Manufacturing and Contract Service Networks: Composition, Optimization and Tradeoff Analysis based on a Reusable Repository of Performance Models

Alexander Brodsky[1], Mohan Krishnamoorthy[1], M. Omar Nachawati[1], William Z. Bernstein[2], Daniel A. Menascé[1]
[1]*Department of Computer Science, George Mason University, Fairfax, Virginia, USA*
[2]*Systems Integration Division, NIST, Gaithersburg, Maryland, USA*
Email: [1]{brodsky, mkrishn4, mnachawa, menasce}@gmu.edu, [2]wzb@nist.gov

***Abstract*— In this paper we report on the development of a software framework and system for composition, optimization and trade-off analysis of manufacturing and contract service networks based on a reusable repository of performance models. Performance models formally describe process feasibility constraints and metrics of interest, such as cost, throughput and CO_2 emissions, as a function of fixed and control parameters, such as equipment and contract properties and settings. The repository contains performance models for (1) unit manufacturing processes, (2) base contract services, and (3) a composite steady-state service network. The proposed framework allows process engineers to (1) hierarchically compose model instances of service networks, which can represent production cells, lines, factory facilities and supply chains, and (2) perform deterministic optimization based on mathematical programming and Pareto-optimal trade-off analysis. We case study the framework on a service network for a heat sink product which involves contract vendors and manufacturers, unit manufacturing process services including cutting/shearing and Computer Numerical Control (CNC) machining with milling and drilling steps, quality inspection, finishing and assembly.**

***Keywords*-performance models; model repository; optimization; decision support; service composition**

I. 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" [1]. One method for realizing a smart manufacturing ecosystem is to employ an over-arching cloud-based infrastructure that supports decision making. Such infrastructure must support a wide variety of analytical tasks across the entire organizational hierarchy, including manufacturing units, cells, production lines, factories, and supply chains [2]. 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" [3]. 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 [3].

We now borrow from Brodsky et al. [4] and Brodsky et al. [5] to describe the limitations of today's technology to achieve the required analysis and optimization capabilities, which can be broadly classified as descriptive ("what happened?"), diagnostic ("why did it happen?"), predictive ("what will happen?"), and prescriptive ("how can we make it happen?"). Descriptive analytics typically involves manipulation of streams of data at different aggregation levels, continuously over time. Some examples of descriptive analytics and its associated data can be found in Richardson [6] and Ündey et al. [7]. Diagnostic analytics includes such tasks as continuous testing for a significant statistical difference between the estimated values compared against observations, and direct application of fault classifiers to detect failures on the manufacturing floor. Research in diagnostic analytics in manufacturing can be found in Souza [8] and Shao et al. [9]. Predictive analytics involves techniques from queuing theory, statistical learning for regression and classification, and stochastic simulation for what-if estimation. Examples of predictive analytics for manufacturing can be found in Lechevalier et al. [10] and Shin et al. [11]. Prescriptive analytics typically involves decision optimization techniques, such as mathematical and constraint programming. Examples of research in prescriptive analytics in manufacturing can be found in Lee et al. [12] and Brodsky et al. [13].

Today, analysis and optimization solutions for manufacturing are typically implemented from scratch, following a linear methodology [14, 15]. This leads to high-cost and long-duration development as well as results in models and algorithms that are difficult to modify, extend, and reuse. A key contributor to these deficiencies is the diversity of computational tools, each designed for a different task such as data manipulation, statistical learning, data mining, optimization, and simulation. Because of this diversity, modeling using computational tools often requires the use of specialized low-level mathematical abstractions. As a result, the same manufacturing knowledge is often modeled multiple times using different specialized abstractions, instead of being modeled once using a uniform abstraction. Furthermore, the modeling expertise required for the low-level abstractions and languages is typically not within the realm of knowledge of manufacturing users.

Recently proposed Brodsky et al. [4] was an architectural design for the fast development of software solutions for de-

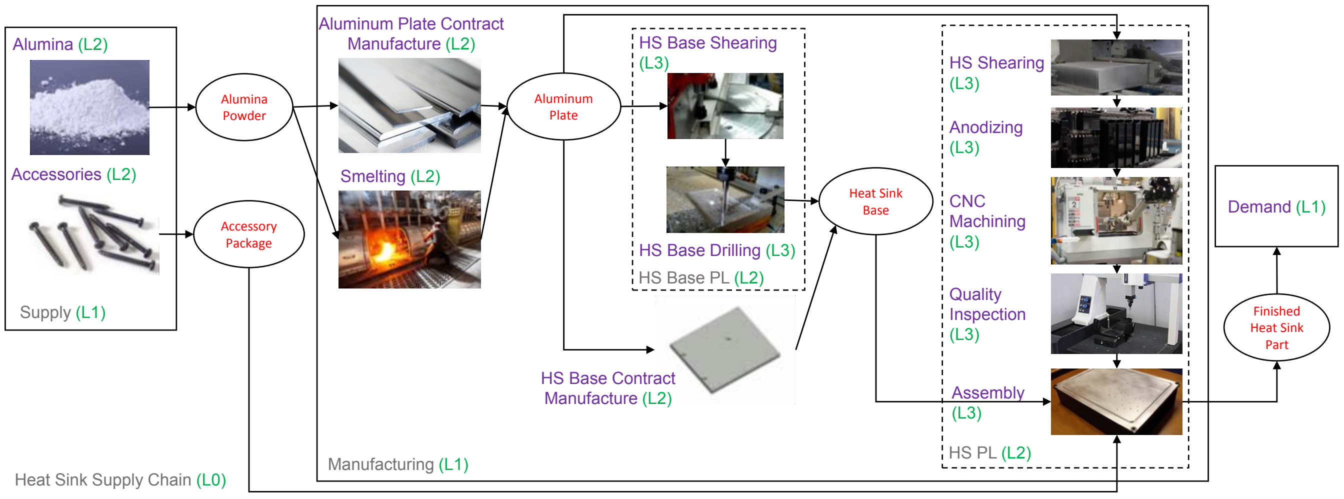


Figure 1. Graphical representation of the Heat Sink service network. Each labeled circle relates to a physical good to be procured. All images represents individual activities. Arrows represent a possible path for procuring a physical good or meeting a specified demand. L# refers to the hierarchical level of the service network composition.

scriptive, diagnostic, predictive, and prescriptive analytics of dynamic production processes. The uniqueness and novelty of the architecture was its middleware layer, which is based on a reusable, modular, and extensible knowledge base (KB) of process performance models. However, Brodsky et al. [4] lacked a systematic design of the unit manufacturing process (UMP) repository and its architecture, and an ecosystem around it. Furthermore, the UMP models were abstracted in Brodsky et al. [4] by piecewise-linear functions whereas real-world process models, which are typically physics-based, require non-linear arithmetic.

To address these limitations, reported in Brodsky et al. [5] was the the development of a software architecture for (1) a reusable repository of UMP performance models, and (2) their analysis and optimization using the Unity Decision Guidance Management System (DGMS). Unity DGMS automatically generates low-level mathematical programming optimization models from simulation-like performance models. The architecture was demonstrated using the case of Computer Numerical Control (CNC) machining. However, Brodsky et al. [5] did not address an important problem of how to compose reusable UMP models into a hierarchy of service networks, including production cells, lines, factories and supply chain. This is exactly the focus of this paper.

More specifically, the contributions of this paper are twofold. First, we extend the functionality of the system developed in Brodsky et al. [5] with performance models for a steady-state service network and a number of base contract services. The extended system allows hierarchical composition of service networks based on a reusable model repository for (1) production services, such as UMPs, assembly and inspection, and (2) contract services, such as vending, manufacturing, packaging, repair and transportation. The uniqueness of the developed system is in its ability to perform service network optimization and Pareto trade-off analysis without the need to manually craft mathematical programming models, all while achieving the quality of optimization results and computational efficiency of mathematical programming solvers, which significantly outperform simulation-based solvers (for e.g., see Amaran et al. [16] and Klemmt et al. [17]). This capability is achieved through the use of Unity DGMS. Second, we conduct a case study of the service network of a large Heat Sink product, and demonstrate its optimization and Pareto trade-off analysis of cost vs. carbon emissions per part.

The rest of the paper is organized as follows. Section II presents the concept of manufacturing and contract service networks, and outlines its possible ecosystem and workflows. Section III presents the system functionality through a case study of a service network of a Heat Sink product. Section IV overviews the software architecture for the system. Section V presents the service network performance model and explains its structure. Section VI concludes with some future research directions.

II. Service Networks: Ecosystem and Workflows

In our proposed framework, we represent the manufacturing process as a network of service-oriented components that are linked together to produce products or product service systems (PSS) to meet some demand. In the performance model knowledge base, we adopt the term *Service Network* to represent this network. Service network stakeholders include both manufacturers and suppliers [18]. In this paper, the following terms are used to describe the various components of a service network.

- **Vendor**: describing an organization that provides a finished product, e.g. raw material, bar stock, or fully realized components.
- **Contract Manufacturer**: describing an organization that provides a manufacturing service, e.g. precision welding, mold-making, and precision machining.
- **Internal Manufacturer**: describing an internal activity "controlled" by the original equipment manufacturer (OEM) of the PSS.
- **Production Line**: describing a chain of Internal Manufacturer activities. It is assumed that a production line is also "controlled" by the OEM.

Using these terms, we can describe various decision paths to realize the delivery of the PSS based on its demand. Fig. 1 provides an example of a service network for a Heat Sink Assembly. This service network was derived to the authors' best knowledge borrowing parts of its configuration from literature [19]. Within the service network, each activity and physical good is represented by an image and a labeled circle, respectively. Vendors can provide raw material, i.e. *Alumina Powder*, or the finished components, i.e. *Accessory Package*. As an example of a contract manufacturer, the *HS Base Contract Manufacture* provides the service of machining a part for the product's final assembly. Within service networks, it is likely that there a multiple paths to successfully provide the same physical good. Often, this is referred to as a multi-echelon supply chain [20]. Procuring the *Heat Sink Base* presents an example of such a situation. The *Heat Sink Base* can either be provided by a contract manufacturer or it can be provided by the OEM's own production line, shown in the dotted box in the middle of the diagram. This production line includes two unit manufacturing processes, namely shearing and drilling. Another such example is the procurement of the *Aluminum Plate*, wherein the OEM operates its own smelting plant or the aluminum bar stock is cut to specifications by another contract manufacturer. The Heat Sink's service network culminates in a relatively complex production line that includes five activities, namely shearing, anodizing, CNC machining, quality inspection, and final assembly.

After formally characterizing the service network, performance models can be constructed that represent each activity depending on its individual characteristics. More information on what parameters of each performance model (PM) can be tuned to find optimal settings for the network are explained in the following section. The concept of representing each activity as a PM introduces the possibility of posing what-if questions, optimizing activity-specific parameters to meet some global objectives, and incorporating advanced analytics while lessening the barrier to access such analysis for domain experts, e.g. process engineers and operation managers.

The idea of invoking composite services to form a network of services in a programmatic way through web services was discussed in Menasce [21]. This along with the work in Menasce [22] discuss how global performance metrics can be computed when analyzing a network of services. The structure of the network and the specific performance characteristics of individual services, specified in terms of contracts, are necessary to compute composite performance metrics.

III. System Functionality and a Case Study

In this section we describe the developed system functionality for the process engineer role, using a case study of the Heat Sink service network depicted in Fig. 1. The key concept in the system is that of a performance model, for a service network and its base service components, which we describe in detail in Section 5. Generally, a PM formally describes process feasibility constraints and metrics of interest, such as cost, throughput and CO_2 emissions, as a function of fixed and control parameters, such as equipment and contract properties and settings. For example, in the Heat Sink service network in Fig. 1, fixed parameters include the type of material used, its dimensions, the number of holes in *HS Base Drilling*, depth of cut in *CNC Machining* etc. On the other hand, the control parameters include the cutting speed in *HS Shearing*, cutting speed in *HS Base Drilling*, the amount produced by the *Aluminum Plate Contract Manufacture* etc.

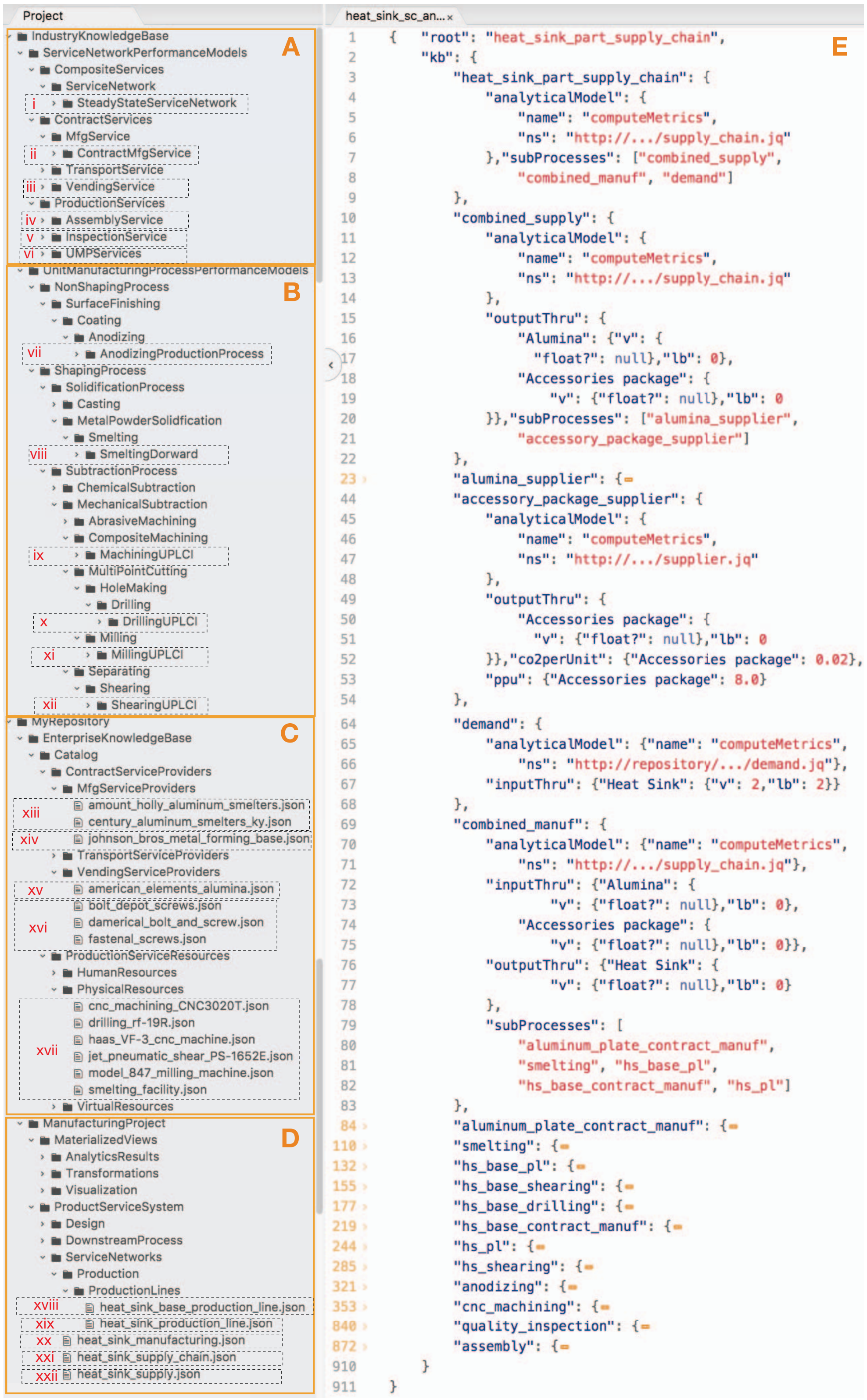
To explain the system optimization functionality in this section, we use the scenario of optimizing the Heat Sink service network shown in Fig. 1. The optimization problem for this scenario is: find all the control parameters of the service network so as to minimize the total cost of the service network operation subject to the satisfaction of all feasibility constraints (of the service network and its components, including UMPs involved.).

To perform optimization and other types of analysis over service networks as the one shown in Fig. 1, a process engineer needs to create a specific instance for the PM input containing fixed and control parameters of the process, but where the control parameters are annotated as unknowns to be found by the system.

The current user interface for a process engineer is through Atom Studio[1], which is an integrated development environment (IDE). However, Atom Studio is not designed for application end users, for whom a web-based user interface may be developed in the future. Fig. 2 shows an Atom window that a process engineer uses to create instances of PM input.

The left pane of Fig. 2 shows an Atom Studio folder tree view with the following key folders: (1) an industry KB containing service network PMs (Fig. 2A) and UMP PMs

[1]https://atom.io/



```
IndustryKnowledgeBase
  ServiceNetworkPerformanceModels
    CompositeServices
      ServiceNetwork
i       SteadyStateServiceNetwork
    ContractServices
      MfgService
ii      ContractMfgService
      TransportService
iii   VendingService
    ProductionServices
iv    AssemblyService
v     InspectionService
vi    UMPServices
  UnitManufacturingProcessPerformanceModels
    NonShapingProcess
      SurfaceFinishing
        Coating
          Anodizing
vii         AnodizingProductionProcess
    ShapingProcess
      SolidificationProcess
        Casting
        MetalPowderSolidfication
          Smelting
viii        SmeltingDorward
      SubtractionProcess
        ChemicalSubtraction
        MechanicalSubtraction
          AbrasiveMachining
          CompositeMachining
ix          MachiningUPLCI
          MultiPointCutting
            HoleMaking
              Drilling
x               DrillingUPLCI
            Milling
xi            MillingUPLCI
          Separating
            Shearing
xii           ShearingUPLCI
MyRepository
  EnterpriseKnowledgeBase
    Catalog
      ContractServiceProviders
        MfgServiceProviders
xiii      amount_holly_aluminum_smelters.json
          century_aluminum_smelters_ky.json
xiv       johnson_bros_metal_forming_base.json
        TransportServiceProviders
        VendingServiceProviders
xv        american_elements_alumina.json
xvi       bolt_depot_screws.json
          damerical_bolt_and_screw.json
          fastenal_screws.json
      ProductionServiceResources
        HumanResources
        PhysicalResources
xvii      cnc_machining_CNC3020T.json
          drilling_rf-19R.json
          haas_VF-3_cnc_machine.json
          jet_pneumatic_shear_PS-1652E.json
          model_847_milling_machine.json
          smelting_facility.json
        VirtualResources
  ManufacturingProject
    MaterializedViews
      AnalyticsResults
      Transformations
      Visualization
    ProductServiceSystem
      Design
      DownstreamProcess
      ServiceNetworks
        Production
          ProductionLines
xviii       heat_sink_base_production_line.json
xix         heat_sink_production_line.json
xx        heat_sink_manufacturing.json
xxi       heat_sink_supply_chain.json
xxii      heat_sink_supply.json
```

```
{   "root": "heat_sink_part_supply_chain",
    "kb": {
        "heat_sink_part_supply_chain": {
            "analyticalModel": {
                "name": "computeMetrics",
                "ns": "http://.../supply_chain.jq"
            },"subProcesses": ["combined_supply",
                "combined_manuf", "demand"]
        },
        "combined_supply": {
            "analyticalModel": {
                "name": "computeMetrics",
                "ns": "http://.../supply_chain.jq"
            },
            "outputThru": {
                "Alumina": {"v": {
                  "float?": null},"lb": 0},
                "Accessories package": {
                    "v": {"float?": null},"lb": 0
            }},"subProcesses": ["alumina_supplier",
                "accessory_package_supplier"]
        },
        "alumina_supplier": {⋯
        "accessory_package_supplier": {
            "analyticalModel": {
                "name": "computeMetrics",
                "ns": "http://.../supplier.jq"
            },
            "outputThru": {
                "Accessories package": {
                  "v": {"float?": null},"lb": 0
            }},"co2perUnit": {"Accessories package": 0.02},
            "ppu": {"Accessories package": 8.0}
        },
        "demand": {
            "analyticalModel": {"name": "computeMetrics",
                "ns": "http://repository/.../demand.jq"},
            "inputThru": {"Heat Sink": {"v": 2,"lb": 2}}
        },
        "combined_manuf": {
            "analyticalModel": {"name": "computeMetrics",
                "ns": "http://.../supply_chain.jq"},
            "inputThru": {"Alumina": {
                    "v": {"float?": null},"lb": 0},
                "Accessories package": {
                    "v": {"float?": null},"lb": 0}},
            "outputThru": {"Heat Sink": {
                    "v": {"float?": null},"lb": 0}
            },
            "subProcesses": [
                "aluminum_plate_contract_manuf",
                "smelting", "hs_base_pl",
                "hs_base_contract_manuf", "hs_pl"]
        },
        "aluminum_plate_contract_manuf": {⋯
        "smelting": {⋯
        "hs_base_pl": {⋯
        "hs_base_shearing": {⋯
        "hs_base_drilling": {⋯
        "hs_base_contract_manuf": {⋯
        "hs_pl": {⋯
        "hs_shearing": {⋯
        "anodizing": {⋯
        "cnc_machining": {⋯
        "quality_inspection": {⋯
        "assembly": {⋯
    }
}
```

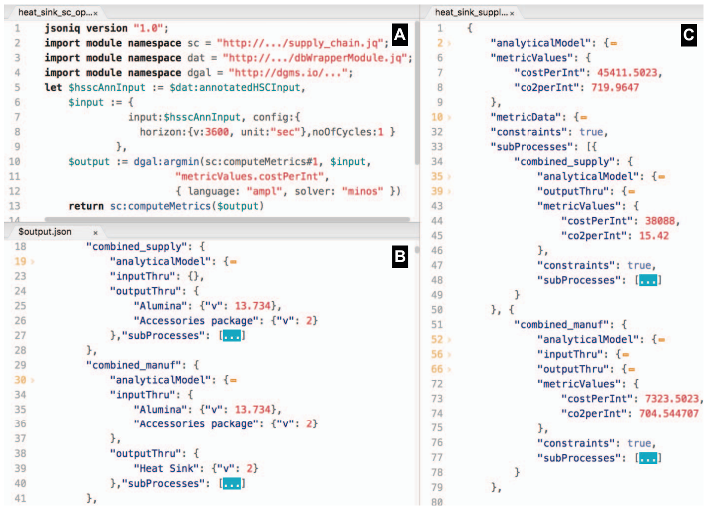
Figure 2. Atom window showing process engineer's view of the system. (A) Industry knowledge base of service network performance models. (B) Industry knowledge base of UMP performance models. (C) Enterprise KB to hold a catalog of variable annotated instances of enterprise artifacts. (D) Process engineer and team's manufacturing project. (E) Snippets of the variable annotated input for the Heat Sink service network.

(Fig. 2B); and (2) a repository for the enterprise KB (Fig. 2C) and a process engineer's manufacturing project (Fig. 2D). The right pane (Fig. 2E) shows the input instance for the Heat Sink service network in Fig. 1. This input instance, which is described in the JavaScript Object Notation (JSON) data format, is created by the process engineer to perform optimization and other types of analysis. We now describe the components in Atom Studio panes in greater detail.

Industry KB: The industry KB contains well-known and validated PMs of UMPs (Fig. 2B) and service network components (Fig. 2A). The UMPs in the industry KB are organized based on the Manufacturing Service Description Language (MSDL) ontology [23]. The process engineer models the Heat Sink service network from Fig. 1 by creating input instances for the PMs stored in the industry KB. For example, the *Smelting*, *HS Shearing*, *Anodizing*, and *CNC Machining* processes in Fig. 1 are instances of the UMP PMs *SmeltingDoward*, *ShearingUPLCI*, *Anodizing-ProductionProcess*, and *MachiningUPLCI* respectively (viii, xii, vii, and ix in Fig. 2B). Similarly the *Alumina*, *HS Base Contract Manufacture*, and *Heat Sink Supply Chain* processes in Fig. 1 are instances of the service network PMs *VendingService*, *ContractMfgService*, and *SteadyState-ServiceNetwork* respectively (iii, ii, and i in Fig. 2A).

Repository KB: The process engineer stores the instances of the PMs from the industry KB into the repository KB. This repository can be divided into enterprise KB (Fig. 2C) and manufacturing project repository (Fig. 2D). The enterprise KB encapsulates the input instances of different UMPs and service network components present in the catalog of the enterprise by capturing their capabilities i.e., capturing instantiated fixed parameters and annotated control parameters of the PM inputs. Such inputs are also called variable annotated inputs. For example the *century_aluminum_smelters_ky* (xiii in Fig. 2C) is a variable annotated input of the contract manufacturing PM, *ContractMfgService* (ii in Fig. 2A). Similarly *bolt_depot_screws* and *cnc_machining_CNC3020T* (xvi, xvii in Fig. 2C) are variable annotated input of the vendor PM *VendingSerive* and the UMP PM of *MachiningUPLCI* respectively (iii in Fig. 2A, ix in Fig. 2B). On the other hand, the process engineer (or the engineer's team) can store a number of different artifacts such as service network compositions, results of the analysis, and other design and downstream processes in the manufacturing project as shown in Fig. 2D. This forms a workspace that the process engineer (or team) can use to store data pertaining to a specific manufacturing project. For instance the process engineer stores the input instance of the production lines and other components of the Heat Sink service network, as well as the results of the optimization problem in the manufacturing project.

Service network composition: The process engineer creates a variable annotated JSON input of the *SteadyStateServiceNetworkProcess* PM (i in Fig. 2A) called



```
jsoniq version "1.0";
import module namespace sc = "http://.../supply_chain.jq";
import module namespace dat = "http://.../dbWrapperModule.jq";
import module namespace dgal = "http://dgms.io/...";
let $hsscAnnInput := $dat:annotatedHSCInput,
    $input := {
              input:$hsscAnnInput, config:{
                horizon:{v:3600, unit:"sec"},noOfCycles:1 }
            },
    $output := dgal:argmin(sc:computeMetrics#1, $input,
                      "metricValues.costPerInt",
                      { language: "ampl", solver: "minos" })
    return sc:computeMetrics($output)
```

```
        "combined_supply": {
            "analyticalModel": {…
            "inputThru": {},
            "outputThru": {
                "Alumina": {"v": 13.734},
                "Accessories package": {"v": 2}
            },"subProcesses": [...]
        },
        "combined_manuf": {
            "analyticalModel": {…
            "inputThru": {
                "Alumina": {"v": 13.734},
                "Accessories package": {"v": 2}
            },
            "outputThru": {
                "Heat Sink": {"v": 2}
            },"subProcesses": [...]
        },
```

```
{
    "analyticalModel": {…
    "metricValues": {
        "costPerInt": 45411.5023,
        "co2perInt": 719.9647
    },
    "metricData": {…
    "constraints": true,
    "subProcesses": [{
        "combined_supply": {
            "analyticalModel": {…
            "outputThru": {…
            "metricValues": {
                "costPerInt": 38088,
                "co2perInt": 15.42
            },
            "constraints": true,
            "subProcesses": [...]
        }
    }, {
        "combined_manuf": {
            "analyticalModel": {…
            "inputThru": {…
            "outputThru": {…
            "metricValues": {
                "costPerInt": 7323.5023,
                "co2perInt": 704.544707
            },
            "constraints": true,
            "subProcesses": [...]
        }
    },
```

Figure 3. (A) JSONiq main to optimize the Heat Sink service network model. (B) JSON snippets of the fully instantiated input for the Heat Sink service network obtained from the *argmin* function. (C) JSON snippets of output metrics and constraints of Heat Sink service network obtained from the *computeMetrcis* function

heat_sink_part_supply_chain to model the service network *Heat Sink Supply Chain (L0)* in Fig. 1. Assuming that the process engineer builds this input instance top-down, the engineer sets the root to *heat_sink_part_supply_chain* and the KB to the knowledge base of sub components of this service network as shown in Fig. 2E. The variable annotated input for composite service network PM such as that of the root process describes the inputs (*inputThru*), outputs (*outputThru*), and sub processes (*subProcesses*) (lines 3-9). The engineer sets the *subProcesses* of the root process to *combined_supply*, *combined_manuf*, and *demand*. These *subProcesses* describe the inputs of the *Supply (L1)*, *Manufacturing (L1)*, and *Demand (L1)* respectively in Fig. 1. The engineer creates the input structures for *combined_supply* (lines 10-22 in Fig. 2E) and *combined_manuf* (lines 69-78 in Fig. 2E) as variable annotated input instance of the *SteadyStateServiceNetworkProcess* PM and includes them into the KB. The engineer also creates a virtual *demand* process and includes it into the KB. (lines 64-68 in Fig. 2E). This process specifies the user-defined (fixed) demand parameter for the fully assembled *Heat Sink*. This value is the minimum number of Heat Sinks that should be produced by the Heat Sink service network.

After setting the top level processes, the engineer continues to build the variable annotated input for lower level components of the Heat Sink service network. The *subProcesses* of the *combined_supply* process is *alumina_supplier* and *acessory_package_supplier* that describe the input to the *Alumina (L2)* and *Accessories (L2)* in Fig. 1. Therefore, next, the engineer either creates a variable annotated input instance of the *VendorService* PM (iii in Fig. 2A) or uses

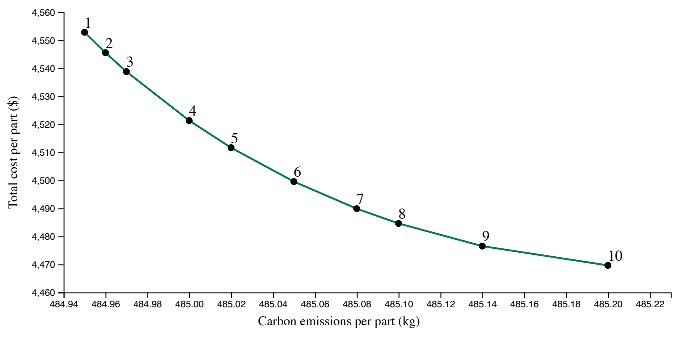


Figure 4. Pareto optimal curve illustrating trade-off between cost and CO_2 emissions per part.

one of the already created vendors from the enterprise KB catalog for eg., see xv, xvi in Fig. 2C, and includes these vendors into the KB as shown in lines 23-54 in Fig. 2E. Similarly, the engineer adds the variable annotated input instance of the *subProcesses* of the *combined_manuf* process by either creating a variable annotated input to the respective PM in the industry KB or by using one of already created instance artifacts from the enterprise KB such as the UMP catalog shown inside xvii of Fig. 2C. Finally, the process engineer stores this variable annotated input for the Heat Sink service network in the manufacturing project as *heat_sink_supply_chain.json* (xxi in Fig. 2D).

Service network optimization: To run the optimization problem of finding the annotated control parameters as to minimize the total cost subject to demand and bound constraints, the engineer writes the JSONiq code as shown in Fig. 3A. This code calls *argmin*, which is an analytical function that solves the deterministic minimization problem. This function takes as argument, the analytical function of the *SteadyStateServiceNetworkProcess* PM called *computeMetrics*. Additionally, the *argmin* function takes the variable annotated input for the Heat Sink service network shown in Fig. 2E, the configuration of the time intervals, the dot separated path of the objective in the output, as well as optimization solver configuration parameters (lines 6-12 in Fig. 3A). The result of *argmin*, if feasible, is a fully instantiated input for the Heat Sink service network where all the annotated control parameters are instantiated to values that minimize the total cost subject to all feasibility constraints. Snippet of the fully instantiated input is shown in Fig. 3B. The output metrics and constraints for the Heat Sink service network can be obtained by running *computeMetrics* over the fully instantiated input as shown in line 13 of Fig. 3A. This output is a JSON structure whose snippet is shown in Fig. 3C.

Trade-off analysis: The process engineer can use the instantiated controls obtained by solving the optimization problem above to set the machines on the manufacturing floor. Additionally, it is also possible to perform trade-off analysis between CO_2 emissions and cost of a part to simulate a real manufacturing scenario through the use of the system. Fig. 4 displays Pareto optimal alternatives for producing the Heat Sink part in terms of cost per part and emissions per part to allow trade-off. The process engineer can use the system described in this section to generate each alternative by solving an optimization problem that minimizes cost per part subject to the corresponding bound to CO_2 emissions. Using the Pareto optimal graph the process engineer can choose the alternative that best suits the objectives of the organization or enterprise that he represents.

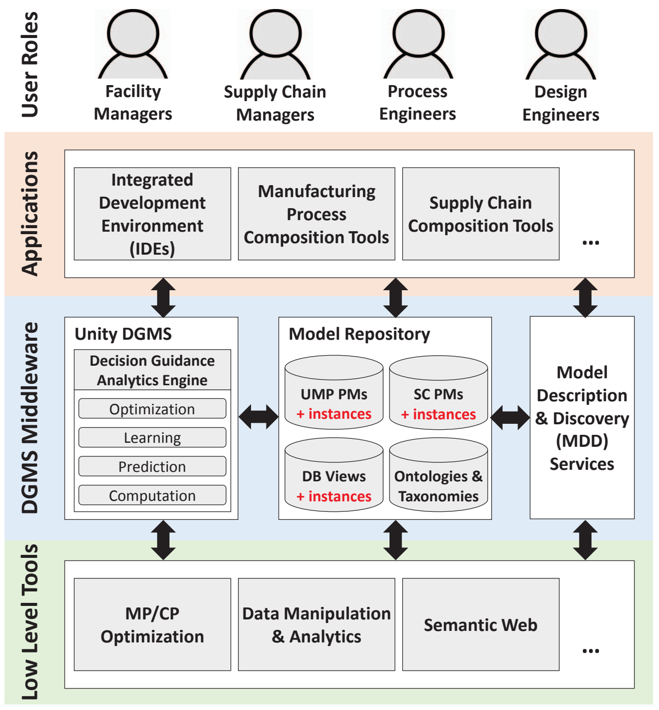


Figure 5. Conceptual overview of the knowledge base housing performance models of unit manufacturing processes (UMP PMs). Adapted from Brodsky et al. [5].

IV. Software Architecture: Reusable Model Repository and Decision Guidance Management System

To implement the proposed system, we adopt the high-level system architecture that we proposed in Brodsky et al. [5], which is based around a reusable model repository and the Unity DGMS [24]. The architecture is depicted in Fig. 5. In this section, we overview this architecture, focusing on the components relevant to the proposed system.

The architecture is designed to support various user roles, as shown in the top-most layer of the diagram in Fig. 5. Among the roles relevant to service network analysis are facility managers, supply chain managers, process engineers, and design engineers. Support for these roles and their respective workflows is handled through a diverse set of high-level tools and external applications.

Situated directly below the user roles layer, the application layer contains both generic and domain-specific user applications that support the variety of workflows performed by the different kinds of users of the system. Currently, the system supports Atom[2] as its primary application interface. Atom is a light-weight, yet highly extensible text editor with a large library of user-contributed plugins.

Moving to the bottom-most layer of the diagram in Fig. 5, the proposed system depends on a number of external and low-level tools to ultimately provide the bulk of its diverse range of capabilities. The categories of low-level tools currently used by the Service Network Analysis system include: (1) solvers for mathematical programming-based optimization, including the IBM CPLEX Optimizer for MILP problems and the MINOS solver for NLP problems, (2) algebraic modeling languages and systems, specifically AMPL and the IBM Optimization Programming Language (OPL), and (3) languages and tools for data manipulation and analysis, primarily the JSONiq language and the Zorba query processor to handle semi-structured JSON data.

Part of the challenge of developing service network analysis systems lies in the complexity of developing high-level tools and applications, which support the different user workflows, from inadequately granular abstractions provided by these low-level tools. Typically, such applications are implemented directly using the low-level tools, from scratch, following a linear development methodology. Furthermore, due to the diversity of low-level tools, applications implemented using one tool are difficult to modify, extend, and reuse with other tools. As a result, the same manufacturing knowledge is often modeled multiple times using different, specialized abstractions, instead of being modeled just once using a single, uniform abstraction.

To overcome these limitations, the architecture is augmented with the DGMS middleware layer, situated between the applications layer and the low-level tools layer in the diagram in Fig. 5. The uniqueness of this solution is that it is centered around a model repository (middle box in the DGMS middleware layer). This provides a uniform, high-level abstraction over different low-level tools and is key in supporting the optimization and trade-off analysis of manufacturing and contract service networks.

Performance models are one of the key artifacts in the model repository. They formally describe process feasibility constraints and metrics of interest (such as cost, throughput and CO_2 emissions) as a function of fixed and control parameters (such as equipment and contract properties and settings).

The model repository for the proposed system contains

[2]https://atom.io/

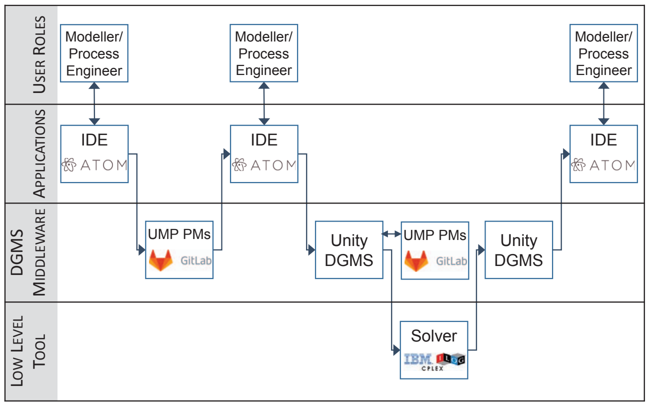


Figure 6. Optimization workflow of performance models in the system, adopted from Brodsky et al. [5].

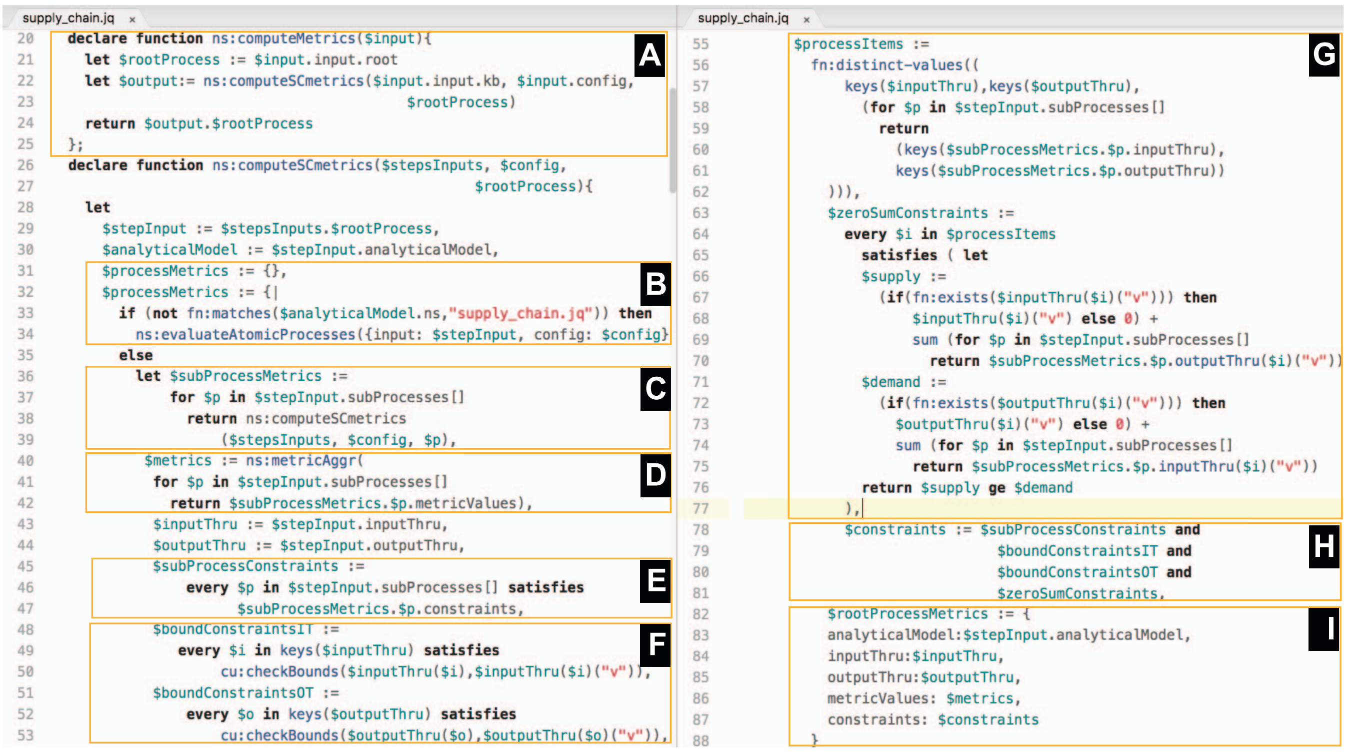
PMs for (1) UMPs, (2) an initial model library of base contract services, and (3) a composite steady-state service network. The repository is also designed to contain data views, ontologies and taxonomies, which have yet to be implemented.

The key technical challenge in realizing a system based on this architecture lies in developing specialized algorithms that automatically translate a uniform, high-level representation of a performance model into the low-level, specialized models required by each of the underlying tools.

The solution to this challenge is based on the Unity DGMS [24], which provides support for different methods of analysis, including optimization and Pareto-optimal trade-off analysis, without the need to manually develop specialized models for low-level tools such as mathematical programming solvers. Within Unity DGMS, the DG analytics engine provides several core analytical operators that can be performed against models in the repository. The implementation of these core analytical operators, which include but are not limited to computation, prediction, learning, simulation, and optimization, involve compilation, symbolic computation and reduction techniques, as well as specialized optimization and learning algorithms.

Fig. 6 shows the workflow followed within the system to optimize the service network in the case study described in Section III. A user such as a modeler or process engineer will first set up the problem on an IDE such as Atom [3]. The user then initiates the optimization request through Atom, which calls Unity DGMS to perform optimization. This is done by calling the DGMS via a REST API from within Atom focused on the file with *argmin* function. Unity DGMS reads all model input and transformation function files required for optimization from the repository in GitLab. Then, Unity DGMS performs automatic translation of the high-level performance models into a lower-level optimization model

[3]https://atom.io/



```
declare function ns:computeMetrics($input){
  let $rootProcess := $input.input.root
  let $output:= ns:computeSCmetrics($input.input.kb, $input.config,
                                           $rootProcess)
  return $output.$rootProcess
};
declare function ns:computeSCmetrics($stepsInputs, $config,
                                              $rootProcess){
  let
    $stepInput := $stepsInputs.$rootProcess,
    $analyticalModel := $stepInput.analyticalModel,
    $processMetrics := {},
    $processMetrics := {|
      if (not fn:matches($analyticalModel.ns,"supply_chain.jq")) then
        ns:evaluateAtomicProcesses({input: $stepInput, config: $config}
      else
        let $subProcessMetrics :=
            for $p in $stepInput.subProcesses[]
              return ns:computeSCmetrics
                  ($stepsInputs, $config, $p),
          $metrics := ns:metricAggr(
           for $p in $stepInput.subProcesses[]
             return $subProcessMetrics.$p.metricValues),
           $inputThru := $stepInput.inputThru,
           $outputThru := $stepInput.outputThru,
           $subProcessConstraints :=
               every $p in $stepInput.subProcesses[] satisfies
                     $subProcessMetrics.$p.constraints,
           $boundConstraintsIT :=
              every $i in keys($inputThru) satisfies
                    cu:checkBounds($inputThru($i),$inputThru($i)("v")),
           $boundConstraintsOT :=
               every $o in keys($outputThru) satisfies
                    cu:checkBounds($outputThru($o),$outputThru($o)("v")),
```

```
              $processItems :=
                fn:distinct-values((
                    keys($inputThru),keys($outputThru),
                      (for $p in $stepInput.subProcesses[]
                        return
                          (keys($subProcessMetrics.$p.inputThru),
                          keys($subProcessMetrics.$p.outputThru))
                    ))),
                  $zeroSumConstraints :=
                    every $i in $processItems
                      satisfies ( let
                      $supply :=
                        (if(fn:exists($inputThru($i)("v"))) then
                            $inputThru($i)("v") else 0) +
                            sum (for $p in $stepInput.subProcesses[]
                              return $subProcessMetrics.$p.outputThru($i)("v"))
                      $demand :=
                        (if(fn:exists($outputThru($i)("v"))) then
                          $outputThru($i)("v") else 0) +
                          sum (for $p in $stepInput.subProcesses[]
                            return $subProcessMetrics.$p.inputThru($i)("v"))
                      return $supply ge $demand
                    ),
                    $constraints := $subProcessConstraints and
                                          $boundConstraintsIT and
                                          $boundConstraintsOT and
                                          $zeroSumConstraints,
                  $rootProcessMetrics := {
                  analyticalModel:$stepInput.analyticalModel,
                  inputThru:$inputThru,
                  outputThru:$outputThru,
                  metricValues: $metrics,
                  constraints: $constraints
                }
```

Figure 7. (A) Analytical function for the service network PM. (B) Evaluate the atomic PMs. (C) Evaluate *subProcess* service network PMs recursively. (D) Aggregate Metrics. (E) *subProcess* constraints. (F) Bound constraint. (G) Part balance constraint. (H) Resulting service network constraint. (I) Returned JSON object.

expressed in the modeling language AMPL[4], and submits it to the MP-based optimization solver. The optimization results are returned to Unity DGMS, which then instantiates the control settings of the input JSON with either optimal values or error codes for infeasible or unbounded problems. The user then gets the optimization results displayed within Atom. The user may then use these results as actionable recommendations for every component of the manufacturing and contract service network.

V. Service Network Performance Models

The system functionality that allows a process engineer to model the Heat Sink service network shown in Fig. 1 and to perform trade-off analysis using Pareto optimal graphs is described in Section III. The service network is modeled as PMs each and PM provides an analytical function that compute the metrics as a function of the fixed parameters and control parameters subject to feasibility constraints. In this section, the analytical functions of the service networks is described using code snippets. Also, the analytical functions of the components of service network such as contract manufacturer, vendor service, and UMP service are briefly described.

[4]http://ampl.com/

The JSONiq analytical function, *computeMetrics* shown in Fig. 7A transforms the service network input structure of fixed and control parameters as the one shown in Fig. 2E to output metrics subject to feasibility constraints as the one shown in Fig. 3C. This function calls *computeSCmetrics* that recursively computes the outputs for all the *subProcesses* of the service network in terms of their respective inputs. For instance, the root process structure of the Heat Sink service network example (*heat_sink_part_supply_chain*) have *subProcesses* of *combined_supply*, *combined_manuf*, and *demand* (see Fig. 2E). The *computeSCmetrics* function will compute the output metrics and constraints of all these *subProcesses* recursively. To accomplish this, the *computeSCmetrics* function first checks if the *rootProcess* JSON structure in the input KB is atomic, i.e., whether the *rootProcess* structure is not of the service network type. Atomic processes in service networks include contract manufacturers, supplier vendor services, and UMP services. If the root process structure is atomic, then it calls *evaluateAtomicProcess* that transforms the input to output metrics and constraints using the analytical function in the respective atomic process PM (see Fig. 7B). On the other hand, if the root process structure is of service network type (e.g., *heat_sink_part_supply_chain*), then the *computeSCmetrics* calls itself recursively with each *subProcess* id as the *root-*

Process, as shown in Fig. 7C.

Due to the standard interface of all the PMs, the output JSON structure is tractable. Hence it is possible to aggregate the output metrics for all the atomic and composite processes in the service network. This is done in the *metricAggr* function (Fig. 7D). Then the constraints of the service network is evaluated as shown in Fig. 7H and it include the satisfaction of the *subProcess* (Fig. 7E), input bounds (Fig. 7F), output bounds (Fig. 7F), and part balance constraints (Fig. 7G). The *subProcess* constraint include bounds on the control parameter and capacity of the *subProcesses* as computed by their respective analytical function. The input and output bound constraints check to see if the input and output flow values are non-negative and within any upper bound specified by the user (e.g., flow capacity). The part balance constraint checks, for each item type, whether the supply of that item is greater or equal to the demand of the same item across the service network. Finally an object that contains the output metrics and constraints of the service network is returned as its transformed result (Fig. 7I).

The atomic *subProcesses* of the service network such as contract manufacturers, supplier vendor services, and UMP services have their own PMs in the industry knowledge base as shown in Fig. 2A and Fig. 2B. Hence the inputs to these atomic processes can be transformed into output metrics and constraints using their respective analytical function. The PM of the contract manufacturer gets as input the products as well as its respective pricing and carbon emission quantities. Given the amount produced, the analytical function of the contract manufacturer PM computes the total cost, carbon emissions, and amount of input items required by the contract manufacturer to manufacture these products. The PM of the vendor service also gets as input the products as well as its respective pricing and carbon emission quantities. For the amount of products supplied, the analytical function of the vendor service PM computes the total cost and carbon emissions incurred for this service. Finally, the UMP service normalizes the output metrics such as cost and carbon emissions from the UMP PM such as the ones in the industry KB (Fig. 2B). We use the UMP PMs from Brodsky et al. [5] to create the UMP service. The UMP service allows the service network and its UMP *subProcesses* to maintain a standard interface that makes it flexible and easy to use for the process engineer.

VI. Conclusion & Future Directions

Described in this paper is a novel software system for composition, optimization and trade-off analysis of manufacturing and contract service networks based on a reusable repository of performance models for (1) unit manufacturing processes, (2) base contract services, and (3) a composite steady-state service network. The uniqueness of the proposed system is in its ability to perform optimization and trade-off analysis on an arbitrary user-composed service network without the need to manually craft mathematical programming models, all while achieving the quality of optimization results and computational efficiency of mathematical programming solvers, which significantly outperform simulation-based solvers.

Many research questions remain open. They include (1) developing a web-based user interface suitable for process engineers and service network managers, (2) extending analytical functions to include prediction, stochastic optimization, and machine learning to calibrate performance models, (3) extending the knowledge base with a searchable interface to allow the user to perform process and service selection, and (3) extending the library of performance models, including for contract services, and generic UMP models, when physics-based models are not available.

Disclaimer

No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial equipment, instruments or materials are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST nor does it imply the materials or equipment identified are necessarily the best available for the purpose.

Acknowledgment

This effort has been sponsored in part under the Cooperative Agreement No.70NANB12H277 between NIST and George Mason University. The work described was funded by the US Government and is not subject to copyright.

References

[1] M. Helu and T. Hedberg, “Enabling smart manufacturing research and development using a product lifecycle test bed,” *Procedia Manufacturing*, vol. 1, pp. 86–97, 2015.

[2] G. Salvendy, *Handbook of industrial engineering: technology and operations management.* Hoboken, NJ, USA: Wiley, Inc., 2001.

[3] D. Wu, M. J. Greer, D. W. Rosen, and D. Schaefer, “Cloud manufacturing: Strategic vision and state-of-the-art,” *Journal of Manufacturing Systems*, vol. 32, no. 4, pp. 564–579, 2013.

[4] A. Brodsky, G. Shao, M. Krishnamoorthy, A. Narayanan, D. Menascé, and R. Ak, “Analysis and optimization based on reusable knowledge base of process performance models,” *The International Journal of Advanced Manufacturing Technology*, pp. 1–21, 2016.

[5] A. Brodsky, M. Krishnamoorthy, W. Z. Bernstein, and M. O. Nachawati, “A system and architecture for reusable abstractions of manufacturing processes,” in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 2004–2013.

[6] J. Richardson, “Gartner bi: analytics moves to the core,” http://timoelliott.com/blog/2013/02/gartnerbi-emea-2013-part-1-analytics-moves-to-the-core.html, 2013, accessed: Sep. 2015.

[7] C. Ündey, C. Ertunç, T. Mistretta, and B. Looze, “Applied advanced process analytics in biopharmaceutical manufacturing: Challenges and prospects in real-time monitoring and control,” *Journal of Process Control*, vol. 20, no. 9, pp. 1009 – 1018, 2010.

[8] G. C. Souza, “Supply chain analytics,” *Business Horizons*, vol. 57, no. 5, pp. 595 – 605, 2014.

[9] G. Shao, S. Shin, and S. Jain, “Data analytics using simulation for smart manufacturing,” in *Proceedings of the 2014 Winter Simulation Conference*, ser. WSC ’14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 2192–2203.

[10] D. Lechevalier, A. Narayanan, and S. Rachuri, “Towards a domain-specific framework for predictive analytics in manufacturing,” in *Big Data (Big Data), 2014 IEEE International Conference on*, 2014, pp. 987–995.

[11] S. Shin, J. Woo, and S. Rachuri, “Predictive analytics model for power consumption in manufacturing,” *Proc 21st CIRP Conference Life Cycle Eng 15:153158*, vol. 15, pp. 153 – 158, 2014.

[12] L. Lee, E. Lapira, B. Bagheri, and K. Kao, “Recent advances and trends in predictive manufacturing systems in big data environment,” *Manufacturing Letters*, vol. 1, no. 1, pp. 38 – 41, 2013.

[13] A. Brodsky, G. Shao, and F. Riddick, “Process analytics formalism for decision guidance in sustainable manufacturing,” *Journal of Intelligent Manufacturing*, vol. 27, no. 3, pp. 561–580, 2014.

[14] C. McLean and G. Shao, “Manufacturing case studies: generic case studies for manufacturing simulation applications,” in *Proceedings of the 35th conference on Winter simulation: driving innovation*. Winter Simulation Conference, 2003, pp. 1217–1224.

[15] B. Denkena, M. Shpitalni, P. Kowalski, G. Molcho, and Y. Zipori, “Knowledge management in process planning,” *CIRP Annals-Manufacturing Technology*, vol. 56, no. 1, pp. 175–180, 2007.

[16] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury, “Simulation optimization: a review of algorithms and applications,” *Annals of Operations Research*, vol. 240, no. 1, pp. 351–380, 2016.

[17] A. Klemmt, S. Horn, G. Weigert, and K.-J. Wolter, “Simulation-based optimization vs. mathematical programming: A hybrid approach for optimizing scheduling problems,” *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 6, pp. 917–925, 2009.

[18] J. C. Aurich, C. Fuchs, and C. Wagenknecht, “Life cycle oriented design of technical product-service systems,” *Journal of Cleaner Production*, vol. 14, no. 17, pp. 1480–1494, 2006.

[19] R. B. Tan and H. H. Khoo, “An LCA study of a primary aluminum supply chain,” *Journal of Cleaner Production*, vol. 13, no. 6, pp. 607–618, 2005.

[20] P. Tsiakis, N. Shah, and C. C. Pantelides, “Design of multi-echelon supply chain networks under demand uncertainty,” *Industrial & Engineering Chemistry Research*, vol. 40, no. 16, pp. 3585–3604, 2001.

[21] D. A. Menasce, “Composing web services: A qos view,” *IEEE Internet Computing*, vol. 8, no. 6, pp. 88–90, Nov 2004.

[22] D. A. Menasce, “Response-time analysis of composite web services,” *IEEE Internet Computing*, vol. 8, no. 1, pp. 90–92, Jan 2004.

[23] F. Ameri and D. Dutta, “An upper ontology for manufacturing service description,” in *ASME Conf. Proc*, 2006, pp. 651–661.

[24] M. O. Nachawati, A. Brodsky, and J. Luo, “Unity decision guidance management system: Analytics engine and reusable model repository,” in *19th International Conference on Enterprise Information Systems (ICEIS)*, 2017, pp. 312–323.