OptiGuide+: An Interactive Recommender System for *Virtual Things*

Tahani Almanie[a], Xu Han[b] Bhavana Posani[c] and Alexander Brodsky[d]
Department of Computer Science, George Mason University, 4400 University Drive, Fairfax, U.S.A.
{*talmanie, xhan21, bposani, brodsky*}*@gmu.edu*

Keywords: Interactive Recommender System, Multi-Objective, Pareto-Optimal, Decision Guidance, Virtual Things, Smart Manufacturing, Optimization.

Abstract: This paper is concerned with markets of *Virtual Things*, which are parameterized specifications of products or services that can be realized and delivered on demand. Reported in this paper is the design and development of *OptiGuide+*, an interactive recommender system designed to guide users in choosing and optimally instantiating *Virtual Things*. *OptiGuide+* enhances decision-making by dynamically integrating user preferences with multi-objective optimization. It supports (1) real-time interactive utility extraction from user-selected preferences on Pareto-optimal alternatives; (2) seamless integration of *virtual things'* specifications; (3) cross-platform web-based deployment capability; and (4) an intuitive user interface for visualizing trade-offs to explore Pareto-optimal alternatives. *OptiGuide+* is unique in its ability to leverage utility-driven decision guidance methodology for markets of parameterized products and services and demonstrate it in real-world applications.

1 INTRODUCTION

In this paper, we report on the design and development of *OptiGuide+*: an interactive recommender system to guide users in choosing and optimally instantiating *Virtual Things* (VT's). *Virtual Things* are parameterized specifications of products, services, or anything of value, associated with analytic models that describe the consumer-interesting characteristics, which can be instantiated and fulfilled on demand (Han and Brodsky, 2022).

To illustrate the concept, consider an example of a virtual bike. Unlike a physical, off-the-shelf product, a virtual bike is defined by a parametric design specification, against which it can be manufactured by a vendor. For a virtual bike, a set of parameters may involve frame material, bike geometry, wheels' size, and drivetrain characteristics. VT is also associated with a set of *constraints* on the parameters and user requirements, such as tire size for riding on snow and weight limits, as well as *objectives*, i.e., key performance indicators (KPIs) for the bike, such as total weight, cost, speed, and feasibility for specific conditions.

[a] https://orcid.org/0000-0002-6017-4841
[b] https://orcid.org/0000-0002-3347-3627
[c] https://orcid.org/0009-0003-4876-3787
[d] https://orcid.org/0000-0002-0312-2105

Consumers face the challenge of selecting the most suitable instantiation of a VT, e.g., a bike capable of riding on snow and optimized according to consumer preferences that balance objectives like cost, performance, and environmental impact. This problem involves extracting a user utility function from (possibly competing) user's objectives and using the extracted utility to recommend an optimal instantiation of the VT.

When repositories (or markets) of VTs are available to consumers, the problem they face is how to choose a particular (optimal) instantiation of one VT out of the set of VTs in the repository. E.g., an instantiation of a virtual bike that can ride on snow, or a procurement package. We are interested in *optimal* VT instances, in terms of the user's utility function, which can be described in terms of a number of KPIs or objectives, e.g., cost, quality and carbon emissions.

Thus, the problem involves two interrelated tasks: how to extract a user utility as a function of KPIs through a short interactive session with the user; and how to choose a specific instantiation of a *Virtual Thing* that optimizes the utility. *OptiGuide+* is designed to address this problem.

There has been prior work on decision guidance systems for *Markets of Virtual Things*, including (Han and Brodsky, 2022; Han and Brodsky, 2023). This is based on the work on Decision Guidance Systems

Almanie, T., Han, X., Posani, B. and Brodsky, A.
OptiGuide+: An Interactive Recommender System for Virtual Things.
DOI: 10.5220/0013470000003929
In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 1*, pages 642-653
ISBN: 978-989-758-749-8; ISSN: 2184-4992

(Brodsky et al., 2016; Nachawati et al., 2017), which are a specialized class of decision support systems that elicit human knowledge and guide the users to make the best possible decisions. Paired with parametric design for novel products (Gingold et al., 2009; Yu et al., 2011; LaToza et al., 2013; Shin et al., 2017; Liu et al., 2019), decision guidance systems are broadly applied to smart manufacturing (Menascé et al., 2015), including product analysis, manufacturing process and procedure optimization (Egge et al., 2013; Shao et al., 2017; Shao et al., 2018; Brodsky et al., 2019; Li et al., 2020), and manufacturing energy and waste reduction (Shao et al., 2011; Griffiths et al., 2016).

The concept of *Virtual Things* (VTs) has been proposed and explored in (Han and Brodsky, 2024) as part of the *DG-ViTh* system, which aims to facilitate optimization and decision-making for parameterized specifications of manufactured products and services. However, *DG-ViTh* lacks interactive mechanisms for dynamic utility extraction, relying instead on predefined utility functions, which limits its adaptability to consumer-driven markets.

Extensive work has also been done on iterative recommender systems based on multi-objective optimization. *CAPORS* introduced a composite recommendation methodology, emphasizing Pareto-optimal trade-offs for arbitrary criteria (Jeffries and Brodsky, 2017). *CAPORS-IUX* extended this by integrating continuous user interaction to refine recommendations iteratively, capturing individual utilities (Jeffries and Brodsky, 2018). However, real-time optimization posed challenges, particularly in complex domains, and both systems lacked robust domain independence.

OptiGuide further advanced the field by introducing preprocessing algorithms to improve efficiency and incorporating a domain-independent architecture. It facilitates user utility extraction by capturing user interactions, providing real time feedback and optimization through a customizable interface (Almanie and Brodsky, 2024). Nevertheless, *OptiGuide*'s framework was not designed to handle VT-specific artifacts, such as parameters or metrics schemas, preventing its application to dynamic *Virtual Things* Markets. Furthermore, its deployment was limited to standalone systems, restricting its broader applicability, and it did not implement the best so far and improvement functionalities.

Bridging these gaps is the focus of this paper. More specifically, we have developed *OptiGuide+*, an interactive recommender system that guides users in choosing and optimally instantiating *Virtual Things*. *OptiGuide+* combines the strengths of *DG-ViTh* and *OptiGuide* to overcome their respective limitations. The core contribution of this paper is the design and development of *OptiGuide+*. This required a number of specific technical contributions as follows:

- Real-time Interactive Utility Extraction: *OptiGuide+* leverages *OptiGuide*'s iterative user interaction capabilities, enabling users to dynamically define and adjust utility functions based on specific objectives and preferences within the VT framework.
- Seamless VT Integration: *OptiGuide+* incorporates VT-specific artifacts, including parameterized specifications and analytic models, enabling it to operate effectively within the *Market of Virtual Things*.
- Web Deployability: The system enhances *OptiGuide*'s deployment capabilities by making it web-accessible, ensuring broader accessibility and scalability across diverse domains and platforms.
- Enhanced User Interface: The system provides a user-friendly interface for visualizing trade-offs, exploring Pareto-optimal solutions, and finalizing decisions, ensuring an intuitive and satisfying user experience.

The integration of these features makes *OptiGuide+* a unique utility-driven interactive decision guidance system capable of addressing the dual challenge of utility extraction and optimal instantiation of *Virtual Things*.

The rest of this paper details the design, implementation, and demonstration of *OptiGuide+*, highlighting its potential to transform decision-making in markets of parameterized products and services. Section 2 reviews the idea of *Virtual Things* as well as the high-level architecture and functionalities of *DG-ViTh*. Section 3 provides an overview of *OptiGuide+* and demonstrates its use through a *Virtual Things* example. Section 4 covers a detailed functional and architectural design and implementation of *OptiGuide+*. We then demonstrate in Section 5 how *OptiGuide+* works to render optimal VT in a web-deployed environment and conclude with future research directions in Section 6.

2 *Virtual Things*: ARTIFACTS AND FUNCTIONS

In this section, we overview the concept of *Virtual Things* (VT's) artifacts and functions more formally

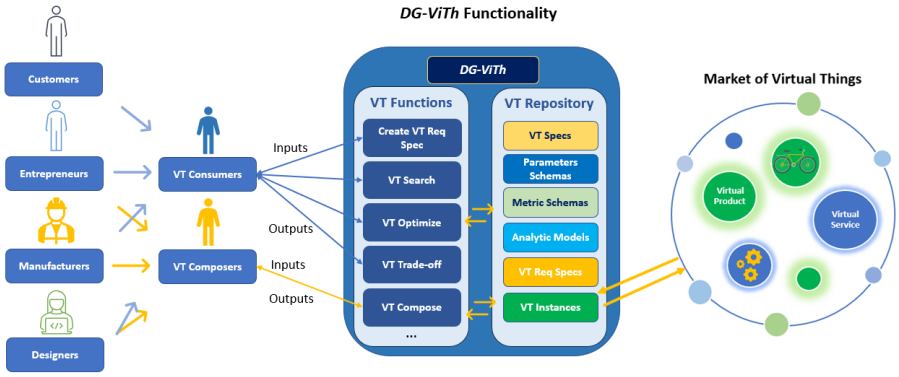

Figure 1: *DG-ViTh* Functionality and User Roles (Han and Brodsky, 2024).

and describe the *DG-ViTh* system. We borrow the description from (Han and Brodsky, 2024).

DG-ViTh (see Figure 1) connects multiple user roles to the functionalities they desire respectively, and acts as the interface and decision guidance engine for the *Market of Virtual Things*. Architecturally, *DG-ViTh* involves the reusable, extendable and modular knowledge base of specs and analytic models in the VT repository, the conventional e-commerce market functions, and the specialized VT functions built on *DGAL* (Brodsky and Luo, 2015) which together power up the *Market of Virtual Things*.

The entrepreneur with innovative ideas can navigate the transition from conceptual ideation, to virtual design and production, and ultimately to concrete market presence, informed and empowered by the decision guidance system of *DG-ViTh*. During the same process, in the whole ecosystem, the capability providers, designers, collaborators, and investors all get accurate, genuine, and timely market information on the reception of the innovative product, so they can use the valuable feedback information to evaluate and further their work and investment. By supporting the creation, searching, and optimization of *Virtual Things*, *DG-ViTh* streamlines the product development workflow and enhances design precision according to user preferences. Its capabilities to construct optimal *Virtual Things* and facilitate information flows in the cloud environment accelerate the realization of innovative ideas.

2.1 Key *Virtual Things* Artifacts

Key artifacts for *Virtual Things* include *VT specs, requirements specs, utility, metrics schemas*, and *parameters schemas*. These artifacts are stored in the repository and used by the *DG-ViTh* system.

2.1.1 vtSpec

The core *Virtual Thing* artifact is a VT design spec (*vtSpec*). Semantically, a *vtSpec* is an analytic model *AM*, which is a (mathematical) function

$$AM : MI \rightarrow MO$$

where

- *MI* is the *AM*'s domain of all possible parameters' inputs, representing all possible instantiations of fixed and controllable parameters relevant to model computation.
- *MO* is the *AM*'s co-domain of all possible model outputs representing computed user-facing metrics and constraints, i.e., all the information that the user needs to judge a VT instance.

Syntactically, *MI* is represented using the notion of the domain associated with a parameters schema (*parametersSchema*), defined formally in (Han and Brodsky, 2024). A *parametersSchema* is a JSON data structure corresponding to *AM*'s input, where all fixed and decision parameters are annotated to indicate their type (e.g., reals, int) as well as optional lower and upper bounds.

Informally, *dom(parametersSchema)* is the set of all JSON structures resulting from replacing parameters' annotations with values within the given bounds from a corresponding domain. Similarly, *MO* is represented using the notion of the domain associated with a *metricsSchema*, also formally defined in (Han and Brodsky, 2024).

The model *AM* is represented syntactically as a Python function that gets, as input, a JSON structure from *MI* and produces, as output, a JSON structure from *MO*. An example of a syntactic representation of a *vtSpec* including its *parametersSchema, metricsSchema* and analytic model *AM*, is shown in Figure 6.

2.1.2 vtReqSpec

VT requirements specification (*vtReqSpec*) is used by users to specify requirements to search for *vtSpecs* or find optimal instances of *vtSpecs*. The semantics of *vtReqSpec* is given by two (mathematical) functions

$$objs : MO \rightarrow O \text{ and } constraints : MO \rightarrow \{T, F\}$$

where

- MO is the domain of *metricsSchema* as previously described (which is also used as a co-domain of *vtSpecs* analytic model).
- O is the *objs*'s co-domain of all possible objectives/KPIs value instantiations according to the objectives schema (*objsSchema*).

While a metrics structure in MO may involve numerous metrics and characteristics of a VT instance, a structure in O holds values for just a (small number) of objectives (also called KPIs), e.g., cost, carbon emission, and manufacturing time. The function *objs* maps the more complex metrics structure in MO into the values for the objectives in O. The Boolean function *constraints* is used to indicate whether a metrics' instance $m \in MO$ is feasible ($constraints(m) = T$) or not ($constraints(m) = F$).

Similar to $MO = dom(metricsSchema)$, the *objs* co-domain $O = dom(objsSchema)$ is defined formally in (Han and Brodsky, 2024). Informally, *objsSchema* is a JSON structure that contains the names of objectives (e.g., cost, carbon emission, manufacturing time), with their values annotated to indicate their type (e.g., real, int) and optional lower and upper bounds. And $dom(objsSchema)$ is the set of all JSON structures resulting from replacing objectives' annotations in *objsSchema* with values within the given bounds from a corresponding domain. An example of *objsSchema* appears in Figure 7 under "objectives" and "schema".

Syntactically, *vtReqSpec* is described as a JSON structure that contains or refers to *metricsSchema*, *objsSchema*, and Python functions *objs* and *constraints* to define the corresponding mathematical functions. It is required and assumed that the Python function *objs* produces an output in O for any input in MO, and that the Python function *constraints* produces a Boolean value T or F for any input in MO. An example of *vtReqSpec* artifact appears in Figure 7.

2.1.3 Utility

If a decision maker would like to find an optimal instantiation of a *vtSpec* in terms of multiple objectives, she needs to specify another artifact - *utility* which describes a linear combination of objectives in the corresponding *vtReqSpec*. The semantics of *utility* (associated with a *vtReqSpec* and its *objsSchema*) is the (mathematical) function

$$U : O \rightarrow \mathbb{R}$$

where O is the domain of *objsSchema* and $\mathbb{R}$ is the set of real numbers. We restrict ourselves to utility functions being linear in objectives. Syntactically, a *utility* artifact includes a *min/max* flag to indicate if a decision maker would like to minimize or maximize it; and the *weights* for each of the objectives in the associated *vtReqSpec* for the computation of a linear combination of the objectives.

2.2 *DG-ViTh* Functions Syntax and Formal Semantics

DG-ViTh functions are a set of functions that specialize in creating, searching, modifying, optimizing, comparing and analyzing the *Virtual Things*. We have defined several key functions with their syntactic signatures and semantics.

2.2.1 vtOptimalInstance(vtSpec, vtReqSpec, Utility)

The semantics of the *vtOptimalInstance* function is to find a VT instance that matches the *vtReqSpec* and is optimal in terms of the user's utility. We have the mathematical functions

$$AM : MI \rightarrow MO$$
$$constraints : MO \rightarrow \{T, F\}$$
$$objs : MO \rightarrow O$$
$$U : O \rightarrow \mathbb{R}$$

corresponding to the semantics of *vtSpec*, *reqSpec* and *utility* in the input. Consider

$$\begin{aligned} \text{ARGMAX} = \underset{mi \in MI}{\text{argmax}} \quad & U(\text{objs}(\text{AM}(mi))) \\ \text{s.t.} \quad & \text{constraints}(\text{AM}(mi)) \wedge \\ & \text{vtMetricBounds}(\text{AM}(mi)) \wedge \\ & \text{reqMetricBounds}(\text{AM}(mi)) \wedge \\ & \text{objsBounds}(\text{objs}(\text{AM}(mi))) \end{aligned}$$

where

- *vtMetricBounds* is the bound constraints of the *vtSpec*'s *metricsSchema*
- *reqMetricBounds* is the bound constraints of the *vtReqSpec*'s *metricsSchema*
- *objsBounds* is the bound constraints of the *vtReqSpec*'s *objsSchema*

To complete the semantics of *vtOptimalInstance*, the output is as follows:

1. If ARGMAX $= \emptyset$, return INFEASIBLE
2. Else, if $(\forall mi \in MI)(\exists mi' \in MI)$ $s.t.\ U(objs(AM(mi'))) > U(objs(AM(mi)))$, return UNBOUNDED
3. Else return $i \in$ ARGMAX

Note that ARGMAX may have more than one element, so in the third case any element $i \in$ ARGMAX, which corresponds to an optimal solution, can be returned.

2.2.2 paretoOptimalDB(vtSpec, vtReqSpec)

This function returns a set of all valid *vtSpec* instances that are Pareto-optimal with respect to *vtReqSpec*. More formally, consider a *vtSpec* instance $i \in vtValidInstances$, where the set *vtValidInstances* is defined by the function *vtValidInstance* that returns all valid instances of the *vtSpec*. Let $o_1(i),\ldots,o_n(i)$ denote the values for each objective's name in *objsSchema* of *vtReqSpec*, as computed by $objs(i)$. We say that i is Pareto-optimal w.r.t. *vtReqSpec* if the following condition is satisfied: there does not exist a valid instance $i' \in vtValidInstances$ such that

- All objective values $o_1(i'),\ldots,o_n(i')$ are at least as good as $o_1(i),\ldots,o_n(i)$, and
- At least one objective value $o_k(i')$ for $1 \leq k \leq n$ is strictly better than $o_k(i)$

Note that strictly better means "$<$" for minimization problems and "$>$" for maximization problems. According to the theoretical semantics above, the set defined by the *paretoOptimalDB* function may be (uncountably) infinite. In practical implementation, this function may return a discretized representation of the Pareto front. Also note that the function of *paretoOptimalDB* easily generalizes to input of a *set* of *vtSpecs*, from a single *vtSpec*.

2.3 *DG-ViTh* Service Architecture

DG-ViTh supports effective and efficient interactions between the users and the *Market of Virtual Things* with a customizable, modular, scalable and user-centric service architecture shown in Figure 2. The users may take upon either the role of VT consumers or VT composers to interact with the system through a graphical user interface (GUI) where available service options are presented. Each service is backed by the corresponding *DG-ViTh* functions implemented in DGAL.

The *DG-ViTh* functions are managed by the decision guidance management system (DGMS) which serves as a platform and connects technical users, such as developers, data analysts, and administrators (Brodsky and Wang, 2008). The DGMS governs the access to a library of reusable, extensible, and modular analytic models that associate with the *Virtual Things*. Through the virtual service network and virtual product assembly, various data artifacts are turned into new *Virtual Things* that match the user's innovative ideas. The *Virtual Things* are described and represented by their dedicated performance analytic models which can be crafted and customized by professional mathematical programmers and model builders, or be machine-generated by the DGMS. In addition, the DGMS offers a comprehensive toolkit for tasks like database management, data mining, simulation, optimization, and more.

3 OVERVIEW OF *OPTIGUIDE+*

OptiGuide+ is an interactive recommender system that builds upon the foundational *OptiGuide* framework. This earlier system is domain-independent and employs multi-objective optimization to guide users in extracting their utility and finding Pareto-optimal recommendations.

OptiGuide+ expands upon this foundation by integrating the VT framework from *DG-ViTh*, enabling real-time interactive utility extraction and seamless integration of VT-specific artifacts. This enhancement allows it to operate effectively within the *Market of Virtual Things*, engaging users to refine their preferences iteratively and ensuring personalized utility-driven decision-making. Moreover, *OptiGuide+* enhances accessibility and scalability by being web-deployable, which broadens its applicability across various domains and platforms. It also enhances the user interface by incorporating more functionalities, making it more intuitive for visualizing trade-offs and exploring Pareto-optimal solutions, thereby enhancing the overall user experience.

We illustrate the methodology of the *OptiGuide+* system with an example involving a procurement package from two virtual supply aggregators. Consider a procurement package that includes items like tables, chairs, and cabinets, each with a specific demand. In our repository (or market) of *Virtual Things*, there are two supply aggregators: Supply Aggregator 1, which includes suppliers 1 and 2, and Supply Aggregator 2, which includes suppliers 3 and 4, each offering different specifications. A recommendation is an optimal instance of one of the *Virtual Things*. This

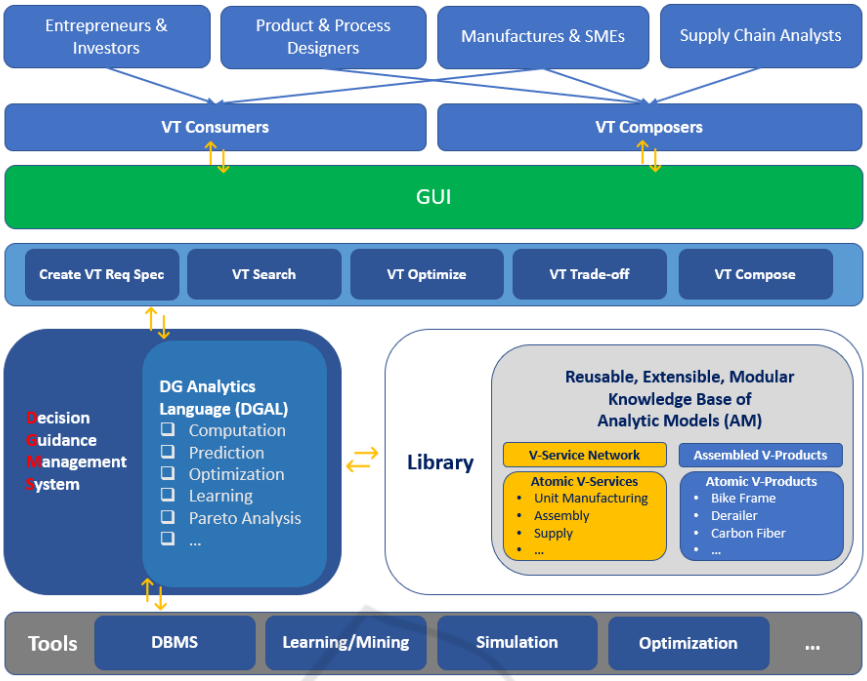

Figure 2: *DG-ViTh* Service Architecture (Han and Brodsky, 2024).

recommendation comprises a set of orders, each of which includes item quantities to be purchased from a specific supplier. These orders must meet demand and availability constraints and optimize multiple objectives, including cost, carbon emissions (CO2), and manufacturing time (manufTime).

The challenge for users is to select the most suitable instantiation of a VT that effectively balances these objectives. This process involves extracting a user utility function from the competing objectives and determining the optimal instantiation of the VT. *OptiGuide+* aims to learn the user's utility through user-system interactions and Pareto-optimal trade-offs between objectives to generate optimal recommendations with the highest predicted utility for the user.

To initialize the *OptiGuide+* recommender system for the given example, users must provide a set of domain-specific artifacts, including the analytic model *AM*, *parameters schema*, *metrics schema*, *vtSpec*, *vtReqSpec*, and specified configuration settings.

The analytic model, as depicted in Figure 3, calculates each objective (cost, CO2, and manufTime) based on the parameters and control variables of the input data. It also defines feasibility constraints to ensure non-negative quantities and to prevent exceeding suppliers' availability limits.

The *parameters schema* specifies the input parameters for the analytic model. Some of these parameters are fixed, while others are control variables, an-

```
def am(input):

    purchaseInfo = input["purchaseInfo"]
    ppu = purchaseInfo["ppu"]
    co2pu = purchaseInfo["co2pu"]
    manufTimePu = purchaseInfo["manufTimePu"]
    available = purchaseInfo["available"]
    qty = purchaseInfo["qty"]

    cost = sum([ppu[s][i] * qty[s][i] for s in qty for i in qty[s]])
    co2 = sum([co2pu[s][i] * qty[s][i] for s in qty for i in qty[s]])
    manufTime = sum([manufTimePu[s][i] * qty[s][i] for s in qty for i in qty[s]])

    constrSeq = [qty[s][i] >= 0 for s in qty for i in qty[s]]

    nonNegQtysConstraint = dgal.all(constrSeq)

    availabilityConstraint = dgal.all([qty[s][i] <= available[s][i] for s in qty
     for i in qty[s]])

    demand = {}
    for i in qty:
        for j in qty[i]:
            demand.update({j: 0})

    supply = {}
    for i in demand: supply.update({i: sum(qty[s][i] for s in qty)})

    constraints = dgal.all([nonNegQtysConstraint, availabilityConstraint])

    return {
        "combinedSupply": supply,
        "cost": cost,
        "co2": co2,
        "manufTime": manufTime,
        "constraints": constraints
        }
```

Figure 3: Analytic Model for Virtual Supply Aggregators.

notated with DGAL for optimal decision-making. In this example, the *Parameters Schema* includes the purchase information variables. For each pair of suppliers and items, the purchase information covers the price per unit, carbon emissions per unit, manufacturing time per unit, and available items, along with the control variable "quantities". Figure 4 depicts the *Parameters Schema* for Supply Aggregator 1.

The *metrics schema* captures all metrics related to

```
{
    "purchaseInfo": {
        "ppu": {
            "supplier1": {
                "chair": 65,
                "table": 90,
                "cabinet": 160
            },
            "supplier2": {
                "chair": 62,
                "table": 100,
                "cabinet": 180
            }
        },
        "co2pu": {
            "supplier1": {
                "chair": 0.7,
                "table": 1.9,
                "cabinet": 2.7
            },
            "supplier2": {
                "chair": 0.9,
                "table": 0.75,
                "cabinet": 3.2
            }
        },
        "manufTimePu": {
            "supplier1": {
                "chair": 0.5,
                "table": 1.25,
                "cabinet": 2.2
            },
            "supplier2": {
                "chair": 0.8,
                "table": 1.5,
                "cabinet": 1.9
            }
        },
        "available": {
            "supplier1": {
                "chair": 60,
                "table": 30,
                "cabinet": 25
            },
            "supplier2": {
                "chair": 90,
                "table": 50,
                "cabinet": 15
            }
        },
        "qty": {
            "supplier1": {
                "chair": {
                    "dgalType": "int?"
                },
                "table": {
                    "dgalType": "int?"
                },
                "cabinet": {
                    "dgalType": "int?"
                }
            },
            "supplier2": {
                "chair": {
                    "dgalType": "int?"
                },
                "table": {
                    "dgalType": "int?"
                },
                "cabinet": {
                    "dgalType": "int?"
                }
            }
        }
    }
}
```


Figure 4: *Parameters Schema* for VT1:Supply Aggregator1.

the user, including cost, CO2 emissions, manufacturing time, combined supply of items (demand), and the satisfaction of constraints for this example, as shown in Figure 5.


```
{
    "combinedSupply": {
        "chair": {
            "dgalType": "intMetric",
            "lb": 140
        },
        "table": {
            "dgalType": "intMetric",
            "lb": 75
        },
        "cabinet": {
            "dgalType": "intMetric",
            "lb": 34
        }
    },
    "cost": {
        "dgalType": "floatMetric",
        "lb": 0,
        "ub": 1000000
    },
    "co2": {
        "dgalType": "floatMetric",
        "lb": 0,
        "ub": 1000000
    },
    "manufTime": {
        "dgalType": "floatMetric",
        "lb": 0,
        "ub": 1000000
    },
    "constraints": {
        "dgalType": "booleanMetric"
    }
}
```


Figure 5: *Metrics Schema* for Virtual Supply Aggregators.

The *vtSpec* for each virtual supply aggregator specifies the collection of the analytic model, *parameters schema*, and *metrics schema* of a VT, as shown in Figure 6.

The *vtReqSpec* artifact captures key information about the objectives and constraints on the metrics. It


```
{
  "@context": {
    "@vtModels": "lib/vThings/vtModels/",
    "@vt": "procurementDgProject/data/vtInstances/",
    "@vtSpecs":"procurementDgProject/vtSpecs/",
    "@vtMetricSchemas":"procurementDgProject/metricSchema/"
  },
  "model": {
    "@functionRef": "lib/vThings/vtModels/procurement/modelAM.py:am"
},
  "parametersSchema": "procurementDgProject/vtSpecs/procurement0/parametersSchem
  "metricSchema": "lib/vThings/vtMetricSchemas/procurement/metricSchema.json"
}
```


Figure 6: *vtSpec* for VT1: Supply Aggregator1.

includes an objectives schema that identifies the objectives, their data types, whether each objective is a minimization or maximization metric, and the lower and upper bounds for each objective. Figure 7 illustrates the *vtReqSpec* for the virtual supply aggregators example.


```
{
  "@context": {
    "@vtModels": "lib/vThings/vtModels/",
    "@vt": "procurementDgProject/data/vtInstances/",
    "@vtSpecs":"procurementDgProject/vtSpecs/",
    "@vtMetricSchemas":"procurementDgProject/metricSchema/"
  },
  "metricSchema":"lib/vThings/vtMetricSchemas/procurement/metricSchema.json",
  "constraints":{
    "@functionRef": "/procurementDgProject/config.py:consts"
  },
  "objectives":{
    "schema": {
      "cost": {
        "dgalType": "floatMetric",
        "minMax": "min",
        "lb": 0,
        "ub": 1000000
      },
      "co2": {
        "dgalType": "floatMetric",
        "minMax": "min",
        "lb": 0,
        "ub": 1000000
      },
      "manufTime": {
        "dgalType": "floatMetric",
        "minMax": "min",
        "lb": 0,
        "ub": 1000000
      }
    },
    "function":{
      "@functionRef": "/procurementDgProject/config.py:objs"
    }
  }
}
```


Figure 7: *vtReqSpec* for Virtual Supply Aggregators.

Finally, the configuration settings for *OptiGuide+* specify the required artifacts, including *vtReqSpec* and *vtSpecs*. The *vtSpecs* structure contains a list of all relevant *vtSpecs* for the example under study. Additionally, the settings define the default objective (e.g., cost) for the user to consider, along with other technical parameters necessary for the system's operation, as shown in Figure 8.


```
{
    "reqSpec": "procurementDgProject/data/specInstances/procurementReqSpecs/
    reqSpecProcurement0.json",
    "vtSpecs": [ "procurementDgProject/vtSpecs/procurement0/vtSpec.json",
    "procurementDgProject/vtSpecs/procurement1/vtSpec.json"
    ],
    "settings": {
      "initialObj": "cost",
      "alpha_entries": 100,
      "alpha_epsilon": 0.0001,
      "unifyObjs_epsilon": 1
    },
    "input_example": "procurement_in.json",
    "folder": "procurementDgProject"
}
```


Figure 8: Configuration Settings for Supply Aggregators.

3.1 System Demonstration

The *OptiGuide+* interface is designed to provide a clear and intuitive layout for analyzing trade-offs and optimizing recommendations among multiple objectives. It facilitates the visual exploration and selection of optimal solutions from a Pareto front, guiding users in choosing and optimally instantiating *Virtual Things*. The interface, as illustrated in Figure 9, consists of the following key sections:

- **Pareto Front Graph:** The upper-left section of the interface displays a scatter plot representing a Pareto front. Each point on the graph represents a solution associated with a specific *Virtual Thing* (e.g., supply aggregator1 or supply aggregator2), showing the current trade-off between utility and another objective (e.g., cost). A toolbar with icons is provided at the bottom of the graph for quick access to additional display options, such as zooming, exporting data, and resetting views.
- **Weights of Current Utility:** This section displays the current weights assigned to each objective, indicating their relative importance in decision-making. These weights influence the calculation of the overall utility.
- **Pareto Front Table:** The upper-right section of the interface features a sortable, interactive table displaying current trade-off recommendations. Each recommendation in the table corresponds to a point on the Pareto front and includes computed objectives and current utility. A "Details" option offers deeper insight into each solution. Additionally, actionable options allow users to mark a solution as "Best" or remove it from the table.
- **Best So Far Section:** The bottom panel of the interface is reserved for recording and displaying the best solutions identified so far, facilitating comparative analysis and final decision-making. Initially, it is blank but will update dynamically.

3.1.1 User-System Interactions

Initially, as shown in Figure 9, users are presented with a graph that plots different solutions on a Pareto-optimal curve, where each alternative optimizes one objective at the expense of others. The current utility is at first calculated with equal weights assigned to each objective. The default objective, represented on the y-axis of the graph, is determined by the configuration settings, shown in Figure 8, which specify "cost" for this example. The recommendation with the highest utility value is displayed in the Pareto Front Table.

The points on the graph are color-coded for distinction, with blue indicating selectable options and red highlighting actively selected points for further analysis, as illustrated in Figure 10. Upon selecting a point on the graph, the system automatically adds the points and displays its information in the Pareto Front Table. A sorting feature above the table allows users to organize solutions based on selected criteria such as utility or cost. Clicking on "Details" opens a hierarchical view of the solution, displaying specific data such as the quantities of items to be purchased from designated suppliers within a supply aggregator. For instance, the solution shown in Figure 10 involves suppliers 3 and 4, associated with supply aggregator2.

When the user selects the third recommendation (Rec 3) as the best choice, the system updates by adding it to the "Best So Far" section. Concurrently, the system recalculates the current utility based on the objective weights of the selected recommendation and updates the Pareto Front Graph and Table to reflect the updated utility.

The "Best So Far" section displays a graph of normalized utility and objectives for each best solution, which facilitates comparison, alongside a detailed information table for that solution. Additionally, for each best solution in this section, the user has three actionable options: "Improve" to generate a new set of recommendations by optimizing a selected objective, "Remove" to delete the solution from the list, or "Accept" to finalize it as the optimal recommendation.

For example, if the user wants to improve the "Best 1" recommendation based on manufacturing time, they select the "manufTime" objective from the table and mark the "Improve" option. Consequently, a new set of recommendations is plotted along the Pareto curve where the y-axis of the Pareto Graph becomes "manufTime", and the x-axis represents the last updated utility. The user can continue refining through this iterative process, as depicted in Figure 11, until deciding to finalize their choice by selecting the "Accept" option for the optimal recommendation.

Figure 12 displays the accepted optimal recommendation and its detailed solution, which involves orders from suppliers 1 and 2, both associated with a specific VT (supply aggregator 1).

4 *OPTIGUIDE+* ARCHITECTURE AND IMPLEMENTATION

In the development of *OptiGuide+*, we utilize the foundational architecture of the original *OptiGuide*, shown in Figure 13. This architecture comprises two primary components: the Recommendation Engine,

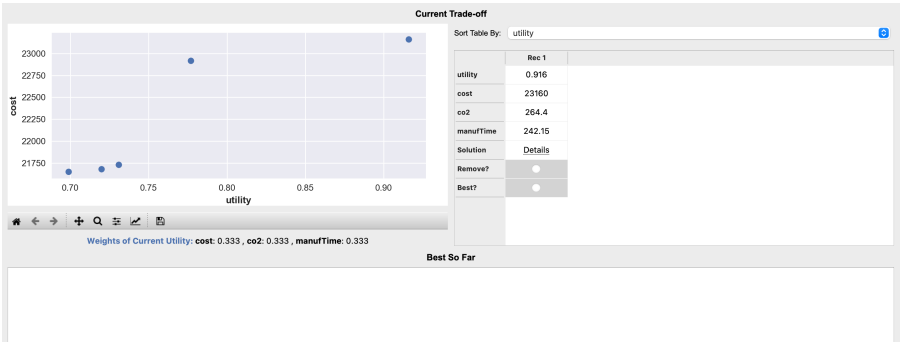

Figure 9: *OptiGuide+* Interface Displaying the Initial Pareto Front Visualization.

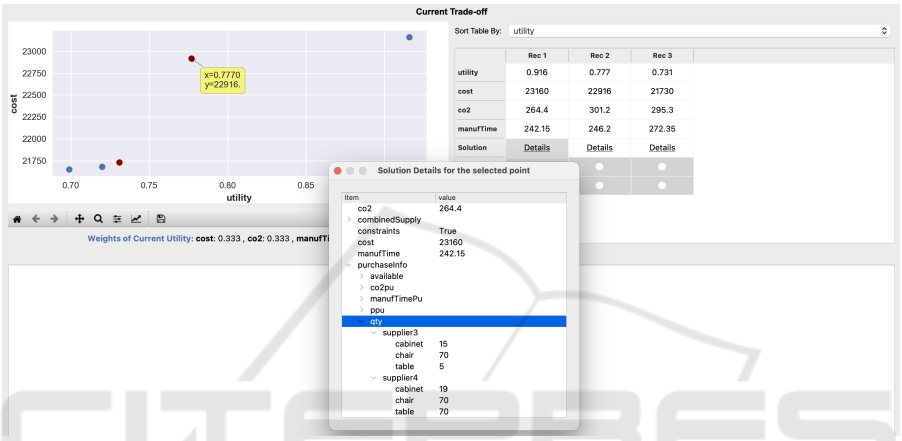

Figure 10: Adding Data Points to the Pareto Front Table and Displaying Solution Details of a Selected Point.

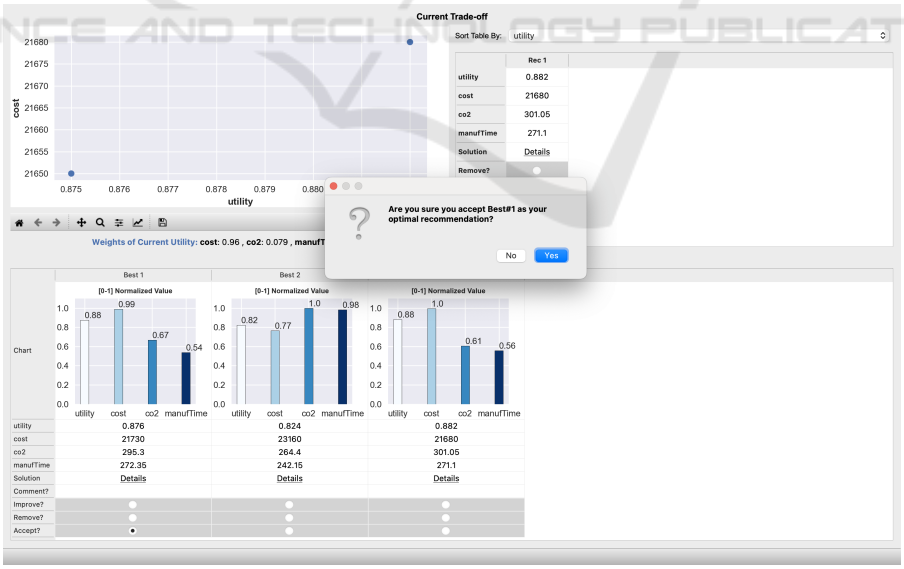

Figure 11: *OptiGuide+* Interface Demonstrating Iterative Improvement and Selection of the Best Solutions.

which handles the *preprocessing phase*, and the Recommendation User Interface, which implements the *runtime phase* (Almanie and Brodsky, 2024).

The Recommendation Engine requires initialization with domain-specific artifacts that include an analytic model, *parameters schema*, *metrics schema*, *vtSpec*, *vtReqSpec*, and configuration settings. Once initialized, it integrates with the DGMS, which uses

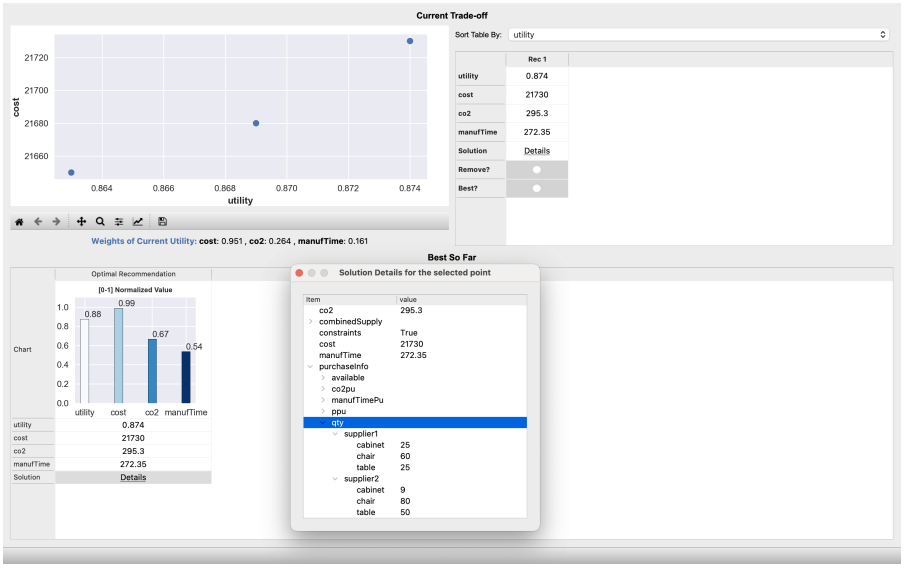

Figure 12: *OptiGuide+* Interface Showing the Accepted Optimal Recommendation and its Solution Details.

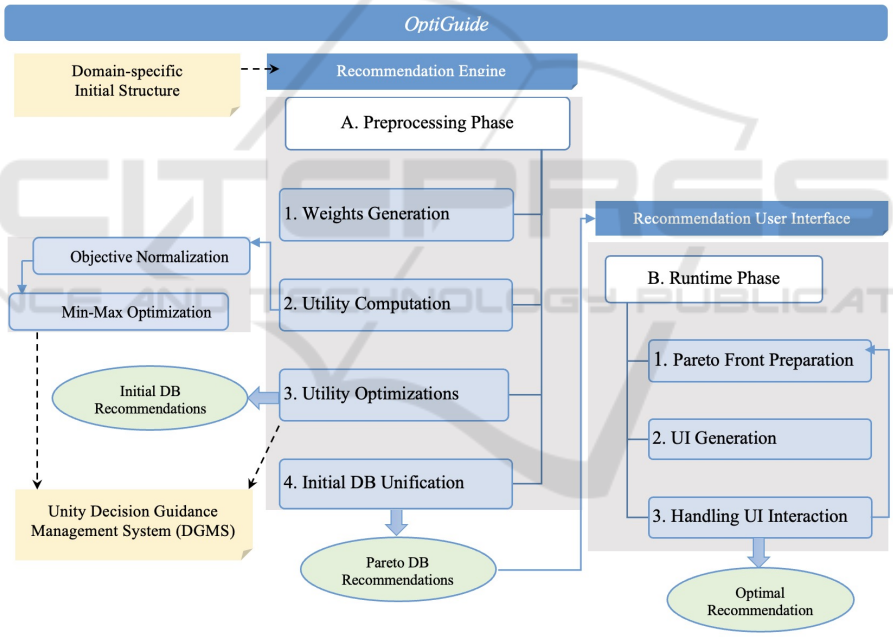

Figure 13: *OptiGuide+* Architecture (Almanie and Brodsky, 2024).

DGAL to generate domain-specific recommendations and compute their objectives based on this initial structure. The goal of the *preprocessing phase* is to create a database for the Pareto front of the recommendation space. This database includes estimated user utility for each Pareto-optimal alternative, which will then be utilized during the *runtime phase*. The *preprocessing phase* involves four steps:

1. **Weights Generation:** This step generates a list of weight combinations, where each weight is associated with an objective. These weights contribute to generating Pareto-optimal solutions during optimization.
2. **Utility Computation:** The utility function normalizes the objectives on a scale from zero to one, where zero represents the worst value and one represents the best. It then computes the weighted sum of these normalized objectives using the generated list of weights.
3. **Utility Optimizations:** This step employs the *paretoOptimalDB* function to identify all *vtSpec*

instances that are Pareto-optimal with respect to *vtReqSpec*. It iterates over the generated list of weights and uses the *vtOptimalInstance* function to find a VT instance that matches the *vtReqSpec* and optimizes the user's utility, resulting in an "initial DB" JSON file containing all feasible recommendations for the given problem.

4. **Initial DB Unification:** This step merges recommendations that share similar or identical objective values but differ in weights, utilizing Euclidean distance and K-Medoids clustering. This process reduces redundancy, resulting in a "Pareto DB" JSON file containing only representative, non-dominated solutions.

The *runtime phase* is designed to facilitate user interaction and learning of an individual's utility to eventually identify their optimal recommendation. This phase includes three components:

- **Pareto Front Preparation:** This component prepares the Pareto front data from "Pareto DB" entries based on the current weights and the objective designated for the y-axis. It computes the current utility, represented on the x-axis, for each entry according to these weights.
- **UI Generation:** This component provides the user with a visual representation of Pareto-optimal solutions.
- **Handling UI Interaction:** It handles all user interactions within the interface. For example, when a user selects a point as "Best" in the table, it retrieves the weights of the corresponding point and updates the GUI to reflect the new utility.

5 *OPTIGUIDE+* CROSS-PLATFORM DEPLOYMENT METHODOLOGY

The deployment of *OptiGuide+* was designed to leverage JupyterLab as the primary platform, capitalizing on its robust support for integrating computational workflows with a graphical user interface (GUI). Users will be able to access the project repository that houses the relevant artifacts and invoke function calls from the GUI. This process ensures an accessible, efficient, and reproducible deployment mechanism, aligning with modern computational standards. The specific steps are as follows:

1. **Platform Preparation:** JupyterLab was selected as the ideal platform for hosting *OptiGuide+* due to its browser-based interface, which ensures platform independence and minimizes setup complexity.
2. **Deployment Initialization:** The deployment process begins with launching JupyterLab from the project directory. This step grants users access to a unified workspace, allowing them to interact with all necessary components of *OptiGuide+*, including file management, script execution, and application interfaces.
3. **Data Preprocessing:** Within JupyterLab, the preprocessing module (mainPreprocessing.py) is executed. This module transforms the input artifacts and pre-computes the data in the underlying Pareto optimal database in preparation for subsequent interactions, ensuring optimal performance of the GUI.
4. **Interface Activation:** After preprocessing, the graphical user interface is initialized by executing the script (optiguideUI.py). This launches the *OptiGuide+* interface in a new browser tab, where users can engage with its features and explore its capabilities in real-time.

By utilizing JupyterLab, this deployment methodology features cross-platform reproducibility, scalability, and user-centric design. The simplified setup and intuitive interface reduce barriers to entry, fostering broader adoption among diverse user groups.

6 CONCLUSIONS

OptiGuide+ has advanced the field of interactive decision guidance systems by specifically addressing the needs of the *Virtual Things* (VT) market. The system leverages multi-objective optimization techniques and real-time user interaction to generate Pareto-optimal recommendations, ensuring that each decision maximizes the user's utility while adhering to practical constraints.

Many interesting research questions remain open. They include (1) the development of more efficient algorithms for Pareto front computation;

(2) exploring new paradigms for utility extraction; and (3) improvements to the user interface with an emphasis on integrating advanced visualization tools. These enhancements will aid users in more effectively comparing the "best so far" options, incorporating domain-specific visualizations such as maps to provide deeper insights into the data.

REFERENCES

Almanie, T. and Brodsky, A. (2024). Optiguide: An efficient domain-independent package recommender system based on multi-objective optimization and user decision guidance. In *Proceedings of the 57th Hawaii International Conference on System Sciences*, page 1528–1535.

Brodsky, A., Krishnamoorthy, M., Bernstein, W. Z., and Nachawati, M. O. (2016). A system and architecture for reusable abstractions of manufacturing processes. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2004–2013.

Brodsky, A. and Luo, J. (2015). Decision guidance analytics language (DGAL) - toward reusable knowledge base centric modeling. In *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 1, Barcelona, Spain, 27-30 April, 2015*, pages 67–78. SciTePress.

Brodsky, A., Nachawati, M. O., Krishnamoorthy, M., Bernstein, W. Z., and Menascé, D. A. (2019). Factory optima: A web-based system for composition and analysis of manufacturing service networks based on a reusable model repository. *International Journal of Computer Integrated Manufacturing*, 32(3):206–224.

Brodsky, A. and Wang, X. S. (2008). Decision-guidance management systems (dgms): Seamless integration of data acquisition, learning, prediction and optimization. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, pages 71–71.

Egge, N. E., Brodsky, A., and Griva, I. (2013). An efficient preprocessing algorithm to speed-up multistage production decision optimization problems. In *46th Hawaii International Conference on System Sciences, HICSS 2013, Wailea, HI, USA, January 7-10, 2013*, pages 1124–1133. IEEE Computer Society.

Gingold, Y. I., Igarashi, T., and Zorin, D. (2009). Structured annotations for 2d-to-3d modeling. *ACM Trans. Graph.*, 28(5):148.

Griffiths, C., Howarth, J., Almeida-Rowbotham, G. D., Rees, A., and Kerton, R. (2016). A design of experiments approach for the optimisation of energy and waste during the production of parts manufactured by 3d printing. *Journal of Cleaner Production*, 139:74–85.

Han, X. and Brodsky, A. (2022). Toward cloud manufacturing: A decision guidance framework for markets of virtual things. In *ICEIS (1)*, pages 406–417.

Han, X. and Brodsky, A. (2023). Optivith: A decision guidance framework and system for design, analysis and optimization of cloud-manufactured virtual things. In Filipe, J., Śmiałek, M., Brodsky, A., and Hammoudi, S., editors, *Enterprise Information Systems*, pages 175–197, Cham. Springer Nature Switzerland.

Han, X. and Brodsky, A. (2024). Dg-vith: A decision guidance system for repositories of virtual things. *Journal of Decision Systems*, 0(0):1–19.

Jeffries, W. and Brodsky, A. (2017). Composite alternative pareto optimal recommender system (capors). In *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 1: ICEIS,*, pages 496–503. INSTICC, SciTePress.

Jeffries, W. and Brodsky, A. (2018). Composite alternative pareto optimal recommendation system with individual utility extraction (capors-iux). In *International Conference on Enterprise Information Systems*.

LaToza, T. D., Shabani, E., and van der Hoek, A. (2013). A study of architectural decision practices. In *6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013, San Francisco, CA, USA, May 25, 2013*, pages 77–80. IEEE Computer Society.

Li, H., Yang, J., Chen, G., Liu, X., yun Zhang, Z., jun Li, G., and hui Liu, W. (2020). Towards intelligent design optimization: Progress and challenge of design optimization theories and technologies for plastic forming. *Chinese Journal of Aeronautics*.

Liu, X., Tang, L., Yi, Y., and Ni, Z. (2019). Intelligent design based on holographic model using parametric design method. *Journal of Ambient Intelligence and Humanized Computing*, 10:1241–1255.

Menascé, D. A., Krishnamoorthy, M., and Brodsky, A. (2015). Autonomic smart manufacturing. *Journal of Decision Systems*, 24(2):206–224.

Nachawati, M. O., Brodsky, A., and Luo, J. (2017). Unity decision guidance management system: Analytics engine and reusable model repository. In *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 1: ICEIS,*, pages 312–323. INSTICC, SciTePress.

Shao, G., Brodsky, A., and Miller, R. (2018). Modeling and optimization of manufacturing process performance using modelica graphical representation and process analytics formalism. *J. Intell. Manuf.*, 29(6):1287–1301.

Shao, G., Brodsky, A., Shin, S.-J., and Kim, D. B. (2017). Decision guidance methodology for sustainable manufacturing using process analytics formalism. *Journal of Intelligent Manufacturing*, 28(3):455–472.

Shao, G., Kibira, D., Brodsky, A., and Egge, N. (2011). Decision support for sustainable manufacturing using decision guidance query language. *International Journal of Sustainable Engineering*, 4(3):251–265.

Shin, S., Kim, D. B., Shao, G., Brodsky, A., and Lechevalier, D. (2017). Developing a decision support system for improving sustainability performance of manufacturing processes. *J. Intell. Manuf.*, 28(6):1421–1440.

Yu, L.-F., Yeung, S. K., Tang, C., Terzopoulos, D., Chan, T. F., and Osher, S. J. (2011). Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30(4):86.