A System and Architecture for Reusable Abstractions of Manufacturing Processes

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

***Abstract*— In this paper we report on the development of a system for managing a repository and conducting analysis and optimization on manufacturing performance models. The repository is designed to contain (1) unit manufacturing process performance models, (2) composite performance models representing production cells, lines, and facilities, (3) domain specific analytical views, and (4) ontologies and taxonomies. Initial implementation includes performance models for milling and drilling as well as a composite performance model for machining. These performance models formally capture (1) the metrics of energy consumption, CO_2 emissions, tool wear, and cost as a function of process controls and parameters, and (2) the process feasibility constraints. The initial scope of the system includes (1) an Integrated Development Environment and its interface, and (2) simulation and deterministic optimization of performance models through the use of Unity Decision Guidance Management System.**

***Keywords*-Knowledge Base, Unit Manufacturing Processes, Repository, Optimization, Reusable Manufacturing Models**

I. Introduction

Citing the Smart Manufacturing Leadership Coalition (SMLC), "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" [1]. These software and computing architectures require capabilities to support the development of analysis and optimization solutions. Additionally, these capabilities need to be designed for multiple operational levels, including manufacturing units, cells, production lines, factories, and supply chains [2].

As indicated by Brodsky et al. [3], the required analysis and optimization capabilities can be broadly classified as descriptive ("what happened?") [4][5], diagnostic ("why did it happen?") [6][7], predictive ("what will happen?") [8][9], and prescriptive ("how can we make it happen?") [10][11]. 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 as well as results in models and algorithms that are difficult to modify, extend, and reuse. A key contributor to these deficiencies is the diversity of computational tools, each designed for a different task such as data manipulation, statistical learning, data mining, optimization, and simulation. Because of this diversity, modeling using computational tools often requires the use of specialized low-level mathematical abstractions. As a result, the same manufacturing knowledge is often modeled multiple times using different specialized abstractions, instead of being modeled once using a uniform abstraction. Furthermore, the modeling expertise required for the low-level abstractions and languages is typically not within the realm of knowledge of manufacturing users.

Recently proposed as a related effort [3] was an architectural design and framework for fast development of software solutions for descriptive, diagnostic, predictive, and prescriptive analytics of dynamic production processes. The uniqueness and novelty of the architecture was its middleware layer, which is based on a reusable, modular, and extensible knowledge base (KB) of process performance models. To demonstrate this, an organization and key structure of the reusable KB was proposed and this design was illustrated by prototyping a decision support system that allows process engineers to hierarchically compose temporal processes and perform deterministic and stochastic optimization of these processes. The term KB and repository are used interchangeably in this paper.

However, this related effort lacked a systematic design of the unit manufacturing process (UMP) repository and possible ecosystems around the repository, as well as a specific architecture for such a repository. Furthermore, it did not address an implementation of a reusable repository and support for populating it with dynamic production processes. While temporal processes and their composition was considered [3], atomic (or lowest level) components were estimated by piecewise-linear functions whereas real-world process models, which are typically physics-based, require non-linear arithmetic to describe them.

Addressing these gaps and limitations is the focus of this paper. More specifically, the contributions of this paper are as follows. First, we propose the concept of a reusable KB of manufacturing process models, its functionality and high-level system architecture capable of supporting future ecosystems around it. The repository is designed to contain (1) unit manufacturing process (UMP) performance models, (2) composite performance models representing production

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

cells, lines, and facilities, (3) domain specific analytical views, and (4) ontologies and taxonomies.

Second, we implement an initial collection of performance models for milling and drilling as well as a composite performance model for machining. These performance models formally capture (1) the metrics of energy consumption, CO_2 emissions, tool wear, and cost as a function of process controls, and (2) the process feasibility constraints.

Third, we develop a system for managing a repository and conducting analysis and optimization on manufacturing models. The initial scope of the system includes (1) an Integrated Development Environment (IDE) and its interface through the use of Atom Studio [13], (2) simulation and deterministic optimization of performance models through the use of Unity Decision Guidance Management System (DGMS), and (3) model management and version control through the use of the standard interface of GitLab [14].

Lastly, we conduct a case study focusing on optimization and trade-off analysis to demonstrate that the system and the proposed architecture are capable of supporting a real manufacturing case and are computationally feasible. The limited case study reflects the machining of a heat-sink part.

The rest of this paper is organized as follows. Sections II and III describe the ecosystem and high-level system architecture of the manufacturing process knowledge base respectively. Section IV presents methods in constructing UMP models for milling, drilling, and machining. Section V introduces a system implementation of the architecture detailing its functionality. Section VI discusses how the system is implemented by showing its workflow for an optimization problem. Section VII describes a case study of a machining sequence for a part. Lastly, Section VIII concludes the paper.

II. The Knowledge Base Ecosystem

The proliferation of sensing technologies in manufacturing allow for new opportunities in monitoring and controlling. These changes have widened the user population for cloud-based services, such as the UMP repository described here. This section details the ecosystem of the KB, including stakeholders and workflows relevant to its incarnation.

A. Scope

The scope of the KB is specific to the manufacturing domain, primarily manufacturing process models. Its core feature is a library of unit manufacturing process models that can be used by various tools and applications for a multitude of manufacturing-related analyses. The plan is for NIST to serve as the curator of the model repository. Researchers and developers will contribute the various tools and applications exploiting the KB contents from different perspectives.

B. Stakeholders

Stakeholders can be classified into three general groups: contributors, consumers, and administrators. Contributors are users that provide the models themselves. Contributers include manufacturing asset vendors, process engineers, systems engineers, standards development organizations (SDO), and researchers. Consumers are those users that extract knowledge from the KB to perform manufacturing-related analysis. Consumers include design & manufacturing engineers, supply chain stakeholders, managers, and procurement decision-makers. Lastly, administrators govern the operation of the KB. The administrators' responsibilities include developing model validation protocols, enforcing governance over the semantic integrity of the KB entries, and the overall organization and management of the KB.

C. Workflows

Use-case scenarios of the UMP repository have been explored in past research [15]. Here, we expand on two of them in detail, including (1) life cycle inventory analysis (LCIA) and (2) integration with computer-aided technologies (CAx).

1) Life Cycle Inventory Analysis: Others have developed databases of UMP models in the context of life cycle inventory (LCI) development for broad dissemination, most notably the Manufacturing Unit Process Life-Cycle Inventory (UPLCI) heuristics database and its underlying framework [16]. One challenge with life cycle inventory analysis (LCIA) is the uncertainty and fidelity of the underlying data. For manufacturing processes, LCIA practitioners rely on highly aggregated data that often describes multiple sources and processes within a single unit process. More prudent incorporation of manufacturing data in sustainability assessment has become an area of focus, shown by the recent standards work, e.g. ASTM International [17] and ISO [18]. However, these efforts remain in a nascent stage and the standards have yet to reach broad dissemination. Thus, there is still limited reliable information available for manufacturers to perform quick analysis. Instead, the onus is on the manufacturers to measure and record individual metrics related to sustainability, e.g. energy consumption, worker safety, and part rework efficiency.

With the inclusion of a performance model repository, the barriers to perform quick sustainability assessment of production systems would be greatly reduced. As of now, in our current implementation, performance models in the knowledge base are based on the proposed schema in the related ASTM standard [19] while life cycle inventory analysis is based on the EcoSpold *de facto* standard [20]. ISO 20140 provides directions for linking components of such manufacturing models to LCIA databases. To extend the use of our model repository, we plan to construct any necessary translators to enable LCIA integration.

2) Computer-aided technologies (CAx Tools): As seen in Fig. 1, the KB supports a number of domain-specific technologies. With recent demand in improving information flow and reuse in manufacturing, e.g. efforts to support the digital thread [21], new manufacturing models must

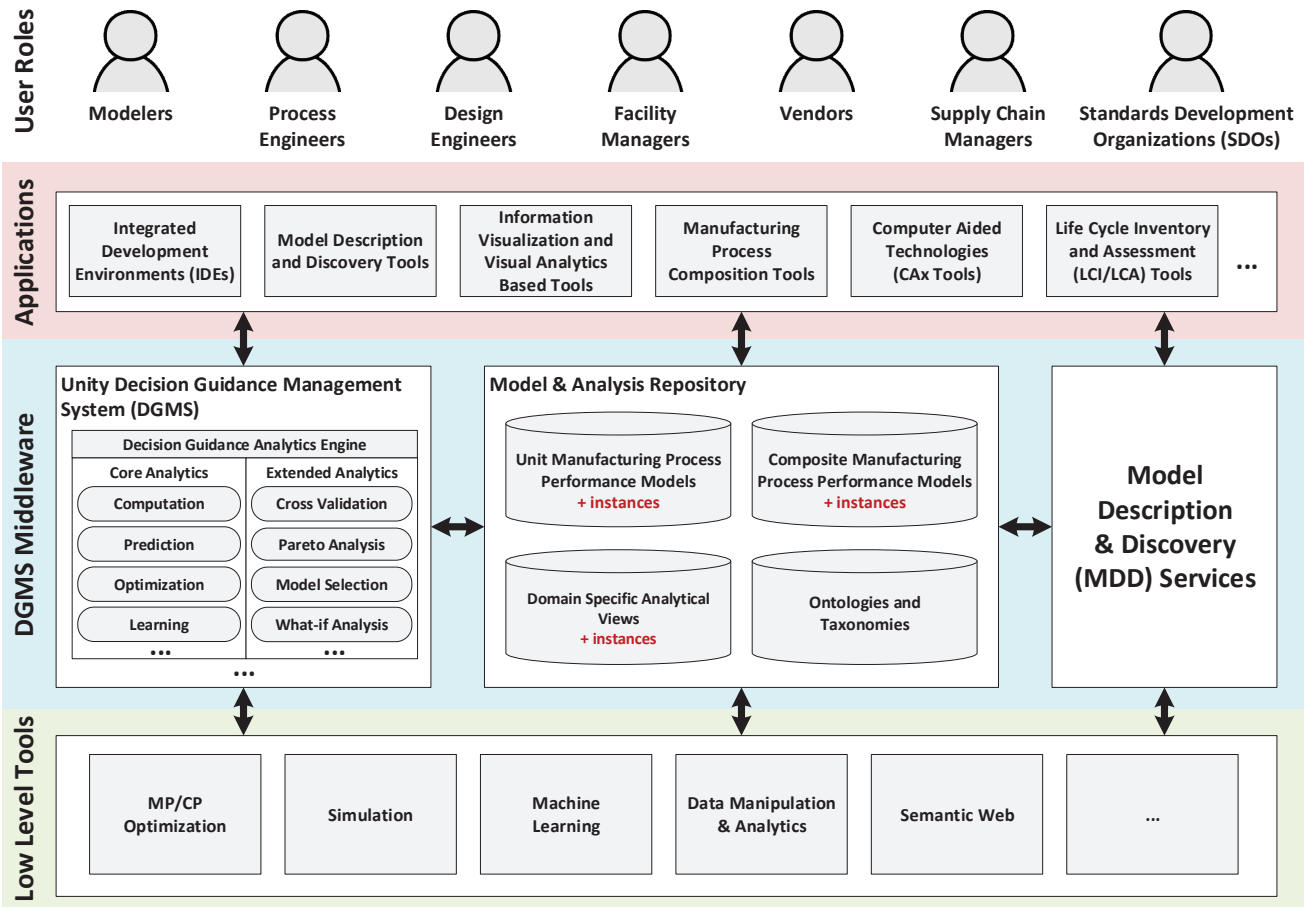

Figure 1. High-level system architecture of the UMP repository and its surrounding components.

be integrated amongst existing computer-aided engineering (CAE) technologies. Here, we allow for the extended use of streaming data to validate or improve existing performance models as well as to develop new models through statistical techniques, e.g. regression analysis. If linked to existing CAD platforms, performance models could also be used to predict cost and required resources.

One challenge is the variety in domain specific CAE tools and in the stakeholders' roles and perspectives. To address this gap, future work will provide a development environment so that KB consumers can develop their own support structures, i.e. a data integration layer through an API. Recently, trends in the cloud-based engineering domain lend themselves to these activities, such as OnShape[1], a fully cloud-based, browser-driven CAD system, and WebGME, a fully web-based modeling environment [22].

[1]https://www.onshape.com/

III. High-Level System Architecture

The high-level system architecture is depicted in Fig. 1. The user roles at the top correspond to the roles of stakeholders described in Section II. The application layer contains generic and domain-specific user applications to support workflows, e.g., LCIA and CAx tools, as well as additional applications such as IDEs, Model Description & Discovery tools, Information Visualization and Visual Analytics tools, and Manufacturing Process Composition tools. The system relies on a number of external, lower level tools that provide a diverse range of capabilities, including MP/CP optimization, simulation, machine learning, data manipulation and analytics as shown in the lower level of Fig. 1. Typically, the application layer apps are implemented using the low-level tools directly. But these solutions are typically implemented from scratch, following a linear methodology. Due to the diversity of low-level tools, apps implemented using a tool are difficult to modify, extend, and reuse. As a result, the same manufacturing knowledge is often modeled multiple times using different specialized

abstractions, instead of being modeled once using a uniform abstraction. These limitations are overcome in this architecture using the middleware in Fig. 1. The uniqueness of the middleware is that it is centered around the model & analysis repository. The middleware layer provides a uniform, high-level abstraction over different low-level tools and is key in supporting analysis and optimization of a reusable, modular and extensible repository of manufacturing models. We now borrow and extend from previous work to describe the major architectural components of the middleware [3][23].

A. Model & Analysis Repository

The model & analysis repository contains (1) UMP performance models, (2) composite process performance models, and (3) domain-specific analytical views, and (4) ontologies and taxonomies. Here, we describe each of them in detail.

The UMP models library contains a classification hierarchy of pre-built performance models for unit manufacturing processes. A performance model represents (1) metrics and performance indicators of a function of process parameters and control variables as well as feasibility constraints of the process. These models may be deterministic or stochastic in the way the input parameters and control variables affect the metrics. For instance, in 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, subject to demand constraint and the bound constraints on the control variables. We further discuss the UMP performance models in Section IV.

A composite process is recursively composed of UMPs or other composite processes and their associated aggregators, information flows, and timing constraints. The composite process performance model library contains performance model templates for such composite processes at different levels of granularity, such as units, cells, lines, and factories.

The domain-specific analytical views are advanced analytical services that can be implemented by a data analyst or domain expert. Examples include a view for the manufacturing-process composition tools to build a composite process performance model to perform metric computation as a function of individual machine metrics and a view for information visualization tools to visualize a pareto-optimal curve that allow the user to make trade-off analysis between competing objectives. Lastly, ontologies and taxonomies, detailed in Section III-C, allow for better model exploration and discovery depending on user expertise.

The performance models in the model & analysis repository are developed in the data manipulation language JSONiq. JSONiq is a language commonly used for querying and manipulating JSON structures. This is analogous to SQL being used for querying and manipulating relational data. Hence, JSONiq is often called "SQL for noSQL data" [24].

B. Unity Decision Guidance Management System

The key technical challenge in realizing a system based on this architecture lies in developing specialized algorithms that automatically translate a uniform, high-level representation of performance models into the low-level, specialized models required by each of the underlying tools. The solution to this challenge is centered around Unity Decision Guidance Management System (DGMS) [25] that provides support for different methods of analysis, such as simulation, optimization and learning, based on reusable analytics models contained in the model & analysis repository. With Unity DGMS, a manufacturing performance model is developed and can be reused for different methods of analysis.

While performance models are expressed directly in JSONiq, the analytics functions, such as prediction, optimization, and learning of the UNITY DGMS, are expressed in the Decision Guidance Analytics Language (DGAL) [26][27]. DGAL extends JSONiq with mechanisms for developing reusable analytical models, as well as advanced analytical services to support model-based descriptive, predictive and prescriptive analytics. The DGAL language is syntactically equivalent to JSONiq, and exposes advanced analytics capabilities as regular JSONiq functions, e.g. *dgal:argmin* and *dgal:argmax* for optimization, and *dgal:learn* for learning. In DGAL, certain numerically or logically-typed properties in the input JSON object can be replaced with decision variables or learning parameters, for which can then be solved by invoking one or more of the analytical services.

To implement analytics functions, such as optimization, the Unity DGMS analytics engine intervenes in the execution flow of JSONiq to recompile analytical performance models into the target low level models. Rather than statically compiling analytical models into different solver-specific languages, such as Optimization Programming Language (OPL) [28], the analytics engine first performs a symbolic execution of the analytical model on an instance of its input to produce an intermediate analytical representation of the optimization problem. Solver-specific models are then generated from the simpler, intermediate analytical representation. More details on Unity DGMS can be found in previous work [25].

The analytics-core methods (compute, predict, learn, simulate, and optimize) are part of the Analytics Engine. Implementing these methods requires reduction and compilation techniques, as well as specialized optimization and learning algorithms. However, once implemented, the analytics-core methods will allow fast implementation of advanced 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 UMP performance models, thus enabling their reusability.

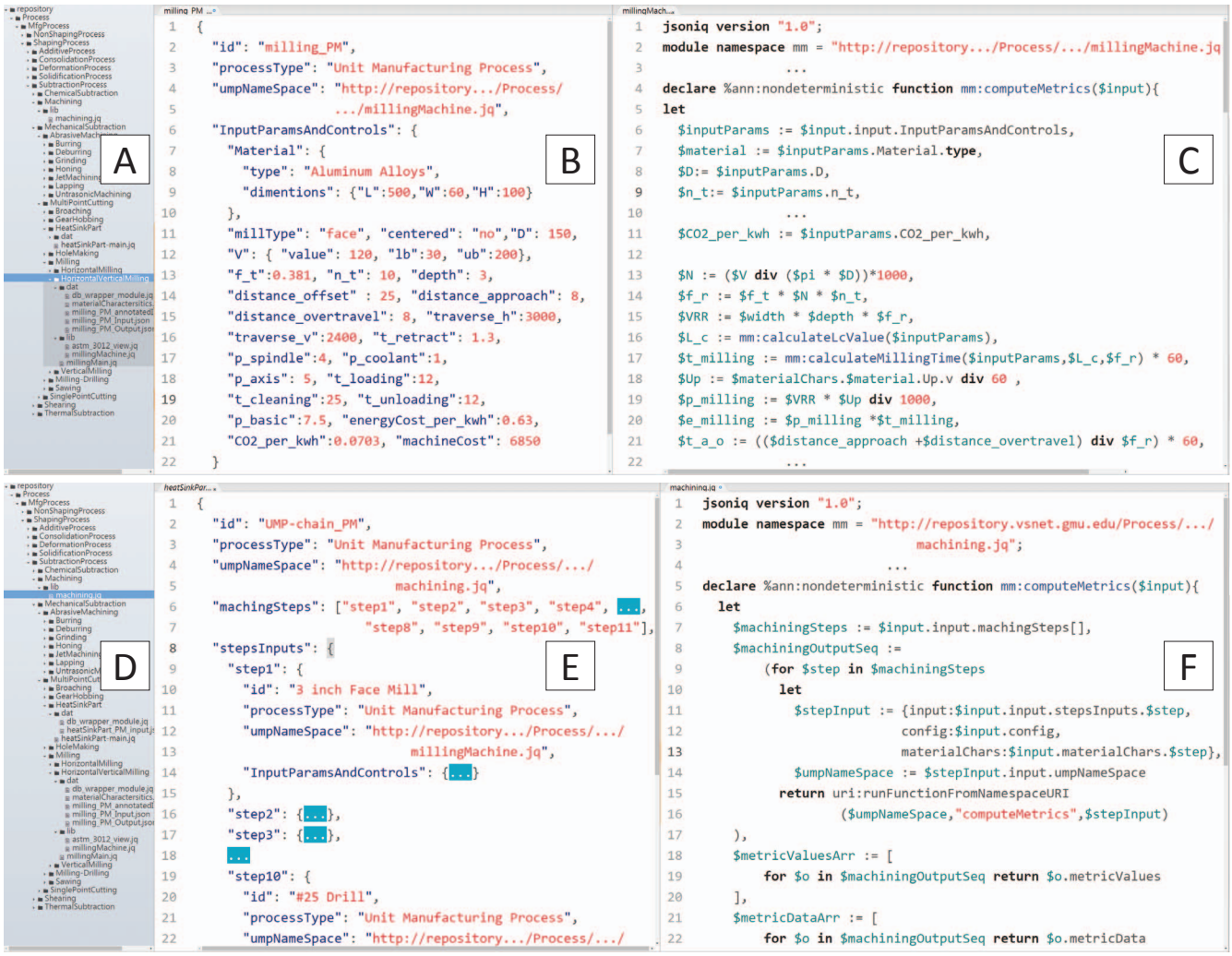

Figure 2. (A) Location of Milling UMP in taxonomy. (B) Example of a single JSON input to instantiate the UMP's transformation function (C). (C) Transformation function of the Milling UMP. (D) Location of Machining Composite UMP. (E) Composition of Milling and Drilling UMPs (e.g. B & C) detailing process plan for the heat sink part. (F) Machining transformation function that accepts composite UMP, E.

C. Ontologies and Description & Discovery Services

The repository may contain thousands of performance model artifacts. To be able to find, discover, and compose these artifacts easily from the repository, we anticipate having ontologies, taxonomies, and description & discovery services. Manufacturing processes can be organized based on accepted taxonomies, e.g. from Todd et al. [29] or from Kalpakjian and Schmidt [30], with the capabilities of each process expressed using a supplier discovery ontological framework [31]. One challenge is developing a formal description of the contributed models and their metadata. If properly defined, knowledge about manufacturing models can be expressed in structured database views. The advantages of using views in a database include the ability to hide complexity from the user, the introduction of an additional security later, and the support they provide to legacy code. In all, these considerations yield the potential for advanced query mechanisms, e.g. natural language queries on the models and their metadata.

IV. UMP Performance Models

We illustrate the use of the proposed model & analysis repository and the analytics services for manufacturing users in Section V. Here, we introduce the performance models for a machining UMP, composed of milling and drilling UMPs. These UMPs are integral pieces of the presented system. Specifically, the machining UMP forms the basis for the case study that we discuss in Section VII. The key idea is that using these UMPs as performance models makes it possible to compute the metrics that are a function of the input parameters and controls subject to feasibility constraints.

In the repository, these UMPs are organized based on the manufacturing taxonomy from Todd et al. [29].

The machining UMP is composed of milling and drilling UMPs. Before showing the performance model for the machining UMP, we look at the milling and drilling UMPs. The top of Fig. 2 shows the milling UMP in a window with three panes: left (A), middle (B), and right (C). Fig. 2A shows the taxonomy of the UMP repository where we can see that the milling UMP is a shaping process that mechanically subtracts by performing multi-point cutting operations. Within each UMP folder, we include dat and lib subfolders that contain related artifacts, including a standard view of the UMP as described in ASTM 3012-16 [19]. The input JSON in Fig. 2B contains the milling UMP parameters such as material characteristics ($Material$, lines 7-10), number of teeth (n_t, line 13), and depth of cut ($depth$, line 13) as well as controls such as cutting speed (V, lines 12). The input JSON also contains the feasibility bounds for the controls (lines 12). The inputs to the UMP transformation function is specific to the model type, i.e. physics or data-based, and model elements.

Finally, the JSONiq code in Fig. 2C contains snippets of the transformation function (*computeMetrics*, line 4) for the performance model that transforms the input parameters and controls to the output metrics along with a boolean that signifies whether the constraints were satisfied or not. The function first reads the parameter and control input values from the input JSON that is passed as a parameter to the function (lines 6-11). Then, the code shows the transformation equations written in JSONiq that transforms the input to output (lines 13-21). In this case, these equations are taken from the Unit Process Life-Cycle Inventory database [32]. The model estimates energy consumption of the milling operation by considering the active cutting time, rate of removed volume, and specific cutting energy (U_p). The relationships of the cutting parameters are mostly dictated by common practice and historical experimentation, e.g. for aluminum alloys ($U_p = 0.98\frac{W}{mm^3}$) feasible cutting speeds include 120-140$\frac{m}{min}$. After computing the transformation equations for this UMP, the code returns the output JSON structure of metrics and constraints (not shown in Fig. 2C).

Similarly, we constructed a drilling UMP. The difference of the energy consumption model used for drilling with milling lies in the the volume removal rate calculation. The explanations for the input JSON and JSONiq transformation function for drilling are similar to those provided for the milling UMP above and hence have been omitted.

To create a performance model for the machining UMP as shown in the bottom half of Fig. 2, we consider an example of the heat sink part. The finished heat-sink part is shown in Fig. 3. In order to produce this part, a number of milling and drilling operations are performed in sequence. The complete list of operations required to produce the heat-sink part is shown in Table I. The first column in the table is the position of the operation in the production sequence of the part. The second column denotes the specific cutting tool required and the third column gives the type of the operation.

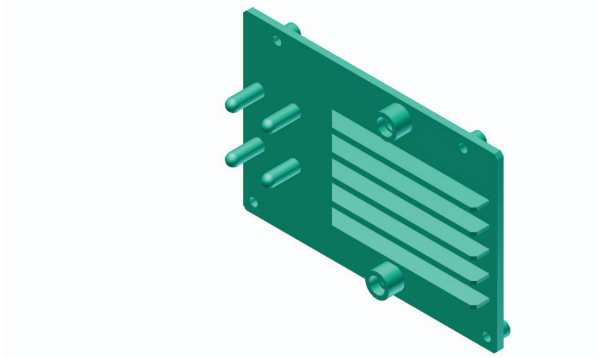

Figure 3. Heat-Sink Part

Table I
SEQUENCE OF OPERATIONS TO PRODUCE THE HEAT-SINK PARK

#	Machining Tool Description	UMP Type
1	3 inch Face Mill	Milling
2	1/2 inch Dia. 2 Flute Stubby Fullerton E.M.	Milling
3	1/4 inch Dia. 2 Flute Stubby Fullerton E.M.	Milling
4	3/16 inch Dia. 2 Flute Stubby Fullerton E.M.	Milling
5	1/4 inch x 45 Chamfer Mill	Milling
6	1/4 inch 2 Flute E.M. With .020 inch x 45 Chamfers	Milling
7	1/4 inch x .093 inch Corner Rounding E.M.	Milling
8	1/4 x 90 Spot Drill	Drilling
9	3/16 inch Drill	Drilling
10	#25 Drill	Drilling
11	1/4 inch Dia. 2 Flute Stubby Fullerton E.M.	Milling

Fig. 2D shows the taxonomy from the perspective of a machining UMP in the left pane. The JSON in Fig. 2E shows the snippet of input for the heat-sink machining example described above. This input contains steps that corresponds to the order of operations given in Table I. Line 6-7 gives the order of the steps and lines 9-15 onwards provide the input snippet of the milling UMP corresponding to the first operation and so on. The structure of these inputs are similar to those that were shown in Fig. 2B but with the values specific to the heat-sink example. Due to brevity, only a snippet of the input is shown here. Finally, Fig. 2F shows the code snippet for a generalized transformation function for a machining UMP (line 5). For each step in the input, this function runs the *computeMetrics* function corresponding to the respective UMP type (e.g., see *computeMetrics* for the milling UMP type in Fig. 2C) to get the output (lines 8-17). Then, the machining transformation function aggregates the metrics and constraints from each step output (lines 18-22) and returns the aggregated metrics and constraints (not shown in Fig. 2F).

```
millingMain.jq
1  jsoniq version "3.0";
2  import module namespace ns= "http://repository.vsnet.gmu.edu/.../millingMachine.jq";
3  import module namespace dgal = "http://mason.gmu.edu/~mnachawa/dgal.jq";
4                          ...
5  let
6    $annotatedInputParams := $dat:annotatedPmInput,
7    $materialChars := $dat:materialCharactersitics,
8    $output := dgal:argmin({config:{noOfCycles:5},input:$annotatedInputParams,
9                 materialChars: $materialChars}, ns:computeMetrics#1,
10                "metricValues.cost.total",
11                { language: "ampl", solver: "minos" })
12   return $output
```

```
milling_PM_Out..
1  {
2     "id": "outFrom milling_PM",
3     "processType": "Unit Manufacturing Process",
4     "umpNameSpace": "http://repository.vsnet.gmu.edu/.../millingMachine.jq",
5     "metricValues": {
6       "cost": {
7           "total": 1.787603241329216,
8           "energy": 0.213826974007662,
9           "wearAndTear": 1.5737762673215538
10      },
11        "productivity": {
12          "totalTime": 98.094680632303977
13        },
14        "sustainability": {
15          "energy": 0.3724395306952631,
16          "co2": 0.026182499007877
17        },
18        "aux": {
19          "input": 3000000,
20          "amountProduced": 1
21        }
22    },
23    "metricData": {...},
24    "constraints": true
25 }
```

Figure 4. Window Showing the Code Snippet to Call *computeMetrics* function (top) and the Output from the function (bottom)

```
millingMain.jq
1  jsoniq version "3.0";
2  import module namespace ns= "http://repository.vsnet.gmu.edu/.../millingMachine.jq";
3  import module namespace dgal = "http://mason.gmu.edu/~mnachawa/dgal.jq";
4                          ...
5  let
6    $annotatedInputParams := $dat:annotatedPmInput,
7    $materialChars := $dat:materialCharactersitics,
8    $output := dgal:argmin({config:{noOfCycles:5},input:$annotatedInputParams,
9                 materialChars: $materialChars}, ns:computeMetrics#1,
10                "metricValues.cost.total",
11                { language: "ampl", solver: "minos" })
12   return $output
```

```
milling_PM_Out..
1  {
2    "id": "milling_PM",
3    "processType": "Unit Manufacturing Process",
4    "umpNameSpace": "http://repository.vsnet.gmu.edu/.../millingMachine.jq",
5    "InputParamsAndControls": {
6      "Material": {
7        "type": "Aluminum Alloys",
8        "dimentions": {"L":500,"W":60,"H":100}
9      },
10     "millType": "face",
11     "centered": "no",
12     "D": 150,
13     "V": {
14       "value": 199,
15       "lb":30,
16       "ub":200
17     },
18     "f_t": 0.381,
19     ...
20   }
```

Figure 5. Window Showing the Code Snippet to Call *argmin* function (top) and the Output from the function (bottom)

V. System Functionality

While the scope of the high-level system architecture, discussed in Section III is broad, we develop an initial system, with limited scope that captures partial functionality of the architecture. This functionality supports modelers or process engineers who are capable of working on an editor. The application layer of the functionality just supports the IDE called Atom [13], which is not designed for the application end users. From the model repository we implement the milling, drilling and machining UMPs that were discussed in Section IV. We implement analytical functions like compute, optimize, and trade-off analysis in the Unity DGMS. From the lower level tools, we use the MP/CP Optimization tools as well as the data manipulation and analytics tools.

The initial system is centered around the model & analysis repository, and the UMP performance models within them. This section shows the operations that can be performed on these models in the system. We describe these operations using the milling UMP performance model. It should be noted that these operations can be performed in a similar way on the drilling and the heat-sink (machining) UMPs.

Users can select UMPs from the repository by scanning through the taxonomy. For instance, we select the milling UMP by navigating in the taxonomy shown in Fig. 2A. After selection, we update or modify the milling UMP's inputs or transformation functions through Atom shown in Fig. 2B&C and then saving these models to the repository. In the current deployment, we first save files locally, then check them into the local Git repository, and lastly push changes to the remote GitLab repository, i.e., the model & analysis repository. The general architecture, however, allows for a repository artifacts to physically reside on any globally accessible server.

The operations of analytical tasks such as compute and optimize can be performed in the system. The computation of the metrics and constraints for the milling UMP performance model is performed by calling the transformation function *computeMetrics* (see Fig. 2C). This function call, shown in top pane of Fig. 4, is handled by Unity DGMS as a call to the *computeMetrics* function in JSONiq that returns back the metrics and constraints output as shown in the snippet on the bottom pane of Fig. 4.

The optimization problem in the milling UMP here is that of finding the control setting of cutting speed (decision variable) as to minimize the total cost (objective) within bounds on the cutting speed (feasibility constraint). To indicate cutting speed as the decision variable, it is annotated with a special key. To do this, the cutting speed (V, line 12 in Fig. 2B) is replaced with the following JSON object (see annotation on line 2).

```
1 "V": {
2         "value":{"float?": null},
3         "lb":30,
4         "ub":200
5 }
```

We will call the milling UMP input JSON with the replaced annotated cutting speed object above as *anno-*

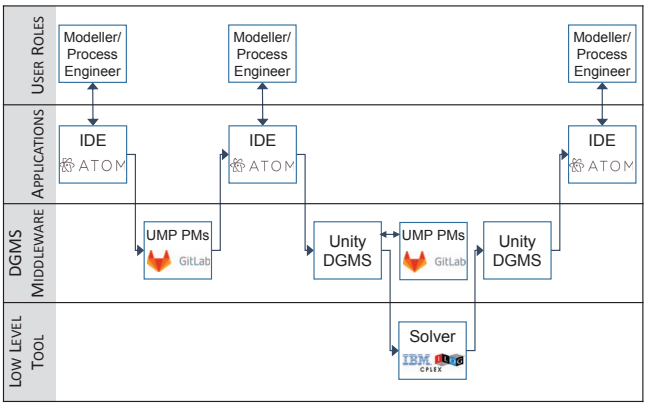

Figure 6. Optimization Workflow of Performance Models in the System

tatedInputParams. A code snippet to call Unity DGMS to perform optimization is shown in the top pane of Fig. 5. The optimization is performed by calling the *argmin* function (lines 8-11) that takes as input of *annotatedInputParams* along with other compute parameters, pointer to the milling transformation function *computeMetrics*, and the JSON path to the objective, which is the value of total cost in the output of *computeMetrics* (see line 7 in the bottom pane of Fig. 4). The output of *argmin* function is a fully instantiated *annotatedInputParams* with the cutting speed (control) set to the optimal value found by the optimization. This output is shown in the bottom pane of Fig. 5. This output can be used as input to the *computeMetrics* function to find the resulting metrics and constraints of the performance model.

VI. System Implementation

In this section, we explain the implementation and technologies used in the system. We do so through an example, the optimization problem for the milling UMP discussed in Section V. Fig. 6 shows the workflow followed within the system to solve such an optimization problem. A user such as a modeler or process engineer will first set up the problem on an IDE such as Atom [13]. We chose Atom since it is designed for maximum customizability and borrows an assortment of UI attributes, shortcuts, and functionality. On the IDE, the user will perform functions such as adding, updating or selecting (from the taxonomy) the input JSON, transformation functions as well as annotating the control settings in the input JSON and calling the *argmin* function to perform optimization. Once the problem is setup the user updates the data and functions into the UMP repository of reusable models that is stored and managed using GitLab [14], a code browser and review tool for the server that works in association with the version control system called Git [33]. This is done by first checking in the changes into Git and then then pushing the changes to GitLab.

The user may use an Atom plugin to call Unity DGMS to perform optimization. This is done by calling the DGMS via a REST API from Atom focused on the file with *argmin* function (top pane of Fig. 5). The DGMS will read all model input and transformation function files required for optimization from the repository in GitLab. Then, the DGMS performs automatic translation of the high-level representation of the performance models into low-level, specialized models required by the underlying tools. Since the optimization problem for the milling UMP example is a mixed integer linear programming problem, the DGMS performs translation as required by the IBM ILOG CPLEX optimization studio i.e., OPL [28]. The optimization results are returned to the DGMS, which then instantiates the control settings of the input JSON with optimal values (or with error codes for infeasible/unbounded). The user then gets the optimization results in Atom. If optimal control settings are found by the DGMS, the user may use them as actionable recommendations on the manufacturing floor.

VII. Case Study

To demonstrate the implementation of the middleware discussed in Fig. 1, we conduct a case study involving the optimization of machining control parameters associated with the part shown in Fig. 3. The purpose here is to exploit the constructed UMPs, i.e. machining, represented in JSONiq and perform tradeoff analysis between CO_2 emissions and cost of a part to simulate a real manufacturing scenario through the use of the system.

A. Assumptions and Limitations

All setup information, initial cutting parameters, and cutting constraints for the test part were procured from the technical data packages, available on the NIST SMS Testbed. The part is assumed to be cut on GF MIKRON HPM600U, a 5-axis simultaneous milling center. More information on the test part, including all CAD, CAM, and inspection data can be found on the NIST SMS Testbed website [34].

The main limitations here lie within the accuracy of the computed metrics (or objective functions). The models for evaluating the cost and sustainability of a given set of control parameters are simplified for the purpose of demonstration. Also, the individual UMP models used in this case study are assumed to be appropriate for modeling individual operations on a 5-axis milling center. In other words, we estimated cutting parameters such as depth of cut as well as power and time required for basic, non-productive phases of milling and drilling. We are aware that modeling the energy consumption of such a complex machine would most likely require historical data to properly estimate such missing parameters. We expect that in the future, these missing parameters would be learned through regression analysis within the DGMS.

B. Optimization Analysis & Key Takeaways

Fig. 7 displays pareto optimal alternatives for machining in terms of cost per part and emissions per part to allow

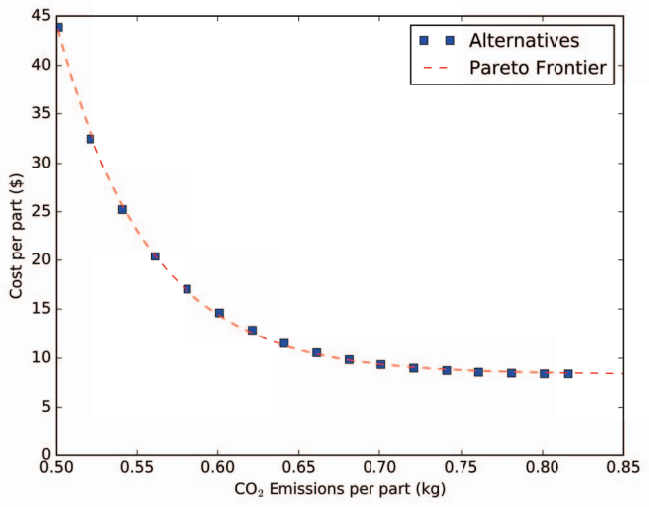

Figure 7. Pareto optimal curve, fitted by a logarithmic function, illustrating trade-off between cost and CO_2 emissions per part.

trade-off analysis. Each alternative is generated by solving an optimization problem that minimizes cost per part subject to the corresponding bound to CO_2 emissions. In this case, alternatives over about 0.75 kg CO_2 per part exhibit negligible differences in cost. Hence, users may choose the alternative closest to this threshold. In the future, we plan to incorporate an interface that will allow for trade-off analysis against traditional manufacturing KPIs, such as part quality and resource availability.

This case study demonstrated the core functionality of the presented system. We were able (1) to develop a reusable UMP model by formally representing process transformations and constraints in JSONiq, (2) to pose an optimization problem to Unity through Atom, (3) to retrieve a JSON output including design alternatives, and (4) to visualize the pareto frontier to aid in overcoming the decision scenario.

VIII. Conclusion

In this paper, we proposed the concept of a UMP repository, associated analytics engine to perform different kinds of analysis and optimization tasks, as well as a KB ecosystem for manufacturing users. The purpose of the ecosystem is to promote the rapid performance-related assessment of a manufacturing system through the reuse of existing manufacturing performance models. The focus of this paper was to extend our previous work [3] by introducing an initial system of the repository. The primary functions of system were demonstrated through a case study of a machining sequence of a heat-sink part. We believe that the case study demonstrates within the limited scope that performing optimization and analysis against performance models in the repository is possible without the need for developing hardwired, low-level models. Future work includes the development of the underlying architecture of the process model repository to enable development of domain-specific tools that can interface with the model descriptions. Additionally, we plan to formally capture each analysis task performed by DGMS through standard representations, such as the Predictive Model Markup Language (PMML) and the Portable Format for Analytics (PFA).

Disclaimer

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

Acknowledgment

This effort has been sponsored in part under the Cooperative Agreement No.70NANB12H277 between NIST and George Mason University. The work described was funded by the US Government and is not subject to copyright. We acknowledge KC Morris, Kevin Lyons, and James Barkeley for their valuable comments and suggestions. We also wish to acknowledge Kevin Li, a former NIST SURF student, for his help in completing the case study.

References

[1] SMLC, "Implementing 21 century smart manufacturing, workshop summary report," https://smartmanufacturingcoalition.org/sites/default/files/implementing_21st_century_smart_manufacturing_report_2011_0.pdf, 2011, accessed: June 2015.

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

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

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

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

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

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

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

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

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

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

[12] C. Gröger, F. Niedermann, H. Schwarz, and B. Mitschang, "Supporting manufacturing design by analytics, continuous collaborative process improvement enabled by the advanced manufacturing analytics platform," in *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*, 2012, pp. 793–799.

[13] "ATOM - a hackable text editor for the 21st century," https://atom.io, accessed: Sep. 2016.

[14] "Gitlab," https://gitlab.com, accessed: Sep. 2016.

[15] W. Z. Bernstein, M. Mani, K. W. Lyons, K. C. Morris, and B. Johansson, "An open web-based repository for capturing manufacturing process information," in *ASME 2016 IDETC/-CIE*. American Society of Mechanical Engineers, 2016.

[16] K. Kellens, W. Dewulf, M. Overcash, M. Z. Hauschild, and J. R. Duflou, "Methodology for systematic analysis and improvement of manufacturing unit process life cycle inventory (uplci) co2pe! initiative (cooperative effort on process emissions in manufacturing). part 2: case studies," *The International Journal of Life Cycle Assessment*, vol. 17, no. 2, pp. 242–251, 2012.

[17] ASTM International, *ASTM E2986-15, Standard Guide for Evaluation of Environmental Aspects of Sustainability of Manufacturing Processes*. ASTM International, 2015.

[18] ISO 20140, *Automation systems and integration - Environmental and energy efficiency evaluation method for manufacturing systems*. International Organization for Standardization, 2013.

[19] ASTM International, *ASTM E3012-16, Guide for Sustainability Characterization of Manufacturing Processes*. ASTM International, 2016.

[20] I. Meinshausen, P. Müller-Beilschmidt, and T. Viere, "The ecospold 2 formatwhy a new format?" *The International Journal of Life Cycle Assessment*, pp. 1–5, 2014.

[21] T. Hedberg, J. Lubell, L. Fischer, L. Maggiano, and A. B. Feeney, "Testing the digital thread in support of model-based manufacturing and inspection," *Journal of Computing and Information Science in Engineering*, vol. 16, no. 2, p. 021001, 2016.

[22] M. Maróti, T. Kecekés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendoszky, and Á. Lédeczi, "White paper: Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure," Vanderbilt University, Institute for Software Integrated Systems, Tech. Rep., 2014.

[23] A. Brodsky and X. S. Wang, "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*, 2008, pp. 71–71.

[24] J. Robie, G. Fourny, M. Brantner, D. Florescu, T. Westmann, and M. Zaharioudakis. (2015) Jsoniq the complete reference. [Online]. Available: http://www.jsoniq.org/docs/JSONiq/html-single/

[25] M. O. Nachawati, A. Brodsky, and J. Luo, "Building decision support systems from web-based marketplaces of reusable analytics models," Available at http://cs.gmu.edu/ tr-admin/papers/GMU-CS-TR-2016-4, Department of Computer Science, George Mason University, 4400 University Drive MSN 4A5, Fairfax, VA 22030-4444 USA, Tech. Rep., 2016.

[26] A. Brodsky and J. Luo, "Decision guidance analytics language (DGAL) - toward reusable knowledge base centric modeling," in *17th International Conference on Enterprise Information Systems (ICEIS)*, 2015, pp. 67–78.

[27] ——, "Decision guidance analytics language (DGAL) - toward reusable knowledge base centric modeling," Available at http://cs.gmu.edu/ tr-admin/papers/GMU-CS-TR-2014-6, Department of Computer Science, George Mason University, 4400 University Drive MSN 4A5, Fairfax, VA 22030-4444 USA, Tech. Rep., 2014.

[28] P. Van Hentenryck, L. Michel, L. Perron, and J.-C. Rgin, "Constraint programming in opl," in *Principles and Practice of Declarative Programming*, ser. Lecture Notes in Computer Science, G. Nadathur, Ed. Springer Berlin Heidelberg, 1999, vol. 1702, pp. 98–116.

[29] R. H. Todd, D. K. Allen, and L. Alting, *Manufacturing processes reference guide*. Industrial Press Inc., 1994.

[30] S. Kalpakjian and S. Schmid, "Manufacturing processes for engineering," 2014.

[31] F. Ameri and L. Patil, "Digital manufacturing market: a semantic web-based framework for agile supply chain deployment," *Journal of Intelligent Manufacturing*, vol. 23, no. 5, pp. 1817–1832, 2012.

[32] "Manufacturing unit process life-cycle inventory heuristics, uplci," http://cratel.wichita.edu/uplci/entry-template/, accessed: Sep. 2016.

[33] "Git on the Server - GitLab," https://git-scm.com/book/en/v2/Git-on-the-Server-GitLab, accessed: Sep. 2016.

[34] "Smart Manufacturing Systems (SMS) Test Bed," https://smstestbed.nist.gov, accessed: Aug. 2016.