Analysis and Optimization in Smart Manufacturing based on a Reusable Knowledge Base for Process Performance Models

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

[1]Computer Science Department,
George Mason University
Fairfax, VA 22030, U.S.A.

[2]System Integration Division Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899, U.S.A.

[3]Department of Mechanical Engineering
University of Maryland
College Park, MD 20742, U.S.A.

brodsky@gmu.edu, guodong.shao@nist.gov, mkrishn4@gmu.edu, anantha@umd.edu, menasce@gmu.edu, ronay.ak@nist.gov

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 the development of automatic 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 performing diagnostic tasks on a composite performance model.**

Keywords- smart manufacturing; data analytics; optimization; reusable knowledge base; process performance models

I. 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 to support developing analysis capabilities and optimization solutions. These capabilities and solutions 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, diagnostic, predictive, and prescriptive analytics [3]. Descriptive analytics typically involves manipulation of streams of data at different aggregation levels, continuously over time. Some examples of descriptive analytics and its associated data can be found in [4, 5]. Diagnostic analytics includes such tasks as continuous testing for a significant statistical difference between the estimated vs. observed values of metrics, and direct application of fault classifiers to detect failures on the manufacturing floor. Research in diagnostic analytics in manufacturing can be found in [6, 7]. Predictive analytics include techniques of stochastic simulation and statistical learning for regression, classification, and what-if estimation. Examples of predictive analytics for manufacturing can be found in [8, 9]. Prescriptive analytics typically involves decision optimization techniques, such as mathematical and constraint programming. Examples of research in prescriptive analytics in manufacturing can be found in [10, 11].

The current manufacturing analytics practice is that every analytical task is often 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 each computational tool 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 uniformly only once. Furthermore, the modeling expertise required for the low-level abstractions and languages is typically not within the realm of knowledge of manufacturing users, e.g., 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.

978-1-4799-9926-2/15/$31.00 ©2015 IEEE

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) a pre-built library of atomic performance models of a variety of unit manufacturing processes, (2) a library of composite performance models, which can be constructed from the atomic models using a GUI, and (3) a library of analytical views and dashboards designed for specific types of analysis for domain-specific users. To illustrate the use of the proposed design and framework, we show how a process engineer could perform diagnosis over a pre-built composite process.

The paper is organized as follows. Section II discusses the needs and challenges encountered in implementing analytics and optimization solutions. Section III provides the design of the architecture and software framework that is based on reusable KB of process performance models. Section IV extends the previous section by exemplifying the reusable KB. Finally, Section V concludes and discusses the future work.

II. Need and Challenges in Implementing Analytics and Optimization Solutions

To discuss the required analysis and optimization capabilities in more detail, we use the diagram of a 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.

A. Descriptive, Diagnostic, Predictive, and Prescriptive Analytics

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

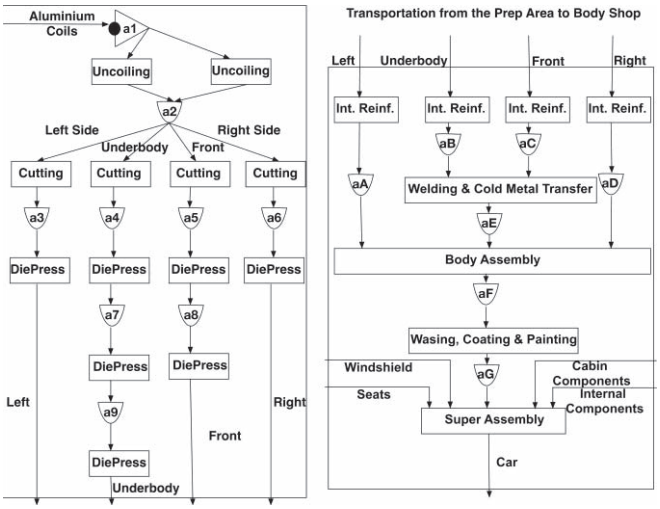

Figure 1. Component diagram for a car manufacturing process.

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 space. 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. 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 that get the energy consumption within the stated bounds and (2) rebalance the workload distribution to meet the production schedule.

As depicted in Fig. 2, a typical software architecture to implement the analysis and optimzaiton capabilities includes (1) a top layer of a domain-specific GUI for manufacturing users, (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). In the next subsection, we describe a variety of tools and then discuss the core implementation challenge in more detail.

B. *Computational Tools and Languages That Support Analytics*

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 [12]
- Data manipulation languages, such as Structured Query Language (SQL) [13], XQuery [14], and JSONiq [15]
- Simulation tools including discrete event simulation, system dynamics simulation, such as Jmodelica (based on Modelica), AnyLogic, and Simulink [16, 17]
- Optimization modelling languages, such as A Modeling Language for Mathematical Programming (AMPL) [18], The *General Algebraic Modeling System* (GAMS) [19], and OPL [20] for MP and CP
- Statistical learning languages and interfaces, such as Predictive Model Markup Language (PMML) [21, 22] and the Portable Format for Analytics (PFA) [23]
- Modeling languages for complex physical systems, such as Modelica [15]

The strengths and weaknesses of these categories of tools are discussed 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*.

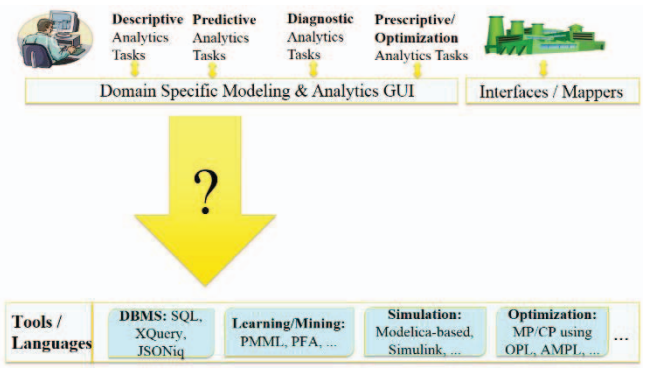

Figure 2. Challenges in implementing analytics functionality using tools.

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) [27]. Furthermore, simulation languages were not designed to be easily integrated with data manipulation languages such as SQL, XQuery, and JSONiq [14].

MP and CP optimization models are built using modeling languages such as AMPL, GAMS or OPL [17-19]. 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, 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.

The Sustainable Process Analytics Formalism (SPAF) [8, 25] 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.

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 effort to address this difficiency is the development of PMML by the Data Mining Group (DMG) [21]. PMML is

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

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 [23], 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 pre-defined 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 [15]. 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 [16]. 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.

C. *Challenge in Developing SM Analysis and Optimization Solutions*

Using the aforementioned tools and languages can improve diverse analysis and optimization tasks. Achieving these improvements, however, will be difficult because each analytics task

- is implemented from scratch, as a one-off effort
- is not modular or reusable
- requires mathematical, operations research, 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

We believe there are two key factors causing these deficiencies. The first is due to the fact that today, in manufacturing analytics practice, analytical tasks are typically implemented following a waterfall methodology of gathering requirements, identifying data sources, developing a model or an algorithm using a range of modeling languages and tools to perform analysis (see Fig. 3). Using this linear task-centric methodology, models and algorithms are difficult to develop, modify, and extend because they are not designed for reusability.

The second, and perhaps the more important factor is the diversity of computational tools that are required to meet our needs. Because of such diversity, developing models using any one of these computational tools typically requires both the knowledge of, and the use of, the specialized, low-level, vendor-specific, mathematical abstractions and languages. As a result, the same manufacturing knowledge is often modeled multiple times using several different specialized abstractions, one for each tool that either generates or consumes that knowledge. Furthermore, the mathematical and modeling expertise required to use these low-level abstractions and languages is not available in the typical factory manufacturing environment.

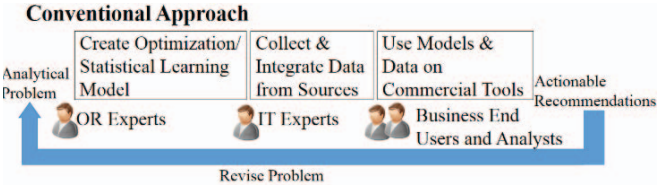

Figure 3. Conventional approach to analytics solutions.

III. Architectural Design and Framework Based on Reusable Knowledge Base of Process Performance Models

To overcome the challenges in the development of SM analysis and optimization capabilities, we advocate a paradigm shift. The key idea is to move from the non-reusable, task-centric, modeling approach to the approach more commonly used in Database Management Systems (DBMS). In a DBMS, a central data repository is updated continuously from multiple sources. DBMS users pose high-level 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 (AM) 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 models, composite models,* and *analytical*

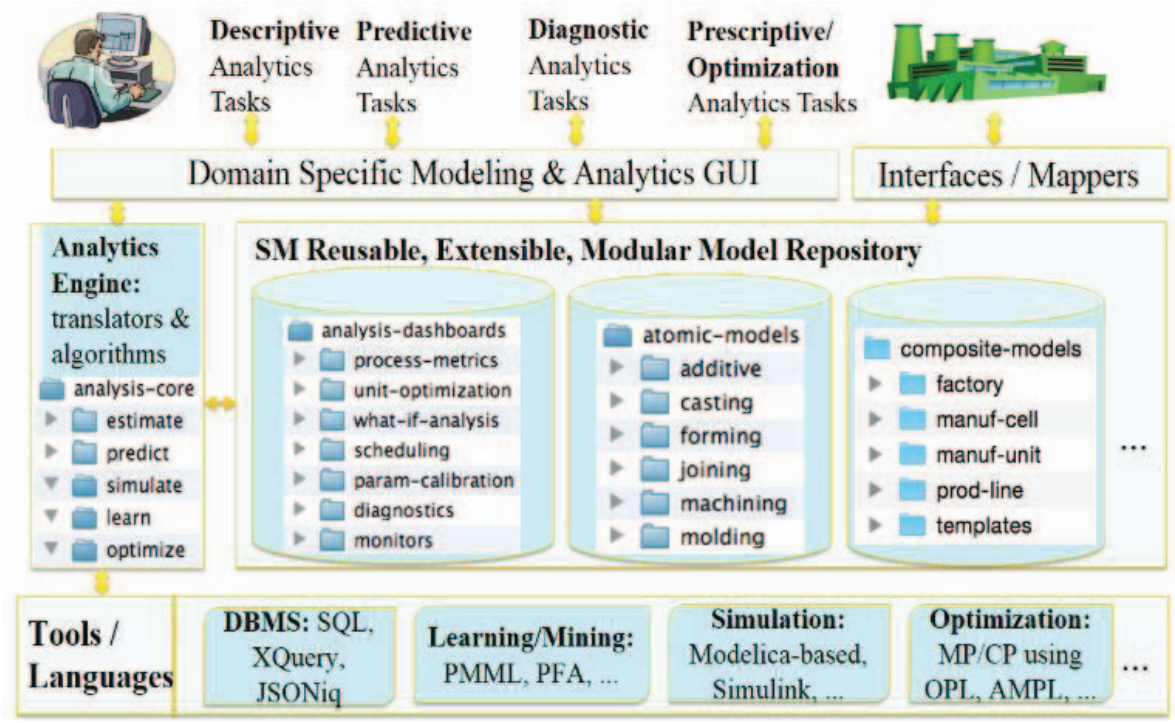

Figure 4. Proposed conceptual architecture.

views, according to types of knowledge or expertise required for developing these models.

A. Atomic Models

A*tomic-models* (in the middle of KB in Fig. 4) require most expertise and effort to build; however, they are also the most reusable ones. The atomic-models 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 sub-process. Each performance model contains the process parameters, control variables, performance metrics, and 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.

There are two types of atomic model: high level and low level. High-level atomic models may look like parameterized templates, while lower-levels models may contain models for vendor-specific equipment. Building atomic models requires expertise in process engineering and computer science. Domain-specific, process-engineering expertise includes an understanding of the equations defining performance metrics and constraints. Computer science expertise includes a working knowledge of a data-manipulation language to encode the data transformation and equations. However, building these models does not require expertise in optimization, MP, or statistical learning.

Once built, the atomic 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 model and historical performance data. Process engineers use these atomic models as the basis for creating composite models of composite processes.

B. Composite Models

A composite process is recursively composed of atomic processes and the associated aggregators, information flows, and timing constraints. The composite-model library contains performance models of such composite processes at different levels of granularity, such units, cells, lines, factories, and enterprises. Process engineers are capable of constructing composite models. To construct composite models, manufacturing process engineers only need 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 models can be constructed by using a drag-and-drop GUI. Fig. 1 is an example of such a model built using a GUI for the Car Manufacturing.

In the flow graph of Fig. 1, the user drags and drops the component models for the sub-processes (the rectangles), work-in-progress inventories (*a2, a3* etc. in the graph), and the flows that connect the sub-processes and the inventories. System-level metrics and feasibility constraints can be determined from the metrics and constraints of its sub-processes, recursively. The metric computation and feasibility constraints evaluation are done by the system using the corresponding atomic and composite models that use the model composition template. Thus, a composite model, just like an atomic 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 models is similar to the use of atomic 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. We use the *JSON* (JavaScript Object Notation) data model [26], which is becoming increasingly popular for analytical data, and its data manipulating language *JSONiq* [27]. Examples of *JSON* and *JSONiq* appear in Section IV.

C. 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 SCADA data, (4) parameter calibration of the power consumption as a function of machine controls, (5) composite-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 were 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 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.

IV. The Reusable Knowledge Base

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 type, composite-model type, or analytical-view type. The 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 model. We also describe general guidelines for developing and storing atomic models. Then, we provide an example for the composite 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 diagnostics.

A. Example of Atomic Model: Injection Molding

We use JSON and JSONiq to describe the atomic process models. 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, then forcing this molten material into a mold (die) where it cools and solidifies [28]. Consequently, the injection molding process consists of three major sub-processes [28-30]: (1) Melting, Injecting, or filling, (2) Cooling, (3) Ejection and resetting. The resulting cycle time, t_{cycle} can be formulated as according to [28-30]:

$$t_{cycle} = t_{inj} + t_{cool} + t_{reset}\,, \qquad (1)$$

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

The operators of the injection molding machines set a multitude of control parameters to ensure good product quality. In general, pressure and temperature parameters are often the ones that cause production problems. 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:

$$E_{melt} = P_{melt} \times \frac{V_{shot}}{Q} \quad , \qquad (2)$$

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 model contains all the process parameters, control variables, constraints, and coefficients. Fig. 5 shows the JSONiq code for computing Key Performance Indicators (KPIs) of the injection molding process using the JSON input data. For instance, Equation (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, FLWOR (For, Let, Where, Order By, Return) expressions and the equations to compute the quantity of interests.

```
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),
                                    ...
   $E_melt := ($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_cycl,
                                    ...
```

Figure 5. JSONiq formulation of the injection molding atomic model.

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 sub-process. Note that computations and equations are encoded in JSONiq with parameters and inputs imported from the JSON data model. In other words, the *process-dependent* variables, as a function of parameters and control variables are encoded in JSONiq.

B. Example of Composite Model: Car Manufacture Prep

In this subsection, we provide an example composite model based on the BTFP Car Manufacturing described in Section II. 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. Fig. 6 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 time setting and defines the temporal setting for the 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*).

```
{   "nameSpace": "http://www.mfda.com/manufacturing_KB/
        composite-models/prod-line/automotive/prep_flat",
    "type": "module",
    {  "id" : "prepProcess",
       "type" : "hierarchicalProcess",
       "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" : {"energy" : 15.84},
       "i_metrics" : ["energy"],
       "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" : {"0" : 28, "1" : 22,
                "2" : 22, "3" : 28, "4" : 33, "5" : 21,...}
          } ...
       ],
       "subProcesses" : [ {"localId" : "Uncoiling1"} ...],
    },
    { "id" : "Uncoiling1",
      "type" : "baseProcess",
       "I" : ["ToUncoiling1"], "O" : [FromUncoiling1"],
       "capacity" : 50,
       "i_metricPWLcoefficients" : {
          "energy" : {"bound1" : 10, "bound2" : 40,
          "slope1" : 5, "slope2" : 7, "slope3" : 9,
          "startX" : 0, "startY" : 1}
       },
       "i_metricValues" : { "energy" : 390.43 },
       "i_metrics" : ["energy"],
       "ii_inputPerOutput" : { "ToUncoiling1" : 1 },
       "itemFlows" : [
          {"id" : "ToUncoiling1",
             "materialType" : "ac",
             "pi_periodQty" : {"1" : 15, "2" : 17, ...},
             "ti_tpAllocation" : {"0" : 29, "1" : 23, ...}
          },
          { "id" : "FromUncoiling1",
             "materialType" : "uac",
             "pi_periodQty" : {"1" : 22, "2" : 17, ...},
             "ti_tpAllocation" : {"0" : 29, "1" : 30, ...}
          }
       ],
       "pi_accumulatedAmount" : {"1" : "39.22",
                     "2" : "30.69","3" :"35.68",...},
       "pi_throughputControl" : {"1" : "15.66",
                     "2" : "16.86","3" :"24.75",...},
       "ti_leftOver" : {"0" : "7.81", "1" : "6.21",
                        "2" : "7.31", ...}
    },
    {  "id" : "timeSetting",
       "noPeriods" : 18,
       "periodLength" : 1,
       "startTP" : 0,
       "type" : "temporalSetting"
    }
}
```

Figure 6. JSONiq formulation of the injection molding atomic model.

The JSON structure also contains subprocess for the Uncoiling 1 machine. This structure provides the parameters, variables, and metrics including 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*) for the Uncoiling 1 machine. The atomic model process 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 sub processes have a similar structure with different parameter, variable, and metric values.

Finally, the JSON structure contains the composite model for the Prep process (lines 3 to 55in Fig. 6) that encapsulates the reference to the time settings and the sub processes discussed above. The structure also contains the parameters, variables, and metrics for the composite 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. 1, input distribution aggregator (*inputAggr*), e.g., a1 in Fig. 1, and output distribution aggregator (*outputAggr*). Due to lack of space, these structures have been left out in Fig. 6. However, more details about these and other BTFP components can be found in [31].

C. Example of Analytical View: Diagnostics

Diagnostic tasks involve the process of detecting faults. This analytical view performs prediction of what is considered to be normal – typically min and max values. It also compares observed metric can be compared against the normal behavior to raise alarms or take corrective actions. One such implementation of diagnostics view may require the process engineer to make this comparison at certain points in time.

The JSONiq code for such a diagnostics function is shown in Fig. 7. The code reuses the atomic or composite modules from the KB to find the normal behavior. This can be done by running Monte Carlo simulations for each analysis window. Additionally, the function also finds the lower and the upper intervals of the expected metric with 95 % confidence. The function returns a JSON object containing the computed predicted values and the observed values over the display window. A process engineer can use this JSON object in a graph-visualization software application to diagnose faults in the system over the display window. The next subsection illustrates the use of such a diagnostics' view against the composite process of the Car manufacturing composite model discussed in the previous subsection.

D. Applying Diagnostics View on a Performance Model

To use the diagnostics view on the Car Manufacturing composite process, a process engineer writes a Diagnostics Analysis object in JSON as shown in Fig. 8. This object refers to the *prepDiagnosticsInput* composite model that is similar to the model discussed in Fig. 6 except that the controls and metrics and not instantiated. Using this analysis object, the diagnostics JSONiq code in Fig. 7 will first instantiate the diagnostics parameters (lines 3 to 14). Then, for each time point in the display window, the predicted energy metric for this analysis object is coded. First, the computed values that were observed at the beginning of the analysis window are initiated (line 20). Second, the control variables (*pi_throughputControl*) are initiated to the observed speeds for the entire analysis window (line 25). Third, the computed values for other time points in the analysis window are annotated as *dexpr* so that the system knows that these values need to be computed during prediction (line 29). Fourth, the predicted energy is computed using the predict function that performs the Monte Carlo simulations on the analysis object to find the expected (predicted) energy and the standard deviation (line 34). Using this information, the lower and upper intervals of energy consumption with 95 % confidence (line 38) are coded. Note that many of the given and derived values are random variables (with associated distributions). Finally, the JSON object of the computed prediction values and the observed values are returned for each time point over the display window (line 40).

A process engineer can use the JSON object returned by the diagnostics view in a chart-drawing software application to visualize the energy diagnostic information. This provides a good diagnostic tool to assess the running of the components on the manufacturing process. The process operator can use this tool to monitor the current metrics coming from the sensors; and, if these metrics go above the maximum or below the minimum, then the operator can determine the component that is at fault, within a certain confidence, and take the necessary steps to mitigate the fault.

V. Conclusion and Future Work

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 performance models, composite performance models, and analytical views and dashboards. Finally, we also showed an application of the diagnostics view over a Car Manufacturing composite model.

The proposed architectural framework and the Analytics engine follows the ideas from the Decision Guidance Analytics Language (DGAL) and framework proposed in [32] which, in turn, builds on prior work on decision

guidance and optimization languages. In particular, the unification of computation and equation syntax comes from CoJava [33], SC-CoJava [34], and DGQL [35]; CoReJava [36, 37], and DGAL and DG-Query [38]. 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 [39]. Finally, the concept of centralized Analytical KB (AKB) is borrowed from our previous work on SPAF [6, 25], which was limited to MP or CP optimization only.

```
let
 (:current time point>=($analysisWindow + $displayWindow):)
  $currentTP := $diagnosticsInput.currentTP,
  (:graph display window:)
  $displayWindow := $diagnosticsInput.displayWindow,
  (:diagnosis analysis window:)
  $analysisWindow := $diagnosticsInput.analysisWindow,
  (:observed Values:)
  $observedValues := $diagnosticsInput.observedValues,
  (:Metric on which diagnostics is to be performed:)
  $metric := $diagnosticsInput.metric
  (:Fetch the performance model $M as indicated by
  analyticalObjectId from KB:)
  $P := KB($diagnosticsInput.analyticalObjectId),
  $diagnosticGraphPoints :=
    for $t in ($currentTP - $displayWindow to $currentTP)
    let
      (:Initial the computed values of $P with values from
      $observedValues at time point $t-$analysisWindow:)
      $P1 := initiateComputedValues($P, $observedValues,
           $t-$analysisWindow),
      (:Initiate the ccontrol variables of $P1 with values
      from $observedValues at time points in time window
      [$t - $analysisWindow, $t]:)
      $P1 := initiateControlVariables($P1,
          $observedValues, $t - $analysisWindow, $t),
      (:Annotate the computed values of $P1 with the string
      "dexpr" for all time points in time window:)
      $P1 := annotateComputedValues($P1, "dexpr",
          $t - $analysisWindow + 1, $t),
      (:Perform prediciton on $P1 for $metric in the time
       window .The returned values are the expected value
       and the standard deviation of the metric:)
      ($expectedValue,$standardDediation) :=
           predict($P1,$metric, $t-$analysisWindow, $t),
      (:Compute the lower and upper intervals of the
      metric with 95 % confidence :)
     ($LB,$UB):=conf($expectedValue,$standardDediation,95),
      (:Return the values as a JSON object for $t:)
      return {$t:{observed:$observedValues.$metric.$t,
                  predicted:$expectedValue,LB:$LB,UB:$UB}}
```

Figure 7. Diagnostics analytical view.

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 on extending 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.

```
{   "type": "Analysis",
    "subType": "diagnostics",
    "id": "CarManufacturingEnergyDiagnostics",
    "analyticalObjectId": "prepDiagnosticsInput",
    "currentTP" : 14,
    "displayWindow" : 10,
    "analysisWindow" :3,
    "observedValues":{"energy":{"1":12,"2":14,"3":10,...}},
    "metric": "energy"
}
```

Figure 8. Diagnostics analysis object for the car manufacturing example.

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.

ACKNOWLEDGMENT

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 United States Government and is not subject to copyright.

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 27, 2015].

[2] Salvendy, G. 2001. Handbook of Industrial Engineering: Technology and Operations Management, Third Edition. John Wiley & Sons, 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: Sept. 17, 2015].

[4] C. Undey, S. Ertun, T. Mistretta, and B. Looze, "Applied advanced process analytics in biopharmaceutical manufacturing: Challenges and prospects in real-time monitoring and control," Journal of Process Control, vol. 20, no. 9, Oct. 2010, pp. 1009 – 1018, doi: doi:10.1016/j.jprocont.2010.05.008.

[5] G. C. Gilvan, "Supply chain analytics," Business Horizons, vol. 57, no. 5, Sep. 2014, pp. 595 – 605, doi: 10.1016/j.bushor.2014.06.004.

[6] G. Shao, S. Shin, and S. Jain, "Data analytics using simulation for smart manufacturing," Proc. 2014 Winter Simulation Conference, ser. WSC '14. Piscataway, NJ, USA: IEEE Press, Dec. 2014, 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] S. Shin, J. Woo, and S. Rachuri, "Predictive analytics model for power consumption in manufacturing," Proc. 21st CIRP Conference on Life Cycle Engineering, vol. 15, no. 0, 2014, pp. 153 – 158, doi: 10.1016/j.procir.2014.06.036.

[9] J. Lee, E. Lapira, B. Bagheri, and H. Kao, "Recent advances and trends in predictive manufacturing systems in big data environment," Manufacturing Letters, vol. 1, no. 1, Oct. 2013, pp. 38 – 41, doi: 10.1016/j.mfglet.2013.09.005.

[10] A. Brodsky, G. Shao, and F. Riddick, "Process analytics formalism for decision guidance in sustainable manufacturing," Journal of Intelligent Manufacturing, Mar. 2014, pp. 1–20, doi: 10.1007/s10845-014-0892-9.

[11] C. Groger, F. Niedermann, H. Schwarz, and B. Mitschang, "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, May 2012, pp. 793–799, doi: 10.1109/CSCWD.2012.6221911.

[12] S. Katz, Y. Labrou, M. Kanthanathan, and K. Rudin, "Method for managing a workflow process that assists users in procurement, sourcing, and decision-support for strategic sourcing," US Patent number US20020174000 A1, Nov. 2002.

[13] S. Harkins and M. P. Reid, "Structured query language," SQL: Access to SQL Server, 2002, pp. 1–5, doi: 10.1007/978-1-4302-1573-8_1.

[14] M. Rys, D. Chamberlin, and D. Florescu, "XML and relational database management systems: The inside story," Proc. International Conference on Management of Data, ser. ACM SIGMOD,May 2005, pp. 945–947, doi: 10.1145/1066157.1066298.

[15] D. Florescu and G. Fourny, "JSONiq: The history of a query language," IEEE Internet Computing vol. 17, no. 5, Sept. 2013, pp. 86–90, doi: 10.1109/MIC.2013.97.

[16] P. Fritzson and V. Engelson, "Modelica – a unified object-oriented language for system modeling and simulation," Proc. European Conference on Object-Oriented Programming, Jul. 1998, pp. 67–90, doi: 10.1007/BFb0054087.

[17] J. Akesson, K. E. Arzen, M. Gafvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with optimica and JModelica.org languages and tools for solving large-scale dynamic optimization problems," Computers & Chemical Engineering, vol. 34, no. 11, Nov. 2010, pp. 1737 – 1749, doi: 10.1016/j.compchemeng.2009.11.011.

[18] R. Fourer, D. M. Gay, and B. W. Kernighan, "AMPL: A mathematical programming language," AT&T Bell Laboratories, Murray Hill, NJ 07974, Tech. Rep., 1987.

[19] A. Brook, D. Kendrick, and A. Meeraus, "GAMS, a user's guide," *ACM SIGNUM Newsletter*., vol. 23, no. 3-4, Dec. 1998, pp. 10–11, doi: 10.1145/58859.58863.

[20] P. Van Hentenryck, L. Michel, L. Perron, and J.-C. Rgin, "Constraint programming in OPL," Principles and Practice of Declarative Programming, ser. Lecture Notes in Computer Science, G. Nadathur, Ed. Springer Berlin Heidelberg, 1999, vol. 1702, pp. 98–116, doi: 10.1007/10704567_6.

[21] Data Mining Group (DMG), "The predictive model markup language (PMML) 4.2," 2014. [Online]. Available: http://www.dmg.org/. [Accessed: June 27, 2015].

[22] A. Guazzelli, W.-C. L., and T. J., PMML in action: unleashing the power of open standards for data mining and predictive analytics. North Charleston, South Carolina: CreateSpace, 2012.

[23] J. Pivarski, "PFA: Portable format for analytics (version 0.6)," 2015. [Online]. Available: http://scoringengine.org/. [Accessed: June 27, 2015].

[24] V. Jain and I. E. Grossmann, "Algorithms for hybrid MILP/CP models for a class of optimization problems," INFORMS Journal on Computing, vol. 13, no. 4, Sept. 2001, pp. 258–276, doi: 10.1287/ijoc.13.4.258.9733.

[25] A. Alrazgan and A. Brodsky, "Toward reusable models: System development for optimization analytics language (OAL)," Technical Report, George Mason University, Fairfax, VA, 22030, Tech. Rep. GMU-CS-TR-2014-4, 2014. [Online]. Available: http://cs.gmu.edu/tr-admin/papers/GMU-CS-TR-2014-4.pdf. [Accessed: June 27, 2015].

[26] G. Fourny, M. Brantner, and F. Cavalieri, "JSONiq the SQL of NoSQL," CreateSpace Independent Publishing Platform, 2013.

[27] 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 27, 2015].

[28] G. Boothroyd, P. Dewhurst, and W. Knight, Product design for manufacture and assembly Third Edition ed2011. Florida, USA: CRC Press, Tyler and Francis Group. XIII, 2011.

[29] J. Madan, M. Mani, J. H. Lee, and K. W. Lyons, "Energy performance evaluation and improvement of unit-manufacturing processes: injection molding case study," Journal of Cleaner Production, vol. 105, no. 1, Oct. 2015, pp.157-170, doi: 10.1016/j.jclepro.2014.09.060.

[30] J. Madan, M. Mani, and K. W. Lyons, "Characterizing energy consumption of the injection molding process," Proc. *International Manufacturing Science and Engineering Conference*, vol. 2, no. 1. Manufacturing Engineering Division, Jun. 2013, pp. 1–13, doi: 10.1115/MSEC2013-1222.

[31] M. Krishnamoorthy, A. Brodsky, and D. Menasce, "Temporal manufacturing query language (tMQL) for domain specific composition, what-if analysis, and optimization of manufacturing processes with inventories," Technical Report, George Mason University, Fairfax, VA, 22030, Tech. Rep. GMU-CS-TR-2014-3, 2014. [Online]. Available: http://cs.gmu.edu/ tr-admin/papers/GMU-CS-TR-2014-3.pdf. [Accessed: June 27, 2015].

[32] A. Brodsky and J. Luo, "Decision guidance analytics language (DGAL) - toward reusable knowledge base centric modeling," Proc. 17th International Conference on Enterprise Information Systems, Apr. 2015, pp. 67-78 doi: 10.5220/0005349600670078.

[33] A. Brodsky and H. Nash, "CoJava: Optimization modeling by nondeterministic simulation," in Principles and Practice of Constraint Programming-CP 2006. Springer, 2006, pp. 91–106, doi: 10.1007/11889205_9.

[34] A. Brodsky, M. Al-Nory, and H. Nash, "Service composition language to unify simulation and optimization of supply chains," Proc. 41st Annual Hawaii International Conference on System Sciences, , Jan 2008, pp. 74–74, doi: 10.1109/HICSS.2008.386 .

[35] A. Brodsky, S. Mana, M. Awad, and N. Egge, "A decision-guided advisor to maximize ROI in local generation utility contracts," Proc. Innovative Smart Grid Technologies, IEEE PES, Jan 2011, pp. 1–7, doi: 10.1109/ISGT.2011.5759185.

[36] A. Brodsky, J. Luo, and H. Nash, "CoReJava: Learning functions expressed as object-oriented programs," Proc. Seventh International Conference on Machine Learning and Applications, 2008. ICMLA '08., Dec 2008, pp. 368–375, doi: 10.1109/ICMLA.2008.144.

[37] J. Luo and A. Brodsky, "Piecewise regression learning in CoReJava framework," International Journal of Machine Learning and Computing, vol. 1, no. 2, 2011, pp. 163–169,, doi: 10.7763/IJMLC.2011.V1.24.

[38] A. Brodsky, S. G. Halder, and J. Luo, "DG-Query, XQuery, mathematical programming," Proc. 16th International Conference on Enterprise Information Systems (ICEIS 2014), 2014.

[39] A. Brodsky and X. Wang, "Decision-guidance management systems (DGMS):Seamless integration of data acquisition, learning, prediction and optimization," Proc. 41st Annual Hawaii International Conference on System Sciences, , Jan 2008, pp. 71–71, doi: 10.1109/HICSS.2008.114.