盾めいた状況は、トール・マグヌソンによってコンピューターを用いた楽器の特徴を認識論的道具と呼んだことに対応する (Magnusson 2009)。

本研究はマグヌソンの議論を引き継ぎ、より焦点を音楽のためのプログラミング言語における、音楽のための最小限の抽象化とは一体なんなのかという問いを *mimum* という言語の実装を通じて検討する。今日既
 常の音楽のためのプログラミングシステムや、それらを用いることで実現される様々な音声合成の手法は依然
 多くが現実世界のメタファー（楽譜や楽器、モジュラーシンセサイザーなど）に依存している。それ自体はマグヌソンの指摘した原理的限界とも言えるものだが、問題はその具体的メタファーの部分はプログラムの意味論としてははじめてから言語に組み込みのものとされており、現実的には C++ のような汎用プログラミング言語で記述されることだ。これは実用的な問題としては、各プログラミング言語ごとにそれぞれの言語が考える（とはいえ現実的には粒度の粗い）“最小限”が多数存在していることで、言語ごとの相互利用がままならないという問題につながる。しかしより根本的な問題として挙げられるのが、具体的メタファーに基づいた言語仕

*7 Research through Design という言葉が作られるきっかけとなった、クリストファー・フレイリングの “Research in Art and Design” では、アートやデザインの領域における研究行為を into（歴史研究など、一般に「研究」と認識されているもの）、through（アクションリサーチや材料研究）、for（アーティストやデザイナーが自らの制作のために行う個人的な探求）という 3 つの接続詞で分類し、このうち “for” は伝達可能な知を生み出すことを一義的な目的としておらず、その知は制作者やあるいは作られた人工物に埋め込まれる (Embodied) とされている (Frayling 1993)。詳しくは第 2 章を参照。

様が存在していることで、コンピューターをメディア装置として使う際の随一の特徴である、自らの機能をプログラミングという手段を用いることで変化、拡張させられるというメタメディアとしての機能に限界が生じてしまうことだ。

メタメディアの思想の源流であり、今日の GUI を中心としたパーソナルコンピューティングの姿に大きく影響を与えたアラン・ケイとアデル・ゴールドバーグによる Dynabook の研究では、コンピューターを使う人が自身でプログラムを書くことによって、自らの道具（ソフトウェア）の機能を自らに合わせて拡張させることが思想の根幹に置かれていた。だが、今日プログラミングという行為の裾野は随分広がったとはいえ、音楽を作るためのソフトウェアの機能をプログラミングを用いて自ら拡張できるような環境は、Ableton 社の Live における、音楽プログラミング環境 Max を内部拡張として用いることができる Max for Live、独自のスクリプト言語を使用できる Reaper のような数少ないソフトウェアを除けば、未だメインストリームとは程遠いと言える。この自己拡張性の少なさは時に、想像できないことはプログラムとして記述のしようがないという限界にとどまらず、実現したいと思っている機能があるにもかかわらず（そして、プログラミングの知識があったとしても）容易に拡張することが難しいという状況を生んでいる。

つまりプログラミング言語についての理解を深めるということは特に音楽という分野においては、我々が音楽の認識を形作る仕組みを理解することでもあり、同時に社会の中における音楽を作るための道具が人間にもたらす自由と制約の仕組みを理解するという2種類の疑問にアクセスすることになる。（もうちょっと補足してもいいかも）

1.5 本論文の貢献

（順番を整理したほうがいいかもしれない）

本論文を通じて学術的に貢献する内容は、大きく分けて3つである。

1つは、“音楽のためのプログラミング言語とは何か？”という、PLfM の存在論の提示だ。PLfM は歴史的にコンピューターを用いて音楽を生成するための技術としてスタートしつつも汎用プログラミング言語の理論を取り込んで発展してきたことにより、~~一重~~に音楽のためのプログラミング言語/環境といってもその応用範囲や想定される使用方法、さらに内部の実装方法まで多岐に及び、単純な比較をすることが難しい。また、評価のための語彙も“表現力が高い”“効率的”“汎用的”などの言葉が共通認識の無いまま慣例的に使われており、実際に何を意味するかもはっきりしないことがある。本稿ではまず音楽プログラミング言語の歴史の変遷を改めて整理した上で、“PLfM にはどんな種類があるのか”、“PLfM の特性はどう記述できるか”といった概念を整理して提示する。

2つ目は、“音楽のためのプログラミング言語を~~音楽~~設計するという行為とは何か？”という問いである。録音技術（音響再生産技術）に端を欲した音楽のフォーマットは現在空間音響技術のためのフォーマットの普及などによって、原音再現という従来の明確な一つの目標を失いつつあるだけでなく、新しさを謳いながら一方でインフラストラクチャの性質による音楽の形式を画一化する傾向を持っている。その環境でコンピュータ上の表現の自由度を最大限担保するためにはプログラミング言語そのものを音楽のためのフォーマットすることが必要になってくる。この背景から、音楽プログラミング言語を設計することは2020年代以降における、テクノロジーを主体的に使う音楽実践の1つのあり方に位置付けることを試みる。さらにこれはデザイン学におけるデザイン実践を通した研究の領域の広がる中で、社会構造自体を長い時間をかけて変化させていくことを試みる動きとも呼応する。これまでデザインリサーチ的手法は Humam-Computer Interaction の研究法の1つとして狭い意味で音楽プログラミング言語にも適用されてきたが、本研究ではプログラミング言語設計という行為をさらに一般化し、音楽土木工学という、誰もがアクセス可能な音楽のためのテクノロジーの変革のための政治的行動という広い領域へ接続する。

3つ目は、筆者が設計/実装した自己拡張性の高い音楽プログラミング言語“mimum”についての設計思想

とその具体的な実装を提示した上で、以上2つの観点からの位置付けを試みることだ。mimum は先述した音楽のためのプログラミング言語の歴史を見直し、汎用プログラミング言語の設計の上に最低限の音楽のために特化した言語機能を備える構造を取ることで音楽のための言語におけるブラックボックスを減らしながらもその実装の単純さと自己拡張性の高さを同時に実現できるように設計されている。

1.6 構成

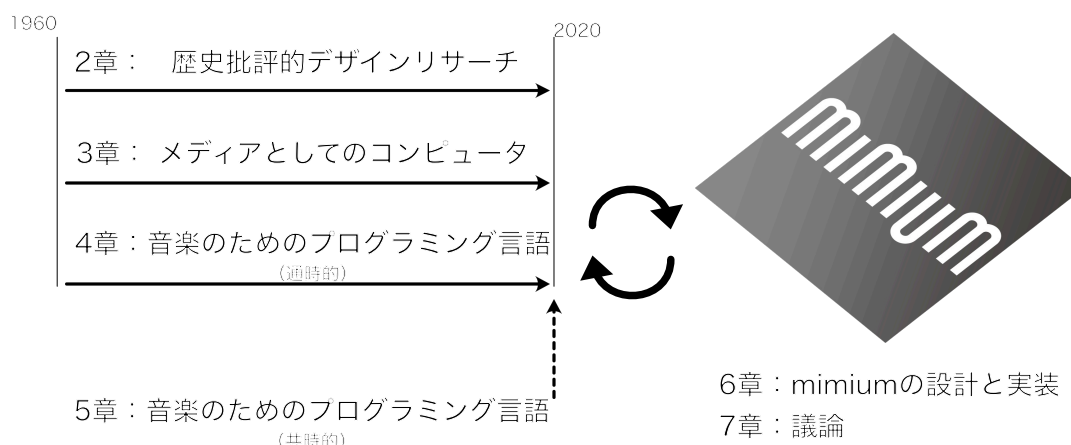


図 1.2: 本書で提示する音楽プログラミングの歴史の概要。

第2章以降は以下のような構成で論じる。本論文全体の構造を図1.2に示した。全体としては、24章で1960年代から2020年までの歴史をデザインリサーチ、メディアとしてのコンピューター、音楽のためのプログラミング言語というそれぞれの視点から振り返る。これらの歴史は例えばデザインリサーチとデザインであれば70年代以降のコンピューターを用いた機器のインタラクションデザインという点で重なりはするものの、概ね独立して読めるような構成に~~な~~っている。第5章は通時的な視点でのPLfMの整理、第6章は実際の言語設計と実装についての解説、第7章はその実装の問題点を起点にした省察である。なお、第2章から4章にかけての歴史的な文脈の整理は必ずしもmimumという言語制作の背景として事前に定まっていたものではないことに注意してもらふ必要がある。第2章で詳しく説明するように本研究の特徴的な点は実践を通じて、これまでとは異なる視点での歴史記述を行うことにあるからだ。

第2章では、まず音楽のためにコンピューティング技術の基礎的要素を再考するという筆者の研究の立脚点をデザインの歴史から説明する。コンピューター上で動作するシステムやインターフェースを作るということは、何か特定の課題が存在し、それを解決するための道具を作り、その効果を測定するといった工学的な視点で捉えられ、実施されてきた。しかしながらこれらの歴史~~は~~同時に、コンピューター以前に始まっていた工業製品のデザインにおける科学的手法と、その合理化に伴う功利主義やデザイナーの主体性の欠如~~という批判~~と~~な~~った文脈と接続され、近年ではヒューマン・コンピューター・インタラクション (HCI) 分野においても、研究領域として、科学と異なる方法論としてのデザインリサーチの文脈が立ち上がりつつある。その中にはシステムや人工物を創造する過程の中ではじめて問題が立ち上がり、それを知見として共有することを目指したデザイン実践を通じた研究 (Research through Design: Rtd) のような文脈や、デザインしたものを直接プロダクトとして利用するのではなく、その人工物を通じて未来の技術環境やあり得るかもしれない別の社会の姿の想像力を喚起し、議論を引き起こすことを目的としたクリティカル・デザイン、スペキュラティブ・デ

に対する

ザインと呼ばれる試みがある。第2章では本研究で目指す、プログラミング言語の実装という実践的経験を通して、音楽とテクノロジーに対する個人の関わり方の異なる可能性（＝音楽土木工学という新しい学問領域）を提起する行為をこうした文脈と接続する。

第3章では、メタメディア装置としてのコンピューターの思想の当初の理想とその現状について、エマーソンの「Reading Writing Interfaces」(Emerson 2014)を参照しつつ、ユーザーが真っ当にソフトウェアをすることで音楽表現を行うことが難しい状況が、テクノロジーを積極的に用いる芸術家のアプローチを、ベンディング（＝技術の誤用）のアプローチへと近づけたことを示す。~~その上で~~、2000年代までは機能していたサーキットベンディングやグリッチのようなアマチュアリズムを伴う意図的な技術の誤用は今日のソフトウェア中心の音楽技術文化の中ではもはや機能せず、技術を深く理解した上で、コンピューターというブラックボックスを自らの手で開いていくようなアプローチへと転換を迫られている。これは結果的に2章で議論するデザイン運動とも並行しており、そのブラックボックスの開き方の具体例を音楽に限らないデザイナーやアーティストの取り組みから比較することで、ブラックボックス性の少ない、インフラストラクチャとしての PLfM の設計を広義の音楽実践として位置付けることを試みる。

第4章ではより具体的な例を挙げつつ、PLfMの歴史を整理し、現在設計すべき言語の方針についての示唆を得ることを目指す。ここでは既存のコンピューター音楽のためのプログラミング環境についての文献を参照しつつも、より個々の言語のインフラストラクチャとしての役割に焦点を当てる。また本研究ではジョージナ・ボーンの80年代における IRCAM における作曲家と技術者の関わりを描いたエスノグラフィや、田中による80年代を中心に発生した、初期パーソナルコンピューターやその後のゲーム機における音声合成 IC を活用した音楽表現であるチップチューン史を参照することで、より制度化（Institutionalized）、もしくは権威づけされたコンピューター音楽と、これまでコンピューター音楽の歴史に含まれることは比較的少なかったチップチューンのような、いわば傍流のコンピューター音楽を意図的に対比することで、より広い意味での（すでに用いた言葉を再利用するなら“弱い”コンピューター音楽の）歴史観を提示することで PLfM の定義をより明確にする。

第5章では PLfM の現代における特性や概念を通時的な視点で整理する。そのために、PLfM の実装の方法の違いをドメイン固有言語のデザインパターン、汎用プログラミング言語における評価語彙の研究を参照しつつ、PLfM を使用するプロセスを Human-in-the-Loop モデルとして提示し、ランタイムのブラックボックスを大きくすれば動的变化に強くなる、小さくすれば表現自体の汎用性が高まる、動的变化と汎用性を両立しようとすると設計が複雑化し実装のコストが嵩むといった根本的トレードオフが存在することを提示する。

第6章では筆者が設計した音楽のためのプログラミング言語 mimium(Matsuura2021)の詳細を記述する。まず第2～4章で示してきた背景を総括し、1. ブラックボックスを可能な限り減らす、2. 新しい表現を生むことを目的としない、3. インフラストラクチャとしてのプログラミング言語の必要性という3つの設計方針を示す。mimium は型推論を伴う静的型付け言語であり、汎用の関数型プログラミング言語の設計や実装に、音楽のための言語仕様を2つ、最小限追加する形で実装されていることが大きな特徴である。1つは関数の実行のスケジューリングを行う @ 演算子、もう1つは内部状態を伴う信号処理を通常の関数と同じよう記述可能な、状態付き関数の記法である。mimium はこの2種類の記法を持つことで、これまではブラックボックスとして与えられていた基本的処理の単位をライブラリとして（実行性能を損なわないまま）実装可能であることを、既存の言語との比較を交えて示す。

第7章では、第2～6章で焦点を絞りながら説明してきた背景を、mimium の現状における実装の問題点を基に各章ごとの話題において振り返る——すなわち、PLfM の設計上存在するトレードオフの選択、PLfM の歴史における mimium の位置付け、音楽実践の1つとしてのプログラミング言語実装、RtD としての PLfM 設計、という順番で視点を広げていながら制作を分析する。この中で PLfM 制作においてブラックボックスの少ない言語を作る方針を突き詰めると、ホスト言語の実装とその上でのライブラリ構築という2種類の作業へ分岐する。そのため、結果的にホスト言語の実装には音楽に関わる作業が少なくなるという分断が発生して

しまう~~よう~~矛盾について説明する。

最終章では、前半で示した PLfM の存在論、歴史と後半で示した mimium の実装とその実装過程の分析をより一般化し、音楽とテクノロジーの狭間で行われる非常に個人的な動機からの問題設定を基に始めた~~ち~~道具づくりを、多くの人間が共有して使うものへと開いていく研究領域を「音楽土木工学」として、~~近年の音楽情報処理研究との対比や計算機科学の動向交えて検討する。今後の研究課題として、音楽という応用領域の中では背景化されてきたオペレーティングシステムやコンピューターアーキテクチャといった計算機の根幹に関わる理論自体の見直しを行うことで、mimium の実装に伴って明らかになった問題の解決の糸口になることを示す。~~

第2章

歴史を記述しなおすデザインリサーチ

2.1 音楽のための道具づくりを研究するとはいったいなんなのか

Do the address issues that are most vexing or pressing to musicians? Furthermore, in terms of technology, these contributions seem to be rather unsophisticated.

この論文は音楽家にとってもっとも厄介な、もしくは差し迫った問題を扱っているのだろうか？ さらに言えば、技術的な観点では、本論文の貢献はかなり洗練されていないように見えます。~~3~~

これは、第6章で扱う、筆者が開発した音楽のためのプログラミング言語 mimium を初めてある国際会議に投稿したときについた査読コメントの一節である。投稿結果はもちろん拒否だった。私はこの査読結果にそれなりに落胆しつつもある程度は仕方ないと諦めをつけていた。第6章で詳細に示すが、mimium の設計で取り組んだ内容はプログラミング言語理論に対してある程度精通していなければその新規性を伝えることが難しく、かといってプログラミング言語理論について研究する学会の中では、信号処理や音楽の文脈を伝えることも限られたページ数の中では難しい。このとき投稿した国際会議は、投稿を受け付けている時期の問題もあって、音響エンジニアリングに関する研究（例えば信号処理や録音再生技術に関わるもの）が中心的な話題のものであった。もちろん音楽のためのソフトウェア開発はそのトピックの中には含まれてはいたが、音楽のためのプログラミング言語はかなり周縁的な話題である事は投稿の時点で疑いようもなかった。

私がやっている事はどうしても技術的に役に立たなさそうで、音楽家にとって興味深いツールになりそうもない、非常に価値の薄い研究をしているのかもしれない。査読コメント読むとどうしてもそうした感情に覆われながらも、一方でこのコメントにはどうしても腑に落ちないところもあった*¹。音楽家にとって最も厄介な、もしくは差し迫った問題とはいったい何なのだろうか。そして、音楽家のための道具を作る研究者は、音楽という利害関係や良悪の基準が非常に多様な分野のための道具を作るときにも、常にもっとも差し迫った問題について考えなければならないのだろうか？ ~~極めて~~

それに加えて、自分の作っている mimium という言語がまったく的外れで理解不能なことをやっているわけでもない~~と~~事は感じられていた。mimium という言語の開発は2019年の未踏IT人材発掘・育成事業（以下、未踏事業）という、経済産業省の沿革団体である情報処理推進機構（IPA）が主導する革新的なソフトウェア制作に対して金銭的補助、またコミュニティやメンターからのフィードバックを9ヶ月掛けて受ける支援事業に採択される事で初期の開発が行われた。

未踏事業は2000年度から歴史を持つ、類似する支援事業の中では比較的古くから続いている事業で、その特徴としては採択されるプロジェクトの幅が非常に広く、かつ即効的な実用性のみを必ずしも求めないことがある。例えば筆者が採択された年では、ジャズ演奏の練習を支援するための音源分離技術を利用した“耳コピ”

*¹ そもそもこの投稿時の査読コメントがこの文章を含めて非常に短く、査読者は一人だけ、技術的コメントはやや的外れなものだったがリパッタル（査読コメントに対する反論を行う機会）が無いなど構造的な問題もあった。

を支援するためのスマートフォン向けアプリケーションのような[△]すぐにも実用的に利用できるものもあれば、秘匿計算（プログラムの入力データを暗号化したまま計算し、暗号化したまま結果を得る技術）のための、実行速度はまだ実用的ではないが実際に稼働する計算機アーキテクチャ、エミュレータ、コンパイラを含めた総合的プラットフォームの開発のような、セキュリティ技術に関連した非常に基礎的[△]だが、実現すれば大きなインパクトをもたらし得る[△]技術を開拓するプロジェクトも含まれる。また事業終了後のアウトプットの形も、論文執筆のようなアカデミックな形だけでなく、事業内で開発したものをもとに起業することも珍しくない。このような応用的な実践と基礎部分の再考という考え方が同居しているコミュニティは[△]後から振り返ってみればかなり珍しいものだ。

筆者のプロジェクトは、採択時のタイトルが「プログラマブルな音楽制作ソフトウェアの開発」となっている通り、その当初のアイデアは音楽のためのプログラミング言語設計とそのソースコードをグラフィカルに編集できるソフトウェアという[△]2つのプロジェクトを並列して行うものになっていた[△]が、同事業での過去の採択者や同じ年に採択された人とディスカッションを重ね、プログラミング言語や計算機アーキテクチャ自体の開発に興味を持つ人からのコメントをもらう中で、筆者の興味関心は徐々にその根幹であるプログラミング言語の設計そのものへと少しずつシフトしていった[△]*2。

もちろん、未踏事業のコミュニティにおいても音楽のためのプログラミング言語の開発というテーマは過去に近い例はほとんどなく、既に存在している Max や Puredata といった既存のツールとの違いの説明や、この言語を作ることが最終的にどういった面白みがあるのかを伝えることは先の学会への論文投稿と同じように課題ではあった。それでも、同事業の成果報告会では、音楽制作ソフトウェアとして実用的に活用するには現実的課題を多く残しつつも、概ね好意的なフィードバックを受けることができた。

あの未踏事業では（その意義を受け入れてもらうのに9ヶ月という長い事業期間があったおかげもあるとは言え）受け入れられ、[△]しかし投稿した論文では差し迫った問題にアプローチしているわけでないという理由で採択されなかった[△]違いとはいったいなんなのだろうか？ [△]図書の

本章ではそうした、音楽のための（とくに、コンピューターを用いた）道具を作るということを研究するのはそもそもいったいどういうことなのか、という問いについて考え、本研究を支える研究プログラムそのものの歴史的文脈を構築する。 [△]コンピューター [△]集の

表現のための道具づくりの研究はとくに、コンピューターの普及が進んで以後[△]ヒューマン・コンピューター・インタラクション（HCI）と呼ばれる、人間が計算機を使う際の相互作用を、主にその境界面：インターフェースを設計することで、[△]その利用方法の拡張を図る学術研究分野で中心的に議論されてきた。

とくに近年では HCI の研究の方法論の中でも、人工物を作る中での発見を知のあり方とするような、科学と異なる方法論の学術研究としてのデザイン、とくにデザイン実践を通じた研究（Research through Design : RtD）と呼ばれる方法論が立ち上がりつつある。

本章の構成としては、まず読者が全体的な研究領域同士の関連性を概観しやすくなるよう、音楽とコンピューティング、とくに本研究で例に挙げるプログラミング言語にまたがる研究分野の簡単な見取り図を紹介する。その上で、今日に至るデザインリサーチの歴史的経緯として、1960年代以降のサイバネティクスの煽りを受けたデザインと科学的手法の合流、デザイン思考のようなプロトタイプ、実験、反省の繰り返しという手法の浸透と、それらへの反動という形で現れた、それまでの支配的な考え方である問題解決主義[△]および統一的方法論を否定する、クリティカル・デザインやスペキュラティブ・デザイン、RtD といった運動を紹介する。RtD の考え方は日本国内での浸透度合いは未だ不十分だが、国際的には HCI 分野の最大規模の国際会議 CHI で 20 年近く議論が繰り返される中で[△]その歴史的文脈は確立しつつある。しかし、音楽とコンピューティングに跨る分野での研究の方法論の議論の中では、RtD は十分に着目されてはいない、もしくは単なる問題解決的

*2 未踏 IT 人材発掘・育成事業：2019 年度採択プロジェクト概要（松浦 PJ）
（<https://www.ipa.go.jp/jinzai/mitou/2019/gaiyou.tk-1.html>）を参照。また同時期に情報処理学会音声情報処理研究会にて同様の案についてのポスター発表を行っている（松浦 and 城 2019a）。

思考に縮退してしまっている。このことをいくつかの文献を交え紹介しよう。

RtD が立ち上がる中で重要視されてきたのは研究を行う中で自らの考え方や、モノや歴史に対する認識が変化する自己反映性：Reflexivity^{*3}を研究の過程に織り込むことである。本研究はその考え方を一歩推し進め、物を作る過程で歴史的視点を再編成するということも知のあり方であるという立場を取る。とくに、メディア論の中で近年立ち上がりつつあるメディア考古学と呼ばれるアプローチや、ダニエル・K・ロスナーの批判的奇譚づくり（Critical Fabulation）（Daniela K Rosner 2018）という考え方と、筆者自身のこれまでの作品制作をもとに、音楽のためのプログラミング言語の設計そのものを研究成果と~~する~~のではなく、その過程で変化した視点をもとに改めて歴史を記述すること~~を~~学術的な知の貢献とするという考え方を~~する~~。

2.2 隣接する学術領域

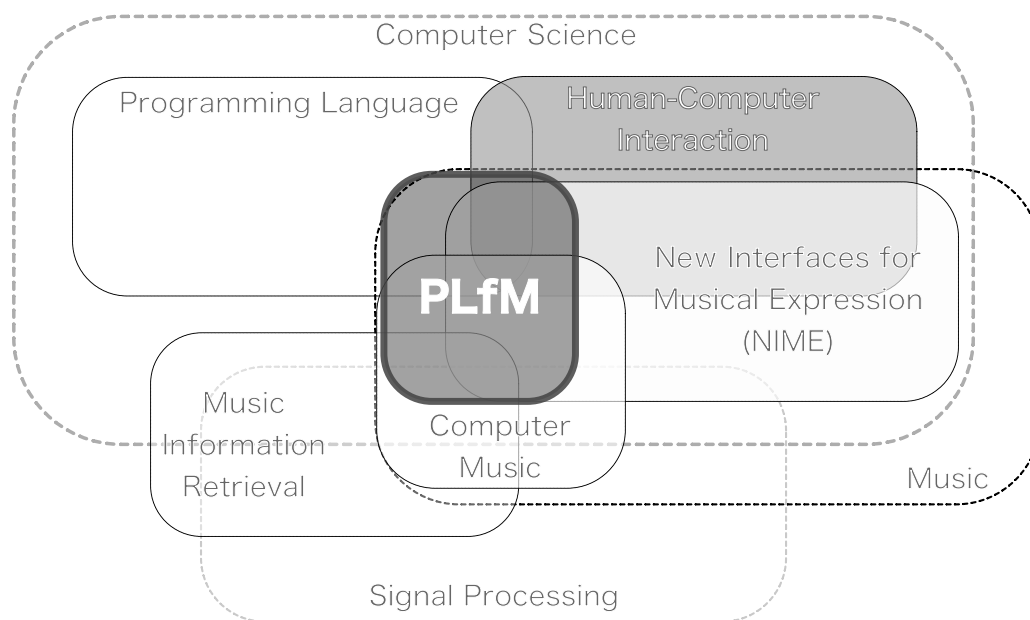


図 2.1: 音楽とコンピューティングに関わる研究領域の見取り図。

まず、音楽のためのコンピューターを用いた道具づくりや、本研究で行う音楽のためのプログラミング言語（Programming Language for Music: PLfM）の研究を取り巻く研究領域について概観しよう。代表的な研究領域の重なりを図 2.1 に示した。

上側の大きな囲み：Computer Science(コンピューター科学) が、計算機についての研究全般を表す領域である。この図に表されていないがコンピューター科学に含まれる領域としては例えばセキュリティやネットワークに関する研究などが挙げられるだろう。Human-Computer Interaction(HCI) と呼ばれる分野は、根本的には数値の計算を行うだけのコンピューターという機械を、人間の生活を補助したり創造性の支援を行うなど様々な~~人との~~関わり方をデザインすることを目的とした分野だと言えるだろう。今日では当たり前となっ

^{*3} Reflexivity は再帰性、反射性、反照性、反省性、（原因と結果の）相互参照性など様々な用語で訳される言葉だが、本稿ではプログラミング言語理論における再帰（Recursion）との不要な混同を避けるため（実際には広い意味で確かに共通している概念ではあるのだが）、自己反映性という用語を用いる。この語はプログラミング言語において、言語自体の記法を自己拡張できる、もしくはその言語自体が処理系に機能変更を命令できるような機能：リフレクション（Reflection）の訳として使われるもので、社会学における Reflexivity の意味としてはプログラミング言語における再帰よりもこちらの方が類似していると~~考えられている~~。

ている、マウスやキーボードの入力をもとに、演算結果として文字や画像などがディスプレイに即時表示される、GUI (Graphical User-Interface) も、それまでのパンチカードやテキスト主体の入力方式しか存在しなかった時代を経て、1970 年代ごろからのコンピューターの道具としての利用法の拡張を試みてきた結果として確立されてきたものである。

また HCI 分野から派生した音楽に関する研究領域として、**New Interfaces for Musical Expression : NIME** (音楽表現のための新しいインタフェース) が挙げられる。NIME は 2001 年に、HCI 分野における世界最大規模の国際会議、CHI の中でのワークショップのひとつとして始まったものだが、現在は独立した国際会議として、単なる研究発表やワークショップのみならず、自作のシステムを用いて行う演奏を中心としたコンサートや、インスタレーション作品展示などが同時に行われている。

音楽のためのプログラミング言語は NIME よりも、**Computer Music** (コンピューター音楽) の分野で長く研究されてきた。これは第 4 章で詳しく触れるが、今日の音楽のためのプログラミング環境の祖先にあたるものの研究は、1950 年代というコンピューター黎明期における、コンピューターを用いて音楽を作ること自体に新規性があった時代の流れを引き継いだものと位置付けられる。そのため、この分野における代表的な国際会議 International Computer Music Conference (ICMC) や、Sound and Music Computing (SMC) Conference、また学術誌である Computer Music Journal (CMJ) における議論は、その制作環境そのものよりも作られた作品についての議論が中心であると言える。

また **Programming Language** (プログラミング言語) の研究は、実用的に用いる際の課題解決に主眼をおいたものと、言語を抽象的に形式化する理論に寄ったものがあるが、前者は音楽に限らないライブプログラミング (実行しながらプログラムを書き換え続ける手法) の研究や、プログラミング体験 (Programming Experience : PX) の改善に主眼をおいたものなどがあり、こうした領域は NIME と同様に HCI の領域と重なってくる。

一方後者のような理論寄りでありながら、音楽への活用を議論する領域としては、ACM SIG-PLAN (Association for Computing Machinery Special Interest Group of Programming Language) の中での、関数型プログラミングと呼ばれる研究領域についての会議 ICFP (International Conference of Functional Programming) の、さらにその中で開催されているワークショップ、FARM (Function, Art, Music, Modeling and Design) で、信号処理を対象とした言語や、Haskell など汎用の関数型言語を用いて実装された音楽のためのライブラリ、作品についての研究がなされている。

また **Signal Processing** (信号処理) も音楽とコンピューティングにおいて重要な話題である。信号処理という分野自体はコンピューター登場以前から、音楽のみならず通信分野で電気信号の伝達のための重要な研究領域である。音楽においては電気信号として空気の振動である音声を表現することが一般的になって以来、任意の周波数 (≡ 音の高さ) の信号を抽出するフィルターを電子回路を用いて作るための理論をはじめとして様々な研究がなされている。電子計算機の普及以降は、Digital Signal Processing : DSP (デジタル信号処理) として、電気信号に変わり離散化/量子化を通して任意の数値の列として (音声などの) 信号を表現、加工することで電気回路だけでは難しかった周波数領域での処理 (スペクトラルプロセッシング) などが可能になるなどその応用範囲が広がっている。とくに音楽に関わる信号処理の研究が中心の国際会議としては DAFx のような会議が挙げられる。

信号処理の中でもより音楽に関係する話題を中心しつつ、かつ音圧信号に限らずより高次の音楽に関わるデータの分析や、それをもとにした再合成などについて研究する分野として、**Music Information Retrieval: MIR** (音楽情報抽出) と呼ばれる領域が、国際会議 ISMIR などを代表として挙げられる。MIR では例えば音声信号からテンポやメロディのような情報や、近年ではより高次の曲調のような情報を解析したり、そうした特徴量を基に音声信号を再合成するような研究、またそうした分野と相性のよい機械学習を用いる音声解析と合成 (例えば、2ch ステレオの音源から個別の楽器音を抜き出すような音源分離、テキスト入力を基に音声を合成する Text-to-Speech など) のような研究が行われている。もちろん機械学習のような手法

は NIME や ICMC においても積極的に用いられるようになってきてはいるのだが、我々にとって確認しておくべき事項としては、MIR のような分野では用いられる中心的なツールは MATLAB や Python、Julia などより自然科学分野における分析に向けたプログラミング言語上で構築されたライブラリが主であり、Max や SuperCollider のようなリアルタイムインタラクションを重視した環境が使われることはあまり多くはないということだろう*4。

我々が mimium において議論しようとしている **Programming Language for Music : PLfM** (音楽のためのプログラミング言語) という枠組みは筆者が独自に定義するものだ。この語が意味するところは文字通りの意味合いではあるのだが、ここでは特に、Computer Music の領域に留まらない、より広い意味での音楽のために使われるプログラミング言語に関する研究という意味合い、~~強調する意味で新しい語を導入している~~ (詳しくは第4章を参照)。

それぞれの会議へのリンクを貼る *ok*。

2.3 デザインリサーチの変遷—デザインサイエンス、デザイン思考、RtD

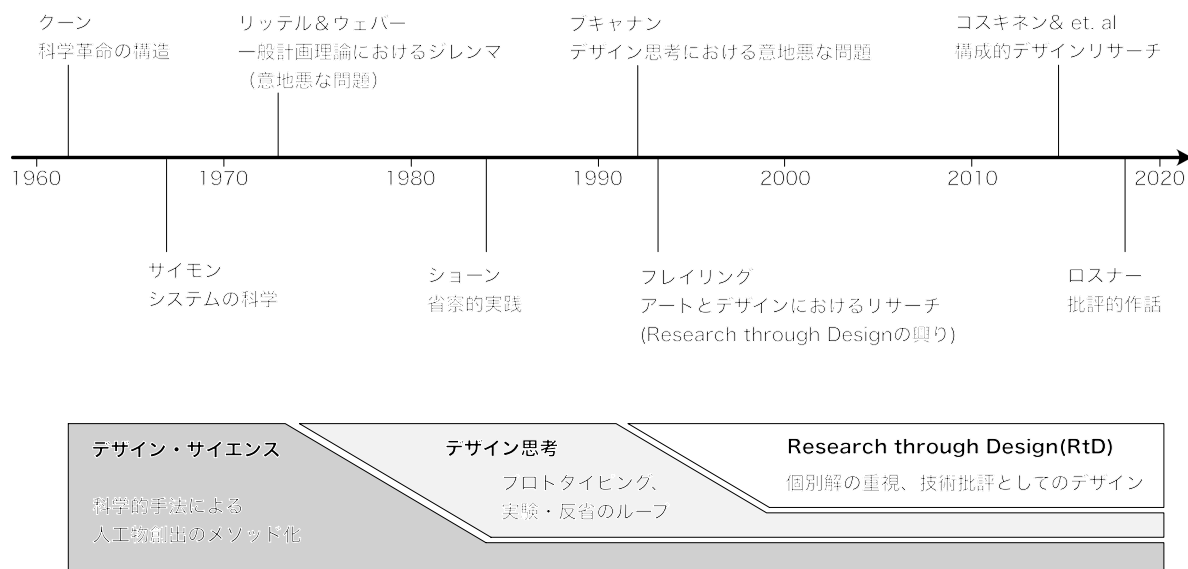


図 2.2: デザインリサーチの歴史の見取り図。

それでは、こうした学問的付置はどのような歴史的経緯で成立してきたのだろうか。本稿ではその成立過程をデザイン学の歴史的変遷を中心にした視点で追いかける。図 2.2 はその大まかな見取り図と運動に影響を与えた代表的文献を時間軸上に配置したものである。

学問としてのデザインの歴史は大きく分けて以下の3段階の潮流に分けられ、現在もそれぞれの方法論が同居して存在している状況がある。

1. デザイン・サイエンス：科学的、統一的手法による良い人工物の創出の方法論の確立

*4 なお、日本国内における音楽のためのプログラミング環境を議論するための場としては、情報処理学会（IPSJ）の音楽情報科学研究会（SIGMUS）などが考えられるが、近年では機械学習などを用いた相対的に高レイヤーでの音声情報処理が議論の中心となっている。SIGMUS からより音楽表現に関する議論に重きをおく形で独立した先端芸術音楽創作学会では（Nishino, Osaka, and Nakatsu 2014）などの数少ない発表を除き、プログラミング環境自体の構築のような研究はほとんど無い。プログラミング言語研究においては、PX の概念自体を提唱者である、産業技術総合研究所の加藤が立ち上げた SIGPX (<https://sigpx.org/>) という勉強会において表現の支援ツールを含めた議論が行われている（筆者も第8回にて第4、5章の議論のベースとなる発表を行なっている）。

2. デザイン思考：共感-問題定義-アイデア創出-プロトタイピング-テストの繰り返し

3. Research through Design(RtD)：統一的手法、客観主義、問題解決主義の否定、作る過程での問題発見

ゲイバーがこのような、統一的で定まった学問プログラムが存在していないデザインという領域をパラダイム成立以前（Pre-Paradigmatic）の研究と表現しているように（W. Gaver 2012）、デザイン学の歴史認識や、デザイン学という学問はそもそもどういった方法論の上で研究されているのかの認識は研究者ごとに大きく異なる。この方法論の不明瞭さは音楽とコンピューティングが交差する研究領域においても同様に存在しており、図 2.1 の付置でいえば、おおむね下側に行くほど客観主義的、問題解決主義的傾向によっていく傾向がある。音楽や表現のための道具作りの研究を行うにあたっては、本章冒頭で引用した査読コメントのように、問題解決主義的研究プログラムがそもそも前提となってしまうという齟齬が発生することは珍しくない。しかしデザインの歴史の中では、ユーザーに対して決定的な解を提供したり、研究を始める前から問題がはっきりとわかっているような考え方の限界が繰り返し指摘されてきた。一見して無関係に思える工学的な研究に関しても、そもそもどのようにして技術的な問題というものが立ち上がるのか、また誰のために、なぜ問題を解決するのかという思索的な問いは常につきまとっている。これは本研究のような表現に関わる工学的研究という領域で比較的わかりやすい形で表出する問いではあるが、根本的にはどんな種類の工学的研究にも関わってくる問題である。

2.3.1 デザインの科学化と科学の社会構築

デザインという人工物（プロダクトやシステム）の創出行為を、科学的手法を応用することで、誰もが普遍的方法を用いて良いものを作る方法論として定式化しようとする運動は 1960 年代ごろに始まっている。

デザイン研究者の水野大二郎は今日までの「デザインリサーチ」の源泉としてデザインサイエンスと呼ばれる 1960 年代の動向を取り上げ、その要因となる社会的背景として、“欧米社会が豊かになるのに併せてデザインされる対象の規模や機能が拡張、複雑化したと同時に、生産の合理化・効率化が求められた時期”であり、“芸術家的あるいは職人的な経験と勘に基づくデザインメソッド（デザインの方法論）を客観的に学術研究の対象とし、デザインを精緻に理解することが求められた”~~からだとする~~（水野 2014）。デザイン・サイエンスの潮流に大きな影響を与えたのが心理学・政治学など人間の意思決定に関する研究者であるハーバート・サイモンによる 1969 年初版の「The Sciences of the Artificial」*5である。同書でサイモンは、経済学や認知心理学における当時の最新の知見を取り入れ、世界をシステムとして捉える見方を人工物の創出一般に適用することを試みた。そこには電子計算機がそもそも求められた理由でもある、シミュレーションという行為を通じて我々が世界について知ることができるという意味での、自然環境との境界面：インターフェースとしての人工物の存在意義が示されていた（Simon1993）。~~コンセプト？~~

実際 1960 年代以降にはデザインに限らず建築や芸術作品を作るにあたって科学的なアプローチを用いるもの~~たち~~によって、言うなればアートやデザインの科学化とも言える現象が起きていた。たとえばデザインサイエンスという言葉を広めた建築家のバックミンスター・フラーや、1968 年に行われた、テクノロジーを用いた芸術作品に関する初めての大きな展覧会である Cybernetic Serendipity 展、1970 年大阪万博における E.A.T が主導したペプシ館の展示のような事象が~~並~~られる。デザインサイエンスやこうしたアート&テクノロジーが 1960 年代に発展した背景には、ノーバート・ウィーナーやロス・アシュビーのような学者らによって立ち上がったサイバネティクスや、フォン・ベルタランフィの一般システム理論のような、人間と機械の相互制御をフィードバック・システムを中心に~~も~~理解しようと試みた学問の隆盛が、~~それ~~そのさらに背景には冷戦による科学技術への国家的投資の過熱があった。今日の人工知能研究のはしりでもあるサイバネティクスの思想は、機械の自動制御にとどまらず、生命の構造をモデル化し、さらには経済や社会までもを大きいひとつの制御可能なシステムと捉えようという形で拡大してきた。~~まとめ~~

*5 直訳すると「人工物の科学」、邦題は「システムの科学」。

この「あらゆるものの科学 (Science of Everything)」は奇妙なことに社会を安定して管理するための手段として国家や軍事産業のテクノクラートたちに受け入れられつつも、同時にフィードバックコントロールを非中央集権的な制御と自己管理と捉えた、ヒッピームーブメントのような左派イデオロギーにも受け入れられたことが知られており (Bernes 2021)。デザインやアートといった分野に限らず、経済学や社会計画といった分野にも大きな影響を与えた*6。サイモンの、「現在の状態をより好ましいものに変える」行為としてのデザインという考え方は、現在まで RtD の文脈を含めてデザインリサーチの論文で多数引用される考え方になっている*7が、この言い回しもサイバネティクスに特有の、世界をフィードバックシステムとしてモデル化し、その出力から目標との差分=エラーを少しずつ増分的に減らしていくことで理想状態へと近づけていくという考え方が色濃く現れたものである。

しかしその一方、同時期に科学論の分野では、自然科学という分野を絶対的真理の探求の営為としてではなく、社会的関係の中に成立する学問として捉える動きが発生している。デザイン学がパラダイム成立以前の学問だというゲイバーの主張をすでに引用したが、このパラダイムという語は1962年のトーマス・クーンが「科学革命の構造」の中で、科学者や研究者の中でのある時期の共通した世界認識といった意味合いで導入された語だ (Kuhn1962)。科学はそれまでポパーが主張してきたように反証可能 (Falsifiable) な命題にのみよって示され、そうでないものは疑似科学にしかなり得ないと説明されてきたが、野家の説明を借りればパラダイム論では「理論は事実によってではなく、競合する別の理論によって打ち倒される」(野家 1981, p69) とされる。その時代で支配的だったパラダイムの中では説明、解決できない問題が出てきたときに全く別の理論体系を採用しなくてはならないときに科学革命=パラダイム・シフトが発生するのである。今日ではパラダイム・シフトという語はおおよそ、これまで常識とされてきた認識が覆されてしまうような状況全般を指すような拡大された使い方をされているが、このパラダイム論は科学論としては、例えばデヴィッド・ブルアの「数学の社会学」に代表されるような、自然科学という分野を絶対的真理の探求の営為としてではなく、それすらも科学者というある特定の社会集団の営みとして社会学的に分析できると捉える社会構築主義という立場に大きく影響を与えた (Fukushima2021)。

つまり、1960~70年代はデザインや芸術に科学的手法が取り入れられる一方で、同時にいわば科学の人文知化とも言える運動も発生していたのである。それゆえ、デザイン・サイエンスの運動もこうした科学論の議論を取り入れながら、あらゆる問題をワンストップで解決できるようなシステムの構築を目指す普遍主義 (Universalism) とも言えるものが徐々に否定されるようになる。

2.3.2 意地悪な問題、自己反映性、デザイン思考

デザイン・サイエンス的思想を支えた、あらゆるもののシステム化という思想の限界は、1973年のデザインの科学的研究者であるリッテルと都市計画研究者であるウェバーによって意地悪な問題 (Wicked Problems) として提起され (Rittel and Webber 1973)、1992年にブキャナンはこれをデザイン学の統一的メソッドづくりの限界の指摘に用いた (Buchanan 1992)。

意地悪な問題は今日のスペキュラティブ・デザインやクリティカル・デザインのような複雑な社会問題に対処することを目指したデザイン運動の議論で必ず現れるほど重要なキーワードのひとつとなっている。リッテルが挙げた意地悪な問題の特徴は以下のようものである。

1. 決定的な定義と解が存在しない

*6 (Rid 2016) も参照。例えば右派的なものであればチリで1970年代に試みられた国家管理システム Cybersyn についてのメディアの研究をみよ (Medina2006) ((Myers 2015) も参照)。左派的なものとしては、Apple 創業者のスティーブ・ジョブズにも大きく影響を与えたスチュアート・ブランドによる「全地球カタログ」の第1号の巻頭にウィーナーの書籍が引用されていることがよく知られる (Spectator)。

*7 (Simon1993)。Research through Design の文脈においてこの考え方を強調している文献としては (Zimmerman, Stolterman, and Forlizzi 2010) がある。

2. 停止するためのルールが存在しない
3. 解は真か偽ではなく良か悪でしかない
4. 解を得るための即時的に、完璧な結果が得られる実験はありえない
5. 全ての問題に対してトライアンドエラーの機会がない（やればやるほど困難な問題ばかりが残っていく）
6. 解決策や許容される策を網羅的に記述することができない
7. 全ての意地悪な問題は本質的にユニークである
8. 全ての問題は他の問題の前兆である
9. 問題の解釈と説明次第で解決策が変わってしまう
10. 意地悪な問題に取り組むものは解決案に責任を負う事になる

意地悪な問題はこの 10 カ条の形そのままで紹介されることが多いが、その本質は問題設定と解決のプロセスにおける自己反映性を無視した合理化の限界という点で共通していることはあまり強調されていない。

意地悪な問題が提起された論文のタイトルは「計画の一般理論におけるジレンマ」というサイバネティクスの残滓が見て取れるタイトルで、政策決定などを科学的根拠と手法に基づいてシステマチックに行うことの限界を示したものだ。リッテルはこの「意地悪な問題」の原点に、合理性（Rationality）に深く根差したパラドックスがあることをあげている（Rittel 1982）。リッテルのいう合理性とは行動する前にその結果を予測することであり、これには 4 種類の矛盾が存在するという。

まず 1 つ目には、（例えば政策決定などで）行動、つまり時間や金銭の投入を実際に開始する前に、その行動が引き起こす価値を最大化するために結果を予測する必要がある。が、その結果の予測のためには何らかの根拠を揃えるための研究活動が必要になる。しかしこの研究活動自体にも時間と金銭を投入しなくてはならないのでその価値の最大化のための基礎研究が……という無限後退が存在する。

2 つ目には、例えばある上司が合理的に仕事をこなそうとしている部下を管理する時、部下が実際に行動しはじめた後に上司がそれを何らかの要求で介入して中止することは、部下にとってはどんな理由であれ自分の決定と異なる外的要因による遮断であり非合理的なものになってしまうということ。

3 つ目は、合理化を進めるほどにある行動のもとに何かが生まれるという予測をするということは、ある種はじめから全ての結末が決まってしまうことを意味し、行動するためのモチベーションをなくしてしまうという矛盾だ*8。

4 つ目は、合理性のために予測を行うということは、サイバネティクスの発想で言えば、世界をなるべく現実と同じように振る舞うモデルを構築しシミュレーションを行うという説明ができるが、このモデルによって生み出した結果が後の世界の構築に影響を与えるのであれば、そのモデルは世界から生まれたものでもあり、世界を生み出すものであることになる。つまりモデルはそのモデル自身の未来を自己参照して記述されなければならないという矛盾だ。

この 4 つの説明を読んだ上で改めて意地悪な問題の十ヶ条を読み直すと、意地悪な問題の例示としてよく挙げられる環境や経済、政治における問題における説明のように、単に関連し合う問題が複雑に絡み合っているからという理由や複数のステークホルダーごとの価値観の違いというような意味合いよりも、ある問題のフレーム設定をする事によって、そのフレーム設定自体が社会に影響を及ぼしてしまい別の問題を発生させたり、問題を記述する事で、記述されなかった問題に後々向き合うことが大変になったりするという因果結合の方に重きが置かれていたと理解する方がより正確である。

とはいえこの合理的未来予測における矛盾は確かにあり得そうな一方で、今日我々が歴史を振り返った時には、科学技術が一見線形的に安定して発展してきたように見える事象も確かに存在しており、直感と異なる●をどう説明できるだろうか。たとえば、ムーアの法則という、IC チップに集積されるトランジスタの密度が

*8 リッテルはこれを、合理的に考えたら我々は将来的にみんな死んでいるのだから我々が今何をどうしようが関係のないことだろうという冗談めいた説明をしている。

時間と共に指数関数的に増加することを予測したもの (Moore1965) は有名な例である。トランジスタの集積密度を上げるための技術的工夫には必ず質的なブレイクスルーが必要であり、そのための技術開発という不確定性を含むはずだが、現実にはかなりの精度で予測の通りの集積密度の発展が成されてきたという事実がある。

この奇妙さは科学社会技術論においてはハロ・ヴァン・レンテらが「科学技術における『期待』の社会学」と銘打った研究の中で、こうした未来予測は、結果の予測というよりもむしろ、産業を発展させるための投資をブーストさせるような“期待”を自ら引き起こすことで産業界などの動向を決定づけるための要因として自己参照的に働き、結果的に技術者たちがそれに応えることによって予測通りの形に落ち着いている、という見方がなされている (山口 and 福島 2019, p7)。ブラウンやレンテらはこの自己反動的なプロセスを図 2.3 の様に入れ子上の過程として説明している (Brown, Rip, and Lente 2003)。

すよ、

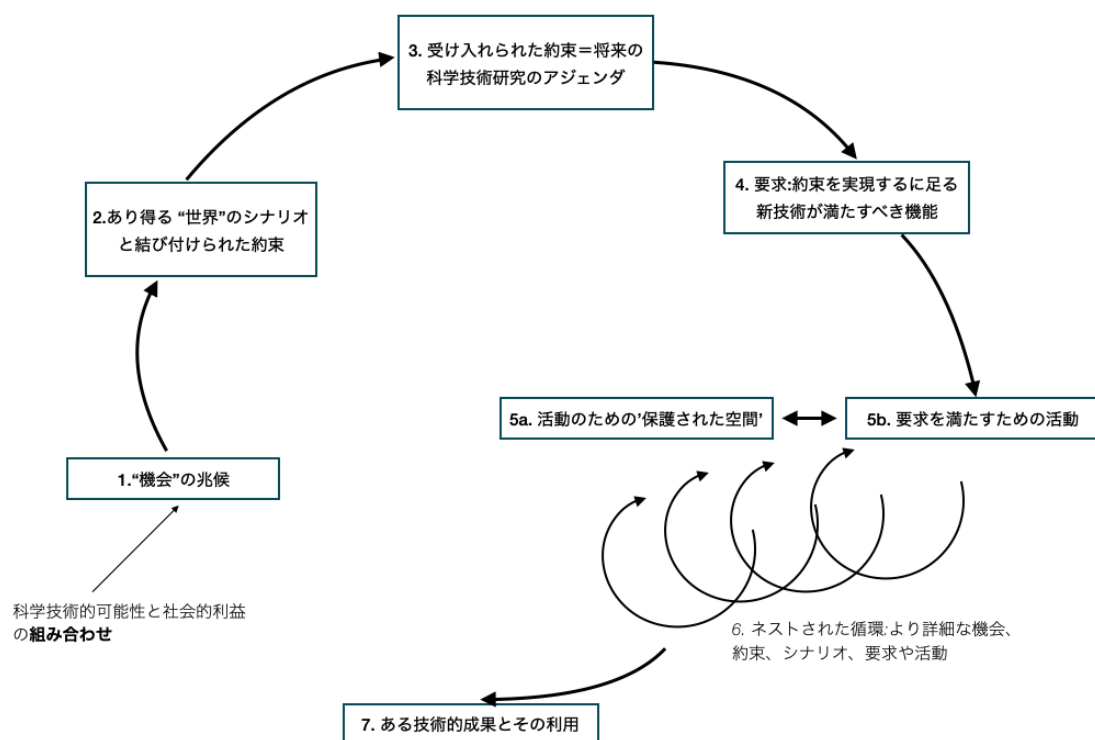


図 2.3: Brown による、技術発展における約束と要求の推移過程を示した図 (Brown, Rip, and Lente 2003) をもとに筆者が作成

何らかの技術的要素が社会的利益を生みそうだという兆候がまずある未来予測 (≡ムーアの法則) を生み、その予測をもとに科学技術の進むべき方向性が決定され、一度それが可能な研究となった場合その研究分野の開発は研究者たちへ“権限が委任 (mandate)”され、科学者にとっては安定して研究開発に臨む保護された領域が作られると同時に、その目標の達成に責任 (≡ムーアの法則に追いつくこと) を負うことになる。そして実際の実現のためにはこのプロセスがより小規模な可能性と兆候、未来予測に基づいて再起的に発生していくというモデルだ。このネストした循環というのはたとえばレイ・カーツワイルのシンギュラリティ論 (Kurzweil) の様な典型的技術決定論者の未来予測という大きなループの根拠にムーアの法則が用いられる状況をよく説明している。

リッテルの意地悪な問題の提起とはこうした期待の社会学の視点では次のように説明し直せる。これまで客

観的かつ合理的な予測のもとに発展してきた科学技術は、本来あり得ないはずの自己参照的未来予測を成立させているように装っているだけであり、実のところは、未来の進むべき方向性を、自らの主体的意図（例えば、ムーアという IC チップを生産する側の人間が自らへの投資を煽ること）という色を薄めつつ浸透させている状況のことだったのだ。意地悪な問題とは詰まるところ、サイバネティクスのフィードバックによる自己調整的社会という虚像のもとシステムチックな思考を推し進めた結果、結局誰がどう、どこへ社会を進めるべきなのかという議論が欠如していることを自ら露呈させたとも言える。

リッテルの問題提起はデザインの科学的方法による実証主義的、統一的方法での画一化の限界を示し、それが 1970 年代から 1980 年代における「デザイン思考」の文脈における、過度な科学的アプローチに陥らないための方策として、プロトタイピング、実験、思考の繰り返しによる問題発見と問題解決の同時遂行を目指すようなアプローチを生んできた。今日デザイン思考と呼ばれるこうした考え方は例えば、IDEO を創業したデイヴィッド・ケリーがスタンフォード大学における授業の中で確立した共感-問題定義-アイデア創出-プロトタイピング-テストの繰り返しの手法のように、デザインを科学的でシステムチックなフィードバックループとしてではなく、実践の過程自体で自らの視点の変化を織り込んだフィードバックループとして捉えるようになった。またドナルド・ショーンの「省察的实践 (Reflective Practice)」も、教育分野を中心に大きな影響を与えた (Schon1983)。ショーンは自分で実際に手を動かしながら考える (Reflection-in-action) こと、実践の過程で自らが変化する自己反映性の重要性を説き、サイバネティクスにおける自動制御のような身体性の欠如を批判した。

しかしこうしたデザイン思考は、1960 年代のデザインサイエンスにおける客観主義を否定はしたものの、今日デザイナーのイメージをアイデアを書いた付箋をホワイトボードに貼ってディスカッションしている人たちに固定してしまったように、あらゆる問題の種類に対して作る（もしくはアイデアを出す）-反省という繰り返しの枠組みだけを与える普遍主義に留まってしまっている。

ブキャナンによる意地悪な問題のデザイン思考プロセスにおける紹介はサイモンのようなデザインの科学化に対するアンチテーゼでもあると同時に、デザイン思考がそのプロセスによって問題を確定させる (Determinate) 方向に働いているが、そもそもリッテルらが提起した意地悪な問題とは、根本的に確定することができない (Indeterminate) ことだったことを強調している。結局人文的アプローチであったとしても、大枠がプロトタイピング、実験、思考の繰り返しという一般化されたフィードバックでは同じ穴の貉であり意地悪な問題には対応できないということだ。ブキャナンはこれを、確定 (Determinate) できない主題であっても特殊 (Particular) 化することはでき、デザイナーは、クライアントがまだ準主題 (Quasi-subject matter) である状態のものを特殊化する事によってエンジニアなどが解決可能な問題にスライドすることができるという提言をしたのだ。

こうした流れのもと 1990 年代以降に、デザイン実践を通じた研究 (Research through Design : RtD) と呼ばれる研究領域が主にデザイン・サイエンス的アプローチの行きづまり（と並行して起きていたデザイン思考の一般的方法論化）を突破するべく、デザインを”不明瞭かつ個別固有の社会・技術的問題を対象とする臨床的、生成的研究” (水野 2017) として位置付けるような運動として現れてきた*9。RtD には、問題を研究する過程で発見すること、必ずしも役に立つものだけを作るわけではないこと、問題解決だけを目的としないこと、社会そのものの変化を視野に入れること、ユーザーではなく参加者/共同製作者への転換、といった様々な特徴を列挙することができるが、これはリッテルの意地悪な問題への動機をもとに考えれば、RtD やその延長線上にあるデザイン運動において着目されている。 “どこへ向かうべきか” という社会の中におけるデ

*9 なおこのような研究として科学と異なる学問プログラムとしての“デザイン”は単にデザインリサーチ (Design Research) と呼称されることもあり、それはマルパスのような丁寧な定義を借りれば“製品やサービスの開発における、多様で複雑な問題の解決策を導くためのデザインプロセスを主に研究する学術領域” (マルパス 2019, p10) ということになるのだが、水野が“日本の産業界においてはデザインリサーチ (= デザイン研究) という言葉が「新規製品またはサービス開発のために文化人類学を応用した一連のユーザ調査とアイデア創出法」と限定的に理解されている”と指摘しているように日本国内では特にこの言葉は稀少化された理解をされていることに注意して、本項では RtD という用語を一貫して用いるようにする。

デザイナーの主体性という軸から派生したものとして捉えることができる。

2.3.3 Design into/through/for Art and Design

Research through Design という言葉が作られるきっかけとなったのはイギリスのロイヤル・カレッジ・オブ・アート（RCA）のクリストファー・フレイリングが1993年に提示した「Research in Art and Design」(Frayling 1993)における以下の三分類である^{*10}。

- Research **into** art and design （アートやデザインに関する研究）——歴史研究／美学研究／デザインやアートに関する諸理論に関する研究^{*11}
- Research **through** art and design （アートやデザインを通じた研究）——材料研究／制作実践（を通じたコミュニケーション）／アクションリサーチ
- Research **for** art and design （アートやデザインのための研究）——制作実践のための学び

ここでフレイリングがデザインだけでなくアートという用語も使ってこうした分け方を提示していたそもその理由は、プログラミング言語を作ることを芸術実践のひとつと捉えようとしている筆者の問題意識とも共通するところがある~~ので触れておこう~~。フレイリングがこの分類を提示した背景はアーティスト、デザイナー、科学者という職業のパブリックイメージとその実際の行為の解離、及びリサーチという行為自体が大学にあって制度化されゆく中で、研究者が行うべき行為とは何なのかを既存のフレームに頼らぬ新しい分類を与えることでその方向性を見出すことを目指したと捉えられる。

フレイリングは映画の中でのアーティスト（ゴッホ）や科学者（フランケンシュタイン博士）の描かれ方を挙げつつ、アーティストは未だに無からの創造を天才的感性を持って行うように、科学者は突然のひらめきに導かれる合理的で完璧な理論を持つ人間のように（時には合理化が行きすぎて狂ったマッドサイエンティストとして）描かれるステレオタイプが残っていることを説明する（これは2021年においてもまだ根強いものだろう）。しかし既に説明したように、1960~1980年にはデザインサイエンスやアート&テクノロジーのように、アーティストやデザイナーも作品やプロダクトを作るにあたって科学的知識やテクノロジーを活用して~~おり~~、~~また~~科学者も社会構築主義的科学論の代表的な分野である科学知識の社会学 (Sociology of Scientific Knowledge:SSK) において分析されてきた、よく観察すれば直感や身体化された経験に頼っている面があったり、合理性を伴わない社会的階級に影響された行動をして~~たり~~と^{*12}、現状のパブリックイメージとの開きは大きくなってきている~~上に~~、同じリサーチという言葉で表される行為の中でも、誰が実質的に何をやっているのかというカテゴリ分けが職能や肩書きをベースにしたものではもはや機能しなくなっているという背景~~があった~~。そこでフレイリングはアーティスト、デザイナー、科学者に共通して行われている営みを into, through, for という視点でいちど結合、再分解することでデザイナー（あるいは、リサーチャーという個人）が今何をすべきかへのヒントを見出そうとしていた。

今日の Research through Design という用語の定着そのものは、カーネギーメロン大学の研究者ジョン・ツィーママンのような、デザインをバックグラウンドにしつつもコンピューター科学の分野で活動する研究者らが、自らの学問領域を明確にするためにフレイリングの議論などを借りることで立ち上がってきた (Zimmerman2007)。RtD の解釈は研究者によって異なるところもあるが、本章の流れに沿って説明すればその特徴は、既に挙げた自己反映性への自覚に起因する二つの転換、問題解決主義 (Solutionism) から問題提起へと客観的観察から共同参加/あるいは介入である。

^{*10} 訳出は水野らによる以下の資料を参考にした。 <https://issuu.com/tacticaldesign/docs/generaloverview/8> (2021/11/23 閲覧)

^{*11} into の代わりに of や about と置き換えられることもある。

^{*12} フレイリングはこの動向の例として SSK の論者の一人であるハリー・コリンズを例に挙げている。

問題解決主義 (Solutionism) から問題提起へ

RtD におけるこれまでのデザイン研究からのひとつの重要な転換は、ユーザーや社会的な問題を解決することを目的とするのではなく、人工物を通じて問題に対しての議論を引き起こしたり、あるいはこれまで着目されてなかった問題を周知することを目指す、デザインにおける批評性の重視という点だ。この批評性の重視という側面では、RtD の中でも特にクリティカル・デザインやスペキュラティブ・デザインといった分野が HCI 研究にも関わる領域として存在する。マルパスはこのような批評性の強調の源流に、1960 年代のイタリアを起点としたアンチ・デザインのような反商業主義的デザインを見出している (マルパス 2019)。 **7-5, 21**

例えば、クリティカル・デザインとスペキュラティブ・デザインという言葉をついオナ・レイビーと共に作ったデザイナーのアンソニー・ダンと、HCI の分野において RtD の批評的側面を問い続けてきたウィリアム (ビル)・ゲイバーは 1997 年に HCI 分野の国際会議 CHI で “The Pillow: デジタル時代の ~~アナログ~~ デザイナー” というタイトルの論文を発表している (A. Dunne and W. W. Gaver 1997)。The Pillow は身の回りの電磁波に反応して表示パターンを変える LCD スクリーンが埋め込まれた ~~枕~~ の形のプロダクトである。もちろんこの枕は実際に製品として売りに出されることを目的としているわけではない。ダンとゲイバーはこの The Pillow という人工物を通じて、当時のインターネットや携帯電話の普及に際して変化しつつある、電磁波という物理的障壁を超えて飛び交うデジタル情報による ~~プライバシー~~ の意識の変化を顕在化させるを試みたのだ。例えばこの枕を置いて眠ろうとした時に、隣の家でテレビを付けたことを The Pillow が感知して光ったとしたら、知らない隣人の生活が電磁波を通じてこの家に侵入してきたことになるだろう。The Pillow 自体が実際に使われることはなくとも、その人工物にはある種のフィクションとして ~~今後起こりうるであろう~~ プライバシーの概念の変化の可能性を伝える説得力が存在している。このように、問題を解決するよりもこれまで気付かれていなかった問題を提起することが Research through Design ~~における~~ 1 つの特徴である。

客観的観察 (Objectivism) から共同参加/あるいは介入 **へ**

RtD におけるもう 1 つの重要な視点は、デザインされたものを使う人：ユーザーをデザイナーが観察することで問題点を洗い出すという図式を否定し、これまでユーザーと呼ばれていた人たちを共同でものを作る人として巻き込んでいく、あるいはユーザーの生活空間に積極的に介入していくという転換である。この観点で代表的に着目されるのが、やはりゲイバーやダン、そしてエレナ・パセンティら提案した Cultural Probe (文化的探針) という考え方である。ゲイバーらは、ちょうど電子回路にオシロスコープのプロブを当ててそこに流れる電圧や電流を測るように、ある文化圏やコミュニティに対する理解を深めるための方法としてこの Cultural Probe を提案した。Cultural Probe はポストカードや地図、使い捨てカメラなどを、質問や使い方のインストラクション (例えばポストカードには “あなたが大事にし続けているアドバイスや気づきについて教えてください” という質問、地図にはよく 1 人で行く場所、誰かと会うために行く場所などをマークしてもらい、カメラには今日初めて出会った人やあなたが今日着るつもり of 服を撮ってもらい、など) を詰め込んだキットを配布し、一定期間後に送り返してもらうものだ (B. Gaver, T. Dunne, and Pacenti 1999)。

こうした Probe の返信を収集した結果、ゲイバーらは送った地域におけるいくつかの社会問題を顕在化させるに至っているが、重要な点 ~~として~~ は、ゲイバーらは問題発見や質的調査のための方法としてこの Probe を提案しているわけではないということだ。 **に近づく**

この背景を考えるためには、同時代におけるユーザー中心デザイン (User-Centered Design: UCD) という動向において、製品設計においてユーザーの抱えるニーズや問題をユーザー本人が正しく把握していたり表現できているわけではない ~~という状態~~ **へ** のアプローチのひとつとして、文化人類学者が一定の功績をなしてきたこと ~~を~~ を理解しておく必要がある。代表的な例としては 1984 年の「プランと状況的行為」で人類学者のルーシー・サッチマンが Xerox PARC において、自動コピー機の利用を支援するシステムの実際の利用状況をエスノメソドロジーの方法論で分析した研究が挙げられる (A. サッチマン 1999)。エスノメソドロジーと **へ**

は概念化されていないレベルでの人間の慣習的行動を会話分析などを用いて明らかにしようとする学問である(サッチマンの研究では映像記録も用いている)。サッチマンは人間の行為が、サイバネティクスの慣習で考えられている、入出力を持った静的システムとして常に計画された行動をとっているわけではなく、常に状況に依存した行動(Situated Action)を取っていることを示した。この研究は、デザイナーが意図した通りユーザーが行動してくれることなどないという事実と、実際の利用とかけ離れた環境における実験ではなく、実際の利用の現場を事細かに分析することの重要性という2つの意味で大きな影響を与えた。1980~1990年代にはPARCや、Apple、Intelのようなコンピューター技術を開発する企業がサッチマンのように人類学者を招くもしくは雇うことで、ユーザーのより深い理解を目指す流れがあった。

そうした意味で Cultural Probe は、実験環境とは異なる形でユーザーが日常的に抱える潜在的な問題を把握するための便利なツールとして(特にヨーロッパ圏)のデザイン研究で広く普及した。しかし、コスキネンらが指摘するように、Probeはあるコミュニティを理解するためのツールというよりも、どちらかといえばよりプライベートな贈り物や手紙のやり取りに近いものであり、あるコミュニティへの介入、あるいは関係を結ぶ側面が十分に理解されていない(Koskinen et al. 2011, p41)。実際、Cultural Probeは1960年代のダダやシチュアシオニストのような、ギャラリーや美術館のような既存の発表の場に留まらない芸術表現を模索した動向に影響を受けていることや、The Pillowのような議論を喚起するデザインの例との関係性を鑑みても、ゲイバーらの狙いの1つはデザイナーが物を作ることによって社会に介入していることに自覚的なことであり、水面に波紋を全く起こさなまま針を落とすことなど不可能であるように客観的なユーザーの理解ができないのであれば、むしろ積極的にユーザーと呼ばれていた人たちの生活の中へ介入していくことを意識的に行うべきだという主張だったと解釈できる。

2.4 音楽とコンピューティングにおける RtD の受容

(ここはちょっと繋がりを直す、もしくは4章へ移動) *この場所ではいいように思いつく*

Research through Design もまた、デザイン思考に起きた方法論としての固定化と似たように、結果的にコンピューター科学という領域における、研究における問題設定とその評価のためのフレームワーク程度に見積もられてしまっており実態は依然としてデザインサイエンス的アプローチと大差ないものも生んできている。特にそれはデザインリサーチ的手法が一般的に受容されていない周縁的分野—例えば我々のような音楽とコンピューティングにおける研究において現れやすい。

例として、我々の研究対象である音楽のためのプログラミング言語研究における、西野による LC 言語の設計を RtD の側面から分析した論文取り上げよう。西野は音楽プログラミング言語設計という行為が、音声合成の手法の確立などの研究と比べれば、実用的なツールとしての目的意識と評価という側面が強く、知への貢献という側面が十分に取上げられていないという問題意識をツィーマンらによる RtD の文献を参照しつつ提起する。西野が開発する LC 言語は、microsound という、音声データを数ミリ秒単位に切り刻み再構成するような音声合成の手法(Roads 2004)が既存の言語では十分に表現できないという具体的事象から、mostly-strongly-timed language という新しい音楽プログラミング言語における概念の創出ができるといふ具体例を示している(Nishino 2012)。しかしながらこの研究ではツィーマンが RtD の解説で多用していた「世界を現状から好ましい(preferred)状態へと変化させる」という部分が強調されすぎて、なぜローズが提案した microsound という手法がコンピューター音楽において重要なのか、という未来の音楽制作のビジョンが所与のものとされている。この Preferred State という用語はサイモンのデザインサイエンスから部分的に肯定できる部分として用いられたものであり(Koskinen2011)、ツィーマンが意図していたところはより自己反映性への自覚であった。ショーンの「省察的实践」(Shaun1973)に触れていたように、デザインサイエンスと異なるアプローチとしての RtD とは、はじめから問題が設定されているのではなく、常に仮定の問題からスタートしつつも、作る過程で問題が浮かび上がる＝自己に世界が反映され、自己の行動が世界に反

映されるループに自覚的になることがキーポイントであり、好ましい世界の状態とは何かについての反芻が欠けてはならなかったはずだ。そういう意味では西野の研究は主張の主体性が欠落しており、単なる問題解決のための手段としてのコンピューター音楽プログラミング言語の開発にしかになっていないと言わざるを得ない。こうした例に限らず、その語の浸透に比して、RtD は研究プログラム（どのような価値基準をもとに研究を進めるか）の議論から研究メソドロジー（どのような手段をもって研究を進めるべきか）の用語として受け取られることも増え、その語が指すものの焦点がぼやけてしまった。

2.5 歴史を作り直すデザイン-メディア考古学とスペキュラティブ・デザイン、批判的奇譚づくり

歴史的出来事は、この「人間的コンテキスト」の中で生成し、増殖し、変容し、さらには忘却されもする。端的に言えば「過去は変化する」のであり、逆説的な響きを弱めれば、過去の出来事は新たな「物語行為」に応じて修正され、再編成されるのである。(野家 1990, p11)

さて、Research through Design のアプローチは例えばクリティカル・デザインやスペキュラティブ・デザインの文脈ではサイエンス・フィクションから影響を受けてきたように、作った物を利用している状況を映像作品として提示するなど、物語を有効活用してきたという特徴も挙げられる。こうした物語のデザインへの活用は特にスペキュラティブ・デザインでは“The Pillow”のよう^ら、未来のありえるかもしれない技術の姿への想像力を喚起するためのツールとして人工物を用いることが多い。しかしサイエンス・フィクションは未来の姿を想像するだけでなく、過去への探索をきっかけとして、例えば電気技術ではなく蒸気機関が極端に発達したスチームパンクのような、あり得たかもしれない現在の姿を描き出すという立場も考えられる。デザイン研究者のジェームズ・オージャーはこの違いを図 2.4 のように思索的未来 (Speculative Futures) とあり得たかもしれない現在 (Alternative Present) という違いとして表している (Auger 2010)。

の (prop)

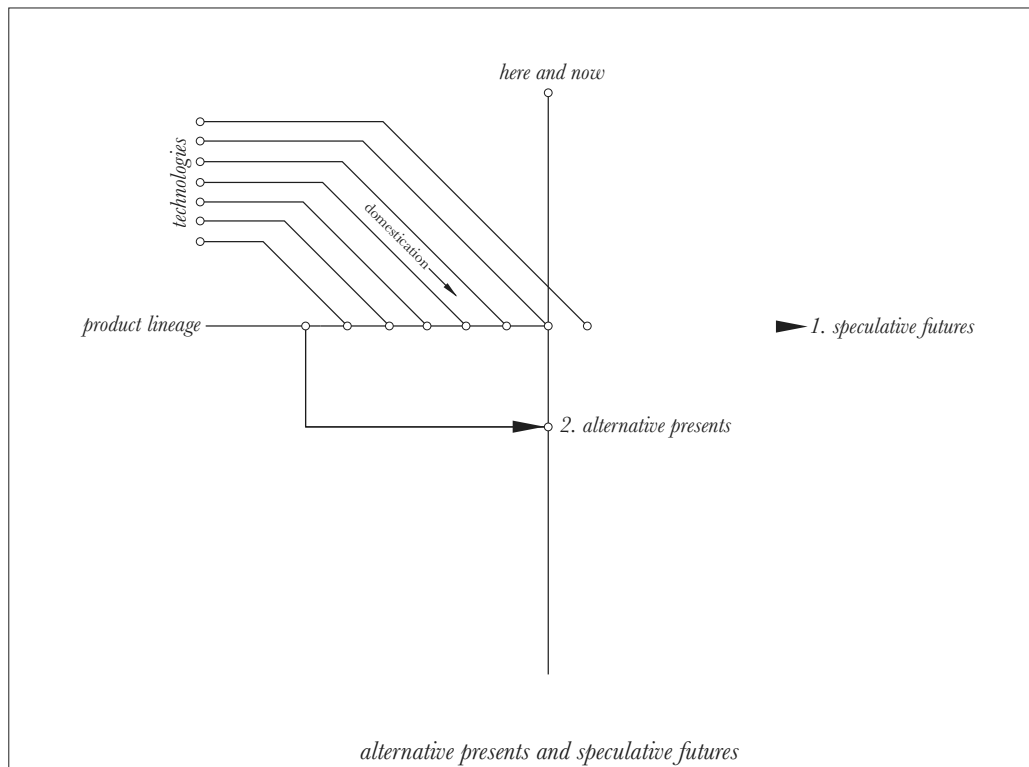


図 2.4: (Auger 2010) より、Alternative Present と Speculative Futures の概念を表した図。

オージャーが指摘するように、未来を思索するデザインは時に現実性から離れすぎることによって人々の共感を得られなかったり、逆に単に物語を作る者の現在の現実に対するステレオタイプの認識を強調するだけにしかかなりえない危うさがある（いったい何人のデザイナーが 2019 年以降のパンデミックの姿を想像できたのだろうか？）。実際、フィクションを用いた未来の思索とは期待の社会学の説明（図 2.3）における 2. 約束作りのプロセスそのままとも解釈でき、結局は未来予測を作るという建前で実際には無意識的に社会的要請に迎合しているだけという状態にもなりかねない。

その点において、過去に参照点をおいたプロジェクトは現実とかけ離れていても、逆説的になぜ現在はそのようなテクノロジーの姿に我々は至らなかったのかという問いの機能が保たれ続ける。

私が本研究で指向するのは Research through Design の中でも、このような、過去へのリサーチから異なる歴史を想像するためのデザインをさらに押し進めた立場—異なる歴史を想像するだけでなく、創造するという立場である。歴史を創造するとは一見事実の改竄のような不誠実な印象を与えるかもしれない。しかしクーンのパラダイム論以降、科学論が反証可能な命題への固執をやめたように、絶対的な歴史的事実というものもまた存在しない。

それどころか、我々はテキストに限らず、今日であれば映像、音声、そしてウェブサイトのような（プリミティブなものも含めて）様々なテクノロジーを通じて現実や歴史の認識を形作る。いわば無数の過去のデータの集合が我々の現実を生み出しているのである。であれば、過去存在はしていたが着目されてこなかったテクノロジー、あるいは時代遅れとされ注目されることなくなったメディアテクノロジーを、現代において作り直すことで我々の歴史や現実の認識自体を作り替えることもまた可能であり、単一的な方向へ進む技術発展に異なる可能性を見出すための手段となりうるはずだ。

本章最後のトピックとして、本研究のアプローチ、実践を通じて歴史を作り直すデザインの参照点として、

アーカイヴ

近年のインタラクショナルデザインの分野からロスナーの批判的奇譚づくり (Critical Fabulation) と、メディア論における 1 つのアプローチ、メディア考古学をインタラクショナルデザインのアプローチと掛け合わせた事例を紹介し、筆者の過去の作品製作と本研究の連続性についても触れることにする。

2.5.1 ロスナーの「批判的奇譚づくり」

ワシントン大学の、デザインという領域における編み物やキルティングのような手工芸との関わりを研究してきたダニエラ・K・ロスナーは、RtD も含めたこれまでのデザインの文脈と異なる方法論として、「批判的奇譚づくり (Critical Fabulation)*¹³」という考え方を提唱している。

ロスナーはデザイン・サイエンス、デザイン思考、RtD の歴史的な文脈を振り返った上で、既存のデザイン研究のパラダイムに浸透する 4 つの考え方を批判した。1 つ目は普遍主義 (Universalism) のようなすべての問題に万能に対応できるような解決策を提示しようとするものであり、これはデザイン・サイエンスの運動から意地悪な問題の提起によって否定されてきたものと対応する。2 つ目は客観主義 (Objectivism)、すなわちデザインを使う対象を客観的对象として決して影響を与えることなく観察する、いわば神の目線で捉えることで問題を洗い出そうとする立場だ。これはデザインサイエンスの時代における考え方とも一致するし、デザイン思考的、ユーザー中心デザインにおいてもユーザーを客観的に理解可能な対象として捉える見方は続いてきた。3 つ目は問題解決主義 (Solutionism) である。これは、デザイン思考からクリティカル・デザインのような問題提起をするためのデザインという転換で否定されてきたものだ。ここまではこれまでの RtD までのデザインの潮流の変化としてすでに説明してきたことだ。だがロスナーはさらに、Research through Design も含めた既存のデザイン文化には、技術を使ったり作ったりする主体をこれまでのパラダイムで言えば、ユーザーやデザイナーという形で (仮に参加型のデザインのような性質を持っていたとしても) あくまで個人の集合として捉えてくる個人主義 (Individualism) が強く根付いていると指摘する。

(この 4 つの批判に関しては図に入れたり冒頭からの説明に入れてしまったほうがわかりやすいかも)

ロスナーはこの個人主義の存在をフェミニズム的視点と動機から説明する。例に挙げるのは 1960 年代冷戦下のアメリカにおける世界初の有人宇宙飛行を実現したアポロ 8 号において、ロケット制御のためのコンピューター：アポロ・ガイダンス・コンピューター (AGC) のプログラムを記憶したメモリの 1 種である、磁気コアメモリ (より正確には、コアロープメモリと呼ばれるもの) の製作である。磁気コアメモリはコアと呼ばれるリング状の磁性体を通した電線に電流を流し、電流が引き起こす磁場によってコアの磁気極性を変えることでバイナリデータを保存する記憶装置の一種である。磁気コアメモリは当時の技術環境においては、物理的に小さく、ロケットの中という物理的振動や熱が激しい環境においても比較的安定して動作するため、主記憶装置として採用された。この磁気コアメモリは、なるべく集積密度を上げるために非常に小さなコアに細いワイヤを通して作られるが、その製造方法は最終的に完全自動化されることはなく、特にアポロ計画のためには服飾工場や時計の製造工場に働く女性工員が生産を担っていた (Shrift 2019)。

有人 (Manned) 飛行で、初の月面往復という革新的事業はこれら多くの無名の女性による集団的創造プロセスなくしては成立し得なかったが、こうした作業は AGC のアーキテクチャを作ったり、あるいは実際にロケットに乗ったりした人と比べると歴史の記述には残りづらい。それは、手作業の中にも、コンピューターアーキテクチャを設計する作業と同じく存在しているはずの様々な知識が、身体的技能と結びつき言語化されないレベルでのモーフレイリングの分類で言えば共有可能でない知、Research for Art/Design と呼ばれていた部分が歴史の中には記述されえないからである。

ロスナーらはこうしたテクノロジーにまつわるヒロイックな言説に基づいて構築された個人主義的歴史像

*13 “Critical Fabulation” の訳出は、この語の起源、ダナ・ハラウェイの SF という語の読み替えのひとつである “Speculative Fabulation” の訳を参考にした。猪口による論考では “思弁的寓話小説” (猪口 2018)、逆巻による 2016 年のハラウェイのインタビューの翻訳では “思弁的奇譚” とされており (Haraway and Sarah 2019)、後者を参考にしつつも、ロスナーの文献内ではその奇譚を作る、語るという動詞の意味合い (Fabulating) が強調されているため “批判的奇譚づくり” とした。

~~が今日のデザインにも強く根付いていることを~~^{示すために}~~ひとつの動機として~~、磁気コアメモリを導電糸やビーズを用いてテキストスタイルとして作成する“Making Core Memory”ワークショップを作り上げた。4x4のビーズにはデータが実際に保存できるようになっており、ワークショップの中ではそのデータをもとに音楽を鳴らしたり、テキストデータとして保存されたデータを Twitter に投稿するなどの応用を交えてその反応を記録した (Daniela K. Rosner et al. 2018)。

ロスナーらのプロジェクトはダンやゲイバーらのクリティカル・デザイン/スペキュラティブデザイン同様に、直接的に役に立つプロダクトを作っているのではなく、デザインされたものを通じて問いかけを行うことを目的としているという点では共通している。しかし、その問いかけはありえるかもしれない未来の姿について夢想するためのものではなく、これまで歴史の中には記述することができなかったコアメモリを作るという手作業を通じたコミュニケーションを引き起こすことで、あったはずなのに認識できていなかった、輝かしい個人の功績ではない、無数の人々の関係性の中に現れる技術的貢献、もしくは、記述されな^{かっ}ために存在しないことになっていた歴史像を浮かび上がらせる試みであるという点で決定的な違いがある。

2.5.2 メディア考古学

また、ロスナーとは異なる視点で^{*14}過去への積極的な調査を基に人工物のデザインへのヒントを得るために、メディア考古学と呼ばれる、既に使われなくなったり廃れたメディア装置への探求を行うアプローチを援用してきた者もいる。

メディア考古学とはそれ自体が明確な定義を与えられることを嫌うような性質を持っているため注意を要する用語ではあるが、ここでの定義は 80 年代や 90 年代以降のエルキ・フータモのようなメディア研究者が試みてきた、映画や映像メディアの正史には普通記述されない、万華鏡のような装置に最新テクノロジーへの熱狂の繰り返しの萌芽を見出す (Huutamo2014)(大久保 2021, p289) ような、歴史の中に埋没したメディアを掘り返すことで所与のものとされている歴史に別の物語を与えるアプローチとしておこう。

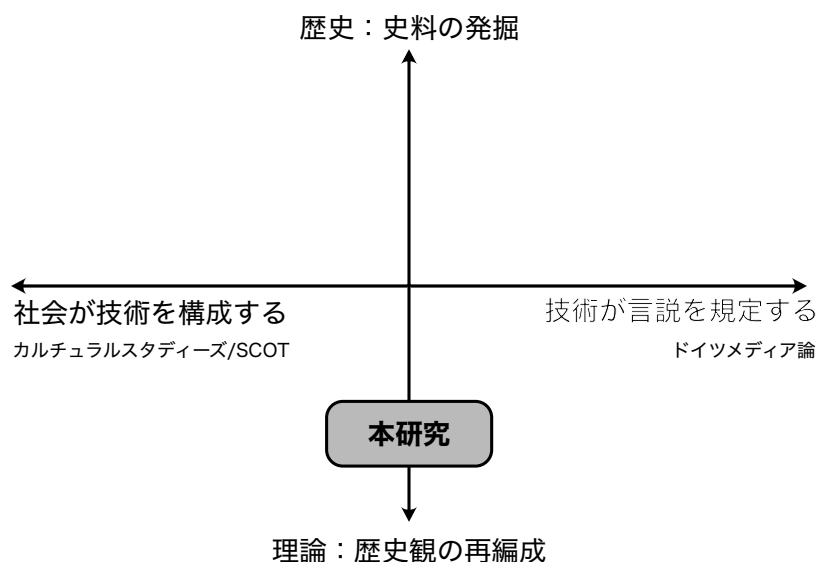


図 2.5: (大久保 2021) をもとに作成した、メディア考古学の立場の広がりを示した図。

^{*14} Rosner は Making Core Memory プロジェクトや Critical Fabulation の説明においてメディア考古学にはほとんど言及していないが、CHI2018 での発表の中では考察の中でユシー・パリッカの名前を挙げ限定的に関連性を見出している (Daniela K. Rosner et al. 2018)。

メディア考古学のアプローチに存在する微妙なスペクトラムを大久保はメディア環境を形成する要因を「社会-技術」のどちらで捉えるか、学術的貢献の形として重視するのが「歴史-理論」のどちらに重きを置くかという2つの視点で整理しており、この視点はどちらもメディア考古学に共通する思想的背景となっているフーコーの「知の考古学」の解釈に大きく影響されるとしている(大久保 2021, p285~289)。図 2.5 にこの立場の広がりの概念を示した。横軸である「社会-技術」の軸では、作られたメディアテクノロジーが既存の社会的言説の影響のもとに生まれると考えるか、メディアテクノロジー自体がテキストではない形で(たとえばキットラーの例を借りれば、レコードや映画、タイプライター(Kittler1999)が)思考や言説を形成すると考える違いがある。この中でも「社会」寄りのアプローチとしては、キャロリン・マーヴィンの「古いメディアが新しかった時」に代表される、STSにおける社会構築主義的立場の1つの社会の技術的構成(Social Construction of Technology:SCOT)という領域とも重なることは大久保も指摘しているが、SCOT的研究の中にはそれほど埋没したメディアの掘り下げといった側面を含まず、あくまでテクノロジーが社会的要請によって決定されるという、アンチ技術決定論的側面の方が大きいものも含まれる^{*15}。

一方縦軸の「歴史-理論」の軸では、これまで取り上げられなかったメディア装置の一次史料の収集と記述に重きを置く、より歴史学的なアプローチか、既存の史料の中から選択し直すことで歴史的記述を再考することに重きを置いているとされる(もっともこれらのアプローチの違いは大久保も強調しているように、対立しているものではなく論者によって重点を置く場所が違うということで、多様なアプローチがあるメディア考古学という領域の見取り図として機能はするという程度のものだと考えた方がいい)。

メディア考古学はメディアの歴史や、メディアが我々の身体や近くに及ぼす影響を考察するための道具としての側面が大きいものの、ニューメディアアートやインタラクショナルデザインのような、作品制作のアプローチのひとつとして古びたメディアに着目することは、メディア考古学という概念が立ち上がる前から行われ続けている。岩井俊雄のゾートロープ的装置を拡張する「時間層」シリーズのような作品群や、ポール・デマリーニスの「エジソン効果」の中で陶器の壺に刻まれた溝をレコードとして扱い音を読み出す試みのような芸術家の取り組みをフタモがメディア考古学的実践のひとつとして位置付けていたり(Huutamo2014)、パリッカとハーツがサーキット・ベンディングという電子回路を意図的に壊すことで楽器として用いるアプローチを同様に分析している(Parikka2014)ように、メディア装置自体を作る、あるいは既存のメディア装置の用途をねじ曲げるような芸術実践は後付的にメディア考古学的と評されることもある。デマリーニスは作品制作中のリサーチにおいて過去の期限の切れた特許資料の中からアイデアを得ていたり、明確に過去のメディアに対するリサーチを行っているが、サーキット・ベンディングのようなアプローチは実践している者からすれば面白い音が鳴るように回路を改造しているだけという側面もあり、実践者自身の過去に対する意識はメディア考古学的実践と位置付けられるものの中にも広がりがあることは注意を要する。

その上で近年では、メディア考古学の方法論を作品製作に明確に自覚的に用いるプロジェクトが、たとえば第1章で既に例示した、情報科学芸術大学院大学(IAMAS)において城一裕が中心となって活動した「車輪の再発明」プロジェクト(城 2016)や、NIMEのような楽器制作の方法論としてもレプリの「Cembalo Scrivano」(Lepri and Mcpherson 2018)を例として現れてきている。本稿ではこの自覚的にメディア考古学を作品制作に導入する立場をメディア考古学的制作と呼ぶ。*このような事例がい*

何より筆者自身が意図的にメディア考古学を作品制作の方法論として用いてきた一人である。例えば2016年の磁気コアメモリよりもさらに古い1950年代の電子計算機黎明期に用いられた、音響遅延線メモリと呼ばれる音波のフィードバックループを用いてデジタルデータを保存する記憶装置を空間に開かれた形で再構築するインスタレーション作品「遅れ/送れ | post/past」では、物質的狀態を固定しないままにデータを保存する音響遅延線メモリを物理的に分離された2台の通信装置として構成することで、通信と記録の不可分性を顕

*15 SCOTの中で本研究と関連するものとしてはたとえばピンチとトロッコによるモーグ・シンセサイザーの歴史的記述を行った「Analog Days」(Pinch2014)や、電子楽器用プロトコルMIDIの規格成立過程を追いかけたディドックの博士研究及び「Mad Skills」(Diduck2018)のような研究が挙げられる。

在化させた (松浦 2016)。

別の例を挙げれば、筆者が 2018 年に、冒頭説明した SFPC 滞在中に作った「Electronic Delay Time Automatic Calculator(EDTAC)」がある。EDTAC は、現在使われているコンピュータは音楽のように時間に関係する処理を日常的に行っているにもかかわらず、その理論的基盤には時間の概念が一切含まれていないことに着目して、「はじめてからコンピュータの設計の中に時間の概念が組み込まれていたらどうなっているか」という問いを提出するために作られた、極めてプリミティブな計算機である (松浦 2019; 松浦 and 城 2019b)。EDTAC はプログラムされた通りに時間を分割しクリック音を鳴らし続けるだけの機能しか持っていないが、ちょうど世界最初の電子計算機である ENIAC がケーブルを配線直すことで物理的にプログラムを構成していたように、抵抗の内蔵されたケーブルと光ファイバーという物質的接続によって分割する時間間隔が決定され、また回路彫刻の手で作られたものとしての存在が全く異なる形式の計算機の可能性を問いかけるようになっている。

(作品の写真入れる?) ... 入っちゃおう!!

こうしたメディア考古学的実践を今日のデザイン研究の中に位置付ける際に注意を要し、かつ自分自身も不満を残し続けてきたのがメディア考古学的制作における作者や作品といった単位の問題である。クリティカル・デザインのような未来についての問いかけを行うデザインは、現実的には役に立ちそうもないものだからこそ、その発表の場としてギャラリーや美術館という場所が用いられることがある。このことはメディア考古学的制作をデザインのアプローチとして用いる場合にも多くの場合共通する点だが、2つの点で問題を招いてしまう。ひとつは Cultural Probe で問題意識に据えられていたように、ギャラリーのようなあらかじめ美術を想定して作られた場にデザインした物を置くことによって、それは人々の日常生活から離れて、単に鑑賞の対象となってしまうことだ。もうひとつはロスナーが批判した個人主義のように、デザインされた物とデザイナー個人の記名性が過剰に結びついてしまい、デザインの内容そのものよりも誰がどういった内容の作品制作を続けているかという点に重きがおかれ、社会の中におけるデザイン運動の中の流れの一つとして捉えることが難しくなってしまうことだ。

メディア研究者の秋吉康晴は「車輪の再発明」プロジェクトの発端となった、レーザーカッターを用いてレコードの溝をデジタルデータから直接生成する城の作品制作の分析を通じて、メディア考古学的作品制作の既存の言説が過度に作者性と結び付けられていることを指摘している (秋吉 2020)。例えばメディア考古学的作品制作を行う代表例としてフータモが例に挙げるポール・デマリーニスの製作の背景には 1960 年代、つまりサイバネティクス全盛期における、アメリカ西海岸のヒッピームーブメントと結びついた DIY テクノカルチャーの存在があった。パリッカラが着目したメディア考古学としてのサーキット・ベンディングも、リード・ガザラのような、ムーブメントの中での代表的存在が過度に照らし出されている一方で、実際には個人レベルでの技法や機材の共有という、集団的創造プロセスの側面が薄められてしまっている。その一方で城の作品制作は、技法自体の公開やドキュメント化を通じて、よりどんな個人でもその制作の再現やさらなる飛躍ができる余地が残されている。秋吉はつまり、フータモやパリッカラはアーティストやサーキットベンダーをいたずらに特別視しすぎており、歴史に個人としては記述されない DIY の実践家との接続を結果的に断ち切っていることを指摘している。

この視点を踏まえると、「車輪の再発明」プロジェクトにおける作者性はより特異なバランスの保ち方をしている。プロジェクトの中では、参加した学生や教員によって多数の「作品」に相当するものが作られ、それぞれにタイトルもつけられているのだが、それだけではなく制作の中で共有して使われた技法にもタイトルがついている。例としては、技法の名前として「予め吹きこまれた音響のない (もしくはある) レコード」その実例として城による、レーザーカッターで任意の周波数の溝を刻んだ扇型のアクリル板を組み合わせることでレコードをステップシーケンサーのように扱う「断片化された音楽」、あるいは同じ技法でも、レコードの溝の上に坂道が形成され、針がスキージャンプのように飛び出すことでランダムに次に再生される溝が変わるといふ、大島による「飛び出せ↑レコード」のような多数のバリエーションが存在する。城はこの技法に名前をつ

ける行為を「作品とツールの間の中間層」を、技法という名のもとに提案することを試みたと説明する。ロスナーの個人主義批判を踏まえれば、メディア考古学的制作に内在する問題点とは、アーティストの特権視するだけでなく、作品（や何らかの DIYムーブメントなど）を特定の個人的アイデンティティに結び付け過ぎてしまうということでもあると理解できるだろう。そして技法に名前をつける行為は、ギャラリーのような場所で展示されることを踏まえた上で作られたものの集団的オーナーシップを表明するための役割を持っていたと捉えることができる。

この作者性の強調と創造プロセスの個人主義的解釈はもちろん、作られたものを分析する側の問題でもあるが、制作をする側にも、特にギャラリーや美術館のような展示というフォーマットを選ぶことによって個人主義的視点を内面化してしまうリスクがつきまとう。

それゆえ例えばロスナーのようなワークショップの形式であったり、「車輪の再発明」のような、技法といった単位での作られたものの提示の仕方の模索はメディア考古学的制作におけるひとつの課題である。本研究における音楽のためのプログラミング言語の開発という行為は、それ自体は道具という、作品以前の存在であるがゆえに、典型的なデザインや工学の文脈においては問題解決主義的視点で捉えられてしまうし、かといって問題解決主義を否定する、議論を起こすことを目的としたクリティカルデザインのプロジェクトのように美術館やギャラリーに置かれることもまたそぐわない。

本研究がプログラミング言語の実装を通じて提示するのは、既存のデザインの対象のように日常の中で使われる道具を、仮に実用を意識して作りはじめはするものの、その最終的な目的は必ずしも実用的であるのではなく、制作やその道具を通じてこれまでの技術やメディア環境に対する認識の変化を促すこと、もしくは制作過程においてデザイナー自身の認識が変化することを目的とするという研究のあり方である。

（音楽土木工学という学問における研究のあり方である位置付け、一般的な高額のなアプローチとは随分違うのでトランジションデザイン）

2.6 小括

ここまでの議論を振り返り、本研究の立脚点を改めて確認しよう。

本章ではまず音楽土木工学という学問を、音楽のためのプログラミング言語を実装する過程で描き出すという研究のあり方を、主にデザインリサーチの変遷を追いかけることで位置付けた。

デザインリサーチは、大きく分けて 1960 年代においてサイバネティクスの影響を受けた、デザインサイエンスにおける科学的、普遍的問題解決手法の探求、1970 年代以降のデザイン思考のような、個人的省察による問題解決、1990 年代以降の Research through Design における問題解決から問題提起、客観的観察から積極的介入といった変遷を見ることができた。

中でも RtD の文脈を引き継ぎ、未来よりも過去に着目する、ありえるかもしれない現在技術・メディア環境を想起させるメディア考古学的制作、これまで歴史的事実の中に含まれなかった身体化された知や周縁化された社会集団に光を当てることで歴史認識を作り替える批判的奇譚づくりといった事例を紹介し、過去の言説や、通常メディア表現には無関係とされるようなインフラストラクチャに着目することによって、メディアテクノロジーの進むべき方向性に異なる可能性を見出すための契機としてのデザイン行為として本研究を位置付けた。

このデザインの行為の変化に通底する視点として重要なのが自己反映性（Reflexivity）への自覚である。道具や作品、人工物を作り出すことは世界や社会に対して何らかの影響を与える、常に中立ではない政治的行動である。ユーザーを神の目線で完璧に客観的に観察することは不可能である。

ここで RtD の言葉が作られる経緯となったフレイリングのアート、デザインにおけるリサーチ行為のカテゴリ分けを再訪すると、この区分は実際のところカテゴリ分けというよりも自己反映性の強さの段階的なものであった。フレイリングは into、through、for の違いを次のように言い換えている。

- INTO: How can I tell that I think till I see what I say?(どうして自分が考えてることを、自分が言うのを観測する前に知ることができるだろうか?)
- THROUGH: How can I tell what I think till I see what I make and do?(どうして自分が考えてることを、自分が作ったものを観測する前に知ることができるだろうか?)
- FOR: How can I tell what I am till I see what I make and do? (どうして自分とは何なのかということを、自分が作ったものを観測する前に知ることができるだろうか?)*¹⁶

フレイリングがいばらの道 (thorny) かつ最も探求すべき場所していたのは実は through よりも for (自らの体験に基づき身体化された知) のことであった。ものを作る過程で現実が構成され、社会が構成され、また自らが構成される。こうした知の形をフレイリングは共有可能な知としてカウントしていなかったが、それはメディア考古学の視点を導入すれば、もしかすると単にテキストの形を持たず、別のメディアの中に埋め込まれた言説として取り出せるものもあるかもしれない。あるいはロスナーの指摘を踏まえれば集団の中で共有はされていたが歴史の中に記述されえなかったもので、実際に手を動かしてみることで、改めて身体化された知として復活させることができるかもしれないものではないだろうか。

私はこの視点に基づいて、実際には音楽のためのプログラミング言語というデザイン対象に取り組むことになる。それは、音楽のためのプログラミング言語を作る、あるいは音楽のためにプログラミング言語を批判的態度を持って作ることによって、音楽家がコンピューターを用いて音楽を作る時の、音楽家の認識論を明確化させることができるのではないか、あるいは、音楽が流通する社会における音楽を下支えするインフラストラクチャの政治性を明確にできるのではないかという観点からの試みである。次の章では、この「なぜ音楽のためのプログラミング言語という対象なのか」という理由を、メディアとしてのコンピューターの扱われ方の歴史の変遷を辿ることで明確にしていく。

持つ

いて

*¹⁶ この原典 (INTO に対応するものがオリジナル) はフレイリングによれば小説家 E.M. Forster のおばが彼に言ったこととされている有名なフレーズだが、実際のところ参照元がはっきりしないことでよく知られている。1926 年の Graham Wallas による “The Art of Thought” が Forster よりも早いものとして挙げられているが、この文のような考え方は当時のヴィトゲンシュタインの論理哲学論考に代表される哲学の言語論的転回 (言語が現実世界を構成すると考える哲学の運動) を反映したものだと言える。
<https://quoteinvestigator.com/2019/12/11/know-say/> (2021-12-12 閲覧)