SPOT: Toward a Decision Guidance System for Unified Product and Service Network Design

Joost Bottenbley[a] and Alexander Brodsky[b]
Department of Computer Science, George Mason University, 4400 University Dr., Fairfax, Virginia, U.S.A.

Keywords: Decision Support, Decision Guidance, Optimization, Manufacturing, Supply Chain, Product Design, Service Network.

Abstract: A major deficiency in the manufacturing ecosystem today is the lack of cloud-based infrastructure that supports the combined decision making and optimization of product design, process design, and supply chain, as opposed to hard wired solutions within silos today. The reported work makes a step toward bridging this deficiency by developing a software framework, prototype and a case study for SPOT - a decision guidance system for simultaneous optimization and trade-off analysis of combined service and product networks, capable to express the combined product, process and supply chain design. SPOT allows users to express, as data input, a hierarchical assembly and composition virtual products and services, i.e., having fixed and control parameters that can be optimized. Virtual services produce a flow of virtual products, such as raw materials, parts of finished products. Like the virtual services, they are associated with analytic models that express customer-facing performance metrics and feasibility constraints, which are used for optimization. The uniqueness of our approach in SPOT is the use of modular simulation-like model for product and service networks, yet optimization quality and computational time of the best available mathematical programming solvers, which is achieved by symbolic computation of simulation code to generate lower-level mathematical programming models.

1 INTRODUCTION

Smart manufacturing can be defined as “the synthesis of advanced manufacturing capabilities and digital technologies to improve the productivity, agility, and sustainability of manufacturing systems” (Helu and Hedberg, 2015). By leveraging the large amount of data from IoT devices and real time analytics, smart manufacturing has become a key element in reducing manufacturing costs, improved efficiency, and time-to-market (KLE,). These reductions have been prevalent enough for some companies to move outsourced manufacturing services back to domestic markets (Shehadi, 2019).

A key approach to realize a manufacturing ecosystem is to employ cloud-based infrastructure that supports decision making and optimization of product design, process design, and supply chain (Brodsky et al., 2017). Such infrastructure must support a wide variety of analytic tasks across the entire organizational hierarchy. This includes manufacturing units, cells, production lines, factories, and supply chains (Salvendy, 2001). Considering the plethora of potential players across an enterprise’s supply chain, it is critical “to effectively and efficiently combine manufacturing services ... [in] multiple-factory production environments” (Wu et al., 2013). Developing tools and methods for service composition remains an on-going research area and requires generic and robust model representations to perform advanced analysis, e.g. optimization (Wu et al., 2013).

There has been extensive work on decision making in product design (e.g., see overview (Klingstam and Gullander, 1999)), process design (e.g., see overview (Stoll, 1986)) and supply chain (e.g., see overview (Garcia and You, 2015)). However, this work is typically done in “silos”. Product design does not take into account manufacturing process design and supply chain. Whereas, often small modifications in product design may result in improving manufacturability and costs with little or no negative effect on customer-facing product characteristics (Eddy et al., 2015; Herrmann et al., 2004). Similarly, manufacturing process design assumes that the product design is

[a] https://orcid.org/0000-0003-1702-9718
[b] https://orcid.org/0000-0002-0312-2105

Bottenbley, J. and Brodsky, A.
SPOT: Toward a Decision Guidance System for Unified Product and Service Network Design.
DOI: 10.5220/0010459707170728
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1*, pages 717-728
ISBN: 978-989-758-509-8

already fixed, limiting design choices, and does not take into account supply chain opportunities. In turn, during supply chain design, both product and process design are assumed fixed, limiting possibly attractive sourcing options. Some work on product design (Molcho et al., 2008), did consider manufacturability. However, this was used as a separate layer of design filtering, as opposed to a comprehensive combined optimization.

As described in (Brodsky et al., 2017), analysis and optimization solutions in manufacturing are typically implemented from scratch, following a linear methodology (McLean and Shao, 2003; Denkena et al., 2007). This leads to high-cost and long-duration development and results in models that are difficult to modify, extend and reuse. This is because different computational tools, i.e., for simulation and optimization, use different low-level mathematical abstractions, which results in re-modeling the same knowledge multiple times.

The work (Brodsky et al., 2016; Brodsky et al., 2017; Brodsky et al., 2019) bridged this deficiency, and also unified optimization of manufacturing processes and supply chain by developing a software architecture and Web-based solution called *Factory Optima*. This tool allows manufacturers to compose, optimize, and perform trade-off analysis of service networks involving both unit manufacturing processes and supply chains. However, their work assumes that product design (for raw materials, part, and finished products) is fixed.

This problem is studied in (Brodsky et al., 2019), which proposed the concept and market place for virtual things (i.e. virtual products and virtual services) based on unified modeling, analysis, and simultaneous optimization of product, process, and supply chain design. However, that work is limited to a theoretical framework, and leaves open the problem of developing generalized models and systems that can support the theoretical framework.

This paper makes the first step on bridging this gap. More specifically, the contribution of this paper is two-fold. First, we develop a software prototype for SPOT [1] - a decision guidance system for simultaneous optimization and trade-off analysis of combined service and product networks, capable to express the combined product, process and supply chain design toward the vision of virtual products and services of (Brodsky et al., 2021). SPOT is based on extending the software framework of Factory Optima (Brodsky et al., 2016; Brodsky et al., 2017; Brodsky et al., 2019) with a model for steady-state *virtual service and product* (VSP) network which allows users to express, as data input, a hierarchical assembly of products and composition of services in terms of (composite or atomic) sub-services. Virtual services produce a flow of virtual products (e.g., raw materials, parts of finished products), which are, like the virtual services, are parameterized with fixed and control variables that effect product and service customer-facing performance metrics and feasibility constraints. The uniqueness of our approach in SPOT (like in Factory Optima) is the use of modular simulation-like model for product and service networks, yet optimization quality and computational time of the best available mathematical programming solvers, which is achieved by symbolic computation of simulation code to generate lower-level mathematical programming models. Second, we demonstrate SPOT on a case study of a (virtual) bicycle product and service network, and describe a typical methodology of its use.

This paper is organized as follows. Section 2 provides an illustration of how SPOT translates a supply chain network which yields any number of products into a V-Service network and a V-Product. In addition, it describes how this V-Things can be used to answer questions about the viability of manufacturing the product. Section 3 describes the enterprise architecture of SPOT, functionalities, common domain users, and types of actionable recommendations SPOT can provide to these domain users. Section 4 provides the formal model of the problem statement, the valid input data structure, the valid output data structure, and the computation of the analytic model. Section 5 demonstrates how domain users can generate a valid input data structure, set the optimization parameters and constraints, and interpret the results. Section 6 details the results of the study and future research directions of interest.

2 VIRTUAL PRODUCT & SERVICE ILLUSTRATION

As described in (Brodsky et al., 2021), a VSP, intuitively, is represented by a parameterized CAD design, e.g., to characterize a customizable consumer product, part or raw material. A VSP represents a parameterized transformation of virtual products into other VSPs, e.g., to characterize a customizable manufacturing process, supply, transportation, logistics or a composed service network.

Each V-thing—product or service—is associated with an analytic model that describes the product and/or service's feasibility and customer-facing metrics/characteristics as a function of the product and/or

[1]SPOT: Services and Products Optimization Tool.

service's (fixed and decision) parameters. For V-products, examples of customer-facing metrics include external dimensions, weight, durability and vacuum efficiency; while examples of internal parameters include internal dimensions, position of fixtures, and type and properties of materials. For V-services, examples of customer-facing metrics include cost-per-unit, total ordered quantities per item, delivery time, carbon emissions per unit, and default risk; while examples of internal parameters include settings for unit manufacturing processes (e.g., CNC machining, injection molding or 3D printing) and selection of and ordered quantities from suppliers and manufacturers.

Intuitively, V-things' customer-facing metrics are all that customers care about when selecting products and services; whereas, customers do not care about, or even understand, V-thing parameters outside the set of customer-facing metrics. Mathematically, an analytic model for a V-thing - product or services — represents a mapping from the parametric space (the input) into the customer-facing metric space (the output.)

In order to create V-services, we focus in this paper on the model development for a V-service network, which allows hierarchical composition of both v-services and V-products, recursively, out of sub-services and sub-products. Based on this model, the SPOT system will make actionable recommendations on the optimal parameter instantiation of V-service and products.

Before defining a formal analytic model in Section 4 for a VSP network, consider the example a special bicycle production depicted in Figure 2. The overall bicycle service network in the example involves supply (depicted on the left), manufacturing (depicted in the middle) and transportation (depicted at the top).

Within the supply part, there are five potential suppliers - including Zoltek and Tradewell LTC - which supply raw materials such as leather, raw steel, and plastic.

We will follow the products made by Tradewell LTC and Zoltek to illustrate the flow of material through the supply chain. As illustrated in Figure **??** Zoltek has two outFlows: CarbonFiber01 and Leather01. These flow ids are listed in Zoltek's outFlows object. Querying the flows object at the top-level will reveal that these flows contain the products carbon fiber and leather. In turn, querying the products object at the top-level provides physical characteristics of the products. For example, the product object carbon fiber has characteristics such as: strength to weight ratio, rigidity, fiber type, thickness dry, fiber areal weight, and density. As indicated by Figure 2, Zoltek's outFlow is given to Swift Transpiration. This agent, a atomic Transportation service (L2), provides transpiration services from Zoltek's location to Bianchi's location. The location information is stored in our shared object at the top-level so that any function can access this data.

Bianchi Corp. is a tier three (L3) manufacturer. Unlike Suppliers, manufactures require products from other agents within the supply chain network to produce products. In this instance, Bianchi requires carbon fiber and leather, as well as other products from with in the supply chain, to make two products: a carbon fiber frame and a leather seat. These two products are transported by the transportation service Fed Ex to the final assembly location where they are finally assembled into a road bike.

Tradewell LTC also is an atomic supplier. The outFlow id of Tradewell LTC is AU01 and contains the product aluminum scrap. This product is transported to Simurgh Iron Company through the transportation service Create Carrier Corporation. Simurgh Iron Company provides the service of manufacturing aluminum billets from raw aluminum scrap. Simurgh Iron is a tier 2 (L2) manufacturer. Western Express transports this output to Garmin, ENVE, Bianchi, and Shimano. These tier three (L3) manufactures use the aluminum to create there respective products as illustrated in Figure **??**. The final assembly location receives its inputs from Fed Ex. Each of these products are considered basic products. Basic products are distinguished from assembled products in the fact basic products do not require assembly. The final assembly service constructs the road bike from the required parts listed in its list of components.

Now consider the decision making problem presented to investors, entrepreneurs, business developers, and other participants in the supply chain. There are several critical decision points these stakeholders would be interested in which could be easily answered by this schematic of the Virtual Bicycle. Entrepreneurs, who may have a considerable amount of domain knowledge of their particular industry, but no knowledge of manufacturing processes, would be aided in their decision to contract with particular providers of supplies and services. Business developers, who's main focus is to maintain and develop new relationships, would be informed about new market participants. Investors would have considerable knowledge about the VSPs cost, time-to-market, regulation requirements, and other KPIs that would be stored in the library of models stored in SPOT's model repository.

3 SPOT DECISION GUIDANCE FUNCTIONALITY & SYSTEM ARCHITECTURE

The developed SPOT Decision Guidance System is designed to provide actionable recommendations on optimal V-services and products instances to domain users including entrepreneurs, investors, product & process Designers, and supply chain analysts as well as to CAD/CAM systems.

SPOT is a middleware software application that facilitates the design of new VSPs by modifying and linking existing VSPs. It enables the simultaneous optimization of products and services by leveraging product-design-compositions and service-network-compositions. In addition, it allows decision guidance analytics in the form of performance metric prediction, Pareto trade-off analysis, and model calibration (training) using historical examples. Finally, it supports an extensible reusable repository of VSPs and analytic models which enable creation of a market of VSPs.

The architecture of the DGS system is illustrated in Figure 1. SPOT will reside on a local, centralized, or distributed compute server and receive the necessary processing request and structured data. Domain users will submit analytic queries and receive answers through the modeling and analytics GUI - see top layer of Figure 1. CAD and CAM software tools can also submit jobs to SPOT through APIs and interface mappers.

Instead of implementing analytics functionality using low-level analytics tools such as optimization, simulation and machine learning (the lower layer in Figure 1), SPOT uses the middleware of *reusable, extensible, modular knowledge base of analytic models* and the Decision Guidance Management System (DGMS). Each analytic model in the KB describes a function that, given fixed and control parameters of service and product network, computes its performance metrics such as cost and delivery time, as well as feasibility constraints. Analytics queries, including prediction and optimization, are posed to DGMS against a particular analytic model. For example, optimization query asks to find an instantiation of all control parameters in the model input, that minimizes or maximizes a performance metric within feasibility constraints computed in the model output.

The SPOT knowledge base includes analytic models for V-Services Network; Atomic V-Services (such as unit manufacturing processes, assembly, contract manufacturers and suppliers and transportation); Assembled V-Products; and, Atomic V-Products. These high-level abstractions encapsulate the construction of a hierarchical service network out of service and product components. We formally define V-service and product analytic models, which are central to SPOT functionality, in Section 4.

Against analytic queries posed by users, DGMS performs symbolic computation and generates low-level models that are then submitted to low-level tools. For example, an optimization query against an analytic model (described as a recursive function in Python) is translated by DGMS to a mathematical programming model which is then submitted to CPLEX - a mixed integer linear programming solver.

To understand the functionality of the system consider the following scenario. You have an entrepreneur who wants to manufacture a custom bicycle. The system will support the entrepreneur by constructing a virtual network and product. First, it will construct the graph for the virtual service network similar to the network depicted in Figure 2. He wants and answer to the question, “Who are the suppliers in the substantiated the supply chain network?”

The system retrieves the relevant models from the modular model repository (e.g. the V-Service Network, Atomic V-Services, Assembled V-Products, and Atomic V-Products) and takes as input a parameterized JSON data structure. The parameterized input describes the structure of the hierarchical bicycle service network (see Figure 2). This data structure contains a variety fixed parameters and control parameters. Control parameters serve as our decision variables. In our example in Section 2, control parameter include the quantity of each input for manufacturers, and which suppliers, manufactures, and transportation services would make up the bike service network. Examples of fix parameter include the required components to assemble the custom bike, the weight of the raw aluminum required for the handle bars, and the locations of each manufacturer. In addition, the SPOT DGS request includes a list of metrics that should be used in the objective function for optimization in the output. The model also has a series of constraints on supply, flow, materials available, delivery time and other constraints limiting the feasibility of the optimal output. DGMS then performs a symbolic analysis of the model, the parameterized input, and objective function and machine generates a mathematical programming model. It then uses solvers, such as condor, cplex, minor, to find the optimal solution. The solution is then returned back to system (see Section 5 for sample output). SPOT then takes the result, looks up the values for the control variables, and populates these values in the input data structure, and returns the substantiated service network to the user/agent.

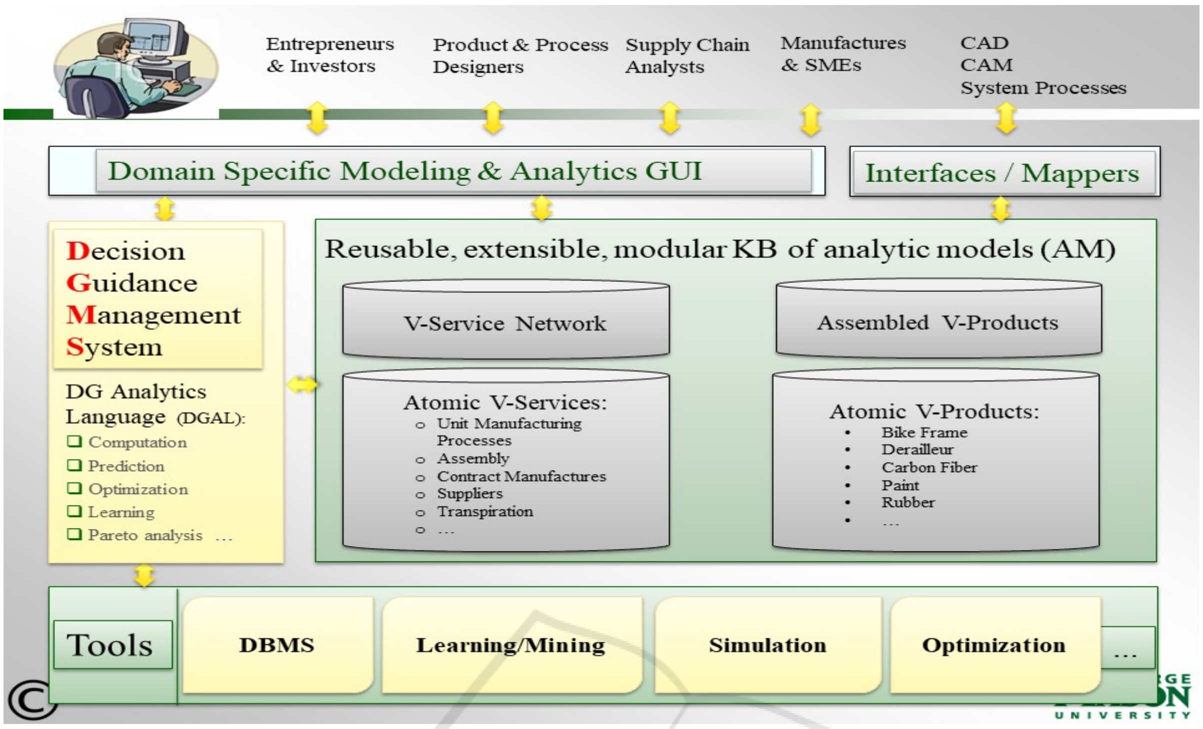


Figure 1: SPOT architecture based on DGMS.

4 ANALYTIC PERFORMANCE MODEL FOR PARAMETERED PRODUCTS AND SERVICES

4.1 Parameterized Optimization Problem

Optimization models for VSPs are predicated on the notion of an analytic performance model (AM). Intuitively, AM describes the *output* of performance metrics (such as weight and cost) and feasibility constraints as a function of *input*. This *input* consists of fixed decision parameters such as product properties, quantities of products purchased, and settings of manufacturing processes. More formally, an AM is a function:

$$AM : I \rightarrow O \qquad (1)$$

where I and O are valid input and output domains. The function AM yields the output $AM(i) \in O$ for $i \in I$ which captures customer-facing metrics.

To find optimal values of decision parameters in the *input*, we can choose an objective function that gives a real value for every output $o \in O$

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

as well as constraint

$$C : O \rightarrow \{T, F\}$$

that yields a Boolean value for every output $o \in O$ to indicate whether it is feasible ($C(o) = T$) or not ($C(o) = F$).

We are interested in solving optimization problems of the form

$$\begin{aligned} \min \backslash \max_{x \in I} \quad & Obj(AM(x)) \\ s.t. \quad & C(AM(x)) \end{aligned} \qquad (2)$$

Note that, while the *objective* and the *constraints* in the optimization problems are expressed in terms of input $i \in I$, our formulation based on AMs allows flexible expression of a range of optimization problems, with no need to modify the AM, which "hides" a possibly involved code that expresses its function. Formalizing the objective function and constraints in this manner leads to a natural decomposition of the problem statement into separate categories that intuitively reflect the product and process specifications.

In the remainder of Section 4 we describe the AM for a (hierarchically) composed service network with flexible products, using the example of a bicycle manufacturing supply chain. To do this, we first describe the structure of a valid output from the AM in Section 4.2, then a valid input in the AM in Section 4.3, and, finally, the computation flow of the AM, in Section 4.4, which maps a valid input instance to a valid output.

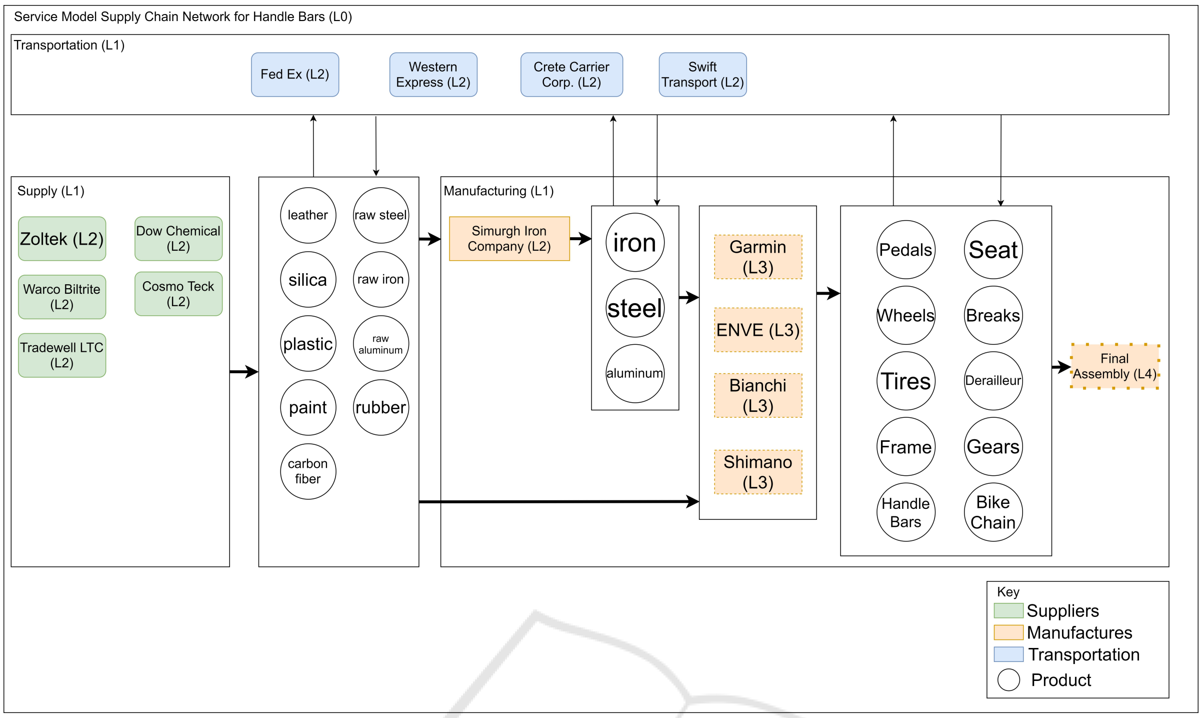


Figure 2: Supply Chain Product and Service Flow.

In the following, we use the following notation of a set of key-value pairs of the form:

$$\{key_1 : value_1, \ldots, key_n : value_n\}$$

where the keys are unique identifiers. Note, that this structure represents a mapping m from the set of keys $\{key_1, \ldots, key_n\}$ to the union $\cup_{i=1}^{n} D_i$ of domains $D_i, i = 1, \ldots, k$, such that $m(key_i) \in D_i$. Thus $m(key_i)$ represents $value_i$, $i = 1, \ldots, n$.

4.2 Model Output Structure

An output o of the analytic model represent metrics of interest and constraints in the form of a set of $key : value$ pairs:

$$\begin{aligned} \{ \quad & metric_1 : value_1 \\ & \ldots , \\ & metric_n : value_n \\ & \text{constraints: True or False} \\ & \text{rootService: root service id} \\ & \text{services: <set of services>} \\ & \text{products: <set of products>} \ \} \end{aligned} \tag{3}$$

as follows.

***Metrics* 1.** through n are additive metrics of the root service (see below) such as the total cost and carbon emissions. Note that these metrics are not fixed as they are aggregated, recursively, from atomic services up.

***Constraints*:** are $True$, if all the feasibility constraints of the service network under the root service and the products associated with it, as described under services, are satisfied.

***rootService*:** is the id of the root service, to distinguish it from the other services, which are described next.

***Services*:** is a set of key-value pairs of the form:

$$\begin{aligned} \{ \quad & sid_1 : service_1, \\ & \ldots , \\ & sid_n : service_n \ \} \end{aligned} \tag{4}$$

where the *sids* are unique service identifiers, including the root service. Each service 1 through n is either *composite* or *atomic*. *Composite* services have sub-services, whereas *atomic* services do not. Each composite service output is a set of key-value pairs of the form:

$$\begin{aligned} \{ \quad & \text{type: “composite”} \\ & metric_1 : value_1 \\ & \ldots , \\ & metric_n : value_n \\ & \text{constrains: True or False} \\ & \text{inFlow: flows of consumed products} \\ & \text{outFlow: flows of produced products} \\ & \text{subServices: set of sub-service ids} \ \} \end{aligned} \tag{5}$$

type indicates the type of the service, and is "composite" here for the composite service.

metrics 1 through n are the same as the metrics in equation (3). Each metric is additive, i.e., it is the sum of the values for this metric across all (child) subServices, described below.

constraints is the conjunction of sub-service constraints, and the flow balancing constraints within the service. These are described formally in Section 4.4.

inFlow and *outFlow* describe the flows of consumed and produced products, respectively. Each is of the following form:

$$\begin{array}{lll} \{\ fid_1 :\{ & \text{qty: real number} & \\ & \text{item: pid}\}, & \\ & \ldots , & (6) \\ fid_n :\{ & \text{qty: real number} & \\ & \text{item: pid}\} & \end{array}$$

where *fids* are unique flow identifiers, where corresponding *qty* is the quantity (being consumed or produced), and *pid* is the unique identifier of a product associated with the flow. *subServices* is a set of ids of sub-services of this composite service.

Each *atomic* service is the same form as equation (5) for *composite* service with a few exceptions. First, the object *subService* is not present because *atomic* services do not have sub-services. Second, *type* maybe any string not logically equivalent to "composite". In our example in Section 2, we use the string "supplier" and "transport" to distinguish *atomic* services from each other. Third, the *metrics*, *constraints*, *inFlows*, and *outFlows* are specific to the *atomic* service. Thus, each atomic service output is a set of key-value pairs of the form:

$$\begin{array}{ll} \{ & \text{type: !=``composite''} \\ & metric_1 : value_1 \\ & \ldots , \\ & metric_n : value_n \qquad (7) \\ & \text{constrains: True or False} \\ & \text{inFlow: flows of consumed products} \\ & \text{outFlow: flows of produced products } \} \end{array}$$

***Products*:** is a set of key-value pairs of the form:

$$\begin{array}{ll} \{ & pid_1 : product_1, \\ & \ldots , \qquad (8) \\ & pid_n : product_n\ \} \end{array}$$

where *pid* are ids of the products. Each product 1 through *n* is of the form

$$\begin{array}{ll} \{ & \text{type: basic or assembled}\} \\ & metric_1 : value_1, \\ & \ldots, \qquad (9) \\ & metric_n : value_n, \\ & constraint : \text{True or False } \} \end{array}$$

where *type* is either an *basic* or *assembled* product. *Assembled* products are constructed from other products within the supply network and are labeled *assembled* under *type*. *Products* also contains a series of constrains. These constraints specify thresholds or acceptable materials for use. Our *products* object is then given to us by the following n-tuple:

4.3 Model Input Structure

A valid VP input instance *i* is a set of key-value pairs

$$\begin{array}{ll} \{ & \text{shared: <general-shared-info>} \\ & \text{flows: <set of flow ids>} \\ & \text{rootService: root service id} \qquad (10) \\ & \text{services: <set of services>} \\ & \text{products: <set of products> } \} \end{array}$$

***Shared*:** contains all data required for computation in multiple layers of security or computation. For example, the location of each manufacturer is required in the computation of transportation services (see Section 2) and it is also required in the optimization of environmental metrics for material selection (see Section 2). The structure of *Shared* is:

$$\begin{array}{ll} \{ & shared_1 : \{\text{real-value, text}\}, \\ & \ldots , \qquad (11) \\ & shared_n : \{\text{real-value, text}\}\ \} \end{array}$$

***Flows*:** contains a list of flow ids for all flows within the supply network. Each flow id has one child which is the product id. Within any supply network there can be *n* such flows. Thus, we define our *flow* object as,

$$\begin{array}{ll} \{ & fid_1 : pid_1, \\ & fid_2 : pid_2, \\ & \ldots , \qquad (12) \\ & fid_n : pid_n\ \} \end{array}$$

***rootService*:** is the id of the root service being evaluated. It is defined the same as in equation (3).

***Services*:** are defined as follows for *composite* and *atomic* services:

$$\begin{array}{ll} \{ & \text{type: `composite'} \\ & \text{inFlow: <set of inflow ids>} \\ & \text{outFlow: <set of flow ids>} \\ & \text{subServices: <set of sub-service ids> } \} \end{array} \qquad (13)$$

$$\begin{array}{ll} \{ & \text{type: `atomic'} \\ & \text{inFlow: <set of inflow ids>} \\ & \text{outFlow: <set of flow ids>} \qquad (14) \\ & \text{ppu\_info: <set of model pricing data>} \\ & \text{qtyInPer1Out: <set of flow ids> } \} \end{array}$$

type indicates if a service is a composite or non-composite.

inFlow and *outFlow* are lists of unique ids indicating which products were consumed and produced during manufacturing. An illustration of the supply flows is illustrated in Section 2. In addition, each element holds a feature labeled *lb* and *units*. *lb* is a real-valued number giving our low bound on the flow. *units* refers to the units the quantity is measured in.

subServices are defined the same way as in equation (5) and only appear in *composite* services.

ppu_info is an n-tuple object with *n* different products being product by the service. For each product produced, a 2-tuple object containing the type of model that should be used for calculating the price per unit and the pricing model's features. These features correspond to each of the products' properties listed in the *products* object. *ppu_info* is only present for *atomic* services.

qtyInPer1Out is an n-tuple object with *n* different products being produced by the service. For each product produced, a product and quantity is listed which indicates the amount required to product one unit of output. This object is only present in services with the type manufacturer.

orders is an n-tuple object with *n* different products being transported by the service. For each product transported, the following data structure is present:

$$\begin{aligned}\{\quad & \text{in: fid} \\ & \text{out: fid} \\ & \text{sender: sid} \\ & \text{recipient: sid} \\ & \text{qty: } \{\text{real value}\}\ \}\end{aligned} \tag{15}$$

where *in* is the flow id of the incoming product to be transported, *out* is the flow ids after the product has been transported, *sender* is the service id of the sender, *recipient* is the service id of the recipient of the product, and *qty* is the amount of product within the shipment.

***Products*:** are defined as follows. Each product p_j in the set P is labeled as either a *basic* or *assembled* product. *Assembled* products are constructed from other products within the supply network and are labeled *assembled* under *type*. *Assembled* products are also given an additional feature that *basic* products do not have; *components*.

Components are lists of product names that are used as input material. Each element in this object contains a value x $\in \mathbb{N}$ which indicates the quantity of that component required for assembly.

Constraints are restrictions on the physical properties of the product. This can either be a real-valued tolerance level of a physical property (like weight or height) or a limitation on the types of materials the product is made out of (like aluminum or carbon fiber).

In addition, *products* has an entry for each physical *property* pertinent for analysis. Our *products* object is defined by the following n-tuple:

$$\begin{aligned}\{\quad & \text{type: basic or assembled} \\ & \text{components: list} \\ & constraint_n : \{\text{real-value, list}\} \\ & constraint_n : \{\text{real-value, list}\} \\ & property_n : \{\text{real-value, text}\}\ \}\end{aligned} \tag{16}$$

4.4 Analytic Model: Computing Output from Input

In this Section we describe the computation of the valid output from a valid input. We describe the computation top-down, recursively, starting with the overall service network analytic model, followed by the composite service, atomic services, composite (assembled) product model, and finally the atomic product models.

***M*odel Output.** The output o of the AM, described in (3) is computed from the input i, described in (10), as follows. Let $services = o(services)$, $products = o(products)$ defined later in this section. Let $rootid = i(rootService)$ and $productIds = keys(i(products))$. Then the values for each metric m_i, $i = 1,...,n$, in the output o is taken from the root service:

$$o(m_i) = services(rootid)(m_i)$$

The constraints are given by

$$constrains = services(rootid)(constraints) \wedge productConstraints$$

where $productConstraints =$

$$\forall sp \in productIds : products(sp)(constraints)$$

Computation of Composite Service. Here we describe the computation of valid output for a composite service in equation (5). Let s be an id of a service in *services* in equation (4). Then, $services(s)$ is a set of key-value pairs in equation (5). The value for key *type* is $services(s)(type)$ = "*composite*". The value $services(s)(m_i)$ for each metric m_i, $i = 1,...,n$ is given by:

$$services(s)(m_i) = \sum_{ss \in SS} service(ss)(m_i)$$

where $SS = services(s)(subServices)$ is the set of sub-service ids.

To describe the computation of *inFlow*, *outFlow*, and *constrains* for service *s*, we first define a number of concepts.

Let

$$serviceFlows = keys(services(s)(inFlow)) \cup keys(services(s)(outFlow)) \quad (17)$$

$$subServiceFlows = \cup_{ss \in SS} keys(services(ss)(inFlow)) \cup \cup_{ss \in SS} keys(services(ss)(outFlow)) \quad (18)$$

$$allFlows = serviceFlows \cup subServicesFlows \quad (19)$$

$$internalFlows = subServicesFlows - serviceFlows \quad (20)$$

be the service flows, the sub-services' flows, all flows, and the internal only flows respectively. For every flow $f \in allFlows$, we define the internal supply of flow f as:

$$internalSupply(f) = \sum_{ss \in SS} outQty(ss, f) \quad (21)$$

where

$$outQty(ss, f) = service(ss)(outFlow)(f)(qty) \quad (22)$$

if $f \in keys(service(ss)(outFlow))$, and 0 otherwise.

Similarly, $internalDemand(f)$ is computed by replacing the key "*outFlow*" with the key "*inFlow*" in equation (21).

The following is the balancing constraint for flow supply and demand, as follows:

$$internalSupplySatisfiesDemands = \forall f \in internalFlows : internalSupply(f) \geq internalDemand(f) \quad (23)$$

Let $inFlowIds = keys(i(services)(s)(inFlow))$ and $outFlowIds = keys(i(services)(s)(outFlow))$ be the inFlow and outFlow ids of service *s*. We define $inFlow = o(services)(s)(inFlow)$ by

$$\forall f \in inFlowsIds : inFlow(f) = internalDemand(f) - internalSupply(f) \quad (24)$$

Similarly, $outFlow = o(services)(s)(outFlow)$ is defined by

$$\forall f \in outFlowsIds : outFlow(f) = internalDemand(f) - internalSupply(f) \quad (25)$$

The *inFlowConstraints* and *outFlowConstraints* are bound constraints given by

$$inFlowConstraints = flowBoundConstraints(inFlow', inFlow) \quad (26)$$

$$outFlowConstraints = flowBoundConstraints(outFlow', outFlow) \quad (27)$$

where $inFlow' = i(services)(s)(inFlow)$ and $outFlow' = i(services)(s)(outFlow)$, respectively taken from the input *i*. The predicate $flowBoundConstraints(flowBounds, flow)$ is given by

$$\forall f \in flows : flows(f)(qty) \geq 0 \ \wedge \ flows(f)(qty) \geq flowBounds(f)(lb) \quad (28)$$

We define *subServiceConstraints* recursively as

$$subServiceConstraints = \forall ss \in services(s)(subServices) : services(ss)(constraints) \quad (29)$$

We now define the overall *constraints* is the conjunction:

$$constraints = internalSupplySatisfiesDemand \wedge inFlowConstraints \wedge outFlowConstraints \wedge subServiceConstraints \quad (30)$$

Finally, define *subService* to be a copy of the *subService* structure defined in (13).

Computation of Atomic Services. For every atomic service with id *s*, a valid output $services(s)$ is of the form described in Structure (7). The extensible library of atomic services initially includes models for *suppliers*, *manufacturers* and *transportation* as follows.

Suppliers: The type $services(s)(type) =$ "*Supplier*". The cost $services(s)(cost)$ is computed as

$$\sum_{o \in outFlows} computePPU(products, ppu\_info) * \text{outflow(o)(qty)} \quad (31)$$

where *products* and *ppu_info* are equal to $products(flows(o))$ and $ppu\_info(flows(o))(product\_ppu)$, *flows* and *products* are from the input Structure (10). The function *computePPU* is a function which produces the price-per-unit of a product as determined by the business operational parameters defined in *ppu_info*, the products physical properties defined in equation (9), and the products required performance thresholds also defined in equation (9).

We now define the value for $inFlow$ and $outFlow$. For our *Supplier* type atomic service, it is assumed that the supplier does not require any products from any other service to produce output. Services that do require input from other services within the supply chain are defined as $Manufactures$.

The inflow $services(s)(inFlow)$ is defined as the empty set of key-value pairs:

$$inFlow(qty) = \{\} \tag{32}$$

Let $outFlow' = i(services)(s)(outFlow)$ where i is the input and $flows = i(flows)$ as defined as in the data structure (10). Then, the outflow $services(s)(outFlow)$ is defined as follows:

$$\begin{aligned}&\forall o \in keys(outFlow')\\ &\quad outFlow(o)(qty) = outFlow'(o)(qty)\\ &\quad outFlow(o)(item) = flows(o)\end{aligned} \tag{33}$$

We now define $constrains$ for a valid supplier atomic service output.

$$\begin{aligned}&constraints =\\ &\quad inFlowConstraints \wedge\\ &\quad outFlowConstraints\end{aligned} \tag{34}$$

The $inFlowConstraint$ and $outFlowConstraint$ values are defined as:

$$flowBoundConstraints(inFlow', InFlow) \tag{35}$$

$$flowBoundConstraints(outFlow', OutFlow) \tag{36}$$

where the function $flowBoundConstraints$ is as defined as before in equation (28), $inFlow'$ and $outFlow'$ are defined in the same fashion as equation (26) and (27).

Manufactures: The type $services(s)(type)$ = "$Manufacturer$." The value for cost $services(s)(cost)$, outflow $services(s)(outFlow)$, and constraints $services(s)(constrains)$ are computed as described in equations (31), (33), and (34) respectively.

To compute inFlow, let $qtyInPer1out$ be $i(services)(s)(qtyInPer1out)$, $outflow$ be $services(s)(outFlow)$, and $inFlow'$ be $i(services)(s)(inFlow)$ where i is the input described in data structure (14).

Then, the inflow $services(s)(inFlow)$ is computed as follows:

$$\begin{aligned}&\forall f \in keys(inFlow'):\\ &services(s)(inFlow)(f) =\\ &\qquad qtyInPer1out(f) * outflow(f)(qty)\end{aligned} \tag{37}$$

Transportation: The type $services(s)(type)$ = "$Transportation$."

Let $inFlow'$, $outFlow'$, $orders$ and $flows$ be $i(services)(s)(inFlow)$, $i(services)(s)(outFlow)$, $i(services)(s)(orders)$ and $i(flows)$ respectively, where i is the input described in Structure 10. Then, the inflow $services(s)(inFlow)$ is defined as follows:

$$\begin{aligned}&\forall f \in inFlow':\\ &\quad inFlow(f)(qty) = \textstyle\sum_{o \in relevantOrders(f)} o(qty)\\ &\quad inFlow(f)(item) = flows(f)\end{aligned} \tag{38}$$

where $inFlow = services(s)(inFlow)$ and $relevantOrders(f) = \{o \in orders \mid o(in) = f\}$. Similarly, outFlow $services(s)(outFlow)$ is defined as follows:

$$\begin{aligned}&\forall f \in outflow':\\ &\quad outFlow(f)(qty) = \textstyle\sum_{o \in relevantOrders(f)} o(qty)\\ &\quad outFlow(f)(item) = flows(f)\end{aligned} \tag{39}$$

where $outFlow = services(s)(outFlow)$ and $relevantOrders(f) = \{o \in orders \mid o(out) = f\}$.

The value for $constrains$ is defined in equation (34) with the flow objects defined in the equations listed above for $inflow$ and $outflow$.

Computation of Assembled Products. Here we describe the computation of valid output for a assembled product in equation (9). Let p be an id of a product defined in equation (16). The, $products(p)$ is a set of key-value pairs in equation (9). The value for key $type$ is $products(p)(type)$ = "assembled". The value $products(p)(m_i)$ for each metric m_i, i = 1, ...,n is given by:

$$products(p)(m_i) = \sum_{c \in C} products(c)(m_i)$$

where $C = products(p)(components)$ is the set of product ids.

Let m_i be a metric calculated in equation (9) for product p. Let lb and ub be the lower-bound and upper-bound for metric m_i. We define the value for $constraints_i$ in the following way:

$$products(p)(lb) \leq m_i \leq products(p)(ub)$$

where $product$ is from the input data structure defined in equation (16).

Computation of Basic Products. The output returned for the computation of basic products is copy of the input described in equation (14).

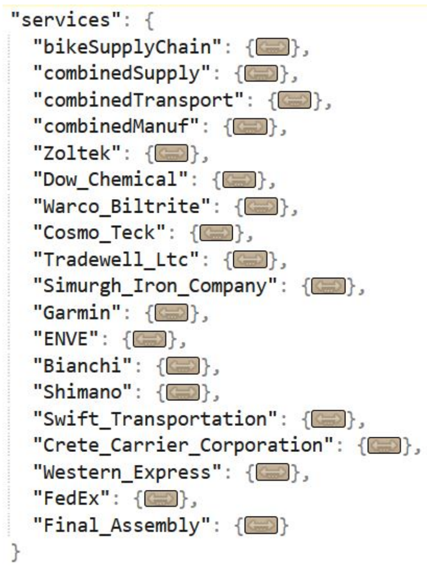
5 SPOT METHODOLOGY BY EXAMPLE

In this Section, we demonstrate how a user can use *SPOT* to optimize over products and services to generate the supply chain for a Virtual Bicycle described in Section 2. The demonstration is organized as follows: (1) a general description of valid input construction, (2) selection of a valid solver, (3) submitting the valid input to SPOT, (4) interpreting the results.

The first task to be completed is the construction of a valid input JSON file. Requirements for this task include a JSON editor, domain knowledge of supply chain network (e.g. Figure **??**) or access to a stored repository which contains the structure of the supply chain network, knowledge of manufactures pricing models or access to the stored repository of manufacturers pricing, and miscellaneous documents required to create a valid input data structure described in Section 4.3.

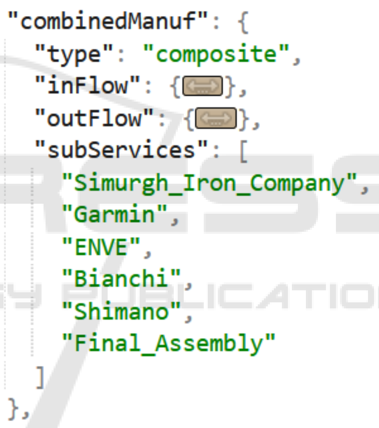
The user will create a JSON object similarly to the one defined in the Data Structure (10). Using a graphic similar to Figure (2), a hierarchical relationship between the different service levels should be created. A the top level (e.g. L0), the *rootService* should be given a name that is demonstrative of the supply chain's purpose. For our example our *rootService* is given the value "bikeSupplyChain". Next, the user will populate the composite service "bikeSupplyChain" in the object *services*. This is done because SPOT views the top-level domain as a composite service. Thus, the *type* for "bikeSupplyChain" is equal to "composite". The object *inFlow* should be an empty object because each of our initial services is a supplier. The outFlow object contains our end product fid (e.g. bike01). Our subService object contain the sids of the service that comprise of the bikeSupplyChain. In our case, we have three services: combinedSupply, combinedManufacturing, and combinedTransportation.

The next level in our services model is L1 which comprise of each of the composite service which make up bikeSupplyChain. For each composite service the user should populate the JSON object with the Structure defined in Data Structure (13) making special care to fill in the subServices object with the desired hierarchical service network model as illustrated in Figure (4). Using a graphic similar to Figure (**??**) each service's flow object should be populated with the correct flows described in Data Structure (12). This process continues for until each composite and atomic service has been added to the service object described in Section (4.3).



```
"services": {
  "bikeSupplyChain": {⟺},
  "combinedSupply": {⟺},
  "combinedTransport": {⟺},
  "combinedManuf": {⟺},
  "Zoltek": {⟺},
  "Dow_Chemical": {⟺},
  "Warco_Biltrite": {⟺},
  "Cosmo_Teck": {⟺},
  "Tradewell_Ltc": {⟺},
  "Simurgh_Iron_Company": {⟺},
  "Garmin": {⟺},
  "ENVE": {⟺},
  "Bianchi": {⟺},
  "Shimano": {⟺},
  "Swift_Transportation": {⟺},
  "Crete_Carrier_Corporation": {⟺},
  "Western_Express": {⟺},
  "FedEx": {⟺},
  "Final_Assembly": {⟺}
}
```

Figure 3: Valid Service Input Data Structure.



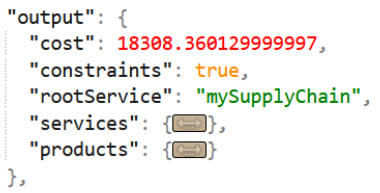
```
"combinedManuf": {
  "type": "composite",
  "inFlow": {⟺},
  "outFlow": {⟺},
  "subServices": [
    "Simurgh_Iron_Company",
    "Garmin",
    "ENVE",
    "Bianchi",
    "Shimano",
    "Final_Assembly"
  ]
},
```

Figure 4: Valid Sub-Service Input Data Structure.

Next, we must analyze our objective function in relation to the constraints and identify if we should use a linear or non-linear solver. The benefits of using a linear solver over a non-linear solver is improved running time. In our case, we find that our objective function is a linear programming problem.

We submit the input data to SPOT by specifying the model, input data, objective function, constraints, the problem type, the solver, and debug flag. The answer to the optimization problem is reported back as a JSON file as described in Section (4.2).

The output in Figure (5) is a valid output data structure from SPOT. The metrics for the optimal selection of services and products are displayed at each of level of the hierarchical supply chain network. Figure (5) displays the top-level metrics. In addition, the Boolean value of the constraints are displayed. Traversing the output down service will reveal each



```
"output": {
  "cost": 18308.360129999997,
  "constraints": true,
  "rootService": "mySupplyChain",
  "services": {⟺},
  "products": {⟺}
},
```

Figure 5: Valid Output Data Structure.

of the metrics for the Sub-Service which comprise of the bikeSupplyChain.

6 CONCLUSION AND FUTURE RESEARCH

This research is the first step toward developing a decision guidance system for combined optimization over product design, process design, and supply chain management all while keeping the lower level mathematical programming code hidden from the domain users. The prototype, SPOT, was successfully tested on a virtual bicycle product and service network. The methodology in creating hierarchical data input, which describes the assembly of product and services using the described input data structure, successfully generated a valid output data structure with the correct optimal solution. This modular composite of the life cycle of product design is unique in the fact that it joins the optimization of traditionally silo'd project spaces and adds to the agility of realizing product and services using distributed manufacturing capacity. Many research questions remain open. Future research directions include expanding the functionality of SPOT, integrating the application with existing design tools such as CAD and CAM, creating a graphical user interface to design the hierarchical relationships.

REFERENCES

Towards agile engineering of mechatronic systems in machinery and plant construction.

Brodsky, A., Gingold, Y., LaToza, T. D., Yu, L.-F., and Han, X. (2021). Catalyzing the agility, accessibility, and predictability of the manufacturing-entrepreneurship ecosystem through design environments and markets for virtual things. In *10th Intern. Conf. on Operations Research and Enterprise Systems (ICORES-2021)*, pages 264–272.

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., Krishnamoorthy, M., Nachawati, M. O., Bernstein, W. Z., and Menascé, D. A. (2017). Manufacturing and contract service networks: Composition, optimization and tradeoff analysis based on a reusable repository of performance models. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1716–1725.

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.

Denkena, B., Shpitalni, M., Kowalski, P., Molcho, G., and Zipori, Y. (2007). Knowledge management in process planning. *CIRP Annals*, 56(1):175 – 180.

Eddy, D. C., Krishnamurty, S., Grosse, I. R., Wileden, J. C., and Lewis, K. E. (2015). A predictive modelling-based material selection method for sustainable product design. *Journal of Engineering Design*, 26(10-12):365–390.

Garcia, D. J. and You, F. (2015). Supply chain design and optimization: Challenges and opportunities. *Computers & Chemical Engineering*, 81:153 – 170. Special Issue: Selected papers from the 8th International Symposium on the Foundations of Computer-Aided Process Design (FOCAPD 2014), July 13-17, 2014, Cle Elum, Washington, USA.

Helu, M. and Hedberg, T. (2015). Enabling smart manufacturing research and development using a product lifecycle test bed. *Procedia Manufacturing*, 1:86 – 97. 43rd North American Manufacturing Research Conference, NAMRC 43, 8-12 June 2015, UNC Charlotte, North Carolina, United States.

Herrmann, J., Cooper, J., Gupta, S., Hayes, C., Ishii, K., Kazmer, D., Sandborn, P., and Wood, W. (2004). New directions in design for manufacturing. volume 3.

Klingstam, P. and Gullander, P. (1999). Overview of simulation tools for computer-aided production engineering. *Computers in Industry*, 38(2):173 – 186.

McLean, C. and Shao, G. (2003). Manufacturing case studies: Generic case studies for manufacturing simulation applications. In *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*, WSC '03, page 1217–1224. Winter Simulation Conference.

Molcho, G., Zipori, Y., Schneor, R., Rosen, O., Goldstein, D., and Shpitalni, M. (2008). Computer aided manufacturability analysis: Closing the knowledge gap between the designer and the manufacturer. *CIRP Annals*, 57(1):153 – 158.

Salvendy, G. (2001). *Handbook of Industrial Engineering: Technology and Operations Management*.

Shehadi, A. I.-H. S. (2019). On the move: manufacturing's return to the developed world. *FDiIntelligence*.

Stoll, H. W. (1986). Design for Manufacture: An Overview. *Applied Mechanics Reviews*, 39(9):1356–1364.

Wu, D., Greer, M. J., Rosen, D. W., and Schaefer, D. (2013). Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing Systems*, 32(4):564 – 579.