Published in final edited form as:
Int J Comput Integr Manuf. 2019 ; 32: . doi:10.1080/0951192X.2019.1570805.

Factory optima: a web-based system for composition and analysis of manufacturing service networks based on a reusable model repository

Alexander Brodsky[a], **Mohamad Omar Nachawati**[a], **Mohan Krishnamoorthy**[a], **William Z. Bernstein**[b], and **Daniel A. Menascé**[a]

[a]Department of Computer Science, George Mason University, Fairfax, VA, USA

[b]Systems Integration Division, NIST, Gaithersburg, MD, USA

Abstract

This paper reports on the development of Factory Optima, a web-based system that allows manufacturing process engineers to compose, optimise and perform 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 representing (1) unit manufacturing processes, (2) base contract services and (3) a composite steady-state service network. The proposed framework allows process engineers to hierarchically compose model instances of service networks, which can represent production cells, lines, factory facilities and supply chains, and perform deterministic optimisation based on mathematical programming and Pareto-optimal trade-off analysis. Factory Optima is demonstrated using a case study of a service network for a heat sink product which involves contract vendors and manufacturing activities, including cutting, shearing, Computer Numerical Control (CNC) machining with milling and drilling operations, quality inspection, finishing, and assembly.

Keywords

Performance models; model repository; optimisation; analytics; decision support; service composition

CONTACT Alexander Brodsky, brodsky@gmu.edu.

Publisher's Disclaimer: 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.

Disclosure statement
No potential conflict of interest was reported by the authors.

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). To realise such a vision, the digitisation of manufacturing environments is a necessity. One method for realising a smart manufacturing ecosystem is to employ an over-arching cloud-based infrastructure that supports decision-making (Wu et al. 2013). Such infrastructure must support a wide variety of analytical tasks across the entire organisational hierarchy, including 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 ongoing research area and requires generic and robust model representations to perform advanced analysis, e.g. optimisation (Wu et al. 2013). get this

Cloud-enabled manufacturing has already shown significant promise through the deployment of distributed computing resources (Xu 2012). In the Prognostics and Health Management (PHM) community specifically, recent developments aim to build fully aware and resilient manufacturing systems that react quickly and appropriately to environmental signals through cloud-based resources (Lee et al. 2014). Wu et al. (2018) demonstrated the use of the cloud for performing advanced data analytics to predict tool wear during machining operations (Wu et al. 2018). However, lack of standard guidelines for the underlying architecture, such as sensor selection, data transmission and database creation, limit cloud-based architectures’ scalability, reproducibility and interoperability (Gao et al. 2015; Lee 2003). This leads to high-cost and long-duration development of manufacturing analysis and optimisation solutions, and results in models and algorithms that are difficult to modify, extend and reuse. A key contributor to these deficiencies is the diversity of underlying computational tools, each designed for a different task such as data manipulation, statistical learning, data mining, optimisation and simulation. Because of this diversity, modelling using computational tools often requires the use of specialised low-level mathematical abstractions. As a result, the same manufacturing knowledge is often modelled multiple times using different specialised abstractions, instead of being modelled once using a uniform abstraction. Moreover, the modelling expertise required for the low-level abstractions and languages is typically not within the realm of knowledge of manufacturing users. A recent state-of-the-art review on cloud manufacturing emphasised these points by listing six primary research directions (He and Xu 2015): (1) additional standards development, (2) multi-cloud integration, (3) security policies, (4) resource utility management, (5) systems integration and (6) stakeholder adoption. Echoing these challenges, Ren et al. presented the primary characteristics associated with fully implementing a cloud manufacturing environment, across the entirety of a large aerospace company operating over 600 subsidiary companies (Ren et al. 2017).

Important

not sure I understand this

Solving the core challenges of the digitisation of manufacturing and promoting flexibility and more rapid innovation can be facilitated by modular simulation and modelling techniques (Brettel et al. 2014). Recently, work has focused on realising fundamental techniques in this direction. For example, Otto et al. presented a simulation-based method that attempts to re-use existing software components used to formally characterise seemingly disparate production realisations to test new reconfiguration designs (Otto, Vogel-Heuser, and Niggemann 2018). Similarly, Denno and Kim presented an ontology-based approach for re-using components of predictive models for optimising the performance of a selective laser sintering process (Denno and Kim 2016). However, while simulation-based modelling approaches have advantages of modularity, reusability and easy visualisation, simulation-based optimisation is inferior to mathematical programming (MP) solvers. In terms of quality of optimisation results and computational efficiency, MP solvers significantly outperform simulation-based solvers for optimisation problems expressed in closed analytic form (Amaran et al. 2016; Klemmt et al. 2009).

In response to these trends, the focus of this paper is the development of a standards-oriented approach for the reusability and reproducibility of manufacturing process models, and composable service network models. This approach is based on adopting an extension of the unit manufacturing process (UMP) information model (Bernstein, Lechevalier, and Libes 2018; ASTM International 2016), a standard representation of process models designed for reusability. The key idea is to allow modular, composable and reusable simulation-like predictive models, yet to achieve the performance of MP solvers through symbolic analysis of simulation code to machine-generate MP models that are optimised using MP solvers.

ML

The proposed approach follows the idea of an architectural design of Brodsky et al. (2017) proposed for the fast development of software solutions for descriptive ('what happened?'), diagnostic ('why did it happen?'), predictive ('what will happen?') and prescriptive ('how to make it happen optimally?') analytics of dynamic production processes based on a reusable, modular and extensible knowledge base (KB) of simulation-like process performance models, and machine translatable into MP models. However, Brodsky et al. (2017) lacked a systematic design of a UMP repository and its architecture, and an ecosystem around it. Furthermore, the UMP models were abstracted in Brodsky et al. (2017) by piecewise-linear functions whereas real-world process models, which are typically physics-based, require non-linear arithmetic.

To address these limitations, a software architecture reported in Brodsky et al. (2016a) was developed for a software architecture for (1) a reusable repository of UMP performance models and (2) their analysis and optimisation using the Unity Decision Guidance Management System (Unity DGMS). Unity DGMS is an integrated analytics platform that aims to simplify the development of intelligent, decision-making systems (Nachawati, Brodsky, and Luo 2017). The architecture was demonstrated using the case of Computer Numerical Control (CNC) machining. However, Brodsky et al. (2016a) did not address the important problem of how to compose reusable UMP models into a hierarchy of service networks, including production cells, lines, factories and supply chain.

To overcome this, in Brodsky et al. (2017) the authors reported on extending the functionality developed in Brodsky et al. (2016a) with a software framework that allows hierarchical composition of service networks based on a reusable model repository for (1) production services, such as manufacturing processes, assembly and inspection and (2) contract services, such as vending, manufacturing, packaging, repair and transportation. However, to compose such service networks and to perform analysis, Brodsky et al. (2017) provided a low-level Integrated Development Environment (IDE) user-interface. Such an interface is not suitable for end users such as process engineers and operators, who do not have a software development background. Lifting this limitation is exactly the focus of this paper.

More specifically, the contribution of this paper is the design and development of Factory Optima, a web-based system that allows manufacturing process engineers to compose, optimise and perform 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 representing (1) unit manufacturing processes, (2) base contract services and (3) a composite steady-state service network.

Factory Optima allows process engineers to hierarchically compose model instances of service networks, which can represent production cells, lines, factory facilities and supply chains, and perform deterministic optimisation and Pareto-optimal trade-off analysis. Factory Optima is demonstrated using a case study of a service network for a heat sink product which involves contract vendors and manufacturing activities, including cutting, shearing, Computer Numerical Control (CNC) machining with milling and drilling operations, quality inspection, finishing, and assembly.

det Optimization

Pareto-Optimal
Trade analysis

The novelty and uniqueness of Factory Optima are in its ability to perform optimisation and trade-off analysis on arbitrary user-composed service networks, similar to flexible and modular simulation-based approaches, yet achieving the quality of optimisation results and computational efficiency of mathematical programming solvers, which significantly outperform simulation-based solvers (e.g. (Amaran et al. 2016) and (Klemmt et al. 2009)). This unique capability of Factory Optima is achieved by machine generation of mathematical programming models, instead of manually crafting them – a demanding task which is typically outside the skill set of process engineers. This paper can be viewed as a significant extension of the authors' conference publication (Brodsky et al. 2017).

The rest of the paper is organised as follows. Section 2 presents the concept of manufacturing and contract service networks, and outlines its possible ecosystem and workflows. Section 3 presents the system functionality through a case study of a service network of a heat sink assembly. Section 4 overviews the software architecture for the system. Section 5 presents the service network performance model and explains its structure. Section 6 concludes with some future research directions.

2. Service networks: ecosystem and workflows

In the proposed framework, the manufacturing activity is represented as a network of service-oriented components that are linked together to produce products or product service

? systems (PSS) to meet some specified demand. In the performance model knowledge base, this network is referred to as the *Service Network*. Service network stakeholders include both manufacturers and suppliers (Aurich, Fuchs, and Wagenknecht 2006). The following terms are used to describe the various components of a service network.

- **Vendor**: describing an organisation that provides a finished product, e.g. raw material, bar stock or fully realised components, at some cost.
- **Contract Manufacturer**: describing an organisation that provides a manufacturing service, e.g. precision welding, mould-making and precision machining, at some cost.
- **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.

These terms are used to describe various decision paths to realise the delivery of the PSS based on its demand. Figure 1 provides an example of a service network for an assembly of a heat sink product, consisting of an aluminium base, a heat sink component and a set of fasteners. This service network was derived to the authors’ best knowledge borrowing parts of its configuration from literature (Tan and Khoo 2005). Within the service network, each activity and physical good is represented by an image and a labelled 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 are multiple paths that provide the same physical good. Often, this is referred to as a multi-echelon supply chain (Tsiakis, Shah, and Pantelides 2001). 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 aluminium 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, anodising, CNC machining, quality inspection and final assembly.

After formally characterising 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, optimising 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 consistent representation across different manufacturing stages offers engineers a reusable toolbox to evaluate process designs and plans. Considering the current trends in cloud-based manufacturing (Wu et al. 2013; Wang, Törngren, and Onori 2015), allowing for on-demand analytics is required for the full realisation of smart manufacturing.

The idea of invoking composite services to form a network of services in a programmatic way through web services was discussed in (Menasce 2004a). This along with the work in (Menasce 2004b) 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. In this work, a service network hierarchy is constructed, representing a network of manufacturing activities recursively. The approach for service network composition is explained in detail in Section 3.3.

3. System functionality and a case study

In this section, Factory Optima's functionality is described from the perspective of a process engineer that is weighing decision alternatives in the context of the Heat Sink service network depicted in Figure 1. The key concept in the system is the *performance model*, for a service network and its base service components, which are described 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 Figure 1, fixed parameters include the type of material used, its dimensions, the number of holes during the *HS Base Drilling* activity, depth of cut in the *CNC Machining* activity, among others. Control parameters in the service network include the cutting speed in an *HS Shearing* activity, cutting speed in an *HS Base Drilling* activity and the amount produced by the *Aluminum Plate Contract Manufacture*, to name a few.

In this section, the Factory Optima system functionality is described using the scenario of composing and optimising the Heat Sink service network shown in Figure 1. To compose this service network, the system needs to recursively capture the hierarchy of processes under the root service network process, and capture the fixed parameters, control parameters and flow of items among the processes using input and output flows. The optimisation problem for this scenario can be described according to the following: find all the control parameters of the service network so as to minimise 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).

The web-based user interface of Factory Optima is shown in Figure 2. The toolbar at the top contains functions to create and manage content, which are *Artifact, Analytics, New, Save, Properties, Manage* and *Run*. To compose a service network as the one shown in Figure 1, a process engineer uses the *Artifact* function from the toolbar to create an artefact for a specific instance for the PM input containing fixed and control parameters of the process. To perform analysis such as optimisation over the service network, a process engineer uses the *Analytics* function from the toolbar to create an analytics input instance for the corresponding analysis type. The *New* function in the toolbar allows the creation of both instance artefacts and analytics input. The *Save, Properties* and *Manage* functions in the toolbar allow the process engineer to view and manage content within Factory Optima. Finally, the *Run* function in the toolbar allows the process engineer to perform analysis such as optimisation by running the currently active analytics input. The subsections below describe the tabs and panes of the Factory Optima user interface, and then demonstrate the composition, optimisation and trade-off analysis of the Heat Sink service network from Figure 1 using this interface.

3.1. Ontology of industry knowledge base

The industry KB contains well-accepted and validated PMs from the industry. These PMs are organised in an ontology and one such organisation is shown in the left pane of Figure 2. In this organisation, the industry KB contains PMs of service network components under *ServiceNetworkPerformanceModel* and PM-based representations of UMPs under *UnitManufacturingProcessPerformanceModel*. The UMPs in the industry KB are organised based on the Manufacturing Service Description Language (MSDL) ontology (Ameri and Dutta 2006). The process engineer models the Heat Sink service network from Figure 1 by creating input instances for the PMs stored in the industry KB. For example, the *Smelting, HS Shearing* and *CNC Machining* processes in Figure 1 are instances of the UMP PMs *Smelting, Shearing* and *CompositeMachining* respectively. Similarly, the *Alumina, HS Base Contract Manufacture* and *Heat Sink Service Network* processes in Figure 1 are instances of the service network PMs *VendingService, MfgService* and *CompositeService* respectively.

3.2. Repository

The process engineer stores the instances of the PMs from the industry KB into the repository. An organisation of the repository is shown in the left pane of Figure 3. In this organisation, the repository is divided into product workspace (see *ProductWorkspace* in the left pane of Figure 3) and enterprise catalogue (see *ACME/Enterprise* in the left pane of Figure 3). The enterprise catalogue encapsulates the input instances of different UMPs and service network components present in 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 *hs_base_contract_manuf* is a variable annotated input of the contract manufacturing PM, *MfgService*. Similarly, *accessories_vendor* and *cnc_machining* are variable annotated input of the vendor PM *VendingService* and the UMP PM of *CompositeMachining* respectively. On the other hand, the process engineer (or the engineer's team) can store a number of different artefacts such as service network compositions, results of the analysis, and other design and downstream processes in the product workspace. This forms a workspace that the process engineer (or team) can use to store data pertaining to a specific manufacturing project or product. 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 optimisation problem in the product workspace.

3.3. Service network composition

In this section, the composition of the Heat Sink service network shown in Figure 1 using the Factory Optima interface is described. The Heat Sink service network is a hierarchical process, and L# in the names of the processes in Figure 1 refers to the hierarchical level of the service network composition. The Heat Sink service network was chosen since it represents a simple supply chain comprising multiple production lines and factory facilities. To compose such a hierarchical process, an engineer recursively initialises its subprocesses as well the input and output item ids of the composite processes at all the hierarchical levels. For instance, Figure 4 shows the subprocesses as well the input and output item ids for the Heat Sink service network at different hierarchical levels of composition. At *L0*, the subprocesses of the Heat Sink service network are initialised with process ids of *supply_11, manufacturing_11* and *demand_11*. At *L1*, the subprocesses of *manufacturing_11* are initialised with process ids of *alum_plate_contract_manu_12, hs_base_pl_12, smelting_12, hs_pl_12* and *hs_base_contract_manuf_12*, and also the inputs to this process is initialised with item ids of *Alumina* and *Accessories package*, and outputs with item id, *Heat Sink*. Such initialisations are recursively performed until all the processes at the leaf of the service network hierarchy are instances of an atomic PM such as *anodising_13* (see *L3* in Figure 4). Finally, the fixed and control parameters in the input instances of the composite and atomic PMs are initialised at all the hierarchical levels within the Heat Sink service network.

To compose the Heat Sink service network in Factory Optima, the process engineer first creates an input instance of the *CompositeService* PM from the *ServiceNetworkPerformanceModel* section of the ontology of industry knowledge base by pressing the *Artifact* function from the toolbar. In Factory Optima, this creates the form as shown in the right pane of Figure 5. Then, the process engineer initialises the configuration of the problem under *config*, specifies the *root* process id of the problem as *heat_sink_part_service_network*, and then proceeds to compose its hierarchy as described above using Figure 4 by initialising the subprocesses as well as the input item ids, output item ids, fixed parameters and control parameters for each subprocess under *kb*. The composed service network is stored as *heat_sink_service_network.json* in the product workspace (see *ProductWorkspace* in the left pane of Figure 5).

In Factory Optima, the process engineer can also import predefined input instances of the subprocesses from the product workspace or the enterprise catalogue. For instance, say that the process engineer used the Factory Optima interface as the one shown in the right pane of Figure 6 to compose the *manufacturing_11* process. The *root* here is *manufacturing_11*, and the *kb* contains all the subprocesses, input item ids and output item ids of the *manufacturing_11* process (e.g. see the subprocesses and item ids with *manufacturing_11* as the root in Figure 4). Additionally, the process engineer initialises the fixed and control parameters in the input instances of the composite and atomic PMs at all the hierarchical levels. The process engineer stores this composed *manufacturing_11* process in the product workspace as *heat_sink_manufacturing.json* (see *ProductWorkspace* in the left pane of Figure 5). When composing the Heat Sink service network, the process engineer can choose the *Import* function above *combined_manuf* (see in right pane of Figure 5) to import the *heat_sink_manufacturing.json* from the product workspace as one of the subprocesses of the Heat Sink service network. In this way, it is possible for the process engineer to reuse input instances of atomic and composite PMs into larger composite processes very easily.

3.4. Service network optimisation

To run the optimisation problem for finding the control parameters of the Heat Sink service network as to minimise the total cost of service network operation subject to feasibility and demand constraints, the engineer first composes the Heat Sink service network as an input instance of the variable annotated *CompositeService* PM. This instance is similar to the instance described in section 3.3 except that the control parameters in this instance are annotated as decision variables. Say that the process engineer saves this instance into the product workspace as *heat sink service network.json*.

The Factory Optima interface to perform optimisation is shown in Figures 7 and 8. The process engineer first chooses the *New* function from the toolbar and then selects the *Optimise* analytics resource as shown in Figure 7(a). This creates a new optimisation view as shown in Figure 7(b). Then, the process engineer inputs configuration and optimisation solver parameters in this optimisation view. After that, the process engineer imports the variable annotated input instance of the Heat Sink service network from the product workspace, i.e. the engineer imports *heat_sink_service_network.json* into the optimisation view using the interface shown in Figure 8(a). Upon importing the variable annotated input instance, this instance gets copied under the *input* of the optimisation view as shown in Figure 8(b). Then, the process engineer can run this optimisation view using the *Run* function from the toolbar. The Factory Optima screen with the result of the optimisation run is shown in Figure 9. This result is a fully instantiated Heat Sink service network input instance where all the annotated control parameters are replaced by values that minimise the objective cost subject to feasibility and demand constraints.

To provide this capability, Factory Optima invokes the optimisation service of Unity DGMS, see Figure 8(b), with the arguments populated in the OptimisationView, as shown in Figures 8(b) and 9. These arguments include the variable annotated input instance, the performance model and the objective to minimise (or maximise). The performance model is implemented as a function in a high-level language, currently JSONiq, that takes the input instance as an argument and computes performance metrics and constraints as a result. In addition, the OptimisationView also allows the process engineer to specify a particular solver to use, as well as any solver-specific configuration parameters.

3.5. Trade-off analysis

The process engineer can use the instantiated controls obtained by solving the optimisation 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. Figure 10 displays Pareto optimal alternatives for producing the Heat Sink assembly 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 optimisation problem that minimises 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 organisation or enterprise that he/she represents.

3.6. Overview of case study

Figure 11 presents an overview of the primary tasks required to compose and then optimise a service network in Factory Optima. One purpose of the case study is to demonstrate each task's delegation. In Figure 11, each task is categorised by whether it is completed by (1) a human user denoted as a person icon, (2) Unity DGMS denoted as a database icon and (3) the Factory Optima client denoted as a computer icon. Section 4 details the 'under-the-hood' interactions between all three.

Finally, it is important to note that, while a relatively simple example of the heat sink service network was used to explain key system functionality, Factory Optima supports hierarchical composition, optimisation and trade-off analysis of arbitrary complex service networks, as long as the atomic performance models (such as UMP and supply chain components) are available in the model repository. Furthermore, when new component models are added to the model repository, no additional changes are required to the service network model or the overall Factory Optima system.

4. Software architecture: reusable model repository and unity decision guidance management system

Factory Optima adopts the high-level system architecture that was proposed in (Brodsky et al. 2016a), which is based around a reusable model repository and Unity DGMS (Nachawati, Brodsky, and Luo 2017). The architecture is depicted in Figure 12. This section overviews this architecture, focusing on the components relevant to Factory Optima.

The architecture is designed to support various user roles, as shown in the top-most layer of the diagram in Figure 12. 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 applications.

Situated directly below the user roles layer, the applications 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. In this layer, the Factory Optima web application serves as the primary application interface for several workflows, including: (1) hierarchically composing model instances of service networks, (2) performing different kinds of analysis against these instances and (3) viewing the results of such analyses. Besides the Factory Optima web application, the system also supports Atom[1] for low-level, text editor-based development. Atom is a light-weight, yet highly extensible text editor with a large library of user-contributed plugins.

The Factory Optima web application is hosted on top of Unity DGMS and is composed of different types of services, including:

[1] https://atom.io/.

- **Process Composition & View Services** that provide support for the instantiation, composition and management of service networks centred around a form-based web interface. Given a particular performance model (e.g. CompositeService), the JSON Schema for its input is first retrieved from the Industry Knowledge base. The schema is used to automatically generate a web-based form for instantiation and composition of input instances. This is done by using and extending the JSON Editor[2] project.

- **Industry Knowledge base Services** that provide support for the construction, manipulation and querying of the Industry Knowledge base. An OWL2 ontology is used to capture and represent a hierarchy of manufacturing concepts, instances and relationships and is based on the MSDL ontology (Ameri and Dutta 2006). The ontology also captures the association between performance models and the JSON Schema documents, which are used both for validation and form generation. Both the OWL API[3] and Apache Jena[4] are used in the implementation of these services.

Where is this?

- **Process Analytics Services** that provide web-application support for performing different kinds of analysis against performance model input instances.

Underlying the above services, Unity Web Application Services provide a RESTful framework for building intelligent, decision-making web applications. These services based on the analytics engine and repository management functionality of Unity DGMS, and also include support for user management, authentication and authorisation. These services are designed around a RESTful, model-view-controller (MVC) pattern and can be consumed both by human users using a web browser interface and by other applications via REST client. This is done by using *Jersey*,[5] a RESTful web services library for Java (JAX-RS).

Moving to the bottom-most layer of the diagram in Figure 12, 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 optimisation, including the IBM CPLEX Optimiser for MILP problems and the MINOS solver for NLP problems, (2) algebraic modeling languages and systems, specifically AMPL and the IBM Optimisation 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,

[2] https://github.com/jdorn/json-editor.
[3] http://owlapi.sourceforge.net/.
[4] https://jena.apache.org/.
[5] https://jersey.github.io/.

applications implemented using one tool are difficult to modify, extend and reuse with other tools. As a result, the same manufacturing knowledge is often modelled multiple times using different, specialised abstractions, instead of being modelled just once using a single, uniform abstraction.

To overcome these limitations, the architecture is augmented with Unity DGMS as a middleware layer, situated between the applications layer and the low-level tools layer in the diagram in Figure 12. The uniqueness of this solution is that it is centred around a reusable model repository (middle box in the DGMS middleware layer) that decouples analytical models from the various kinds of analysis that can be performed on them. This provides a uniform, high-level abstraction over different low-level tools and is key in supporting the optimisation and trade-off analysis of manufacturing and contract service networks.

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

The solution to this challenge is based on the Unity DGMS (Nachawati, Brodsky, and Luo 2017), which provides support for different methods of analysis, including optimisation and Pareto-optimal trade-off analysis, without the need to manually develop specialised models for low-level tools such as mathematical programming solvers. Within Unity DGMS, the Unity 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 optimisation, involve compilation, symbolic computation and reduction techniques, as well as specialised optimisation and learning algorithms.

Performance models are one of the key artefacts 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 PMs that characterise (1) UMPs, (2) a library of base contract services and (3) a composite steady-state service network. In addition to performance models, the model repository is also designed to contain data views, ontologies, schemas and taxonomies.

To conclude this section, the workflow is now described, which is followed within the system to optimise the service network in the case study discussed in Section 3. The workflow is depicted in Figure 13. A user such as a modeller or a process engineer will first set up the problem by constructing a new RunInput instance for Optimisation (argmin). As shown in Figures 7 and 8, the user imports an existing performance model input instance from the repository to use as input for the RunInput instance (argmin). The user then initiates the optimisation request through the Factory Optima by executing the Run function, which calls Unity DGMS to perform optimisation. The Unity engine reads all model input and transformation function files required for optimisation from the repository in GitLab. Then, Unity DGMS performs automatic translation of the high-level performance models into a lower-level optimisation model expressed in the modelling language AMPL,[6] and submits it to the MP-based optimisation solver. The optimisation 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 optimisation results in the same form used for input, however where all decision variables are replaced with instantiated values to together minimise the objective. The user may then use these results as actionable recommendations for every component of the manufacturing and contract service network.

5. Service network performance models

The system functionality that allows a process engineer to model the Heat Sink service network, shown in Figure 1, and to perform trade-off analysis using Pareto optimal graphs is described in Section 3. The service network is modelled as a network of PMs, and each PM provides an analytical function that computes the metrics as a function of the fixed parameters and control parameters subject to feasibility constraints. In this section, the analytical function of the service network is described using a flowchart. The code of the analytical function is given in A. Also, the analytical functions of the components of service network such as contract manufacturer, vendor service and internal manufacturer, captured as a UMP PM, are briefly described.

The flowchart of the service network PM is shown in Figure 14. This PM provides an analytical function that transforms the service network input structure containing subprocesses, input items, output items, fixed parameters and control parameters as the one created using the Factory Optima interface shown in Figure 5 to output metrics subject to feasibility constraints. This function recursively computes the outputs for all the processes in *subProcess* of the service network in terms of their respective inputs. For instance, the root process structure of the Heat Sink service network example contains a *subProcess* with processes of *supply_ 11, manufacturing,_11* and *demand_11* (see *L0* in Figure 4). The analytical function will compute the output metrics and constraints of all the processes in the *subProcess* recursively. To accomplish this, the analytical function first checks if the processes within *subProcess* are atomic or composite of a service network type. Atomic processes in service networks include contract manufacturers, supplier vendor services and internal manufacturing activities. If the root process structure is atomic, then it calls the analytical function of the atomic process PM that transforms the respective inputs of the atomic process into its output metrics and constraints. On the other hand, if the processes within *subProcess* is composite of service network type (e.g. root process of the Heat Sink service network), then the analytical function of the service network PM calls itself to recursively compute the output metrics and constraint of the composite process in *subProcess*.

Then, the output metrics of all the atomic and composite processes in *subProcess* is aggregated. This is possible due to the standard interface of all the atomic and composite process PMs in a given service network. Then, the constraints of the processes within

[6] http://ampl.com/.

subProcess as computed by their respective analytical function is aggregated. Additionally, constraints that describe the interaction between these processes is also validated and aggregated. These constraints include the satisfaction of the input flow bound, output flow bound and item balance zero sum constraints. The input and output flow 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 item balance zero sum 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, the output metrics and constraints of the service network is returned as its transformed result.

The atomic *subProcesses* of the service network such as contract manufacturers, supplier vendor services and internal manufacturing activities have their own PMs in the industry knowledge base. Hence, the inputs to these atomic processes can be transformed into output metrics and constraints using their respective analytical functions. The PM of the contract manufacturer gets as input the products as well as its respective pricing and carbon emission quantities, for example. Given the amount produced, the analytical function of the contract manufacturer PM computes the total cost, carbon emissions and the 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 internal manufacturer service normalises the output metrics such as cost and carbon emissions from the UMP PM such as the ones in the industry KB. The UMP PMs from (Brodsky et al. 2016a) are used to model the internal manufacturer service. The internal manufacturing service allows the service network and its UMP-based representation in *subProcess* to maintain a standard interface that makes it flexible and easy to use for the process engineer.

6. Conclusion and future directions

This paper presented Factory Optima, a novel software system for composition, optimisation and trade-off analysis of manufacturing and contract service networks based on a reusable repository of performance models representing (1) unit manufacturing processes, (2) base contract services and (3) a composite steady-state service network. The uniqueness of Factory Optima is in its ability to perform optimisation 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 optimisation results and computational efficiency of mathematical programming solvers, which significantly outperform simulation-based solvers.

One of the primary goals of this work was to demonstrate the feasibility of performing analytical tasks on standard reusable representations of manufacturing process models. The authors envision the eventual realisation of a shared and open UMP repository to support smart manufacturing activities across both industry and academia (Bernstein et al. 2016). Research activities currently ongoing that would support the dissemination and use of Factory Optima include (1) a web application supporting the consistent recording of standard UMP models, (2) review protocols to ? validate which model should be curated across federations of users and (3) automated translation of standard static UMP representations, as shown in ASTM E3012–16 (ASTM International 2016), towards operational code in the form of functional models, such as the UMP PMs presented throughout this paper.

With respect to Factory Optima itself, many research directions remain open. They include (1) integrating system affordances with other applications curating product and process specific data, e.g. manufacturing execution systems, enterprise resource planning software and shop floor streams as described in (Helu and Hedberg 2015), (2) extending analytical functions to include prediction, stochastic optimisation and machine learning to calibrate performance models based on such data and (3) extending the library of performance models, including for contract services and for representing generic UMP models, when physics-based models are not available.

Acknowledgement

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.

Funding

This work was supported by the National Institute of Standards and Technology [Cooperative Agreement No. 70NANB12H277].

Appendix A. JSONiq Code for Service Network Performance Model

```
declare function ns : computeMetrics ($input){

   let $rootProcess := $input . input . root

   let $output:= ns : computeSCmetrics

          ($input . input .kb, $input . config, $rootProcess)

   return $output

   };

   declare function ns : computeSCmetrics

                               ($stepsInputs, $config, $rootProcess){

   let

     $stepInput := $stepsInputs . $rootProcess,

     $analyticalModel := $stepInput . analyticalModel,

     $processMetrics := { },
$processMetrics := { |
  if (not fn : matches($analyticalModel . ns,
                  “service network . jq “)) then
   ns : evaluateAtomicProcesses
                    ({input : $stepInput, config : $config})
  else
   let
    $subProcessMetrics := { |
       for $p in $stepInput . subProcesses [ ]
          return {$p :(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
         }
         return { | $rootProcessMetrics, {subProcesses :
                        $subProcessMetrics}|}
       |}
       return $processMetrics
}
```

References

ASTM International 2016 ASTM E3012-16, Guide for Sustainability Characterization of Manufacturing Processes. West Conshohocken, PA: ASTM International.

Amaran S, Sahinidis NV, Sharda B, and Bury SJ. 2016 “Simulation Optimization: A Review of Algorithms and Applications.” Annals of Operations Research 240 (1): 351–380. doi:10.1007/s10479-015-2019-x.

Ameri F, and Dutta D. 2006 “An Upper Ontology for Manufacturing Service Description” In ASME 2006 IDETC/CIE, edited by Kumar Vijay, 651–661. New York, NY: American Society of Mechanical Engineers.

Aurich JC, Fuchs C, and Wagenknecht C. 2006 “Life Cycle Oriented Design of Technical Product-Service Systems.” Journal of Cleaner Production 14 (17): 1480–1494. doi:10.1016/j.jclepro.2006.01.019.

Bernstein WZ, Lechevalier D, and Libes D. 2018 "UMP Builder: Capturing and Exchanging Manufacturing Models for Sustainability." In ASME 2018 International Manufacturing Science and Engineering Conference, College Station, TX.

Bernstein WZ, Mani M, Lyons KW, Morris KC, and Johansson B. 2016 "An Open Web-Based Repository for Capturing Manufacturing Process Information" In ASME 2016 IDETC/CIE, V004T05A028. New York, NY: American Society of Mechanical Engineers.

Brettel M, Friederichsen N, Keller M, and Rosenberg M. 2014 "How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective." International Journal of Mechanical, Industrial Science and Engineering 8 (1): 37–44.

Brodsky A, Shao G, Krishnamoorthy M, Narayanan A, Menasce D, and Ak R. 2017 "Analysis and Optimization Based on Reusable Knowledge Base of Process Performance Models." International Journal of Advanced Manufacturing Technology 88: 337–357.

Brodsky A, Krishnamoorthy M, Nachawati MO, Bernstein WZ, and Menasce DA. 2017 "Manufacturing and Contract Service Networks: Composition, Optimization and Tradeoff Analysis Based on a Reusable Repository of Performance Models." 2017 IEEE International Conference on Big Data (Big Data) Boston, MA.

Brodsky A, Krishnamoorthy M, Bernstein WZ, and Nachawati MO. 2016a "A System and Architecture for Reusable Abstractions of Manufacturing Processes." 2016 IEEE International Conference on Big Data (Big Data) Washington, DC, December 2004–2013.

Denno P, and Kim DB. 2016 "Integrating Views of Properties in Models of Unit Manufacturing Processes." International Journal of Computer Integrated Manufacturing 29 (9): 996–1006. doi: 10.1080/0951192X.2015.1130259.

Gao R, Wang L, Teti R, Dornfeld D, Kumara S, Mori M, and Helu M. 2015 "Cloud-Enabled Prognosis for Manufacturing." CIRP Annals 64 (2): 749–772. doi:10.1016/j.cirp.2015.05.011.

Helu M, and Hedberg T. 2015 "Enabling Smart Manufacturing Research and Development Using a Product Lifecycle Test Bed." Procedia Manufacturing 1: 86–97. doi:10.1016/j.promfg.2015.09.066. [PubMed: 28664167]

Klemmt A, Horn S, Weigert G, and Wolter K-J. 2009 "Simulation Based Optimization vs. Mathematical Programming: A Hybrid Approach for Optimizing Scheduling Problems." Robotics and Computer-Integrated Manufacturing 25 (6): 917–925. doi:10.1016/j.rcim.2009.04.012.

Lee J 2003 "E-Manufacturing Fundamental, Tools, and Transformation." Robotics and Computer-Integrated Manufacturing 19 (6): 501–507. doi:10.1016/S0736-5845(03)00060-7.

Lee J, Fangji W, Zhao W, Ghaffari M, Liao L, and Siegel D. 2014 "Prognostics and Health Management Design for Rotary Machinery Systems - Reviews, Methodology and Applications." Mechanical Systems and Signal Processing 42 (1–2): 314–334. doi:10.1016/j.ymssp.2013.06.004.

Menasce DA 2004a "Composing Web Services: A QoS View." IEEE Internet Computing 8 (6): 88–90. doi:10.1109/MIC.2004.57.

Menasce DA 2004b "Response-Time Analysis of Composite Web Services." IEEE Internet Computing 8 (1): 90–92. doi:10.1109/MIC.2004.1260710.

Nachawati MO, Brodsky A, and Luo J. 2017 "Unity Decision Guidance Management System: Analytics Engine and Reusable Model Repository." 19th International Conference on Enterprise Information Systems (ICEIS), edited by Hammoudi Slimane, Smialek Michal, 312–323. Porto, Portugal.

Otto J, Vogel-Heuser B, and Niggemann O. 2018 "Automatic Parameter Estimation for Reusable Software Components of Modular and Reconfigurable Cyber-Physical Production Systems in the Domain of Discrete Manufacturing." IEEE Transactions on Industrial Informatics 14 (1): 275–282. doi:10.1109/TM.2017.2718729.

Ren L, Zhang L, Wang L, Tao F, and Chai X. 2017 "Cloud Manufacturing: Key Characteristics and Applications." International Journal of Computer Integrated Manufacturing 30 (6): 501–515. doi: 10.1080/0951192X.2014.902105.

Salvendy G 2001 Handbook of Industrial Engineering: Technology and Operations Management. Hoboken, NJ: Wiley.

Tan RBH, and Khoo HH. 2005 "An LCA Study of a Primary Aluminum Supply Chain." Journal of Cleaner Production 13 (6): 607–618. doi:10.1016/j.jclepro.2003.12.022.

Tsiakis P, Shah N, and Pantelides CC. 2001 "Design of Multi-Echelon Supply Chain Networks under Demand Uncertainty." Industrial & Engineering Chemistry Research 40 (16): 3585–3604. doi: 10.1021/ie0100030.

Wang L, Törngren M, and Onori M. 2015 "Current Status and Advancement of Cyber-Physical Systems in Manufacturing." Journal of Manufacturing Systems 37 (Part 2): 517–527. doi:10.1016/j.jmsy.2015.04.008.

Wu D, Jennings C, Terpenny J, Kumara S, and Gao RX. 2018 "Cloud-Based Parallel Machine Learning for Tool Wear Prediction." Journal of Manufacturing Science and Engineering 140 (4): 041005. doi:10.1115/1.4038002.

Wu D, Greer MJ, Rosen DW, and Schaefer D. 2013 "Cloud Manufacturing: Strategic Vision and State-Of-The-Art." Journal of Manufacturing Systems 32 (4): 564–579. doi:10.1016/j.jmsy.2013.04.008.

Xu X 2012 "From Cloud Computing to Cloud Manufacturing." Robotics and Computer Integrated Manufacturing 28 (1): 75–86. doi:10.1016/j.rcim.2011.07.002.

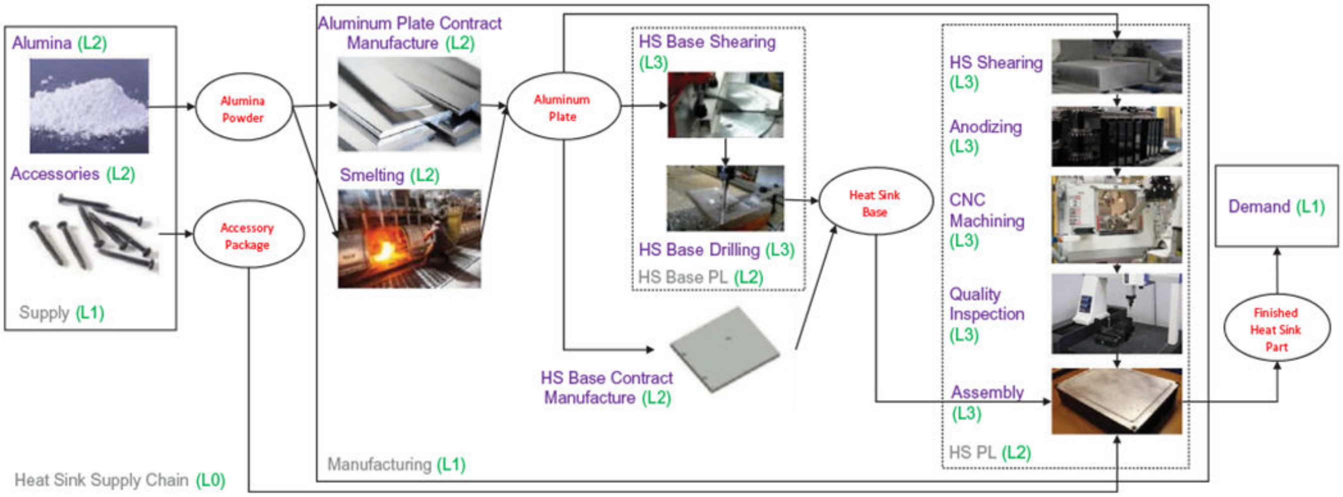
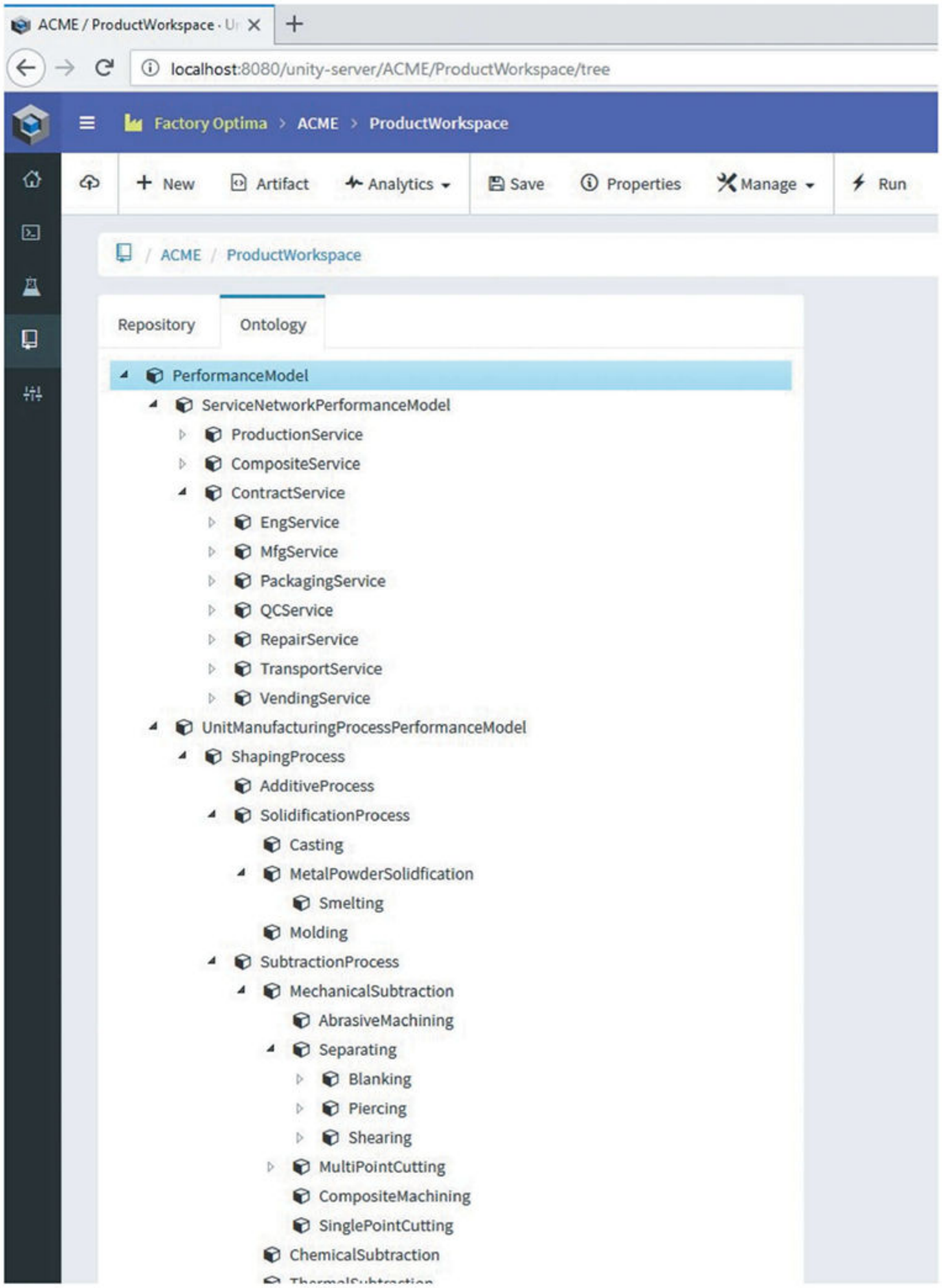


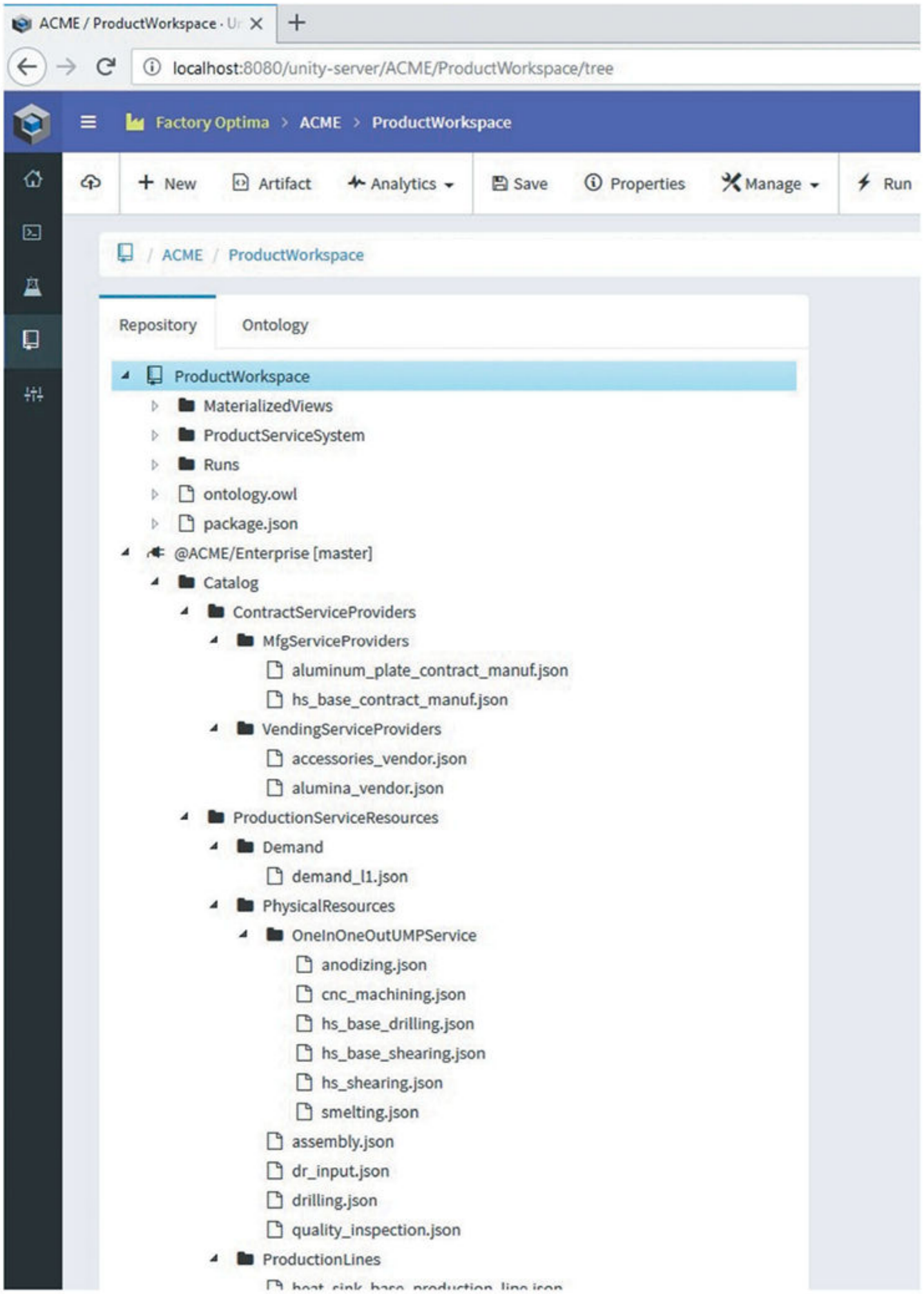
Figure 1.

Graphical representation of the Heat Sink service network, adopted from (Brodsky et al. 2017). Each labelled circle relates to a physical good to be procured. All images represent 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.


ACME / ProductWorkspace · U
localhost:8080/unity-server/ACME/ProductWorkspace/tree
Factory Optima > ACME > ProductWorkspace
New
Artifact
Analytics
Save
Properties
Manage
Run
ACME / ProductWorkspace
Repository
Ontology

- PerformanceModel
  - ServiceNetworkPerformanceModel
    - ProductionService
    - CompositeService
    - ContractService
      - EngService
      - MfgService
      - PackagingService
      - QCService
      - RepairService
      - TransportService
      - VendingService
  - UnitManufacturingProcessPerformanceModel
    - ShapingProcess
      - AdditiveProcess
      - SolidificationProcess
        - Casting
        - MetalPowderSolidfication
          - Smelting
        - Molding
      - SubtractionProcess
        - MechanicalSubtraction
          - AbrasiveMachining
          - Separating
            - Blanking
            - Piercing
            - Shearing
          - MultiPointCutting
          - CompositeMachining
          - SinglePointCutting
        - ChemicalSubtraction
        - ThermalSubtraction

Figure 2.
Factory optima screen with the ontology tab selected.


ACME / ProductWorkspace · U
localhost:8080/unity-server/ACME/ProductWorkspace/tree
Factory Optima > ACME > ProductWorkspace
New
Artifact
Analytics
Save
Properties
Manage
Run
ACME / ProductWorkspace
Repository
Ontology

```
◢ ProductWorkspace
    ▷ MaterializedViews
    ▷ ProductServiceSystem
    ▷ Runs
    ▷ ontology.owl
    ▷ package.json
◢ @ACME/Enterprise [master]
    ◢ Catalog
        ◢ ContractServiceProviders
            ◢ MfgServiceProviders
                aluminum_plate_contract_manuf.json
                hs_base_contract_manuf.json
            ◢ VendingServiceProviders
                accessories_vendor.json
                alumina_vendor.json
        ◢ ProductionServiceResources
            ◢ Demand
                demand_l1.json
            ◢ PhysicalResources
                ◢ OneInOneOutUMPService
                    anodizing.json
                    cnc_machining.json
                    hs_base_drilling.json
                    hs_base_shearing.json
                    hs_shearing.json
                    smelting.json
                assembly.json
                dr_input.json
                drilling.json
                quality_inspection.json
            ◢ ProductionLines
                heat_sink_base_production_line.json
```

Figure 3.

Factory optima screen with the repository tab selected.

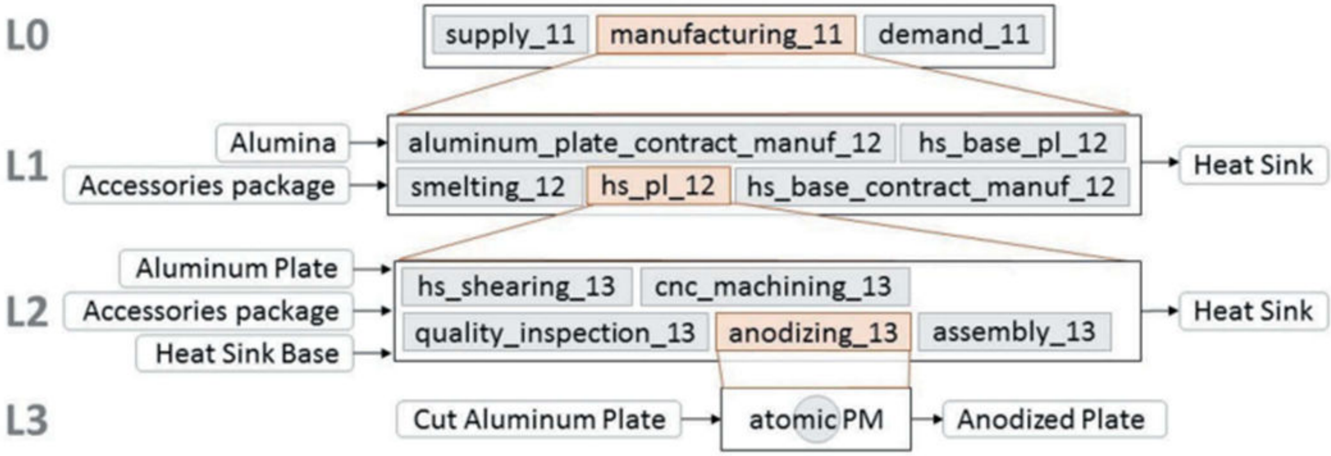
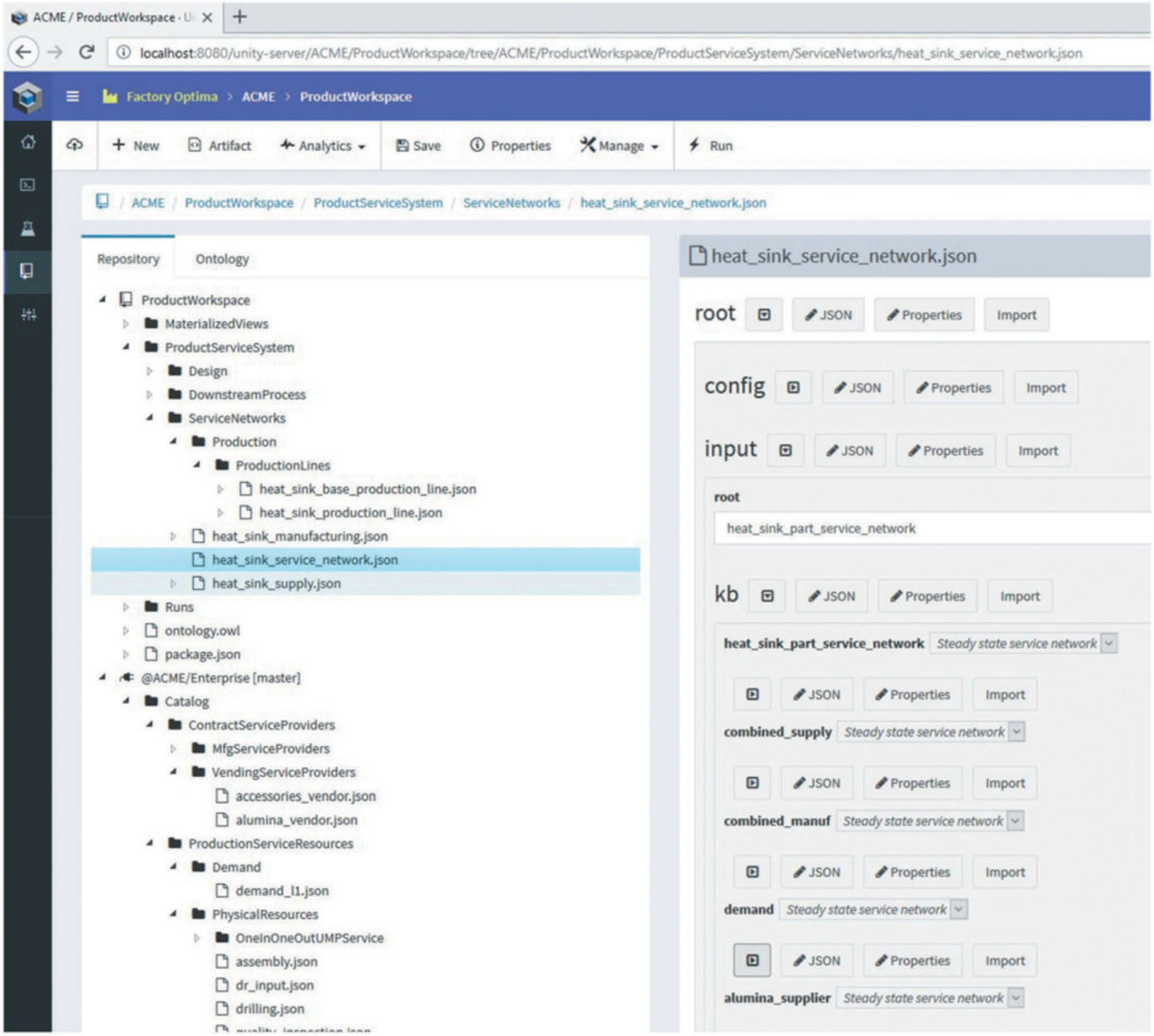


Figure 4.

Heat Sink service network at different levels of hierarchica3l composition.


ACME / ProductWorkspace · U
localhost:8080/unity-server/ACME/ProductWorkspace/tree/ACME/ProductWorkspace/ProductServiceSystem/ServiceNetworks/heat_sink_service_network.json
Factory Optima > ACME > ProductWorkspace
New
Artifact
Analytics
Save
Properties
Manage
Run
ACME / ProductWorkspace / ProductServiceSystem / ServiceNetworks / heat_sink_service_network.json
Repository
Ontology
heat_sink_service_network.json
root
JSON
Properties
Import
config
input
kb

```
ProductWorkspace
  MaterializedViews
  ProductServiceSystem
    Design
    DownstreamProcess
    ServiceNetworks
      Production
        ProductionLines
          heat_sink_base_production_line.json
          heat_sink_production_line.json
      heat_sink_manufacturing.json
      heat_sink_service_network.json
      heat_sink_supply.json
  Runs
  ontology.owl
  package.json
@ACME/Enterprise [master]
  Catalog
    ContractServiceProviders
      MfgServiceProviders
      VendingServiceProviders
        accessories_vendor.json
        alumina_vendor.json
    ProductionServiceResources
      Demand
        demand_l1.json
      PhysicalResources
        OneInOneOutUMPService
        assembly.json
        dr_input.json
        drilling.json
        quality_inspection.json
```

root: heat_sink_part_service_network

heat_sink_part_service_network: Steady state service network

combined_supply: Steady state service network

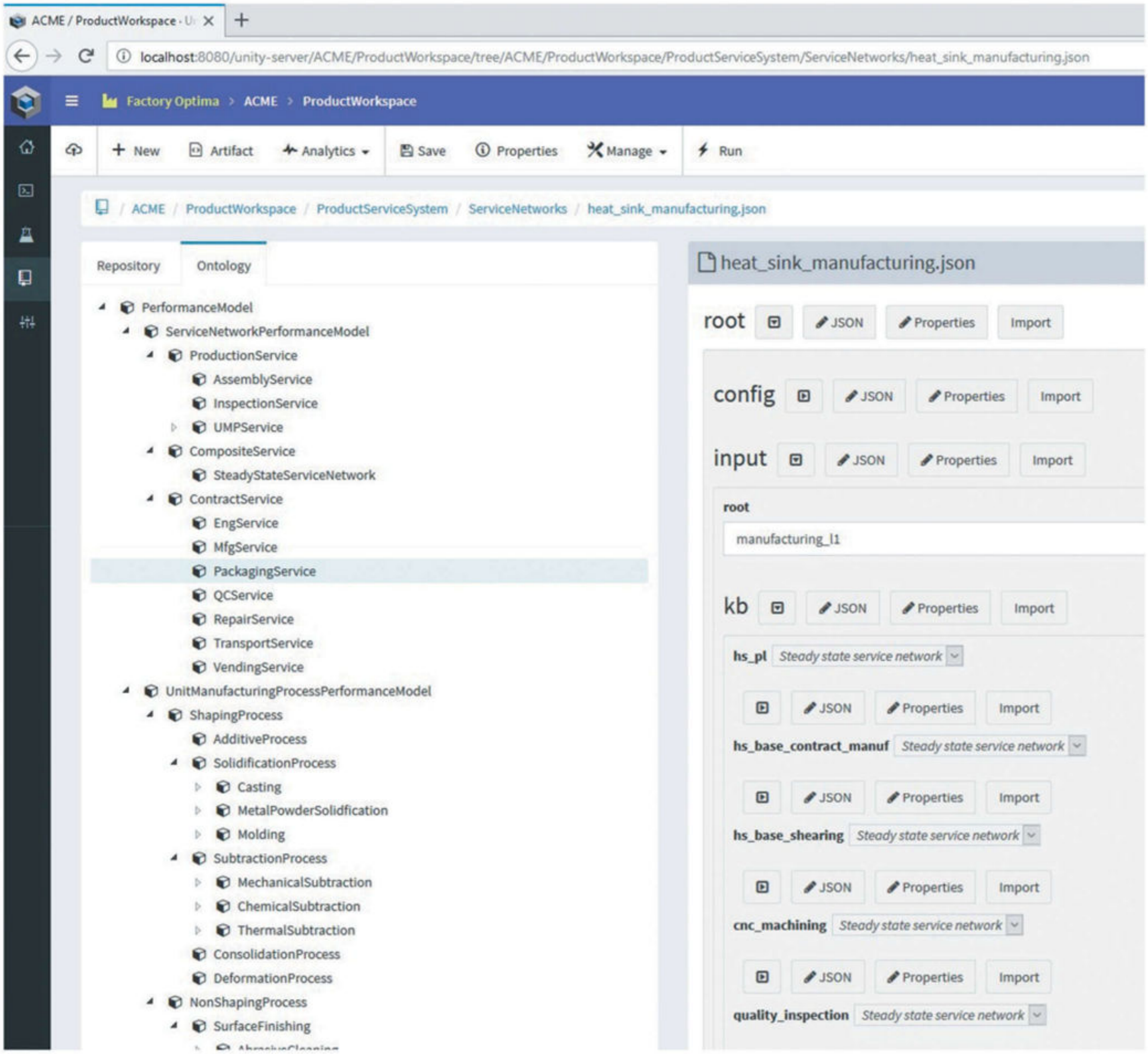
combined_manuf: Steady state service network

demand: Steady state service network

alumina_supplier: Steady state service network

Figure 5.

Factory Optima screen showing the composition of the heat sink service network (right) by reusing the input instances of its subprocesses from the product workspace and enterprise knowledge base under the repository tab (left).



ACME / ProductWorkspace - U
localhost:8080/unity-server/ACME/ProductWorkspace/tree/ACME/ProductWorkspace/ProductServiceSystem/ServiceNetworks/heat_sink_manufacturing.json
Factory Optima > ACME > ProductWorkspace
New
Artifact
Analytics
Save
Properties
Manage
Run
ACME / ProductWorkspace / ProductServiceSystem / ServiceNetworks / heat_sink_manufacturing.json
Repository
Ontology
heat_sink_manufacturing.json
JSON
Properties
Import

```
PerformanceModel
  ServiceNetworkPerformanceModel
    ProductionService
      AssemblyService
      InspectionService
      UMPService
    CompositeService
      SteadyStateServiceNetwork
    ContractService
      EngService
      MfgService
      PackagingService
      QCService
      RepairService
      TransportService
      VendingService
  UnitManufacturingProcessPerformanceModel
    ShapingProcess
      AdditiveProcess
      SolidificationProcess
        Casting
        MetalPowderSolidfication
        Molding
      SubtractionProcess
        MechanicalSubtraction
        ChemicalSubtraction
        ThermalSubtraction
      ConsolidationProcess
      DeformationProcess
    NonShapingProcess
      SurfaceFinishing
```

root

config

input

root: manufacturing_l1

kb

hs_pl: Steady state service network

hs_base_contract_manuf: Steady state service network

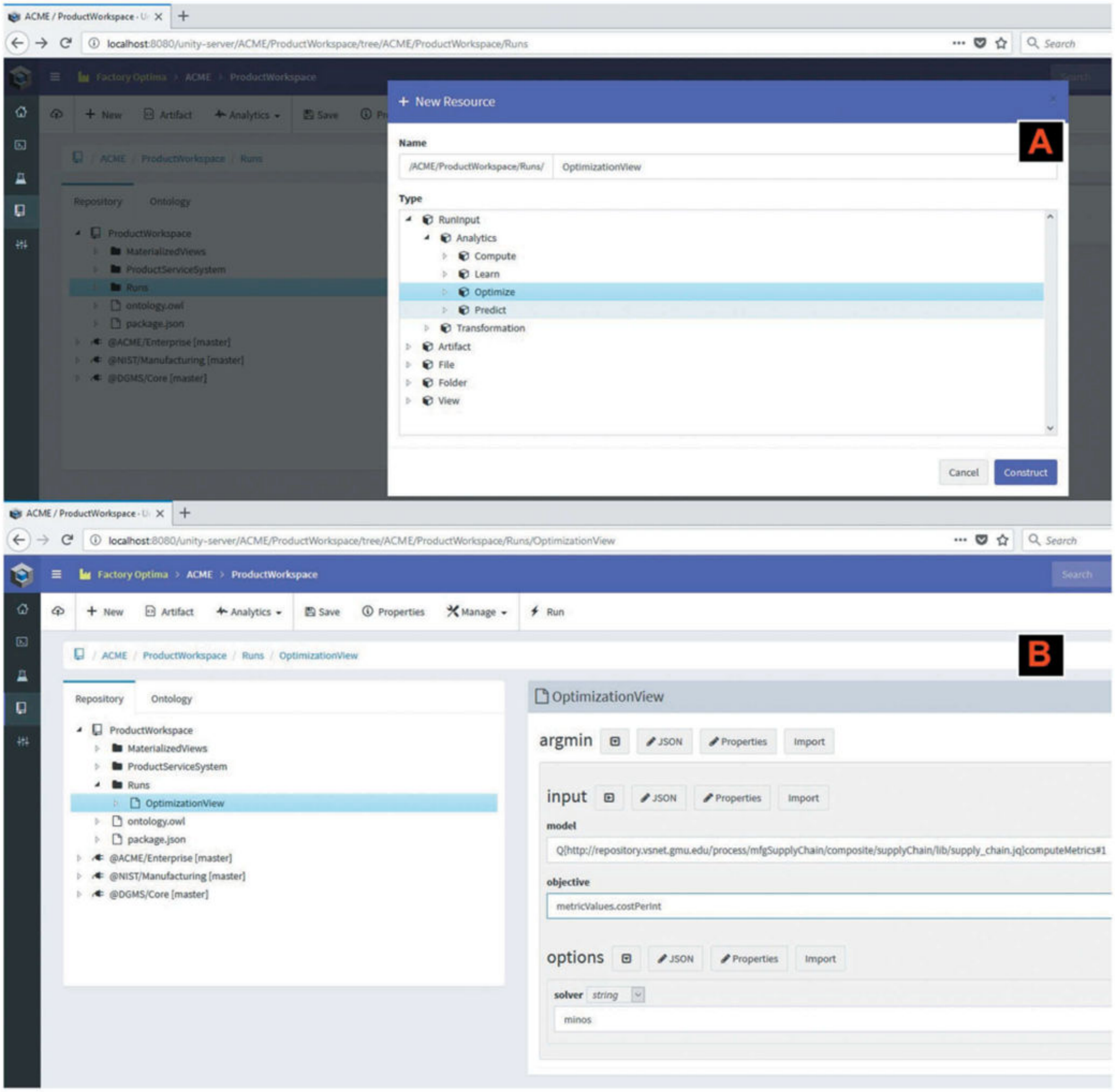
hs_base_shearing: Steady state service network

cnc_machining: Steady state service network

quality_inspection: Steady state service network

Figure 6.

Factory Optima screen showing the composition of the *manufacturing l1* composite process (right) by creating input instances of the PMs under the ontology tab (left).



Name: /ACME/ProductWorkspace/Runs/ OptimizationView

Type: Optimize

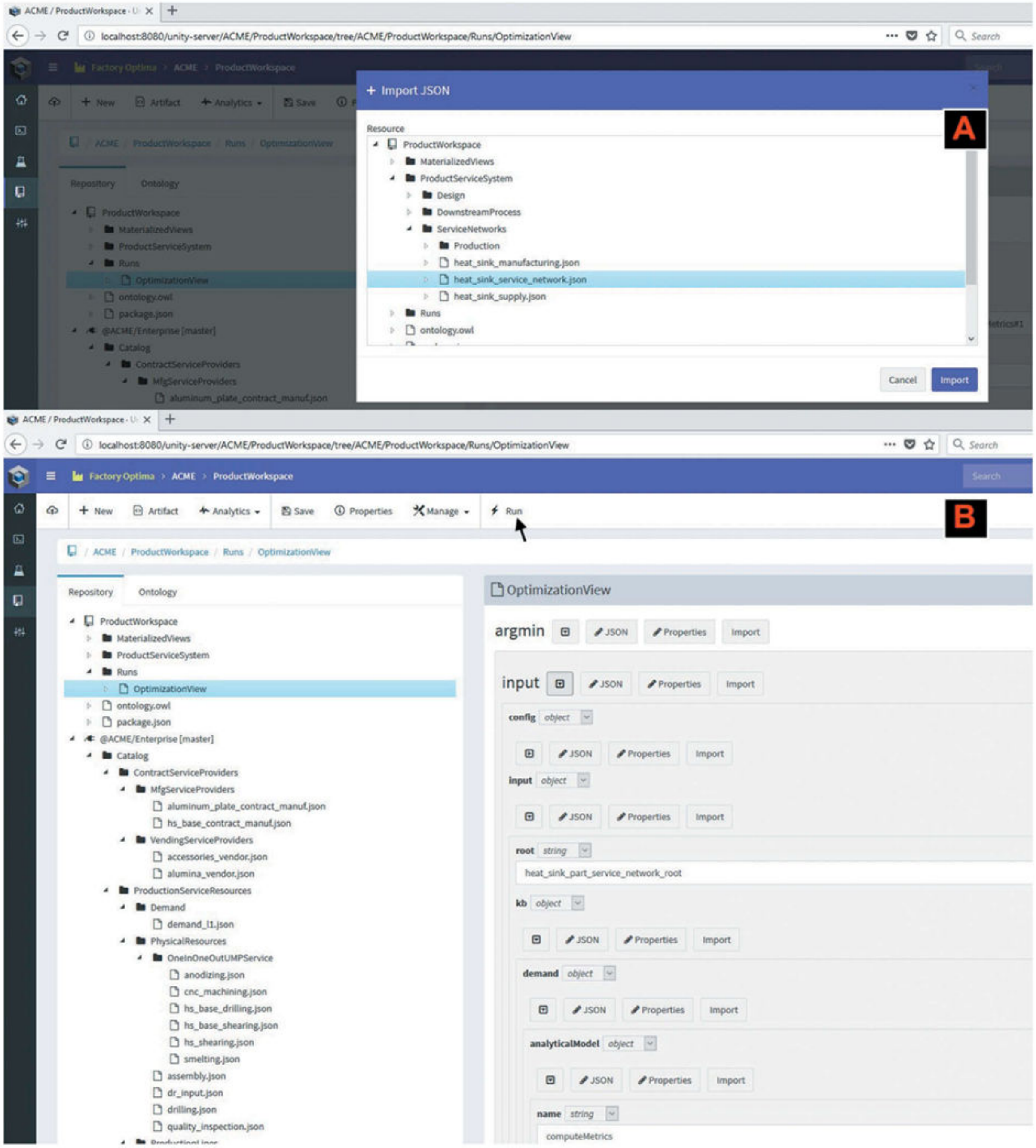
model: Q{http://repository.vsnet.gmu.edu/process/mfgSupplyChain/composite/supplyChain/lib/supply_chain.jq}computeMetrics#1

objective: metricValues.costPerInt

solver: string minos

Figure 7.

Factory Optima screens showing the creation and initialisation: (a) Creation of a new optimisation view and (b) Initialisation of the optimisation view (right) and storing the view under *Runs* in the product workspace (left).

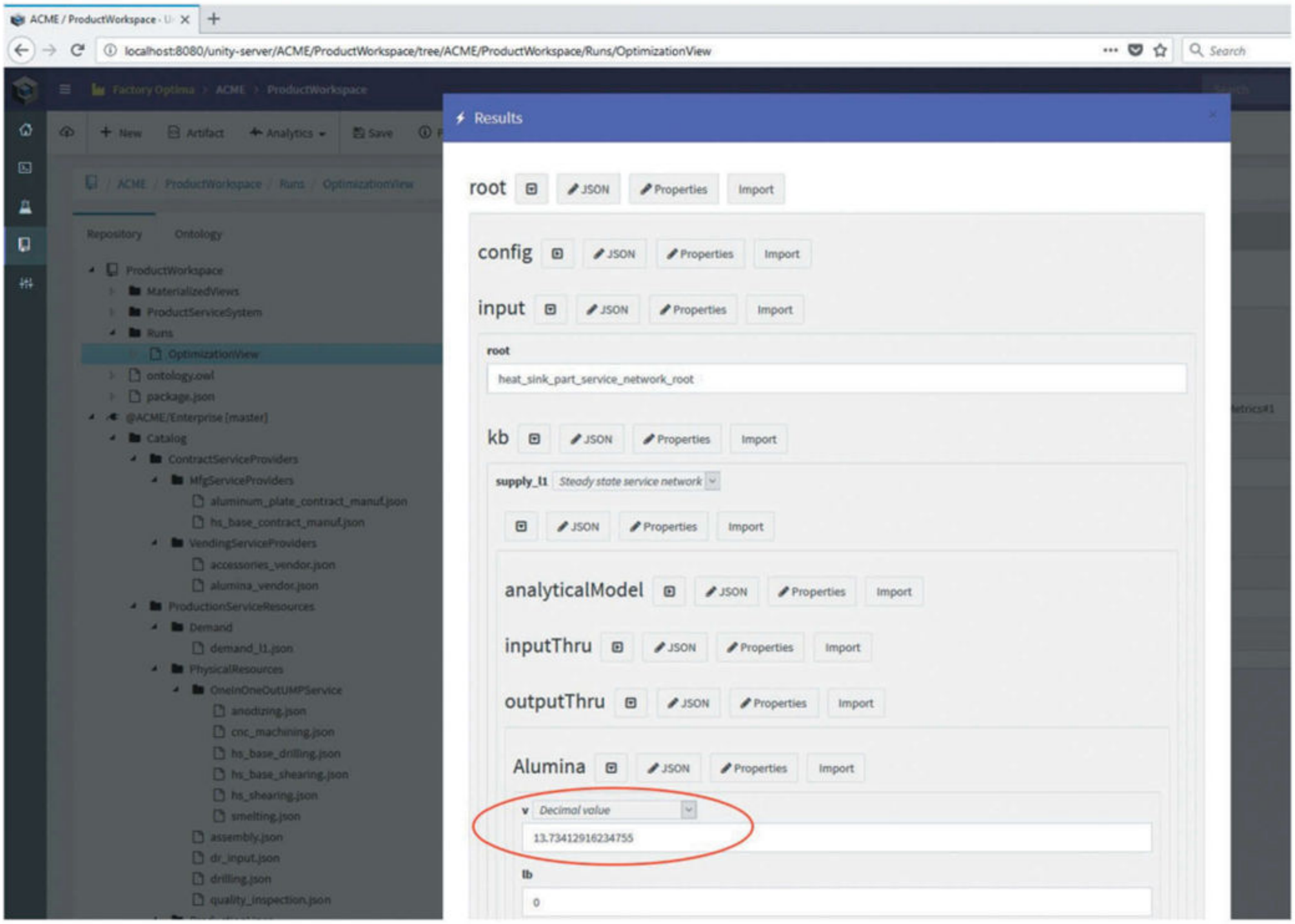


heat_sink_part_service_network_root

computeMetrics

Figure 8.

Factory Optima screens showing the instance import and running an optimisation view: (a) Import the heat sink service network input instance into the optimisation view and (b) Optimisation view after importing the heat sink service network input instance (right) and use of the *Run* function from the toolbar to run the optimisation view.



localhost:8080/unity-server/ACME/ProductWorkspace/tree/ACME/ProductWorkspace/Runs/OptimizationView

root: heat_sink_part_service_network_root

supply_l1: Steady state service network

v: Decimal value 13.73412916234755

lb: 0

Figure 9.

Factory optima screen showing the results after running optimisation on the heat sink service network.

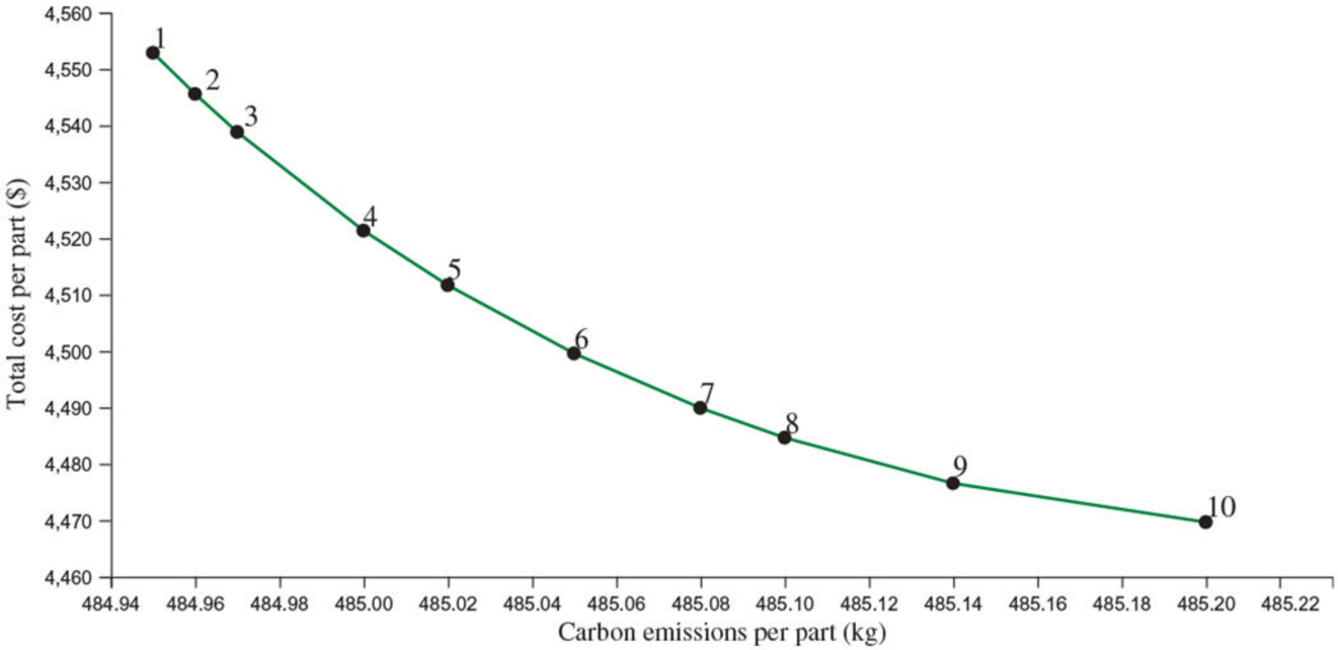


Figure 10.

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

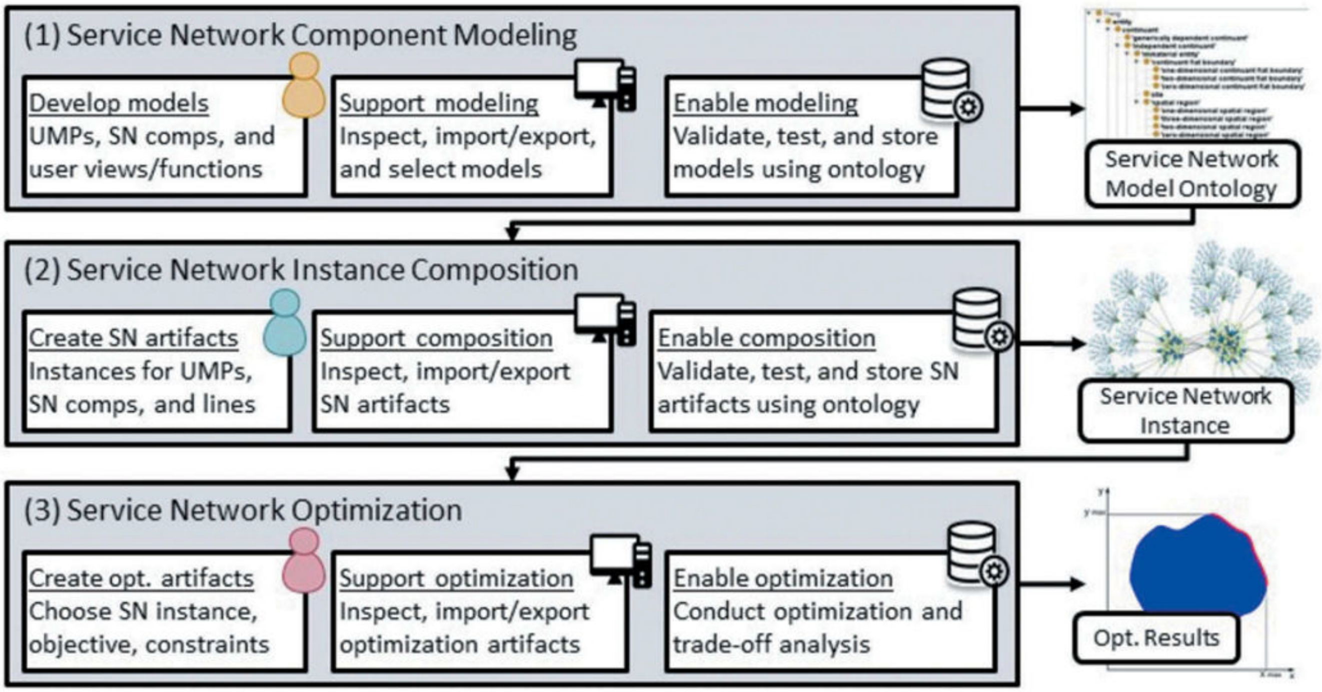

(1) Service Network Component Modeling
Develop models
UMPs, SN comps, and user views/functions
Support modeling
Inspect, import/export, and select models
Enable modeling
Validate, test, and store models using ontology
Service Network Model Ontology
(2) Service Network Instance Composition
Create SN artifacts
Instances for UMPs, SN comps, and lines
Support composition
Inspect, import/export SN artifacts
Enable composition
Validate, test, and store SN artifacts using ontology
Service Network Instance
(3) Service Network Optimization
Create opt. artifacts
Choose SN instance, objective, constraints
Support optimization
Inspect, import/export optimization artifacts
Enable optimization
Conduct optimization and trade-off analysis
Opt. Results

Figure 11.

Overview of tasks performed in the case study completed through factory optima. The user icons specify different classifications of users, including modellers (orange), process engineers (blue) and the decision-maker (red). The factory optima client and DGMS are represented as computer and database icons, respectively.

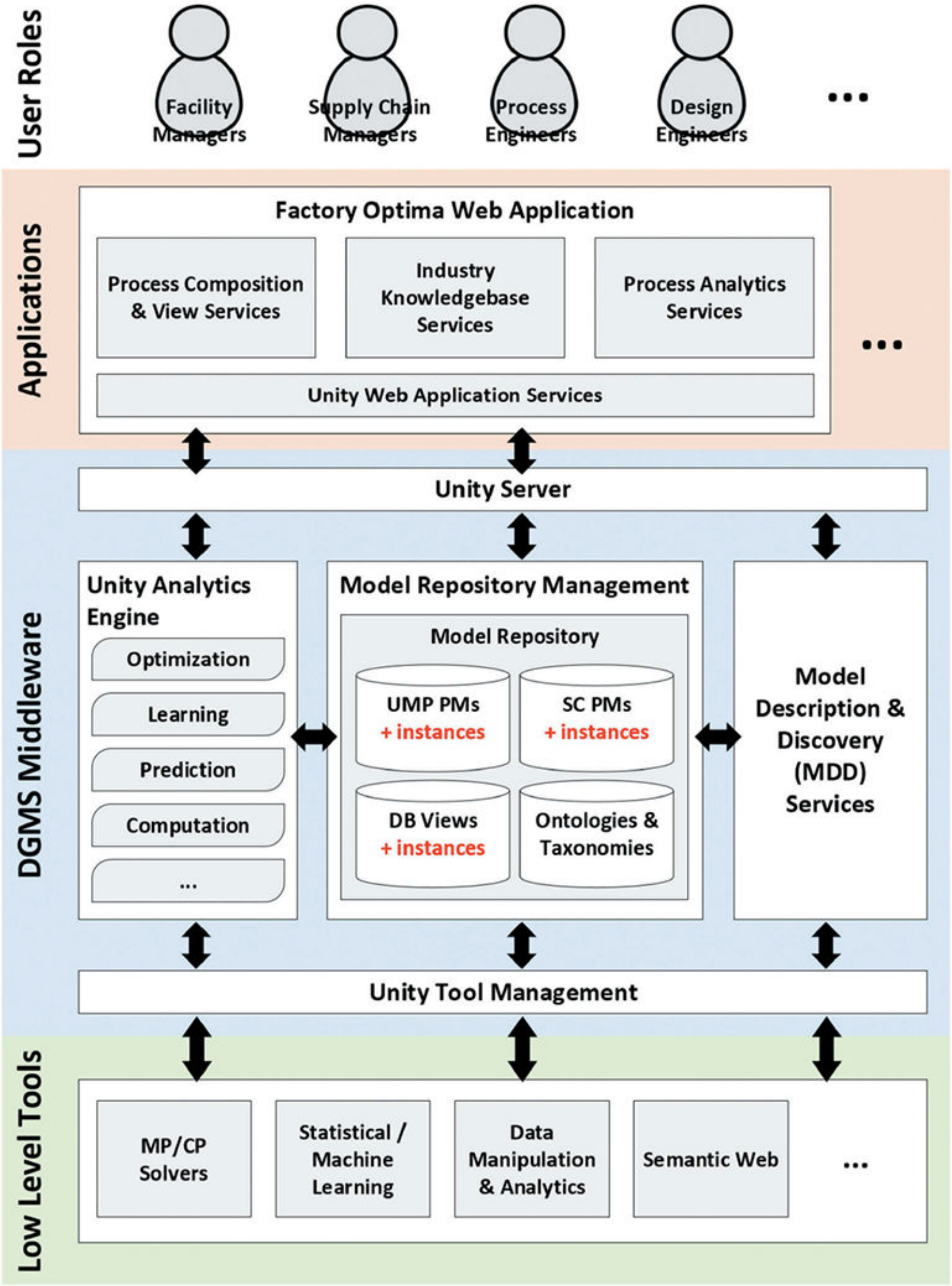


Figure 12.
Conceptual architecture of factory optima and underlying software system based on reusable model repository and Unity DGMS. Adopted from (Brodsky et al. 2017).

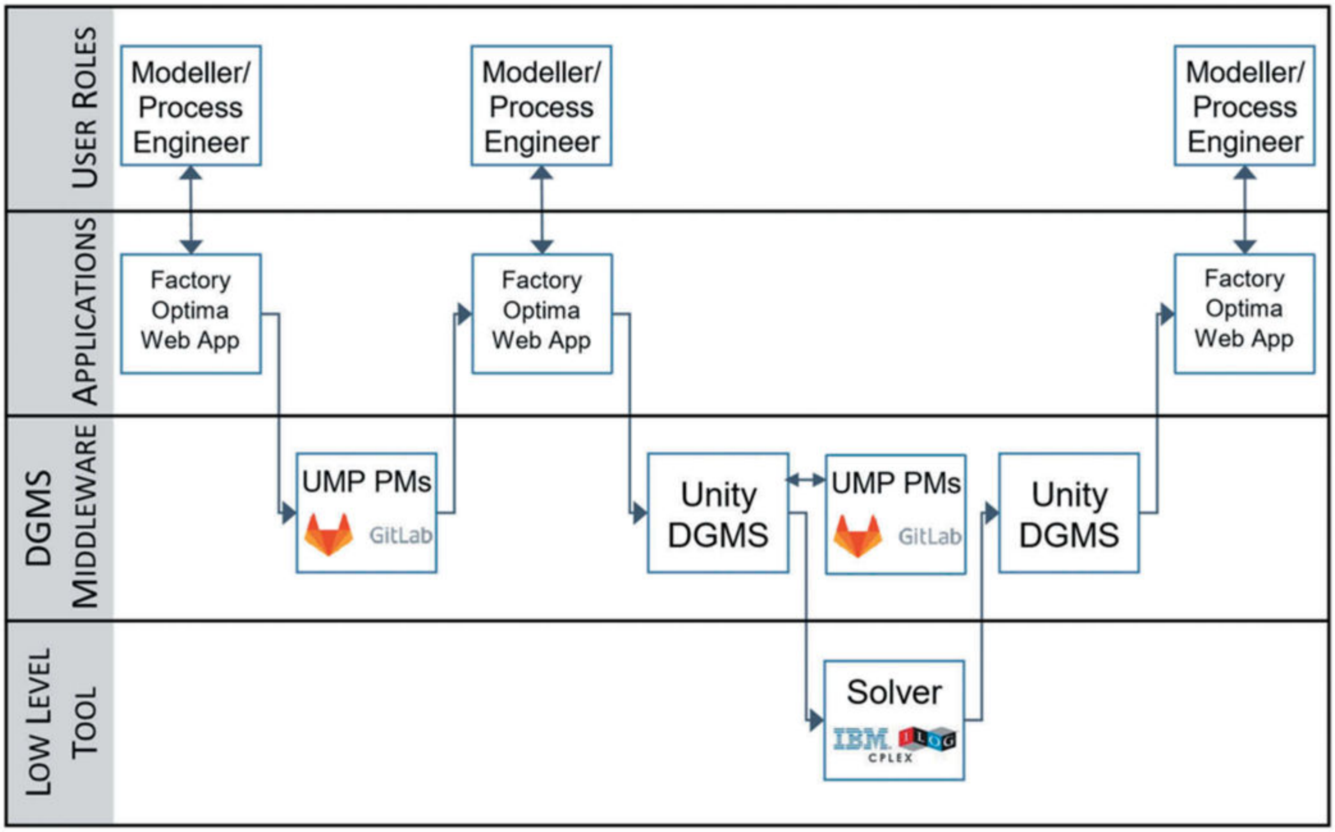


Figure 13.

Optimisation workflow of performance models in the system, adopted from (Brodsky et al. 2016a).

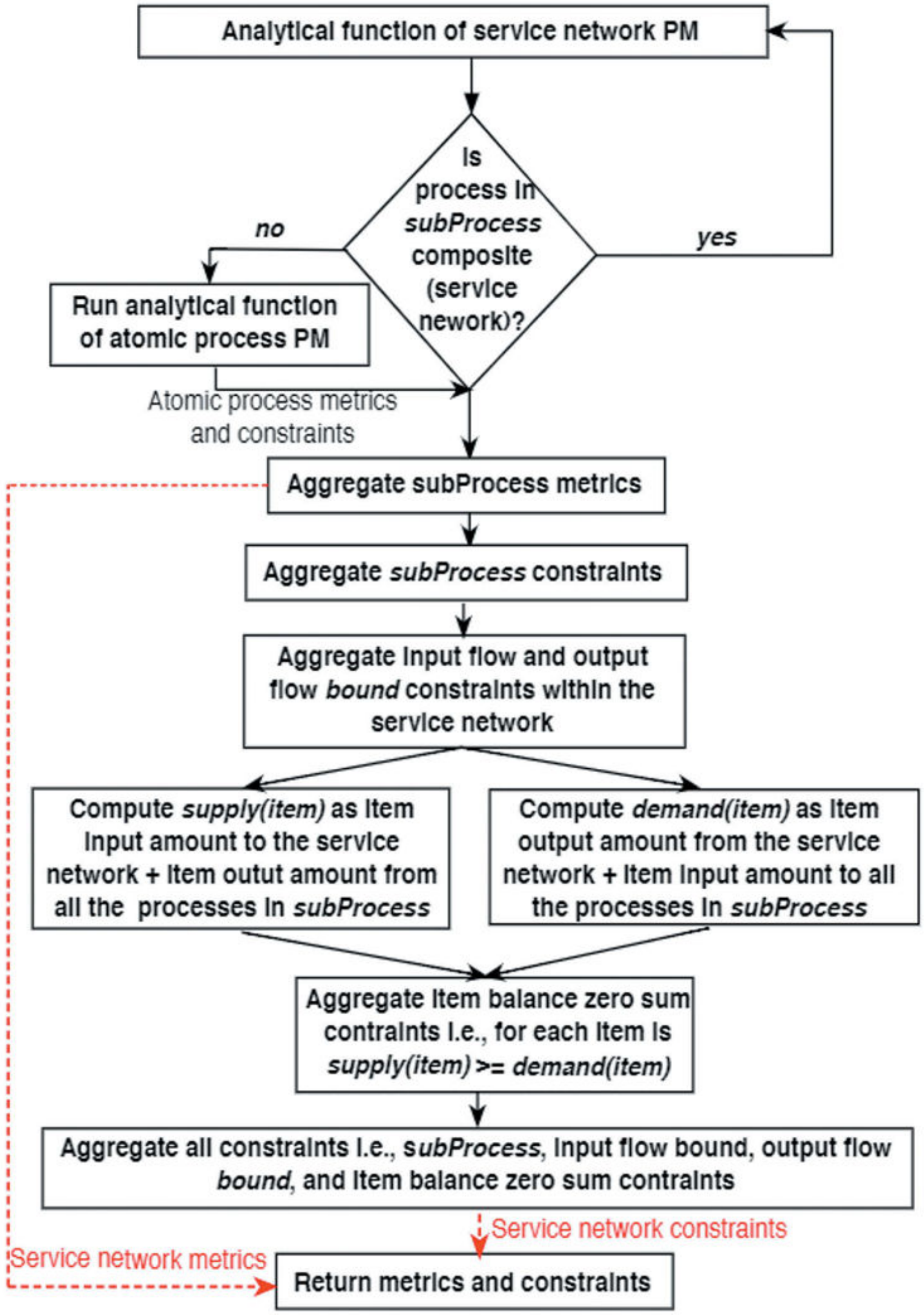

Analytical function of service network PM
Is process in *subProcess* composite (service nework)?
no
yes
Run analytical function of atomic process PM
Atomic process metrics and constraints
Aggregate subProcess metrics
Aggregate *subProcess* constraints
Aggregate input flow and output flow *bound* constraints within the service network
Compute *supply(item)* as item input amount to the service network + item outut amount from all the processes in *subProcess*
Compute *demand(item)* as item output amount from the service network + item input amount to all the processes in *subProcess*
Aggregate item balance zero sum contraints i.e., for each item is *supply(item)* >= *demand(item)*
Aggregate all constraints i.e., s*ubProcess*, input flow bound, output flow *bound*, and item balance zero sum contraints
Service network constraints
Service network metrics
Return metrics and constraints

Figure 14.

Overview of the analytical function of the service network PM.