Int J Adv Manuf Technol (2017) 88:337–357
DOI 10.1007/s00170-016-8761-7

ORIGINAL ARTICLE

Analysis and optimization based on reusable knowledge base of process performance models

Alexander Brodsky[1] · Guodong Shao[2] · Mohan Krishnamoorthy[1] · Anantha Narayanan[3] · Daniel Menascé[1] · Ronay Ak[2]

Received: 13 January 2016 / Accepted: 8 April 2016 / Published online: 23 April 2016

Abstract In this paper, we propose an architectural design and software framework for fast development of descriptive, diagnostic, predictive, and prescriptive analytics solutions for dynamic production processes. The proposed architecture and framework will support the storage of modular, extensible, and reusable knowledge base (KB) of process performance models. The approach requires developing automated methods that can translate the high-level models in the reusable KB into low-level specialized models required by a variety of underlying analysis tools, including data manipulation, optimization, statistical learning, estimation, and simulation. We also propose an organization and key structure for the reusable KB, composed of atomic and composite process performance models and domain-specific dashboards. Furthermore, we illustrate the use of the proposed architecture and framework by prototyping a decision support system for process engineers. The decision support system allows users to hierarchically compose and optimize dynamic production processes via a graphical user interface.

Keywords Smart manufacturing · Data analytics · Domain specific user interface · Optimization · Reusable knowledge base · Process performance models

✉ Guodong Shao
guodong.shao@nist.gov

Alexander Brodsky
brodsky@gmu.edu

Mohan Krishnamoorthy
mkrishn4@gmu.edu

Anantha Narayanan
anantha@umd.edu

Daniel Menascé
menasce@gmu.edu

Ronay Ak
ronay.ak@nist.gov

1 Department of Computer Science, George Mason University, Fairfax, VA, USA

2 Engineering Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA

3 Department of Mechanical Engineering, University of Maryland, College Park, MD, USA

1 Introduction

Smart manufacturing (SM) requires the collaboration of advanced manufacturing capabilities and digital technologies to create highly customizable products faster, cheaper, and greener. According to [1], "Next-generation software and computing architectures are needed to effectively mine data and use it to solve complex problems and enable decision-making based on a wide range of technical and business parameters." These software and computing architectures need capabilities to support the development of analysis and optimization solutions. These capabilities need to be designed for multiple operational levels, including manufacturing units, cells, production lines, factories, and supply chains [2].

The required analysis and optimization capabilities can be broadly classified as descriptive (what happened?) [3, 4], diagnostic (why did it happen?) [5, 6], predictive (what will happen?) [7, 8], and prescriptive (how can we make it happen?) analytics [9–11]. However, the current manufacturing practice is that analysis and optimization solutions are typically implemented from scratch, following a linear methodology. This leads to high-cost and long-duration development and 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 typically requires the use of specialized low-level mathematical abstractions and languages. As a result, the same manufacturing knowledge is often modeled multiple times using different specialized abstractions, instead of being modeled only 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 such as operators and process engineers.

Addressing the described limitations of current practice is the focus of this paper. More specifically, the contributions of this paper are as follows. First, we propose an architectural design and framework for fast development of software solutions for descriptive, diagnostic, predictive, and prescriptive analytics of dynamic production processes. The architecture adopts (1) the top layer of domain-specific modeling and analytics' graphical user interface (GUI) and (2) the low-level layer of computational tools. The uniqueness and novelty of the proposed architectural design and framework is its middleware layer, which is based on a reusable, modular, and extensible knowledge base (KB) of process performance models. Reusability of modular KB models could lead to considerable reduction in the development cost, time, and the required level of expertise. The key technical challenge lies in the development of a middleware analytics engine. This engine comprises algorithms and automatic methods that translate high-level uniform representations of performance models in the reusable KB into low-level specialized models required by each of the aforementioned underlying tools.

Second, we propose the organization and the key structure of the reusable KB, which consists of (1) an extensible library of atomic process performance models of unit manufacturing processes, (2) a library of composite process performance models, which can be constructed from the atomic process performance models using a GUI, and (3) a library of analytical views and dashboards designed for specific types of analysis for domain-specific users.

Third, to illustrate the use of the proposed design and framework, we prototype a decision support system that allows process engineers to (1) hierarchically compose dynamic production processes via a GUI and (2) perform deterministic and stochastic optimization of dynamic production processes. Users can pose optimization queries against atomic or composite process performance models without the need of mathematical or optimization modeling. The deterministic optimization is implemented by automatic translation of performance process models into formal optimization models expressed in Optimization Modeling Language (OPL) and solved using the *International Business Machines* (IBM) Corporation CPLEX mixed integer linear programming (MILP) solver, as described in [12]. The stochastic optimization is implemented by a heuristic algorithm from [13] based on a series of deterministic approximations, which significantly outperforms existing algorithms based on stochastic simulation. The graphical domain-specific modeling environment is implemented using the generic modeling environment (GME) [14].

The paper is organized as follows. Section 2 discusses the needs and challenges encountered in implementing analysis and optimization solutions. Section 3 provides the design of the architecture and framework that is based on reusable KB of process performance models. Section 4 extends the previous section by exemplifying the reusable KB. Section 5 introduces the prototype for the domain-specific SM decision support system. Section 6 discusses the implementation architecture for the prototype. Section 7 gives a more detailed discussion on related work and its limitations that we address in the paper. Finally, Section 8 concludes and discusses future work.

2 Required analysis and optimization capabilities

To discuss the required analysis and optimization capabilities in more detail, we use the diagram of the Tesla car manufacturing process example as depicted in Fig. 1. Aluminum coils are the input of the manufacturing process and are fed into two uncoiling machines that work in parallel to flatten the coils into aluminum plates. The plates are then sent to four different cutting machines to prepare for the four parts of a car: the left side, the underbody, the front, and the right side. After being cut, the aluminum plates are sent to die press machines after which they will be reinforced and welded. After assembly, the finished body is then washed, coated, and painted before the final operations are performed to produce a car.

Different analysis and optimization capabilities are required to analyze the performance of the production line and to achieve SM goals. These capabilities can be classified as descriptive, diagnostic, predictive, and prescriptive/optimization analytics.

Descriptive capabilities are needed to create a temporal sequence of sensor data automatically or semi-automatically. In the car manufacturing process, examples of sensor data include (a) line speeds of the uncoiling machines; (b) CO_2 emissions, water consumption, energy consumption, and temperature of the individual machines or the entire plant; and (c) levels of the work-in-progress inventories. This collected data may be filtered and aggregated over time and manufacturing levels. In addition, some preprocessing or transformation of certain sensor data may be performed to improve visualization.

Diagnostic capabilities are needed to detect undesirable deviations from what is considered normal behavior.

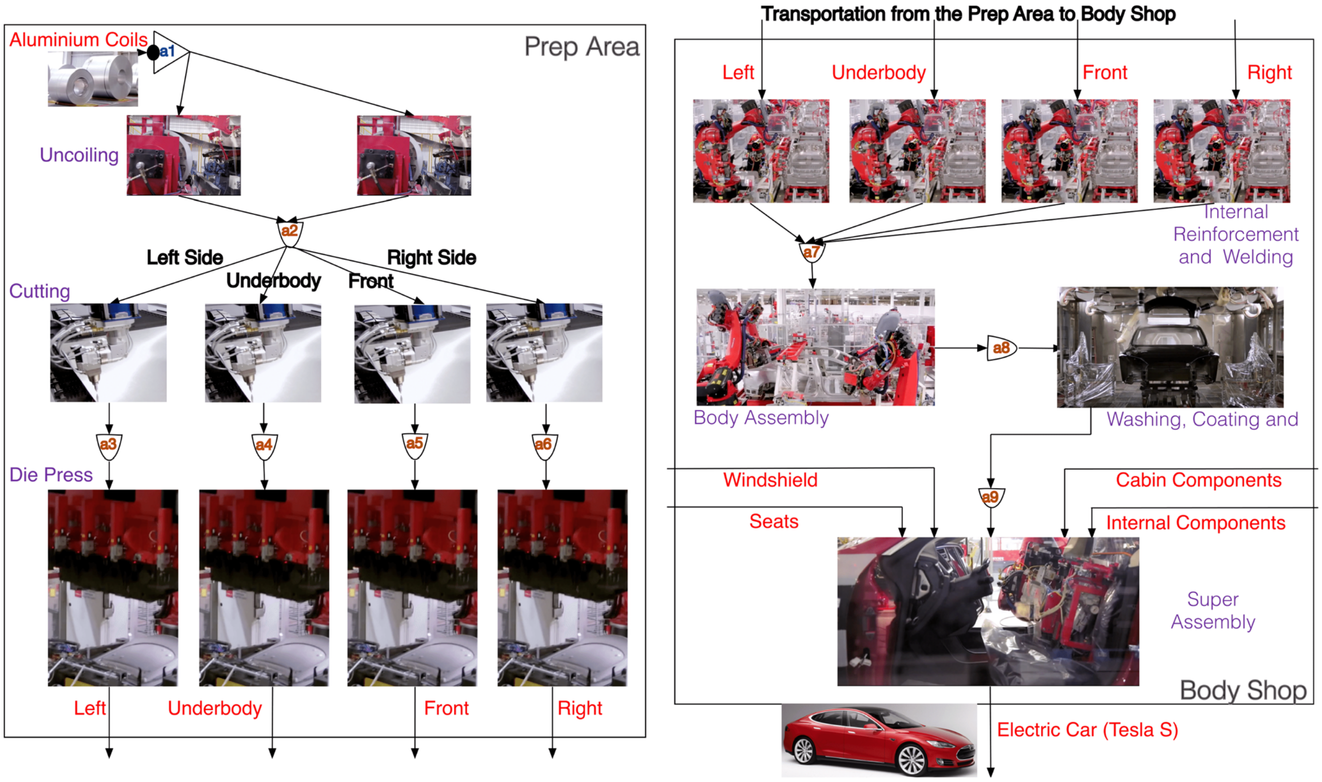


Fig. 1 Tesla car manufacturing

Detecting such deviations requires continuous testing for any significant statistical difference between the predicted and observed values of important metrics. The data needed for this testing comes from the descriptive tasks described above. For instance, after determining the minimum and maximum acceptable values of a metric, such as total energy consumed for the uncoiling machine, the testing can detect if the observed value obtained from sensor data is within these bounds. If it is not, then the process operator will be notified and asked to find the causes of the problem and take corrective actions to eliminate them.

Predictive capabilities are needed to estimate and learn the values of various performance metrics as a function of machine and process controls. These capabilities often come in the form of statistical learning techniques such as regression analysis. For example, a production engineer may want to learn the energy consumption of a die press machine as a function of its pressing speed and nominal pressure. The engineer may use the learned results to predict future performance of the process. The engineer might also use this result together with a stochastic simulation to predict the increase in the energy consumption of the die press machine if, for example, the pressing speed is increased by 15 %.

Prescriptive capabilities, which include optimization techniques such as mathematical programming (MP) and constraint programming (CP), are needed to choose among alternative actions. For instance, upon discovering a spike in total energy consumed and fixing the machine's parameters, the operator may need to (1) determine new machine settings within the allowed bounds and (2) rebalance the workload distribution to meet the production schedule and minimize the total energy consumption.

As depicted in Fig. 2, a typical software architecture to implement the analysis and optimization capabilities includes (1) a top layer of a domain-specific GUI for manufacturing users and (2) a bottom layer of specialized tools and languages. The core implementation challenge, however, lies in the translation of the top-layer tasks into the low level of abstractions of the tools at the bottom layer (the question mark in Fig. 2). We will describe a variety of tools and then discuss the core implementation challenge in more detail in Sect. 7.

3 Architectural design and framework based on reusable KB of process performance models

To overcome the challenges in developing SM analysis and optimization capabilities, we advocate for a paradigm shift from the nonreusable modeling approach, which is a linear task-centric methodology of gathering requirements, identifying data sources, and developing one of models and algorithm using a range of modeling languages and tools to perform analysis (see Fig. 3) to a new one. The key idea is to adopt the approach more commonly used in database management

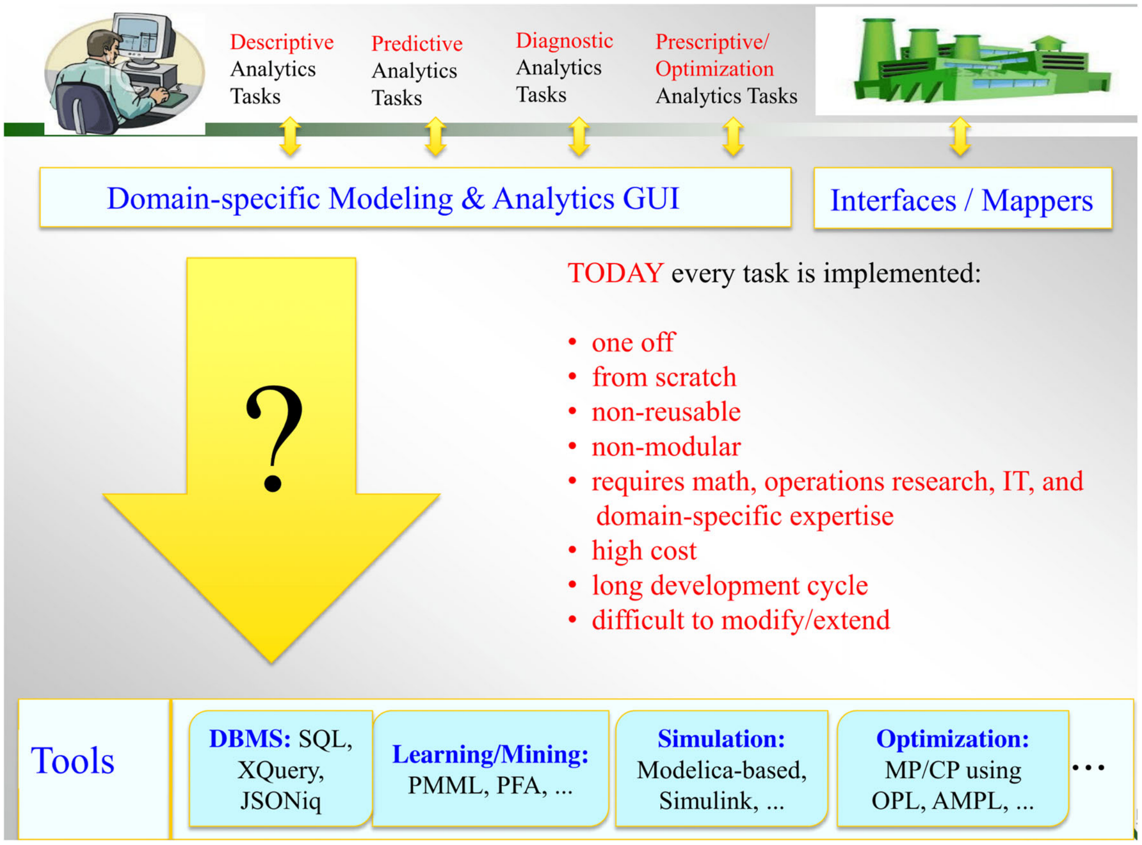

Descriptive Analytics Tasks
Predictive Analytics Tasks
Diagnostic Analytics Tasks
Prescriptive/ Optimization Analytics Tasks
Domain-specific Modeling & Analytics GUI
Interfaces / Mappers
?
TODAY every task is implemented:
• one off
• from scratch
• non-reusable
• non-modular
• requires math, operations research, IT, and domain-specific expertise
• high cost
• long development cycle
• difficult to modify/extend
Tools
DBMS: SQL, XQuery, JSONiq
Learning/Mining: PMML, PFA, ...
Simulation: Modelica-based, Simulink, ...
Optimization: MP/CP using OPL, AMPL, ...
...

Fig. 2 Challenges in implementing analytics functionality using tools

systems (DBMSs). In a DBMS, a central data repository is updated continuously from multiple sources. DBMS users pose declarative data manipulation queries—using one of several query languages—against the database. The DBMS engine translates the declarative queries into efficient, low-level, data processing code. In the case of SM analysis and optimization solutions, there is a need to manage not only the data, but also the analytical knowledge. Rather than posing declarative data manipulation queries, our goal is to pose analytical queries needed to execute descriptive, diagnostic, predictive, and prescriptive/optimization tasks.

To achieve this goal, we have developed the SM analysis and optimization conceptual architecture shown in Fig. 4. The uniqueness and novelty of the proposed architecture is that it is centered on a reusable, modular, and extensible KB of process performance models (the middle layer in Fig. 4).

The key technical challenge in realizing a system based on this architecture lies in developing specialized algorithms that automatically translate the high-level, uniform representation of manufacturing models in the KB into the low-level, specialized models required by each of the underlying tools. The analytical models (AMs) in the KB are mathematical models that represent data, schema, parameters, variables, functions, constraints, and uncertainty. Using these models is easy and done through the aforementioned analytical queries. However, creating these reusable models requires multiple levels of knowledge and expertise. We classify these models in the KB into three libraries of *atomic process performance models*, *composite process performance models*, and *analytical views*, according to types of knowledge or expertise required for developing these models.

3.1 Atomic models

Atomic models, i.e., atomic process performance models (in the middle of KB in Fig. 4), require the most expertise and effort to build; however, they are also the most reusable ones. The atomic model library contains a classification hierarchy of prebuilt performance models for atomic manufacturing processes. An atomic process is an end process in which there is no subprocess. Each performance model contains the process parameters, control variables, performance metrics, and

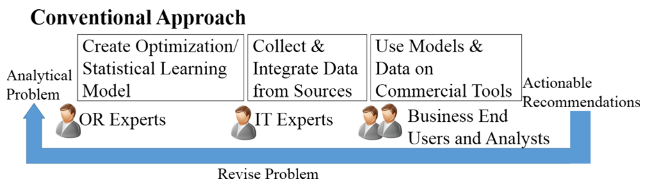

Conventional Approach
Analytical Problem
Create Optimization/ Statistical Learning Model
Collect & Integrate Data from Sources
Use Models & Data on Commercial Tools
Actionable Recommendations
OR Experts
IT Experts
Business End Users and Analysts
Revise Problem

Fig. 3 Conventional approach to analytics solutions

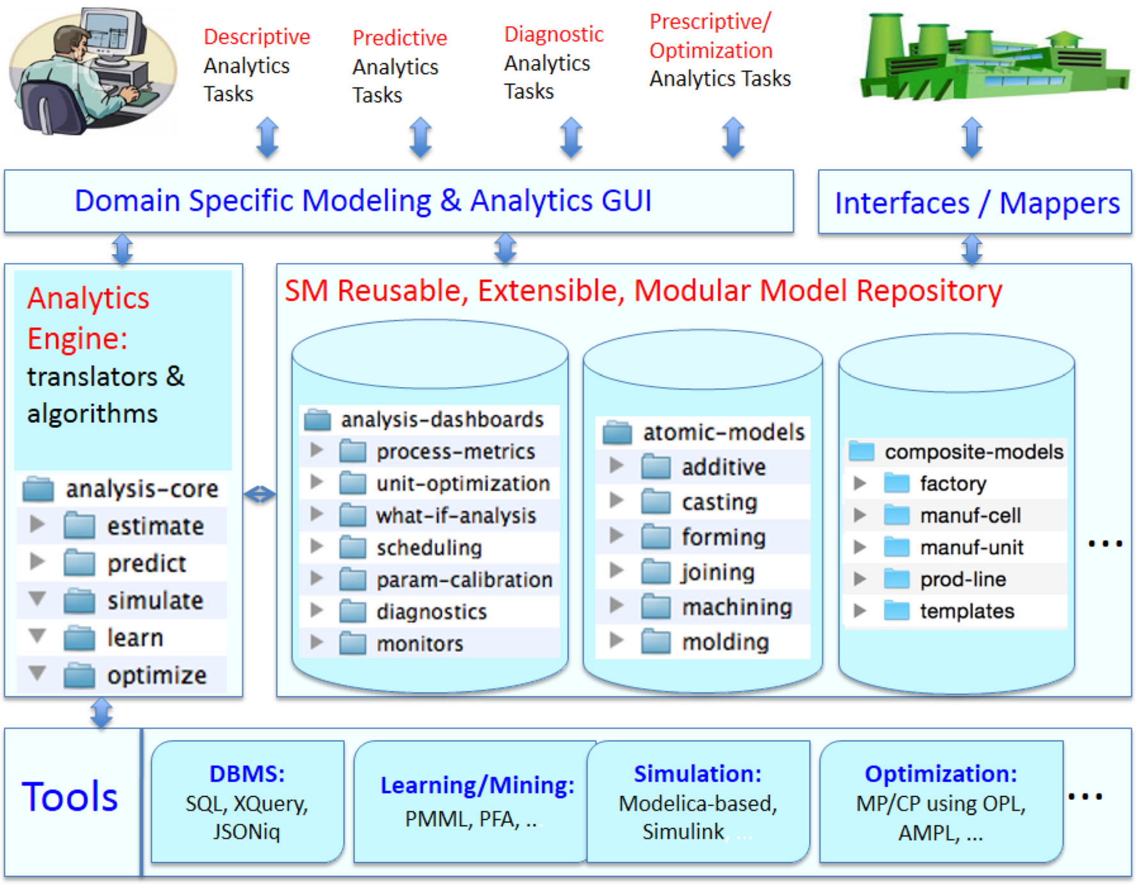

Descriptive Analytics Tasks
Predictive Analytics Tasks
Diagnostic Analytics Tasks
Prescriptive/ Optimization Analytics Tasks
Domain Specific Modeling & Analytics GUI
Interfaces / Mappers
Analytics Engine: translators & algorithms
analysis-core
estimate
predict
simulate
learn
optimize
SM Reusable, Extensible, Modular Model Repository
analysis-dashboards
process-metrics
unit-optimization
what-if-analysis
scheduling
param-calibration
diagnostics
monitors
atomic-models
additive
casting
forming
joining
machining
molding
composite-models
factory
manuf-cell
manuf-unit
prod-line
templates
...
Tools
DBMS: SQL, XQuery, JSONiq
Learning/Mining: PMML, PFA, ..
Simulation: Modelica-based, Simulink ...
Optimization: MP/CP using OPL, AMPL, ...
...

Fig. 4 Proposed conceptual architecture

feasibility constraints—as well as quantification of uncertainty associated with metrics and constraints. For instance, for the injection molding process, the metrics of energy consumption per part, cycle time, and throughput can be expressed as a function of (1) *parameters* such as the number of cavities, the volume of the part, and the material characteristics and (2) *control variables* such as injection pressure and flow rate.

Building atomic process performance models requires knowledge in process engineering and data manipulation languages. Domain-specific, process engineering knowledge includes an understanding of the equations defining performance metrics and constraints. The knowledge of data manipulation languages is needed to encode data transformation and equations. However, building atomic process performance models does not require expertise in optimization, MP, or statistical learning.

Once built, the atomic process performance models can be used by both end users and process engineers. Operational end users will invoke analytics-core functions of *compute*, *predict*, *learn*, *simulate*, and *optimize* to run the models. They will also use the various analytical views described later in this section to monitor the performance of the process, against both the atomic process performance model and historical performance data. Process engineers use these atomic process performance models as the basis for creating composite performance models of composite processes.

3.2 Composite models

A composite process is recursively composed of atomic processes and the associated aggregators, information flows, and timing constraints. The composite process performance model library contains performance models of such composite processes at different levels of granularity, such units, cells, lines, factories, and enterprises. Constructing composite process performance models is the usual task of manufacturing process engineers (MPEs); to construct such models, an MPE only needs to specify the processes involved in the design and the rules for composition. Rules specify the flow of materials, parts, products, and information through the processes. Composite process performance models can be constructed simply by using a drag-and-drop GUI. Figure 5 shows an example of such a model built using a GUI for the Tesla car manufacturing (first explained in Fig. 1).

System-level metrics and feasibility constraints can be determined from the metrics and constraints of its subprocesses, recursively. The metric computation and feasibility constraint evaluation are done by the system using the corresponding

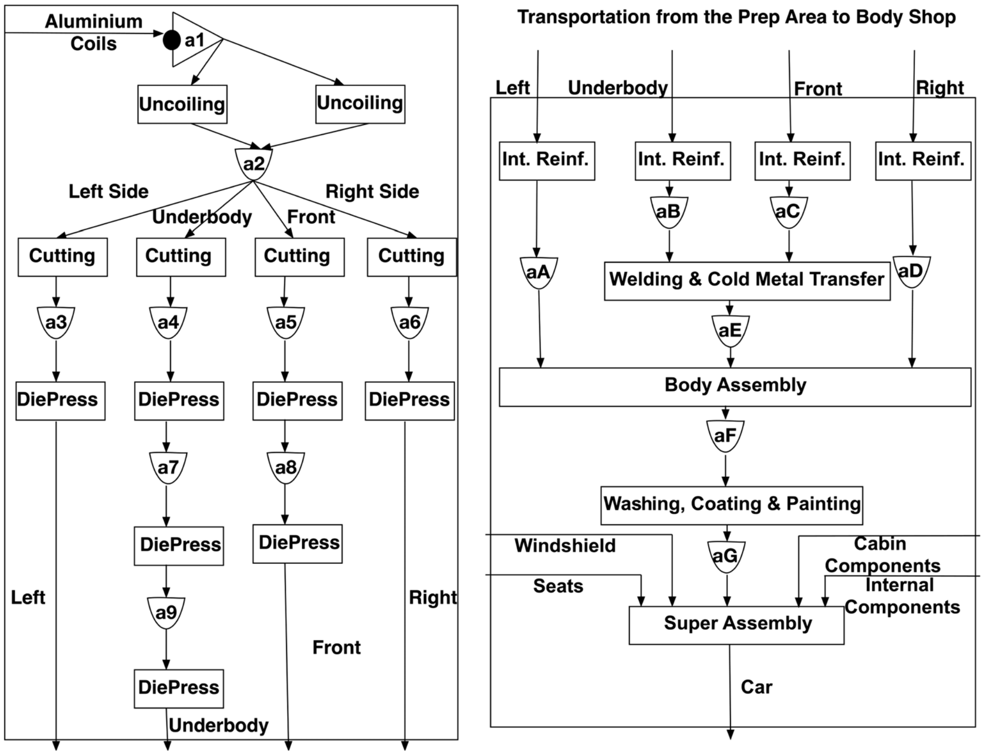


Fig. 5 Tesla car manufacturing composite process performance model diagrams

atomic and composite process performance models that use the model composition template. Thus, a composite process performance model, just like an atomic process performance model, can be thought of as having the same characteristics, namely, parameters, control variables, metrics, and feasibility constraints. It can be used recursively for analysis or as a component of a higher-level process.

Once built, the operational use of composite process performance models is similar to the use of atomic process performance models; i.e., end users can use the analytics-core functions or analytical views to perform analytical tasks on these models. Analytical views are dashboard-like templates, which are implemented using the analytics-core functions (compute, predict, learn, simulate, and optimize) together with a data manipulation language. As noted above, this idea is similar to how relational database views are constructed from database tables using SQL.

3.3 Analytical views

Analytical views can be implemented by a data analyst who has the required analytics knowledge and can use a data manipulation language such as SQL or JSONiq. However, the analyst does not need to have any expertise in mathematical modeling, domain knowledge, or equation writing. Examples of analytical views include (1) dashboard of the energy consumed over a period of time, (2) diagnosis of the statistical difference between the expected and observed power consumption, (3) visualization of the supervisory control and data acquisition (SCADA) data, (4) parameter calibration of the power consumption as a function of machine controls, (5) composite process performance model metric computation as a function of individual machine metrics, (6) scheduling of a job to meet the demand, (7) optimizing the machine operations to minimize the power consumption such that the demand is satisfied, and (8) what-if analysis to find the impact on demand satisfaction and power consumption if one of the machines was switched off.

The analytics-core methods (compute, predict, learn, simulate, and optimize) are part of the *Analytics Engine* (see Fig. 4). Implementing these methods involves and requires reduction and compilation techniques, as well as specialized optimization and learning algorithms. However, once implemented, the analytics-core methods will allow fast and easy implementation of domain-specific analytical views, without the need to understand the lower-level abstractions of the underlying computational tools. Furthermore, they allow manufacturing end users to directly pose analytical queries against the atomic and composite process performance models, thus enabling their reusability. A more detailed description of the reusable KB along with some guidelines to model the AMs is described in the next section.

3.4 JSON and JSONiq

There are a number of data models and corresponding data manipulation languages that we can choose from. They include (1) the relational model and SQL; (2) the XML data model and

XQuery; and (3) the JavaScript Object Notation (JSON) data model [15] and JSONiq [16]. We believe that the “flat” relational model and SQL are not sufficiently flexible for modeling manufacturing processes and systems. Out of XML and JSON, we decided to use the *JSON* data model because it is more compact and lightweight than the XML data model and because it is broadly used today for data analytics and data integration. This subsection briefly overviews JSON and JSONiq. Examples of *JSON* and *JSONiq* appear in Sect. 4.

JSON is a lightweight, data interchange format that facilitates structured data interchange between all programming languages. It defines a small set of structuring rules for the portable representation of structured data. JSON format is text only, just like XML. Therefore, JSON is not only easy for machines to parse and generate but also easy for humans to read and write [15].

JSON Schema is a JSON media type for defining the structure of JSON data. It provides a contract for what JSON data is required for a given application and how to interact with data. JSON Schema can be used to validate if a given JSON document (an instance) satisfies a certain number of criteria. At its core, the JSON schema is made up of data structures such as object, array, number, string, Boolean, and null. With these simple data types, all kinds of structured data can be represented. JSON Schema itself is written in JSON. JSON Schema is data itself, not a computer program. Using JSON data schema, one is able to provide guidelines and data formats needed to create a JSON data model for the inputs (parameters and control variables) of either an atomic unit process or a composite process. In addition, through JSON data schema, one can represent the input constraints for the values of the KB modules in the data model.

JSONiq is a query and processing language specifically designed for JSON data models. The main source of inspiration behind JSONiq is XQuery, which has been proven to be a successful and productive query language for semi-structured data such as XML [15]. JSONiq, however, can do more than queries; it can describe data processing programs created from transformations, selections, joins, data enrichment, information extraction, information cleaning, and so on [16]. In addition, A JSONiq program is an expression; the result of the program is the result of the evaluation of the expression [15].

4 Reusable KB

The proposed architecture contains a KB that contains multiple AMs, which may include data, schema, parameters, variables, functions, constraints, and uncertainty. Modules in the KB are of three types: atomic model (atomic process performance models) type, composite model (composite process performance model) type, or analytical view type. The atomic process performance model (atomic model) library will map to the different types of manufacturing machines or processes. The atomic model library can be organized differently for different use cases. In this section, we show one such organization. In addition, we provide an example JSON input structure and JSONiq physics equations for the injection-molding atomic process performance model. We also describe general guidelines for developing and storing atomic process performance models. Then, we provide an example for the composite process performance model based on the *Buffered Temporal Flow Processes* (BTFP) by giving the JSON code snippets of the different components of the composite processes. Finally, we give an example for the analytical view of a deterministic optimization that can be used against the atomic or composite process performance models in the KB.

4.1 Example of atomic model library

Figure 6 shows an example organization of the atomic models. These models may include a range of manufacturing processes such as casting, forming, joining, machining, molding, and additive manufacturing. Casting process involves pouring a liquid material into a mold, which contains a hollow cavity of the desired shape, and then allowing the liquid to solidify. Examples of casting include die casting and sand casting. Forming process uses suitable stresses to deform plastic materials to produce different shapes. Examples of forming include bending, pressing, and rolling. Joining process connects metals together to create parts, assemblies, or large-scale structures. Examples of joining include sintering and soldering. Machining process removes material from a workpiece to produce parts based on NC programs. Examples of machining include milling and drilling. Molding is the process of shaping liquid or pliable raw material using a mold. Examples of molding include foam molding and injection molding. Additive manufacturing is a process that fabricates products by adding layer-upon-layer of material. Examples of this process include 3D printing and powder-based fusion. To understand how the processes are modeled and stored in the KB, we use injection molding as an example next.

4.2 Example of atomic process performance model: injection molding

We use JSON and JSONiq to describe the atomic process performance models (Fig. 7). An example JSON input structure and JSONiq physics equations are provided for the injection molding machine. The injection molding process consists of heating thermoplastic material until it melts and then forcing this molten material into a mold (die) where it cools and solidifies [17]. Consequently, the injection molding process consists of three major subprocesses: (1) melting, injecting, or filling;

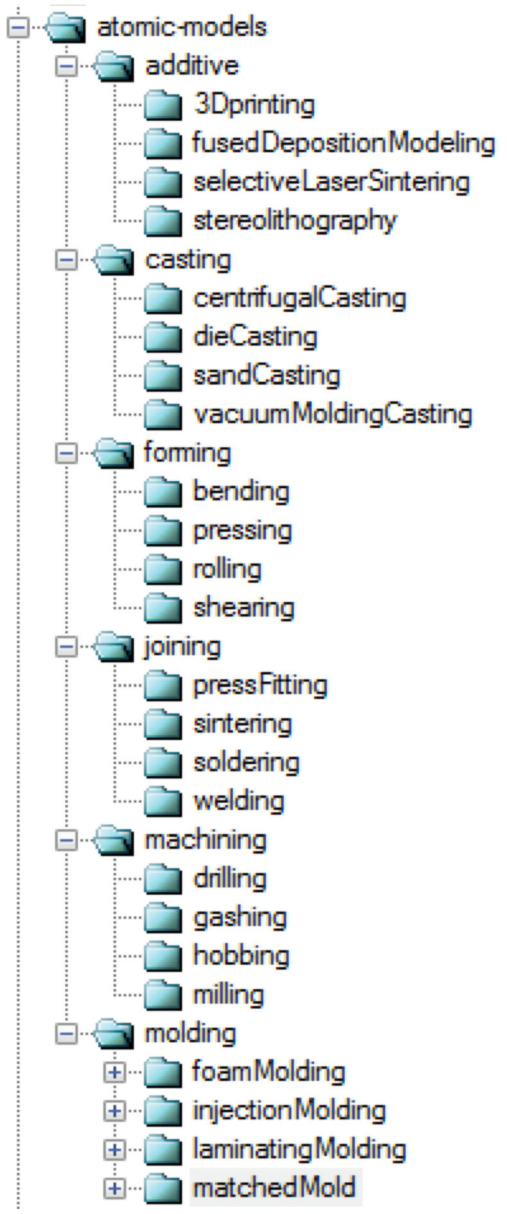


Fig. 6 An illustration of the atomic model classification hierarchy

(2) cooling; and (3) ejection and resetting. The resulting cycle time, t_{cycle} can be formulated as according to [17–19]:

$$t_{\text{cycle}} = t_{\text{inj}} + t_{\text{cool}} + t_{\text{reset}} \tag{1}$$

where t_{inj} is the injection time, t_{cool} is the cooling time, and t_{reset} is the reset time.

Using process parameters, machine parameters, and material parameters, one can estimate a number of key performance metrics including cycle time, energy use, water consumption, and part throughput. For example, the energy required for melting E_{melt}one-shot volume of plastic is as follows:

$$E_{\text{melt}} = P_{\text{melt}} \times \frac{V_{\text{shot}}}{Q} \tag{2}$$

where (P_{melt}) is the power consumed by melting, V_{shot} is the shot volume, and Q is the flow rate of plastic.

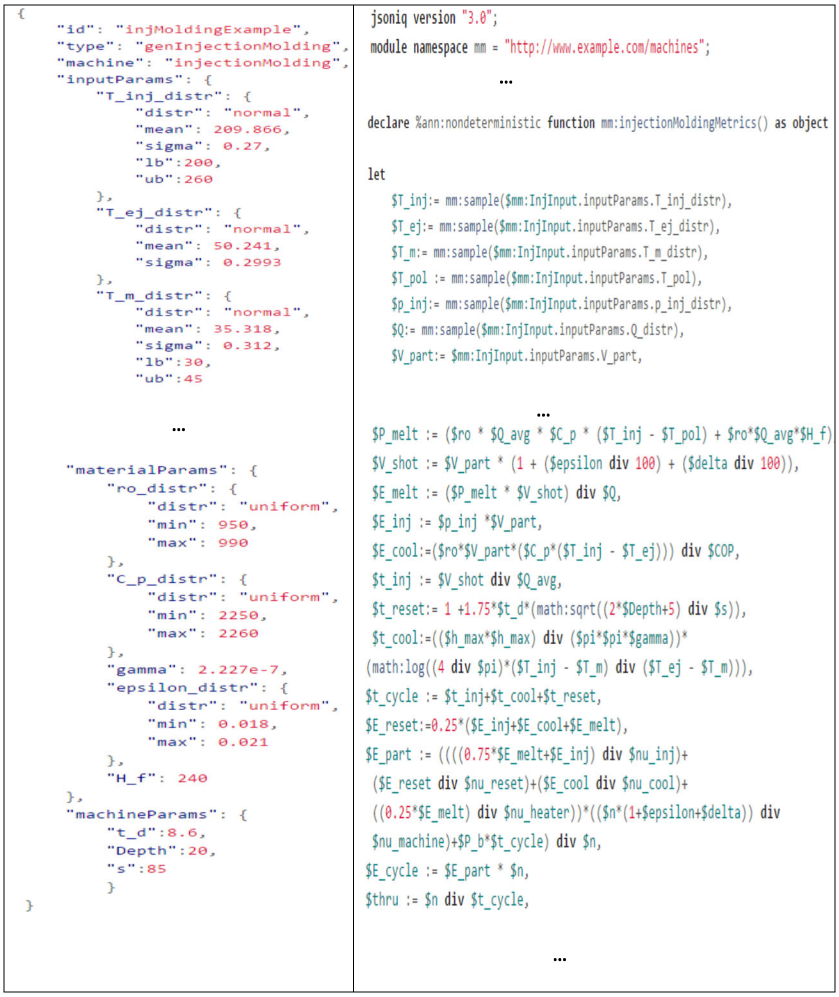
The JSON structure defined for the atomic process performance model contains all the process parameters, control variables, constraints, and coefficients. Figure 5 shows the JSONiq code for computing key performance indicators (KPIs) of the injection molding process using the JSON input data. For instance, Eq. (2) is encoded as a variable, $*E_melt*, in JSONiq and illustrated in the figure as $*E_melt*:= ($*P_melt* * $*V_shot*) *div* $*Q*. Note that values defined by the function *sample* in Fig. 5 including $*T_inj*, $*T_ej*, and $*Q* are random variables and so are all the derived variables such as $*E_melt*. The JSONiq structure includes three parts: top, middle, and bottom. At the top, there is a query header where we define namespace and import the relevant modules; in the middle, we extract and transform the data from the JSON document into JSONiq, and at the bottom, we define the functions, for, let, where, order by, return (FLWOR) expressions, and the equations to compute the quantity of interests.

Each process may have dependent variables including metrics and KPIs such as total cycle time, energy consumption, and cost. Using the JSON document, we create the inputs, the functions, and the equations needed to compute these dependent variables for each subprocess. Note that computations and equations are encoded in JSONiq with parameters and inputs imported from the JSON data model; the *process-dependent* variables, as a function of parameters and control variables, are encoded in JSONiq.

4.3 Example of composite process performance model: Tesla prep

In this subsection, we provide an example composite process performance model based on the BTFP Tesla car manufacturing described in Sect. 2. BTFP is a class of processes where the states of the machines, inventories, and the whole process change over time until process completion. BTFP can be used to model either atomic machines or an entire manufacturing floor. In the latter case, BTFP processes need to capture the variables, metrics, and constraints of all the entities on the manufacturing floor. Figure 8 shows the associated JSON structure including all the parameters, variables, metrics, and temporal information for both atomic and composite processes.

The JSON structure is an analytical module object for the prep composite process. This structure contains the Tesla time setting and defines the temporal setting for the Tesla prep process. In BTFP processes, time is divided into time intervals of duration Δt, where time intervals start and end at time points (t). A time interval (also known as a period) is denoted by $p_{i+1} = (t_i,\ t_{i+1})$. In this example, there are 18 periods (*noPeriods*) and the time starts from time point 0 (*lastTP*).



```
{
    "id": "injMoldingExample",
    "type": "genInjectionMolding",
    "machine": "injectionMolding",
    "inputParams": {
        "T_inj_distr": {
            "distr": "normal",
            "mean": 209.866,
            "sigma": 0.27,
            "lb":200,
            "ub":260
        },
        "T_ej_distr": {
            "distr": "normal",
            "mean": 50.241,
            "sigma": 0.2993
        },
        "T_m_distr": {
            "distr": "normal",
            "mean": 35.318,
            "sigma": 0.312,
            "lb":30,
            "ub":45


                    …

     "materialParams": {
         "ro_distr": {
             "distr": "uniform",
             "min": 950,
             "max": 990
         },
         "C_p_distr": {
             "distr": "uniform",
             "min": 2250,
             "max": 2260
         },
         "gamma": 2.227e-7,
         "epsilon_distr": {
             "distr": "uniform",
             "min": 0.018,
             "max": 0.021
         },
         "H_f": 240
     },
     "machineParams": {
         "t_d":8.6,
         "Depth":20,
         "s":85
         }
 }
```

```
jsoniq version "3.0";
module namespace mm = "http://www.example.com/machines";
                    …

declare %ann:nondeterministic function mm:injectionMoldingMetrics() as object

let
   $T_inj:= mm:sample($mm:InjInput.inputParams.T_inj_distr),
   $T_ej:= mm:sample($mm:InjInput.inputParams.T_ej_distr),
   $T_m:= mm:sample($mm:InjInput.inputParams.T_m_distr),
   $T_pol := mm:sample($mm:InjInput.inputParams.T_pol),
   $p_inj:= mm:sample($mm:InjInput.inputParams.p_inj_distr),
   $Q:= mm:sample($mm:InjInput.inputParams.Q_distr),
   $V_part:= $mm:InjInput.inputParams.V_part,


                    …
 $P_melt := ($ro * $Q_avg * $C_p * ($T_inj - $T_pol) + $ro*$Q_avg*$H_f)
 $V_shot := $V_part * (1 + ($epsilon div 100) + ($delta div 100)),
 $E_melt := ($P_melt * $V_shot) div $Q,
 $E_inj := $p_inj *$V_part,
 $E_cool:=($ro*$V_part*($C_p*($T_inj - $T_ej))) div $COP,
 $t_inj := $V_shot div $Q_avg,
 $t_reset:= 1 +1.75*$t_d*(math:sqrt((2*$Depth+5) div $s)),
 $t_cool:=(($h_max*$h_max) div ($pi*$pi*$gamma))*
(math:log((4 div $pi)*($T_inj - $T_m) div ($T_ej - $T_m))),
$t_cycle := $t_inj+$t_cool+$t_reset,
$E_reset:=0.25*($E_inj+$E_cool+$E_melt),
$E_part := ((((0.75*$E_melt+$E_inj) div $nu_inj)+
 ($E_reset div $nu_reset)+($E_cool div $nu_cool)+
 ((0.25*$E_melt) div $nu_heater))*(($n*(1+$epsilon+$delta)) div
 $nu_machine)+$P_b*$t_cycle) div $n,
$E_cycle := $E_part * $n,
$thru := $n div $t_cycle,


                    …
```

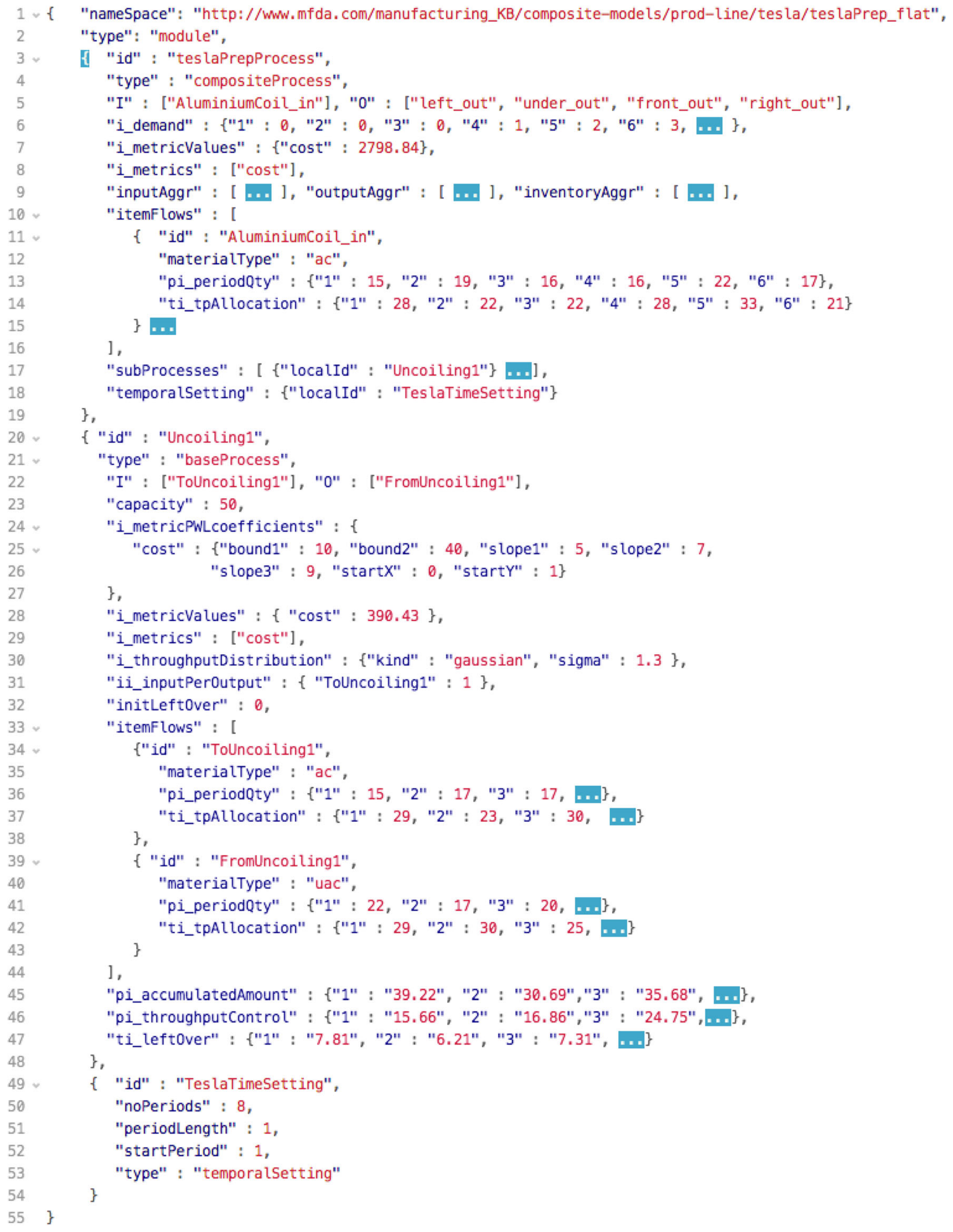
Fig. 7 Injection molding atomic process performance model: **a** JSON data representation and **b** JSONiq formulation

The JSON structure also contains the subprocess for the uncoiling 1 machine. This structure provides the parameters, variables, and metrics for the uncoiling 1 machine. They are inputs (*I*), outputs (*O*), machine *capacity*, metric type (*i-_metrics*), metric values (*i_metricValues*), number of inputs required per output produced (*ii_inputPerOutput*), the incoming and outgoing flow objects (*itemFlows*), the speed (control variable) of the machine at each period (*pi_throughputControl*), the amount of items accumulated in each period (*pi_accumulateAmount*), and the number of items left over in each period (*ti_leftOver*). The atomic process performance model may also have coefficients such as those for piecewise linear functions for calculating the cost metric (*i_metricPWLcoefficients*). The speed of the machines may be stochastic, and the parameters to the random value function are a part of the machine model (*i_throughputDistribution*). Although only the structure of the uncoiling 1 machine is shown here, the other subprocesses have a similar structure with different parameter, variable, and metric values.

Finally, the JSON structure contains the composite process performance model for the prep process (lines 3 to 19 in Fig. 8) that encapsulates the reference to the time settings and the subprocesses discussed above. The structure also contains the parameters, variables, and metrics for the composite Tesla prep process such as its inputs (*I*), outputs (*O*), the process demand (*i_demand*), metric type (*i_metrics*), metric values (*i_metricValues*), and the incoming and outgoing flow objects (*itemFlows*). Additionally, the prep process also contains the structures for the storage and distribution aggregators (*inventoryAggr*), e.g., a2 in Fig. 5, input distribution aggregator (*inputAggr*), e.g., a5 in Fig. 1, and output distribution aggregator (*outputAggr*). Due to lack of space, these structures have been left out in Fig. 8. However, more details about these and other BTFP components can be found in [12].

4.4 Example of analytical view: deterministic optimization

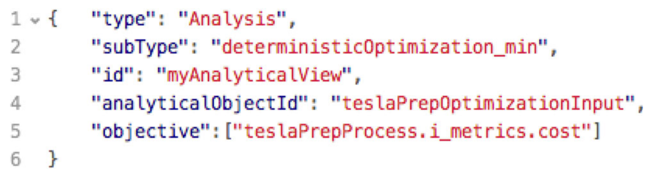
Analytical views allow users to perform analytical tasks on all of the models described above. Figure 9 shows the deterministic optimization analytical view in the JSON structure for the *teslaPrepOptimizationInput*, which is similar to the one described in Fig. 8. The difference is that the control variables



```
{   "nameSpace": "http://www.mfda.com/manufacturing_KB/composite-models/prod-line/tesla/teslaPrep_flat",
    "type": "module",
    {  "id" : "teslaPrepProcess",
       "type" : "compositeProcess",
       "I" : ["AluminiumCoil_in"], "O" : ["left_out", "under_out", "front_out", "right_out"],
       "i_demand" : {"1" : 0, "2" : 0, "3" : 0, "4" : 1, "5" : 2, "6" : 3, ... },
       "i_metricValues" : {"cost" : 2798.84},
       "i_metrics" : ["cost"],
       "inputAggr" : [ ... ], "outputAggr" : [ ... ], "inventoryAggr" : [ ... ],
       "itemFlows" : [
          {  "id" : "AluminiumCoil_in",
             "materialType" : "ac",
             "pi_periodQty" : {"1" : 15, "2" : 19, "3" : 16, "4" : 16, "5" : 22, "6" : 17},
             "ti_tpAllocation" : {"1" : 28, "2" : 22, "3" : 22, "4" : 28, "5" : 33, "6" : 21}
          } ...
       ],
       "subProcesses" : [ {"localId" : "Uncoiling1"} ...],
       "temporalSetting" : {"localId" : "TeslaTimeSetting"}
    },
    { "id" : "Uncoiling1",
      "type" : "baseProcess",
       "I" : ["ToUncoiling1"], "O" : ["FromUncoiling1"],
       "capacity" : 50,
       "i_metricPWLcoefficients" : {
          "cost" : {"bound1" : 10, "bound2" : 40, "slope1" : 5, "slope2" : 7,
                   "slope3" : 9, "startX" : 0, "startY" : 1}
       },
       "i_metricValues" : { "cost" : 390.43 },
       "i_metrics" : ["cost"],
       "i_throughputDistribution" : {"kind" : "gaussian", "sigma" : 1.3 },
       "ii_inputPerOutput" : { "ToUncoiling1" : 1 },
       "initLeftOver" : 0,
       "itemFlows" : [
          {"id" : "ToUncoiling1",
             "materialType" : "ac",
             "pi_periodQty" : {"1" : 15, "2" : 17, "3" : 17, ...},
             "ti_tpAllocation" : {"1" : 29, "2" : 23, "3" : 30,  ...}
          },
          { "id" : "FromUncoiling1",
             "materialType" : "uac",
             "pi_periodQty" : {"1" : 22, "2" : 17, "3" : 20, ...},
             "ti_tpAllocation" : {"1" : 29, "2" : 30, "3" : 25, ...}
          }
       ],
       "pi_accumulatedAmount" : {"1" : "39.22", "2" : "30.69","3" : "35.68", ...},
       "pi_throughputControl" : {"1" : "15.66", "2" : "16.86","3" : "24.75",...},
       "ti_leftOver" : {"1" : "7.81", "2" : "6.21", "3" : "7.31", ...}
     },
     {  "id" : "TeslaTimeSetting",
        "noPeriods" : 8,
        "periodLength" : 1,
        "startPeriod" : 1,
        "type" : "temporalSetting"
     }
}
```

Fig. 8 JSON structure for the Tesla prep composite process performance model

(e.g., *throughputControl* of the uncoiling 1 process) and the dependent variables (e.g., *periodQty* of the itemFlows) are annotated to be determined by the optimization engine. Using this structure, the analytical view defines the type of analysis to be performed on the Tesla prep process. In this example, it is MILP deterministic optimization. This JSON



```
{   "type": "Analysis",
    "subType": "deterministicOptimization_min",
    "id": "myAnalyticalView",
    "analyticalObjectId": "teslaPrepOptimizationInput",
    "objective":["teslaPrepProcess.i_metrics.cost"]
}
```

Fig. 9 JSON structure for the deterministic optimization analytical view

structure may also contain other parameters pertaining to the selected analysis type, such as the objective function (*objective*). Analytical views may have additional parameters relevant to a specific analysis.

5 Prototype of SM decision support system

To illustrate the use of the proposed design and framework, we prototype a decision support system that allows process engineers to (1) compose hierarchically dynamic production processes via a GUI and (2) perform deterministic and stochastic optimization of dynamic production processes. Before discussing the design and implementation of the prototype (the *How*), we first discuss the system functions (the *What*), which are based on users' needs. In this section, we discuss users' needs and roles and the system's functions and semantics. In some sense, functions are associated with the roles and semantics expresses the needs. We also describe a typical case scenario that involves users' needs, roles, system's functions, and semantics.

5.1 High-level system functionality and key user roles

The key system function is to enable all analytical tasks such as what-if analyses and performance optimizations to be executed timely and accurately. The idea is that new analytical tasks will not need to be implemented entirely from scratch every time a user requests one. Rather, new tasks will be implemented using pre-existing, reusable, "component" models of all machines and all processes. These component models are stored in a prebuilt model library and used to construct the composite process models. The same composite process models can be used by different users performing different roles for different analytical tasks.

From the previous discussion, it is clear that functions are executed by users with different roles. There are, as shown in Fig. 10, four roles: analytical view modeler, atomic process performance modeler, composite process performance modelers, and manufacturing end users.

An analytical view modeler designs and describes analytical views in terms of the core analytical functions. Analytical view modelers can develop different descriptive, diagnostic, predictive, and prescriptive analytical views.

An atomic process performance modeler creates physics-based models that describe the operations of machines and atomic process performance models that predict the outputs of the machines. Both model types must be based on either expert knowledge of the domain or the knowledge from other atomic process performance models.

A composite process performance modeler must understand the process plan to (1) determine the required atomic or composite process performance models and (2) the materials and information that flow among those models. Composite performance modelers have access to a pre-built library that contains atomic process performance model components as well as higher-level composite process performance models.

A manufacturing end user must submit a request to the system in the form of declarative analysis queries against previously constructed process models. Of course, manufacturing users can fill some or all these roles. For example, MPEs can play all the roles, whereas manufacturing operators and business managers can only play the role of manufacturing end users. Manufacturing end users can perform various analyses of the manufacturing processes that have been defined by other modelers.

To describe the functionality of the system in more detail, consider GUI screen captures of the system depicted in Figs. 11 and 12. The screen is split into four parts. The left part is a window for a KB of manufacturing models, organized into folders by analytical views, atomic process performance model, and composite process performance model. In the middle is the workspace window used for constructing composite processes and performing analytical tasks. The right part is a window used to input data for the component selected in the workspace or display the results of the component selected. The top part contains buttons to dispatch the manufacturing processes and analytics view to the analysis engine. We now describe each of these parts and the corresponding functionality.

5.2 The KB

The KB/repository (left window in Fig. 11) consists of libraries of analytical views, core analytical functions, models of machines available at a manufacturing facility, and models of composite manufacturing processes. Note that manufacturing process models may be complex and involve an arbitrary hierarchy of subprocesses. At the end of that process will be an atomic process for every machine within the manufacturing facility.

There is a one-to-one correspondence between the library described here and the SM KB described in Sect. 3. There are folders containing analytical views, the analytical core, atomic models (i.e., atomic process performance models), and composite models (i.e., composite process performance models). The analytic view folder contains analytical view functions such as monitors, dashboard, and scheduling. The analytical core folder contains functions such as compute, predict, learn, simulate, and optimize. The atomic model folder contains a classification hierarchy of prebuilt performance models for atomic UMP. Finally, the composite model folder contains performance models of composite processes at different levels of granularity—manufacturing units, cells, lines, factories, and supply chains.

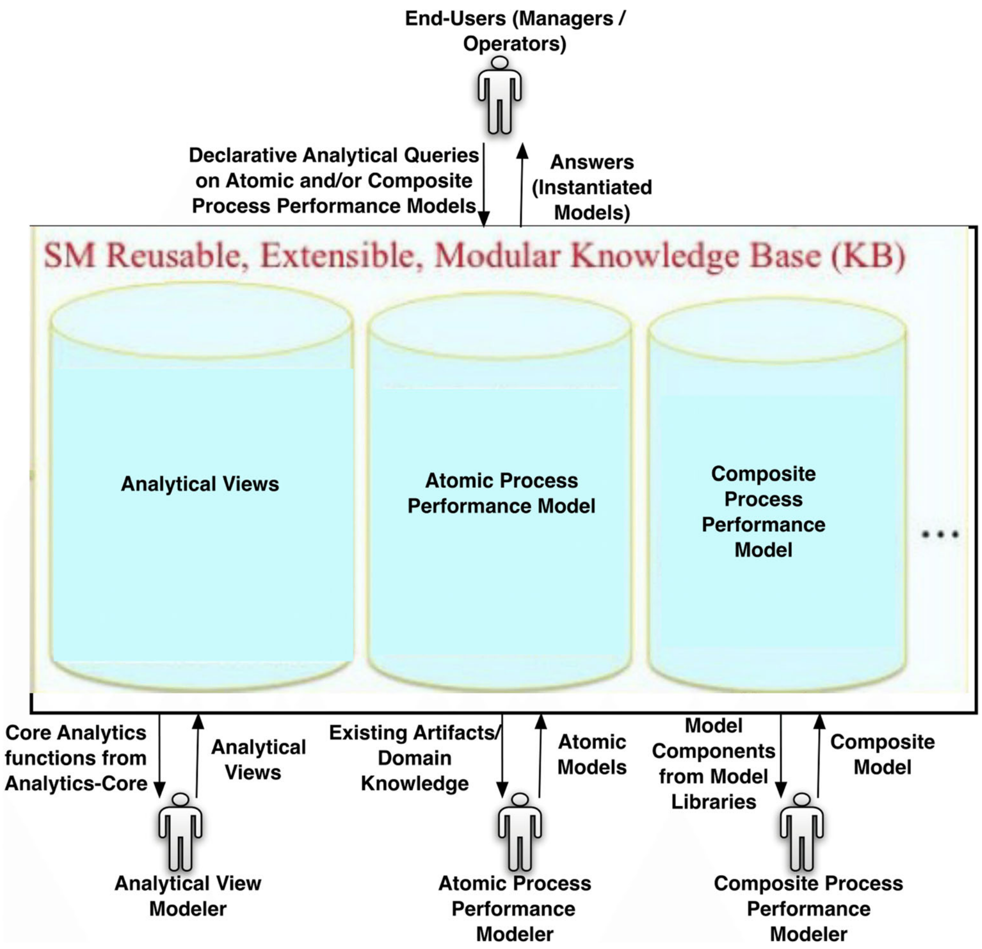

End-Users (Managers / Operators)
Declarative Analytical Queries on Atomic and/or Composite Process Performance Models
Answers (Instantiated Models)
SM Reusable, Extensible, Modular Knowledge Base (KB)
Analytical Views
Atomic Process Performance Model
Composite Process Performance Model
Core Analytics functions from Analytics-Core
Analytical Views
Existing Artifacts/ Domain Knowledge
Atomic Models
Model Components from Model Libraries
Composite Model
Analytical View Modeler
Atomic Process Performance Modeler
Composite Process Performance Modeler

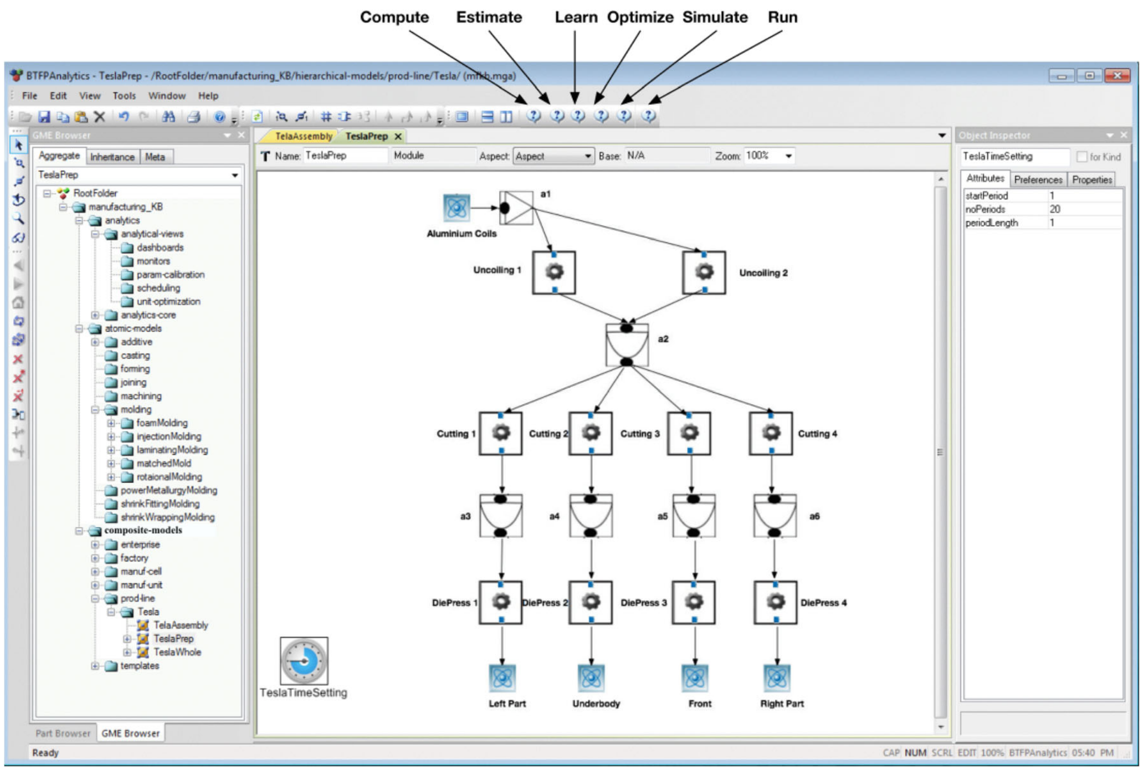
Fig. 10 Potential roles of a domain-specific SM DSS

While implementation of performance models for atomic and composite processes may be complex, their meaning for manufacturing modelers and end users is much simpler. Here is what manufacturing users need to understand.

1 *Correspondence between models and their physical counterparts—machines or processes*—There will be a computational model, either atomic or composite, for every physical machine and process used in a manufacturing facility.
2 *Input and output flows*—Output flows correspond to things produced by machines and processes; input flows correspond to things consumed by the machines and processes. Each machine and process may have multiple input and output flows. These flows act as interfaces and can be either informational or physical.
3 *Process parameters and controls*. Each atomic process performance model of a machine will include a number of parameters. Some parameters describe the machine's behavior, and other parameters describe the machine's direct control capabilities.
4 *Process-dependent variables as a function of parameters and controls*. Each process may have dependent variables including metrics and KPIs. Users need to understand what the metrics mean but are not required to understand the mathematics behind it.
5 *Process feasibility constraints*. Each process may have real feasibility constraints that limit their capabilities and performance. Atomic models capture these constraints, which are only an approximation of their real-world counterparts.

5.3 Process composition scenario

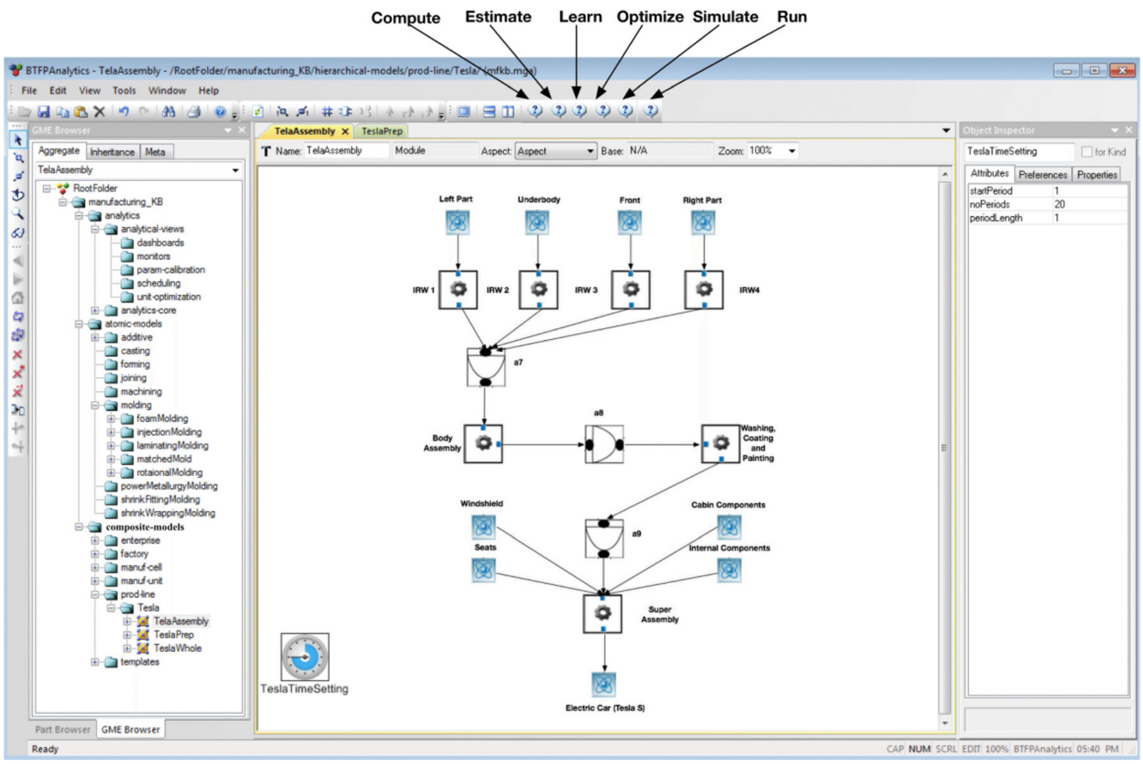
To describe what users need to understand about composite processes performance models, we focus on the Tesla prep composite process and a related performance objective: find the speed of different machines that minimizes cost. The atomic model folders of the model library include a built-in model template for the machines that enable the production of the Tesla car. Using these atomic process performance models and aggregators, a composite process performance modeler can easily be created to minimize the production cost. Below is a typical case scenario to define a new composite process by a composite process performance modeler:



startPeriod	1
noPeriods	20
periodLength	1

Fig. 11 Drag and drop screen capture of the proposed system showing composition of Tesla prep

1 The modeler (1) identifies the required subprocesses according to the process plan, (2) selects their appropriate models from the library, and (3) drags and drops them on the workspace using the GUI. For example, in Fig. 11, there are ten subprocesses: two uncoilings, four cuttings, and four die press. Note that the modeler can use these



startPeriod	1
noPeriods	20
periodLength	1

Fig. 12 Drag and drop screen capture showing composition of Tesla assembly

atomic process performance models to formulate composite process performance models or he or she can use an existing relevant composite process performance model to build a larger model. For example, if the modeler wanted to show the entire Tesla car manufacturing floor in a composite process performance model, then the modeler would use the composite process performance models of Tesla prep (Fig. 11) and Tesla assembly (Fig. 12) as subprocesses.

2 The modeler identifies the input flows to the subprocesses and the output flows from them. In the example, all processes incur a cost and here are the material inputs and outputs for each process:

- Uncoiling process: The input is aluminum coil and the output is an uncoiled aluminum plate
- Cutting process: The input is the uncoiled aluminum plate and the output is a cut aluminum part for the left, right, front, or the underbody.
- Die press process: The input is the aluminum part, and the output is the aluminum part of appropriate shape

3 If the composite process performance model is of BTFP type (as shown in Fig. 11), then the modeler identifies the aggregators (buffers) required between the processes. There are typically two types of physical aggregators: inventory and transportation. For instance, in Fig. 11, a1 is a transportation aggregator distributing the aluminum coils among the uncoiling machines; a2 is inventory aggregator storing coiled and uncoiled aluminum plates.

4 The user connects subprocesses in the workspace using the *connector* components (directed arcs). Each connector signifies the flow of a particular item (material, part, product, energy, etc.). Implicitly, they also signify the balance of the input and output flows. Figure 11 shows the workspace after the completion of this step for the example process.

5 The modeler creates a uniquely named module (with a unique namespace) for the created composite process and stores it in the library under a selected folder for future use.

The panel to the right of Fig. 11 is where the data and the parameters of the existing subprocesses, aggregators, and flows can be instantiated or modified. This can be done by clicking the appropriate icon and instantiating or modifying its associated data structure, displayed in right window panel. Some of these parameters may be decision variables in an optimization problem; they would be marked as "dvars" in the atomic process performance model data structure. This window panel also serves as a display for the results when the analytics task succeeds. To view the results, the end user can click on the subprocess of interest and the associated value will be shown in this window panel.

5.4 Performing analytics tasks

The user can perform different analytics tasks such as compute, predict, learn, simulate, and optimize by selecting the process models of interest and clicking the buttons at the top of Fig. 11. In addition, the user can perform what-if analysis, diagnostics, and optimization using an analytical view created by the analytical view modeler in the KB. To do this, the user drags the appropriate analytics view into the workspace and then presses the "run" button on the top panel in Fig. 11. Here, we discuss two key functionalities that are available to the manufacturing user: compute and optimize.

5.4.1 Compute

The user invokes the "compute" function against a predefined process model. To perform computation, all parameters, control variables, and subprocesses must be instantiated. For the Tesla example, the only control variables are the speeds of uncoiling, cutting, and die press machines. The compute function performs the following actions:

1 Compute each process-dependent variable from the process parameters and controls
2 Evaluate each process feasibility constraint to *true* or *false* from the process parameters and controls. If all feasibility constraints evaluate to true, we say that the process instance is *feasible*.
3 Create a copy of the instantiated process where all dependent variables and feasibility constraints are instantiated to constants computed in 1 and 2.
4 Optionally store the resulting instantiated process model with a new unique name in the manufacturing model library under a folder of users' choice.

5.4.2 Optimize

The user invokes the "optimize" function against a predefined process model. To perform optimization, all model parameters, except control variables, must be instantiated. For each model, its parameters will be displayed at the right-hand side of the window. In addition, control variables and dependent variables are displayed along with their minimum and maximum bounds, which, by default, are negative and positive infinity if the variable is not constrained. The user can choose bounds by adding constraints on any or all variables. The user can also assign a particular variable to be constant by setting both minimum and maximum bounds to be the same value. Then, the system performs the following tasks:

1 If there does *not* exist a feasible instantiation of process control variables, an instantiation that would make all

feasibility constraints evaluate to true, the system reports status *infeasible* to the user. For example, if the throughput demands on the output flow are too high for the limited production capacity of the machines in the composite process performance models, it is infeasible.

2 Otherwise, if a feasible instantiation exists, the system finds the optimal instantiation of all control variables for the given optimization problem. For example, for the process in Fig. 11, a problem is to find the values for all control variables that minimize the cost while meeting all feasibility and demand constraints.
3 The system computes the instantiated process instance with the instantiation of control variables as described in the *compute* function.
4 The user can then store the resulting instantiated (optimal) process instance, under a new unique name, in the manufacturing model library for future use. The instantiated model has the process controls with optimal values.

Next, we describe the overall system architecture, explain all the system components (what is under the hood), and briefly discuss the implementation of the domain-specific SM KB in the next section.

6 Prototype implementation architecture

The implementation architecture for the prototype of the domain-specific SM DSS is shown in Fig. 13. As described in Sect. 3, the proposed architecture allows the manufacturing end users to pose and solve manufacturing problems using a GUI that provides a standardized view of the SM KB. The GUI is based on the generic modeling environment (GME) and contains the SM KB. End users will use the GUI to build models and perform analytics on the model using an analytical view. GME contains the meta-model that defines the rules and constraints for building the model. Additionally, GME will also contain existing atomic and composite process performance models that the user can use to build new models. A number of analytical views such as deterministic optimization, stochastic optimization, estimation, learning, and simulation in the SM KB are available for users to use.

The left half of the architecture in Fig. 13 describes dispatching of the job to the underlying tools, and the right half shows the results returned from the tools to GME. The GME components of the model built using the GUI and the analytical view are translated into a standard data format such as JSON. The models are then dispatched via the web services to the backend JSONiq engine. The JSONiq engine determines the type of analytics to be performed and converts the JSON model and analytics parameters into the input required by the analytical tools. After the analytical tools derive a solution to the problem, this solution is translated back into

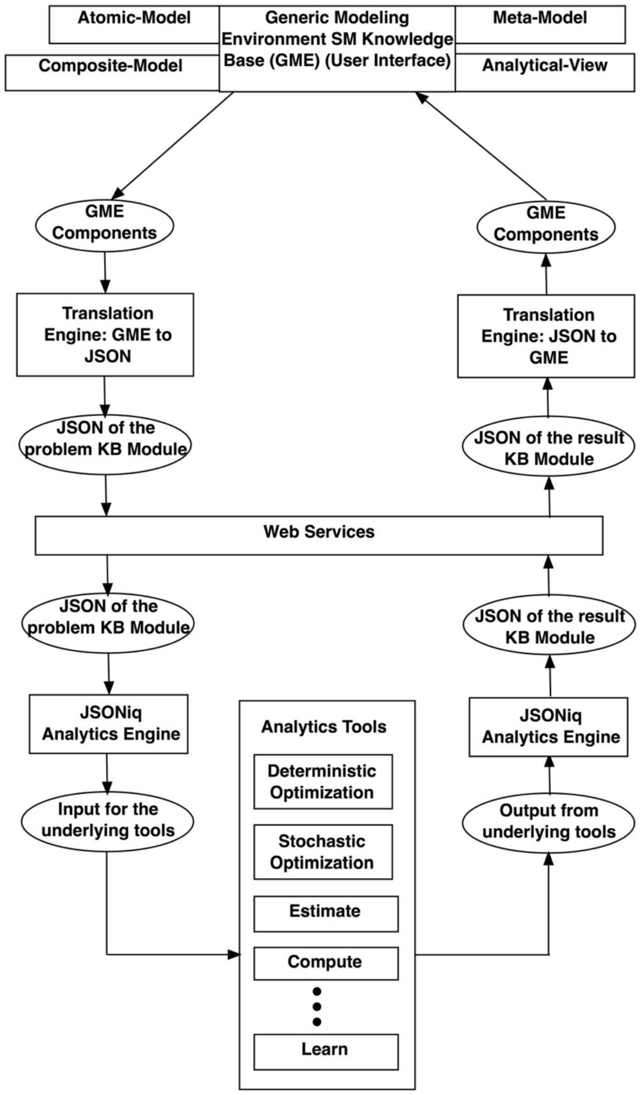

Atomic-Model
Generic Modeling Environment SM Knowledge Base (GME) (User Interface)
Meta-Model
Composite-Model
Analytical-View
GME Components
GME Components
Translation Engine: GME to JSON
Translation Engine: JSON to GME
JSON of the problem KB Module
JSON of the result KB Module
Web Services
JSON of the problem KB Module
JSON of the result KB Module
JSONiq Analytics Engine
Analytics Tools
JSONiq Analytics Engine
Deterministic Optimization
Input for the underlying tools
Stochastic Optimization
Output from underlying tools
Estimate
Compute
Learn

Fig. 13 Overall implementation architecture

JSON via the JSONiq analytics engine. The JSON is then converted into GME components for display at the GME frontend. We now describe the system components in greater detail.

6.1 Domain-specific modeling environment (user interface)

A “domain-specific modeling environment” (DSME) is a visual interface that a domain expert can use to build computable representations of systems in their domain. The DSME provides (1) visually intuitive icons pointing to models that represent elements from the domain, (2) the ability to graphically arrange and connect these models when needed, and (3) a library of models that can be used to represent a wide range of systems within the domain. The DSME is designed to be intuitive for domain experts who only understand how systems in their domain are constructed. This means that users do not need a deep understanding of the analytics performed on these systems.

The DSME for BTFP was designed using GME, a tool for creating custom DSMEs. A DSME is specified in GME by constructing a unique meta-model, which describes the various objects, properties, and relationships in the domain. The meta-model for BTFP defines objects such as *processes*, *aggregators*, and *flowports*, which can be used to build models of manufacturing systems. Figure 14 shows a portion of the BTFP meta-model. This portion of the meta-model specifies that a *process* can be of two types, atomic and composite. Composite processes can contain atomic or other composite processes. The figure also shows that processes can contain flowports, which can be of type input or output. *Flowports* can be connected to aggregators to form connections that model the flow of material or parts between processes and aggregators. Other details of the meta-model are omitted for brevity.

Once the meta-model has been built, GME can be used to build the actual models of real-world objects in the domain. This is possible because GME can be reconfigured to function as a DSME for the BTFP domain. Manufacturers can build representations of their systems by constructing an *instance model* in the BTFP domain within GME. GME provides a palette of the objects that are defined in the meta-model. The user can use these objects to construct the instance models. Figure 15 shows an instance model constructed in the BTFP domain in GME. This model shows an input port called "plywood_in" at the top. The material flows through this port to the input aggregator "a1." The aggregator allocates the material to the two processes "sand1" and "cut1." The visual model makes it easy for users to understand how the overall system looks and operates. After constructing the instance model, the user may invoke a variety of analytical tools and perform analytical tasks to derive further insights about the system.

6.2 Model translation

The instance model in GME provides a generic graphical representation of the system. Before this representation can be used as the basis for executing analytical algorithms, it is translated into a format that will be understood by the analytical software applications. The BTFP instance models in GME are translated into a JSON representation, which can then be input to the analytics algorithms on the back-end server. This translation is executed at the click of a button on the GME interface. In addition to specifying the instance model, the domain-specific environment for BTFP in GME also allows users to specify their analytics objectives and the type of analytics that they would like to perform.

The JAVA program that handles the model translation (1) packages all the information into a single JSON file and (2) sends it to the analytics engine. The response from the analytics engine is also in the form of a JSON file. This is translated back into a GME model and shown visually to the end user.

6.3 JSONiq analytics engine

The JSONiq analytics engine takes the JSON model as input, interprets it into the required tasks, and transforms each task to the appropriate input for the underlying tool. The JSONiq

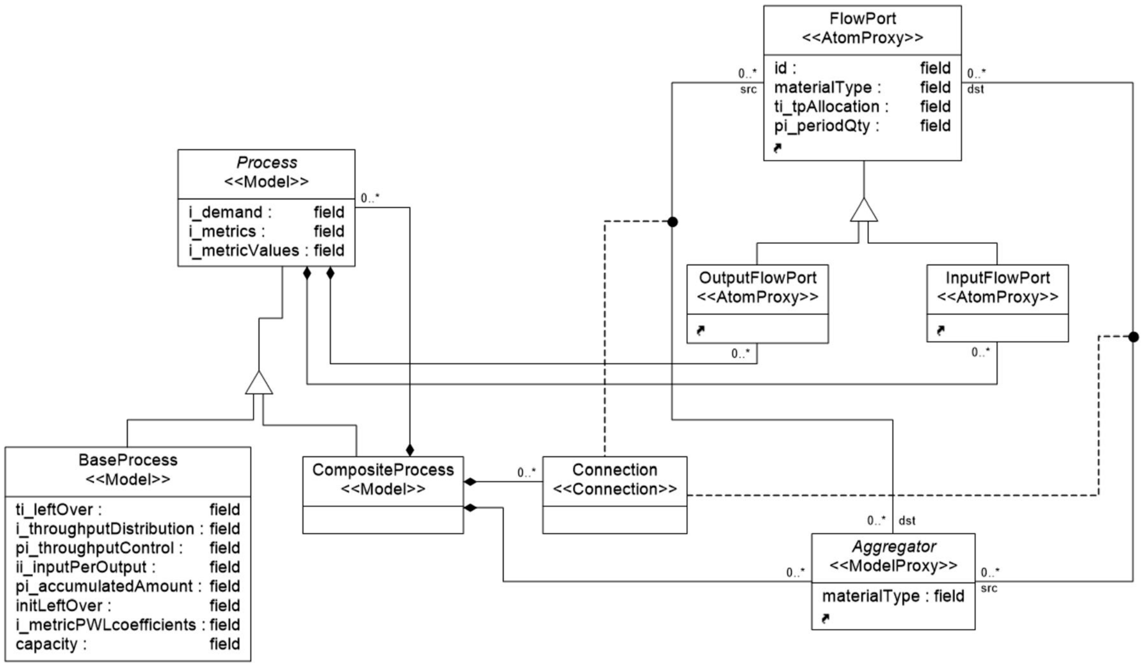


Fig. 14 A portion of the BTFP meta-model

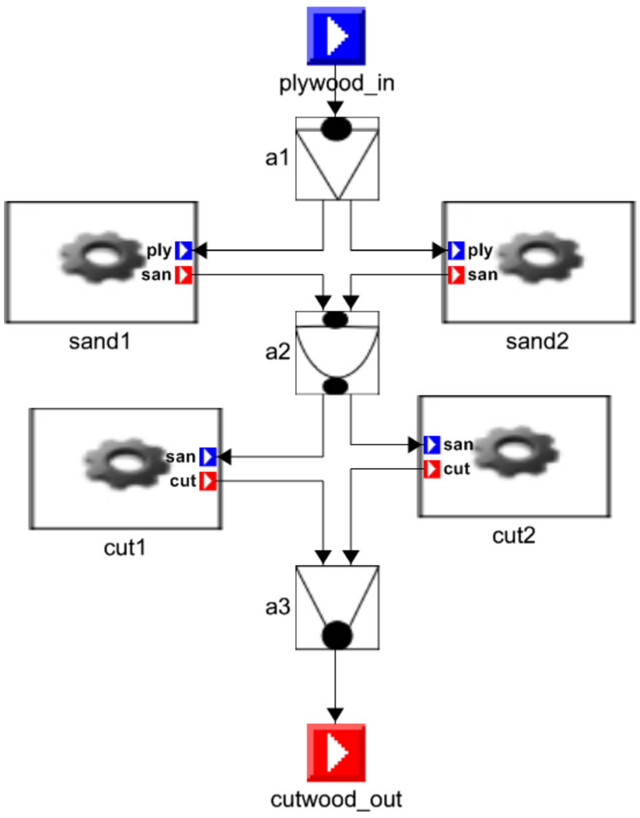


Fig. 15 An instance model constructed in the BTFP domain in GME

interpreter uses the configuration object provided in the input of JSON analytics task to decide the kind of composition to use for the composite process performance model. The composition takes all the parameters, variables, metrics, and constraints and embeds them into a standardized JSONiq data structure. The JSONiq engine will then use the configuration object to decide the type of analytics tool to use and convert this data structure and analytics parameters as inputs for that tool. For instance, if IBM CPLEX is used for MILP deterministic optimization, the JSONiq engine will convert the data structure into the OPL model and data files. After the optimization result is derived, the JSONiq engine will do the opposite transformation, i.e., from the tool output to the result JSONiq data structure. Finally, the engine will convert the resulting data structure into the output JSON file and send it back to the user via the web service interface.

6.4 Analytical tools

As described in Sect. 2, there are a number of tools available to solve analytical problems and we describe two such tools as follows.

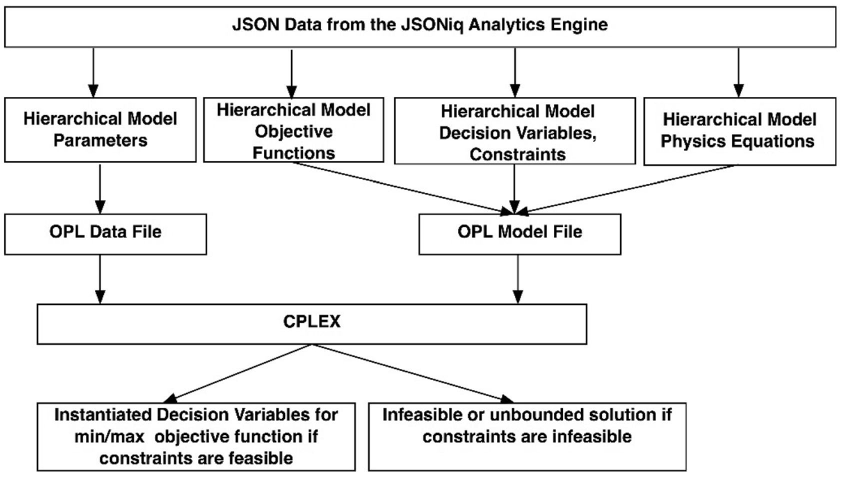
6.4.1 Tool 1: deterministic optimization using IBM CPLEX MILP solver

The first tool is IBM CPLEX (OPL studio) for deterministic MILP optimization. The BTFP processes' composition is used. Figure 16 shows the transformation of the incoming analytical task to the input generation for IBM CPLEX. A number of different optimization problems can be formulated and solved as a MILP. The model parameters are translated into an OPL data file. The decision variables, constraints, physics equations, and objective function are translated into an OPL model file. There are two possible results of the optimization: no feasible solutions or the optimal solution.

6.4.2 Tool 2: stochastic optimization using iterative heuristic optimization simulation algorithm

The second tool is the iterative heuristic optimization simulation (IHOS) algorithm for stochastic optimization discussed in [11]. An overview of this algorithm is given in Fig. 17. To model and query a manufacturing process in the real world, it is critical to consider the stochastic nature of the variables of

Fig. 16 Deterministic MILP optimization using CPLEX (OPL Studio)


JSON Data from the JSONiq Analytics Engine
Hierarchical Model Parameters
Hierarchical Model Objective Functions
Hierarchical Model Decision Variables, Constraints
Hierarchical Model Physics Equations
OPL Data File
OPL Model File
CPLEX
Instantiated Decision Variables for min/max objective function if constraints are feasible
Infeasible or unbounded solution if constraints are infeasible

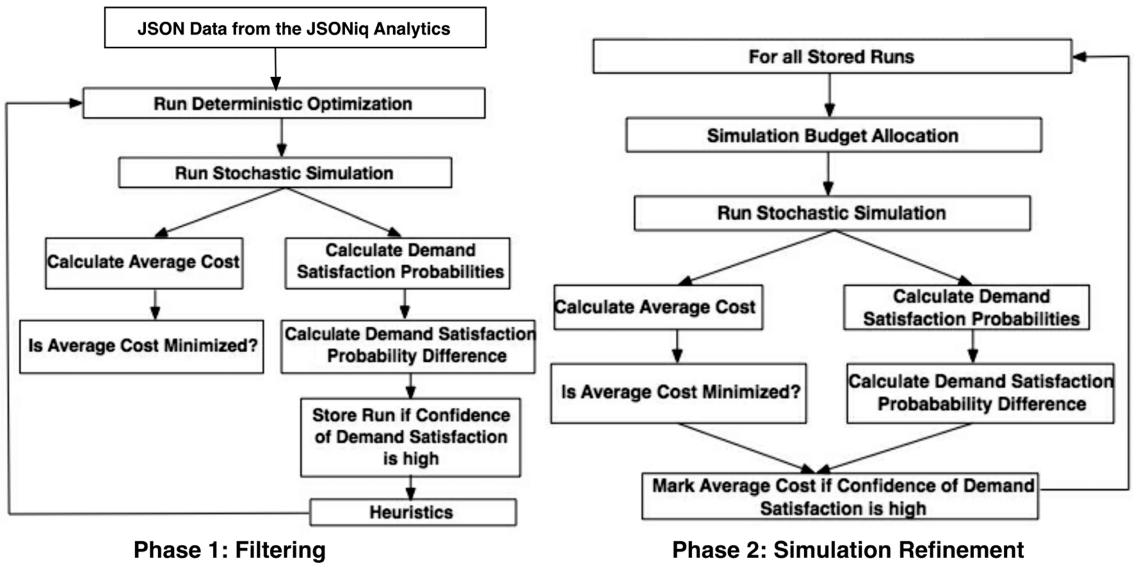

JSON Data from the JSONiq Analytics
Run Deterministic Optimization
Run Stochastic Simulation
Calculate Average Cost
Calculate Demand Satisfaction Probabilities
Is Average Cost Minimized?
Calculate Demand Satisfaction Probability Difference
Store Run if Confidence of Demand Satisfaction is high
Heuristics
Phase 1: Filtering
For all Stored Runs
Simulation Budget Allocation
Run Stochastic Simulation
Calculate Average Cost
Calculate Demand Satisfaction Probabilities
Is Average Cost Minimized?
Calculate Demand Satisfaction Probabability Difference
Mark Average Cost if Confidence of Demand Satisfaction is high
Phase 2: Simulation Refinement

Fig. 17 Stochastic optimization using iterative heuristic optimization simulation algorithm

the process model. In this algorithm, the JSON data from the JSONiq engine is fed to a two-phase IHOS algorithm. Using this data, the algorithm approximates the total cost of running the entire hierarchical process when the actual throughput of the machines is stochastic. The main idea behind these algorithms is multiple iterations of optimization through a series of stochastic simulations.

For the example, the optimization solver finds the expected throughputs of the machines such that the demand is satisfied at each period while minimizing the cost. The stochastic simulation adds noise to the mean throughputs and checks whether the probability of satisfying the demand at each period lies within a predetermined confidence interval. A heuristic is used to vary the demands such that the optimizer can give more realistic mean throughputs in the following iteration. A number of candidate mean throughputs are simulated to ensure (1) the probability of satisfying the demand remains within the desired confidence interval and (2) the cost is approximately minimal. In this way, the algorithm uses the model knowledge in both optimization and stochastic simulation to provide an optimal setting for the throughputs of the machines. The process operator can then apply the optimal setting in production.

The authors conducted an initial experimental study to compare the proposed algorithm with four simulation-based optimization algorithms: Nondominated Sorting Genetic Algorithm 2 (NGSA2) [20], indicator-based evolutionary algorithm (IBEA) [21], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [21], and Fast Pareto Genetic algorithm (FastPGA) [22]. The study shows that IHOS significantly outperforms the other algorithms in terms of optimality of results and computation time; in particular, in 64 s, the cost achieved by IHOS is 5 % of the cost achieved by competing algorithms. For the total run time, in 1500 s, the cost achieved by IHOS is 80 % of the cost realized by the competing algorithms. More details about the results can be found in [12].

7 More related work and its limitation

In this section, we further discuss related work mentioned in the Sect. 1.

The implementation of the analysis and optimization capabilities typically uses a variety of computational tools, which are shown at the bottom layer in Fig. 2. They include

- Domain-specific end-user-oriented tools, e.g., strategic sourcing optimization modules within procurement applications [23]
- Data manipulation languages, such as Structured Query Language (SQL) [24, 25], XQuery [26, 27], and JSONiq [28]
- Simulation tools and languages including discrete event simulation, system dynamics simulation, such as JModelica (based on Modelica), AnyLogic, and Simulink [29, 30]
- Optimization modeling languages, such as A Modeling Language for Mathematical Programming (AMPL) [31], The *General Algebraic Modeling System* (GAMS) [32], and OPL [33] for MP and CP
- Statistical learning languages and interfaces, such as Predictive Model Markup Language (PMML) [34–36] and the Portable Format for Analytics (PFA) [37]
- Modeling languages for complex physical systems, such as Modelica [29].

Discussion on the strengths and weaknesses of these categories of tools as compared to our proposed architecture is as follows:

Domain-specific tools are designed for, and usually do a good job in executing, a particular well-defined task in a particular industry sector. For example, tools that support manufacturing scheduling would not be used to schedule visits to a doctor's office. Nor do these domain-specific tools support compositionality, which is defined to be the ability to make optimal *system-wide* performance predictions from optimal predictions of the *systems components*, whereas the architecture proposed in this paper is designed for composability.

Simulation tools, on the other hand, are usually applicable to many different tasks in many different industry sectors.[1] They have this advantage because of their modeling expressivity, flexibility, and object-oriented (OO) modularity. While simulation models and tools cannot solve optimization problems by themselves, they can be used in a heuristically guided, trial-and-error optimization technique. However, for problems expressed in closed analytical forms, such simulation-based optimization techniques are inferior to optimization techniques based on MP or CP, e.g., mixed-integer linear programming (MILP) [38]. Furthermore, simulation languages were not designed for easy data manipulation the way that data manipulation languages such as SQL, XQuery, and JSONiq [28] are, whereas the architecture proposed in this paper allows the use of the best available tools and algorithms including MP.

MP and CP optimization models are built using modeling languages such as AMPL, GAMS, or OPL [31–33]. These languages and techniques use a range of sophisticated algorithms that leverage the mathematical structure of optimization problems. As a result, they significantly outperform simulation-based optimization, in terms of optimality and execution time. However, MP and CP optimization models are not modular, extensible, and reusable and do not support compositionality. They also do not support low-level granularity of simulation models. Furthermore, some MP- or CP-based optimization algorithms have high worst-case computational complexity. Therefore, these algorithms may not scale up for large-size optimization problems, whereas the architecture proposed in this paper does allow modularity, extensibility, reusability, and composability.

The Sustainable Process Analytics Formalism (SPAF) [10] was proposed to make optimization modeling more modular and extensible, akin an OO simulation model. However, SPAF was not designed to be easily integrated with tools that perform data manipulation, statistical learning, or predictive analytics. We did address these limits in the proposed architecture.

[1] Individual simulation models, however, do not possess this capability.

Similarly, the statistical learning languages and tools were not designed for easy data manipulation; others such as SQL, XQuery, and JSONiq are significantly better. A recent standardization effort by the Data Mining Group (DMG) to address this deficiency is their development of PMML [39]. PMML is an XML-based standard language used to represent predictive and descriptive models, as well as pre- and post-processed data. PMML allows for the interchange of data models among different tools and environments, mostly by avoiding proprietary issues and incompatibilities. Besides neural networks and decision trees, PMML allows for the representation of many other data-mining models.

PFA [40], similar to PMML, is a JSON-based specification for statistical models, but whereas PMML's focus is on statistical models in the abstract, PFA's focus is on the scoring procedure itself. PMML can only express a fixed set of predefined model types whereas PFA represents models and analytic procedures more generally by providing generic programming constructs.

Modeling languages for complex physical systems are designed to reuse knowledge. Modelica, for example, allows a detailed level of abstraction, including OO code and differential equations [29]. Modelica by itself is not a language for performing optimization, learning, or prediction. But, there are tools such as JModelica for simulation and Optimica for simulation-based optimization [30]. However, because of the low level of abstraction allowed in Modelica, general Modelica models cannot be reduced automatically to MP or CP models that can be solved by MP or CP solvers.

As a result of the discussed limitations such as each analytics task is implemented from scratch, as a one-off effort; is not modular or reusable; requires mathematical, operations research (OR), and domain expertise that are not within the realm of manufacturing users; is high cost; requires a long development cycle; is difficult to modify or extend, the development of analysis and optimization solutions today is important.

8 Conclusion

In this paper, we proposed an architectural design and framework for fast development of software solutions for descriptive, predictive, diagnostic, and prescriptive analytics of dynamic production processes. We also proposed an organization of, and key structure of, a reusable KB, which consists of three libraries: atomic process performance models, composite performance models, and analytical views and dashboards. Finally, we also showed a prototype of a decision support system to demonstrate the principles of the proposed architectural framework.

The proposed architectural framework and the analytics engine follow the ideas from the Decision Guidance

Analytics Language (DGAL) and framework proposed in [41] which, in turn, build on prior work on decision guidance and optimization languages. In particular, the unification of computation and equation syntax comes from CoJava [42], SC-CoJava [43], and DGQL [44], CoReJava [39, 45], and DGAL and DG-Query [40]. These languages are designed to add deterministic optimization and machine learning to Java, SQL, and XQuery code, respectively. Additions are implemented via automatic reduction to MP, CP, or specialized algorithms. In addition, DGAL fits into the framework of, but is more general than, Decision Guidance Management Systems proposed in [46]. Finally, the concept of centralized analytical KB (AKB) is borrowed from our previous work on SPAF [10], which was limited to MP or CP optimization only.

The results reported in this paper are only a first step toward reusability and modularity in SM analysis and optimization. We plan to work on extending the analytics engine with reduction algorithms, stochastic simulation, statistical learning, and uncertainty quantification based on the recent advances in these areas. We also plan to extend optimization algorithms for dynamic production processes with more refined unit process performance models. Furthermore, we plan to prototype an AKB on an industry case study for process performance models and systematic guidelines for its creation, extension, and reusability for the diverse analytics tasks. Finally, we plan to work on case studies with industrial partners to demonstrate the productivity gains of the proposed architectural design and framework.

Acknowledgments This effort has been sponsored in part under the Cooperative Agreement No.70NANB12H277 between NIST and George Mason University and Cooperative Agreement No. 70NANB14H250 between NIST and University of Maryland, College Park. The work described was funded by the US Government and is not subject to copyright.

Compliance with ethical standards

Disclaimer No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial software systems are identified in this paper to facilitate understanding. Such identification does not imply that these software systems are necessarily the best available for the purpose.

References

1. SMLC, "Implementing 21 Century Smart Manufacturing, workshop summary report," 2011. [Online]. Available: https://smartmanufacturingcoalition.org/sites/default/files/implementing_21st_century_smart_manufacturing_report_2011_0.pdf. Accessed: June 2015.
2. Salvendy, G. (2001) Handbook of industrial engineering: technology and operations management, Third Edition. Wiley, Inc. ISBN: 9780471330578.
3. J. Richardson., "Gartner BI: analytics moves to the core," 2013. [Online]. Available: http://timoelliott.com/blog/2013/02/gartnerbi-emea-2013-part-1-analytics-moves-to-the-core.html. Accessed: Sep. 2015.
4. Undey C, Ertun S, Mistretta T, Looze B (2010) Applied advanced process analytics in biopharmaceutical manufacturing: challenges and prospects in real-time monitoring and control. J Process Control 20(9):1009–1018. doi:10.1016/j.jprocont.2010.05.008
5. Gilvan GC (2014) Supply chain analytics. Bus Horiz 57(5):595–605. doi:10.1016/j.bushor.2014.06.004
6. Shao G, Shin S, Jain S (2014) "Data analytics using simulation for smart manufacturing," Proc. 2014 Winter Simulation Conference, ser. WSC '14. IEEE Press, Piscataway, pp 2192–2203. doi:10.1109/WSC.2014.7020063
7. D. Lechevalier, A. Narayanan, and S. Rachuri, "Towards a domain-specific framework for predictive analytics in manufacturing," Proc. IEEE International Conference on Big Data,, Oct 2014, pp. 987–995, doi: 10.1109/BigData.2014.7004332.
8. Shin S, Woo J, Rachuri S (2014) Predictive analytics model for power consumption in manufacturing. Proc 21st CIRP Conference Life Cycle Eng 15:153–158. doi:10.1016/j.procir.2014.06.036
9. Lee J, Lapira E, Bagheri B, Kao H (2013) Recent advances and trends in predictive manufacturing systems in big data environment. Manuf Lett 1(1):38–41. doi:10.1016/j.mfglet.2013.09.005
10. A. Brodsky, G. Shao, and F. Riddick, (2014) "Process analytics formalism for decision guidance in sustainable manufacturing," J Intell Manuf, pp. 1–20, doi:10.1007/s10845-014-0892-9.
11. C. Groger, F. Niedermann, H. Schwarz, and B. Mitschang, (2012) "Supporting manufacturing design by analytics, continuous collaborative process improvement enabled by the advanced manufacturing analytics platform," Proc. IEEE 16th International Conference on Computer Supported Cooperative Work in Design, pp. 793–799, doi:10.1109/CSCWD.2012.6221911.
12. M. Krishnamoorthy, A. Brodsky, and D. Menasce, (2014) "Temporal manufacturing query language (tMQL) for domain specific composition, what-if analysis, and optimization of manufacturing processes with inventories," Department of Computer Science, George Mason University, Fairfax, VA, 22030, Tech. Rep. GMU-CS-TR-2014-3. [Online]. Available: http://cs.gmu.edu/tr-admin/papers/GMU-CSTR-2014-3.pdf
13. M. Krishnamoorthy, A. Brodsky, and D. Menasce, (2015) "Optimizing stochastic temporal manufacturing processes with inventories: an efficient heuristic algorithm based on deterministic approximations," in Proceedings of the 14th INFORMS Computing Society Conference, pp. 30–46.
14. A. Ledeczi, M. Maroti, A. Bakay, G. Gabor Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, (2001) "The generic modeling environment," in Workshop on Intelligent Signal Processing, IEEE 2nd International Conference on, May 2001.
15. Fourny, M. Brantner, and F. Cavalieri, "Jsoniq the SQL of NoSQL," CreateSpace Independent Publishing Platform, 2013.
16. J. Robie, G. Fourny, M. Brantner, D. Florescu, T. Westmann, and M. Zaharioudakis, "Jsoniq the complete reference," 2015. [Online]. Available: http://www.jsoniq.org/docs/JSONiq/html-single/. Accessed: June 2015.
17. Madan J, Mani M, Lee JH, Lyons KW (2014) Energy performance evaluation and improvement of unit-manufacturing processes: injection molding case study. J Clean Prod 1(1):1–13
18. J. Madan, M. Mani, and K. W. Lyons, "Characterizing energy consumption of the injection molding process," in ASME 2013 International Manufacturing Science and Engineering Conference, vol. 2, no. 1. Manufacturing Engineering Division, 2013, pp. 1–13.
19. S. Nannapaneni and S. Mahadevan, (2014) "Uncertainty quantification in performance evaluation of manufacturing processes," in

Big Data (Big Data), 2014 I.E. International Conference on, pp. 996– 1005.
20. E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: (2001) improving the strength pareto evolutionary algorithm," Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, Tech. Rep. 103.
21. E. Zitzler and S. Kunzli, (2004) "Indicator-based selection in multiobjective search," in Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, 2004. Springer, pp. 832–842.
22. H. Eskandari, C. D. Geiger, and G. B. Lamont, (2007) "FastPGA: a dynamic population sizing approach for solving expensive multiobjective optimization problems," in 4th International Conference on Evolutionary Multi-Criterion Optimization, 2007, ser. Lecture Notes in Computer Science, vol. 4403. Springer, pp. 141–155.
23. Katz, Y. Labrou, M. Kanthanathan, and K. Rudin, (2002) "Method for managing a workflow process that assists users in procurement, sourcing, and decision-support for strategic sourcing," US Patent number US20020174000 A1.
24. S. Harkins and M. P. Reid, (2002) "Structured query language," in SQL: Access to SQL Server, pp. 1–5.S.
25. S. Harkins and M. P. Reid, (2002) "Structured query language," SQL: Access to SQL Server, pp. 1–5, doi:10.1007/978-1-4302-1573-8_1.
26. D. D. Chamberlin and R. F. Boyce, (1974) "Sequel: a structured English query language," in Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control, ser. SIGFIDET '74, New York, NY, USA, pp. 249–264.
27. M. Rys, D. Chamberlin, and D. Florescu, (2005) "XML and relational database management systems: the inside story," Proc. International Conference on Management of Data, ser. ACM SIGMOD, pp. 945–947, doi:10.1145/1066157.1066298.
28. Florescu D, Fourny G (2013) JSONiq: the history of a query language. IEEE Internet Comput 17(5):86–90. doi:10.1109/MIC.2013.97
29. P. Fritzson and V. Engelson, (1998) "Modelica—a unified object-oriented language for system modeling and simulation," Proc. European Conference on Object-Oriented Programming, pp. 67–90, doi:10.1007/BFb0054087.
30. Akesson J, Arzen KE, Gafvert M, Bergdahl T, Tummescheit H (2010) Modeling and optimization with Optimica and JModelica.org languages and tools for solving large-scale dynamic optimization problems. Comput Chem Eng 34(11):1737–1749
31. R. Fourer, D. M. Gay, and B. W. Kernighan, (1987) "AMPL: a mathematical programming language," AT&T Bell Laboratories, Murray Hill, NJ 07974, Tech. Rep.
32. Brook A, Kendrick D, Meeraus A (1988) GAMS, a user's guide. SIGNUM Newsl 23(3–4):10–11
33. Van Hentenryck P, Michel L, Perron L, Rgin J-C (1999) Constraint programming in OPL. In: Nadathur G (ed) Principles and practice of declarative programming, vol 1702, Lecture notes in computer science. Springer, Berlin Heidelberg, pp 98–116
34. Guazzelli A, Zeller M, Lin WC, Williams G (2009) PMML: an open standard for sharing models. R J 1(1):60–65
35. The Data Mining Group (DMG), "The predictive model markup language (PMML) 4.2," [Online]. Available: http://www.dmg.org/, 2014, Accessed: June 2015.
36. Guazzelli A (2012) W.-C. L., and T. J., PMML in action: unleashing the power of open standards for data mining and predictive analytics. CreateSpace, North Charleston, South Carolina
37. J. Pivarski, "PFA: Portable Format for Analytics (version 0.6)," [Online]. Available: http://scoringengine.org/, 2015. Accessed: June 2015.
38. Jain V, Grossmann IE (2001) Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS J Comput 13(4):258–276
39. Brodsky, J. Luo, and H. Nash, (2008) "CoReJava: learning functions expressed as object-oriented programs," in Machine learning and applications, 2008. ICMLA '08. Seventh International Conference on, pp. 368–375.
40. A. Brodsky, S. G. Halder, and J. Luo, (2014) "DG-Query, XQuery, mathematical programming," in 16th International Conference on Enterprise Information Systems (ICEIS 2014).
41. A. Brodsky and J. Luo, (2015) "Decision guidance analytics language (DGAL)—toward reusable knowledge base centric modeling," in Proceedings of the 17th International Conference on Enterprise Information Systems, 2015.
42. A. Brodsky and H. Nash, (2006) "CoJava: optimization modeling by nondeterministic simulation," in Principles and practice of constraint programming-CP 2006. Springer pp. 91–106.
43. A. Brodsky, M. Al-Nory, and H. Nash, (2008) "Service composition language to unify simulation and optimization of supply chains," in Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, pp. 74–74.
44. A. Brodsky, S. Mana, M. Awad, and N. Egge, (2011) "A decisionguided advisor to maximize ROI in local generation amp; utility contracts," in Innovative Smart Grid Technologies (ISGT), 2011 I.E. PES, pp. 1–7.
45. Luo J, Brodsky A (2011) Piecewise regression learning in CoReJava framework. Int J Mach Learn Comput 1(2):163–169
46. A. Brodsky and X. Wang, (2008) "Decision-guidance management systems (DGMS): seamless integration of data acquisition, learning, prediction and optimization," in Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, pp. 71–71.