J Intell Manuf (2017) 28:1421–1440
DOI 10.1007/s10845-015-1059-z

Developing a decision support system for improving sustainability performance of manufacturing processes

Seung-Jun Shin · Duck Bong Kim · Guodong Shao · Alexander Brodsky · David Lechevalier

Received: 16 July 2014 / Accepted: 23 February 2015 / Published online: 3 March 2015

Abstract It is difficult to formulate and solve optimization problems for sustainability performance in manufacturing. The main reasons for this are: (1) optimization problems are typically complex and involve manufacturing and sustainability aspects, (2) these problems require diversity of manufacturing data, (3) optimization modeling and solving tasks require specialized expertise and programming skills, (4) the use of a different optimization application requires re-modeling of optimization problems even for the same problem, and (5) these optimization models are not decomposed nor reusable. This paper presents the development of a decision support system (DSS) that enables manufacturers to formulate optimization problems at multiple manufacturing levels, to represent various manufacturing data, to create compatible and reusable models and to derive easily optimal solutions for improving sustainability performance. We have implemented a DSS prototype system and applied this system to two case studies. The case studies demonstrate how to allocate resources at the production level and how to select process parameters at the unit-process level to achieve minimal energy consumption. The research of this paper will help reduce time and effort for enhancing sustainability performance without heavily relying on optimization expertise.

S.-J. Shin · D. B. Kim (✉) · G. Shao · D. Lechevalier
Systems Integration Division, Engineering Laboratory,
National Institute of Standards and Technology,
100 Bureau Drive, MS 8260, Gaithersburg,
MD 20899-8260, USA
e-mail: kim.duckbong@nist.gov

A. Brodsky
Department of Computer Science, George Mason University,
4400 University Drive, MS 4A5, Fairfax, VA 22030-4444, USA

Keywords Sustainable manufacturing · Process optimization · Decision supporting system · Sustainable process analytics formalism · Energy consumption

Introduction

As environmental concerns such as climate change, energy security, and scarcity of resources have grown, manufacturing industries have shown more interest in sustainable manufacturing (OECD 2009). Sustainability performance is becoming a major indicator in manufacturing industries for improving energy efficiency and reducing environmental burdens. In response to these industrial needs, there has been significant research in optimization, which refers to finding feasible solutions to obtain extrema for the values of important objectives (Deb 2001).

Diaz and Dornfeld (2012) implemented machine tool scheduling by using a discrete-event simulation for optimizing cost and environmental impact in a flexible manufacturing system. Diaz et al. (2010) implemented green machining strategy by optimizing process parameters. Fang et al. (2013) solved flow shop scheduling with peak power constraint using mathematical programming and a combinational approach. Winter et al. (2013) developed multi-objective optimization models that identified process parameters in grinding operations. Aggarwal et al. (2008), Hanafi et al. (2012), Bhushan (2013), and Kuram et al. (2013) studied the selection of cutting parameters for minimizing power consumption with consideration of surface roughness and tool wear. Li et al. (2013a) suggested a comprehensive model to analyze carbon emission of a machining process quantitatively to achieve energy-driven process optimization. Li et al. (2013b) found optimal cutting parameters that minimize machining time, energy consumption, and surface roughness in sculptured part machining. Jia et al. (2014) proposed the optimization method that applies the Therblig-based energy model through mapping of machining activities with energy attributes. Lei et al. (2014) integrated a traditional process planning approach with the green process concept that considers several sustainability indicators such as tool wear, energy consumption, fluid consumption, fluid toxicity, and equipment's noise. Their efforts have contributed to the success of utilizing optimization approaches to advance sustainability performance objectives. However, the application of these optimization approaches remains a major challenge for manufacturers, especially small and medium-sized manufacturers, due to the complexity and expertise required for formulating and solving optimization problems.

Therefore, there is a need to easily formulate optimization models (i.e., a set of representation models including an optimization problem, its associated manufacturing processes and data, and optimization strategy) and to obtain optimal solutions by using a Decision Support System (DSS), which is "a computer-based program application that helps users incorporate business and organizational decision making activities (Vinodh et al. 2014)". Manufacturers can use commercial DSSs, i.e., existing optimization applications, for this purpose. It is more efficient to use a DSS that enables manufacturers to formulate their optimization problems easily and intuitively and, in turn, to obtain optimal solutions for sustainability performance of manufacturing processes. However, there are no such DSSs in the domain of sustainable manufacturing, which are capable to resolve the following challenges.

First, optimization models for sustainable manufacturing generally need to deal with problems at multiple manufacturing levels such as the production level and/or the unit-process level (i.e., a process element of a production process). For example, there are optimization problems for minimal energy consumption by resource allocation at the production level and by process parameter selection at the unit-process level. Moreover, these optimization models are typically complex and involve many constraints from both productivity and sustainability aspects such as production throughput and energy consumption. Second, the diversity of manufacturing data and the complex relationships among those data make it difficult to formulate optimization problems. The mathematical relationships need to be represented correctly and also solvable. It is not easy to represent manufacturing data necessary for formulating the problems due to various characteristics of the data in terms of time domain, format and data source (e.g., time-series or discrete-event, structured or unstructured, and dynamic or static). For example, process plan data have a static characteristic that is independent of time domain while machine monitoring data are dynamic, time-sensitive, and related to a machine tool's actions. These two heterogeneous data may cause data processing and matching problems. Third, formulating and solving an optimization problem generally need specialized expertise and skill, e.g., Mathematical Programming (MP), Constraint Programming (CP), Genetic Algorithms or Simulated Annealing. This work also requires fundamental knowledge of existing software and the underlying algorithm principles (Biegler 2010). However, manufacturers may not have this expertise and skill. Fourth, manufacturers normally design their incompatible optimization models, i.e., application-specific models. This means that the use of a different optimization application will require re-modeling even for the same problem. Fifth, manufacturers have to create their optimization models again and again unless these models or model components can be reused.

To address these challenges, this paper presents the development of a DSS that enables manufacturers to: (1) formulate optimization problems at multiple manufacturing levels, (2) represent manufacturing data and performance metrics, (3) create compatible and reusable models, and (4) easily derive optimization solutions for improving sustainability performance. More specifically, the contributions of this paper are as follows. First, we propose a decision-making procedure that can be applied to manufacturing practices and design considerations to map this procedure to system implementation. Second, we propose a system architecture for the behavior (activities in a logical progression), the function (functional components for the activities), and the data (data components for representing an optimization model and solution). Third, we implement a prototype of the DSS based on the designed architecture. Finally, we apply the prototype to two case studies to demonstrate the effectiveness of the DSS. The two case studies include a resource allocation problem at the production level and a parameter selection problem at the unit-process level. For representing optimization models and solutions, we adopt the Sustainable Process Analytics Formalism (SPAF) that was developed at National Institute of Standards and Technology (NIST), which "allows formal modeling of modular, extensible and reusable process components and enables the optimization of sustainability performance based on mathematical programming (Brodsky et al. 2014)". The key functions of the DSS are to: (1) provide an efficient procedure to derive optimal solutions for sustainability performance, and (2) formulate compatible and reusable optimization models for various optimization applications.

"Related work" section of this paper reviews the literature relevant to DSSs in manufacturing and the data structure of SPAF. "A decision-making procedure for improving sustainability performance" section describes the decision-making procedure for improving sustainability performance. "Design consideration" section presents design considerations for architecting the DSS, called Sustainability Performance Optimization Tool (SPOT) and “Architecture design” section introduces the architecture design of SPOT. “Implementation” section describes the prototype implementation of SPOT and “Case study” section describes the two case studies and discussion. “Conclusion” section provides a summary and conclusions of our work.

Related work

This section reviews DSSs within manufacturing domain and introduces the data structure of SPAF.

Decision support systems in manufacturing

A wide range use of DSSs has served to the improvement of productivity performance in production line design, process and machine tool selection, outsourcing determination, condition-based maintenance and so on. For examples, Vitanov et al. (2007) presented a DSS that obtained a quasi-optimal solution for cell line formation. Battaia et al. (2012) developed a DSS that assisted finding the configuration of a production line. Yurdakul et al. (2014) developed a DSS that provided the most economical net-shape primary process. Giachetti (1998) developed a DSS to solve a material and manufacturing process selection problem. Arslan et al. (2004), Taha and Rostam (2012) and Alberti et al. (2011) developed their DSSs for machine tool selection by using multi-criteria weighted average, fuzzy analytic hierarchy process, and machine characteristics and performance tests, respectively. Leng et al. (2014) proposed a DSS for parts machining outsourcing. Yam et al. (2001) implemented a DSS that supplemented conventional condition-based maintenance by adding fault diagnosis and predicting equipment deterioration. Their DSSs have shown to assist manufacturers to improve productivity performances across enterprise and manufacturing process levels; however, they have not related to the improvement of sustainability performance.

Recently, a few references such as Le and Pang 2013, Zhu et al. 2014, Vinodh et al. 2014 and Iqbal et al. 2014 have introduced DSSs relevant to support decision-making on sustainability performance in manufacturing. Their efforts have demonstrated the usefulness of the relevant DSSs for improving sustainability performance. However, these references are different in the scope of this paper. Le and Pang (2013) proposed a unified DSS architecture that segmented power data and clustered the power into groups according to the process operational states. This reference focused on energy- and cost- efficiency and decision reliability, but it requires expertise of operational research for generating energy-based diagnosis and prognosis models. Zhu et al. (2014) developed a model-predictive Linear Programming (LP) optimizer to minimize energy lost with application to energy management in production environments; however, this reference only utilized the optimization technique for the specific problem. Vinodh et al. (2014) implemented a DSS that performed fuzzy logic-based sustainability evaluation for measuring the sustainability level of a manufacturing organization and did not much consider the compatibility and reusability of optimization models. Iqbal et al. (2014) developed a fuzzy-based DSS that consists of optimization and prediction for trade-off among energy consumption, tool life, and material removal rate.

Data structure of sustainable process analytics formalism (SPAF)

SPAF has been designed to allow formal modeling of modular, extensible and reusable process and data components in a unified modeling environment (Brodsky et al. 2014). For such a purpose, SPAF provides the hierarchical data structure to represent process flow, data and mathematical specification of metrics for sustainability-related optimization problems (Shao et al. 2014). As shown in Fig. 1, ‘context’ presents the model name and context data and is globally accessed by all the components. ‘Flow’ describes entities of inputs and outputs of a unit-process. ‘Flow aggregator’ presents merges or splits of the inputs and outputs of a unit-process. ‘Process’ presents a unit-process that may consist of a single process or its subordinate processes. ‘Metrics aggregator’ provides a set of sustainability metrics.

A decision-making procedure for improving sustainability performance

A decision-making procedure, i.e., an optimization procedure, is necessary to formulate an optimization problem and to solve the optimization problem. When a well-structured approach is taken, the complexity of an optimization procedure can be reduced. This section introduces a decision-making procedure to improve sustainability performance based on the interaction with SPOT.

An overall procedure

An optimization procedure typically consists of problem formulation and problem encoding (Dimopoulos and Zalzala 2000). The problem formulation step specifies the problem and system boundary or constraints; meanwhile the problem encoding step determines an optimal solution through process modeling, data collection, optimization solving and validation. In our approach (as shown in Fig. 2), these steps can be taken by the interaction with SPOT through its user interfaces. SPAF makes it possible to represent optimization model components in a modular and compatible way.

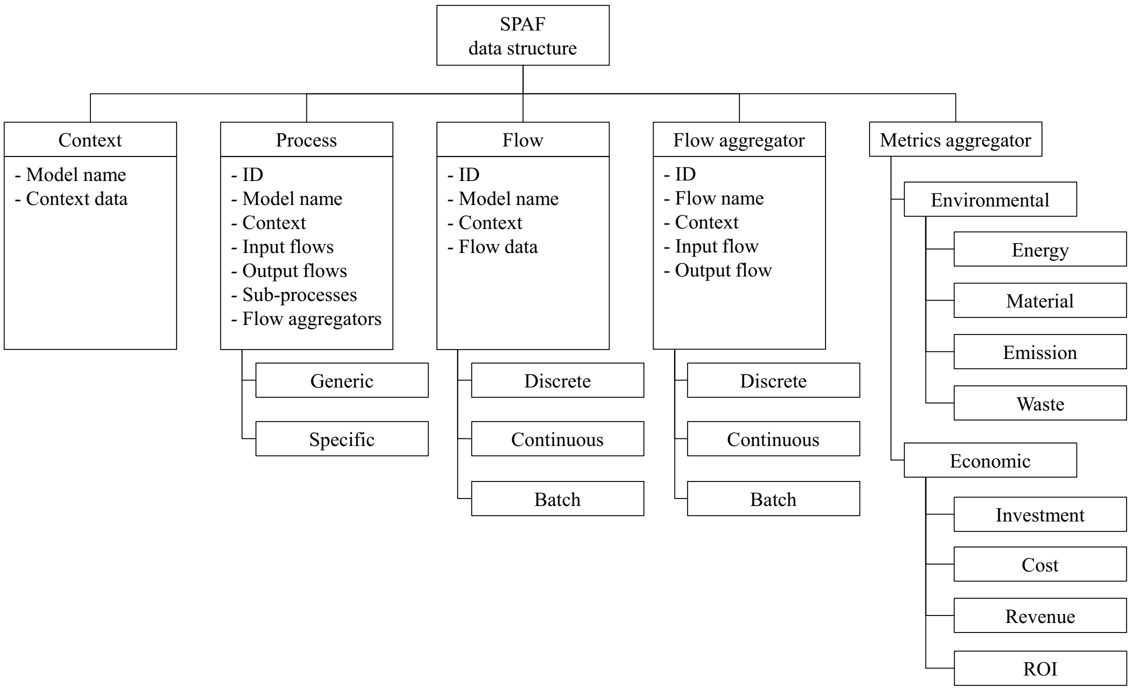


Fig. 1 A data structure of SPAF (Brodsky et al. 2014)

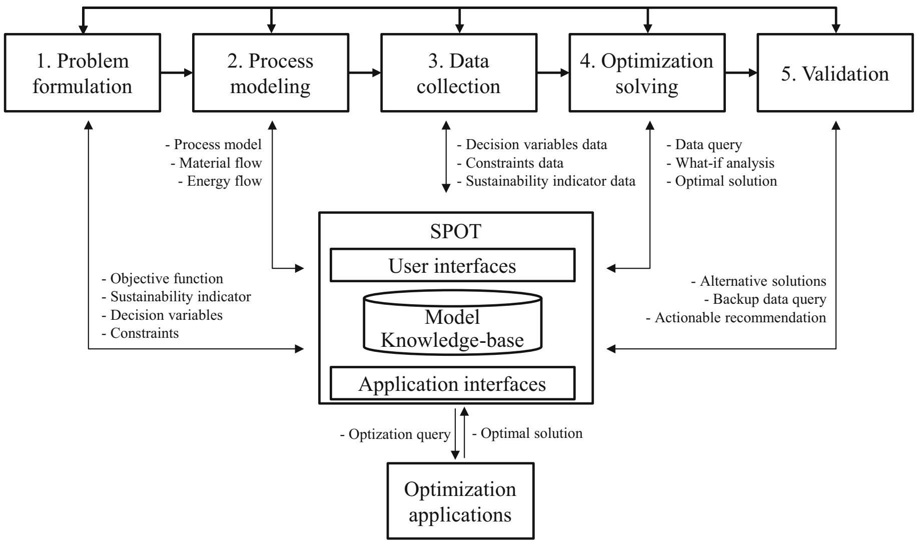


Fig. 2 A schematic optimization procedure for improving sustainability performance

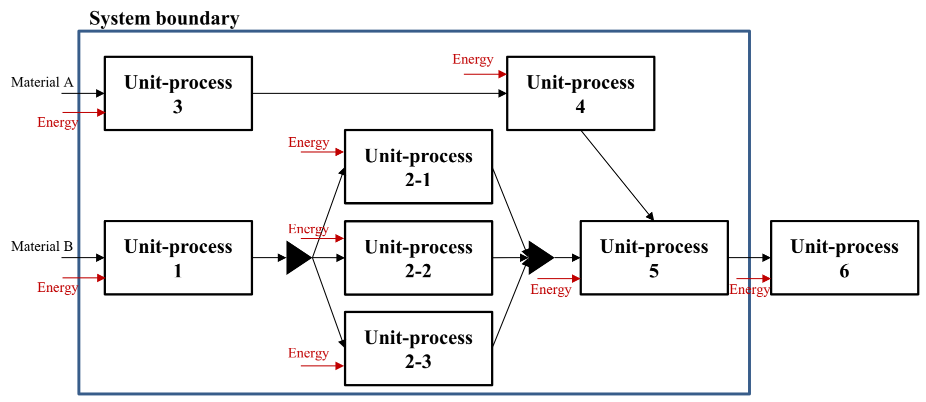
The model knowledge-base stores and retrieves these model components for their reuse. SPOT interacts with optimization applications that can derive the optimal solutions for the given problem.

Figure 2 presents a schematic optimization procedure for manufacturing domain. Each step can concurrently progress, take an iterative way with others and even provide feedback from the last step to the first one. In the following subsections, we describe the details of these steps and their corresponding examples at the production and the unit-process levels for easy understanding.

Step 1: problem formulation

When manufacturers consider the improvement of sustainability performance, they normally conceptualize an engineering problem based on their requirements. Because this engineering problem forms informal and conceptual statement at the initial stage, it needs to be transformed to formal and mathematical representation for formulating an optimization problem. As ‘problem formulation’ associates with this work, it specifies the problem and system boundary or constraints. The current problem model may not be fully for-

Fig. 3 An example of a production process model and system boundary



mulated because its manufacturing process model and data are not instantiated yet. We need to revisit this step after the completion of each subsequent step to check for consistency of the optimization model. This step outputs the definition of the following elements (Deb 2001):

- Objective function: a scalar quantitative performance measure that needs to be minimized or maximized
- Boundary or constraints: a feasible region that defines limits of performance
- Decision variables: multiple parameters adjusted to satisfy the constraints

Production level: when a production engineer needs to minimize the total energy consumption by allocating resources appropriately at the production process level, the engineer can set: 1) an objective function to minimize energy consumption, 2) constraints to minimum and maximum throughputs at each unit-process of the production process, and 3) a decision variable to the number of throughput at a unit-process, as expressed in Eq. 1.

$$\begin{aligned}&\mathit{Minimize}\quad E(x_1,\ \ldots,\ x_n),\\&\mathit{subject\ to}\quad \begin{cases}E(x)=\sum_{i=1}^{n}E_i(x_i),\\ x_i^{(L)}\le x_i\le x_i^{(U)}\end{cases}\end{aligned}\tag{1}$$

where, x: the number of throughput, $E_i(x_i)$: energy consumption at each unit-process, n: the number of unit-processes, $x^{(L)}$: minimum throughput, and $x^{(U)}$: maximum throughput.

Unit-process level: in the case of a machining process, when a process engineer needs to minimize the average power demand (the average electrical power consumed during machining time) by the best selection of process parameters, the engineer can set an objective function to minimize power demand, constraints to the ranges of the process parameters and decision variables to the process parameters. For example, the power demand equation assumes to be a quadratic linear regression model, as expressed in Eq. 2.

$$\begin{aligned}&\mathit{Minimize}\quad P(x_1,\ \ldots,\ x_n)\\&\mathit{subject\ to}\quad \begin{cases}P(x)=\gamma_0+\sum_{i=1}^{n}\gamma_i x_i+\sum_{i\le j}^{n}\gamma_{ij}x_i x_j+\varepsilon,\\ x_i^{(L)}\le x_i\le x_i^{(U)}\end{cases}\end{aligned}\tag{2}$$

where: x: process parameter, $P(x)$: the average power demand at a machining process, n: the number of process parameters, $x^{(L)}$: the minimum value of a process parameter, $x^{(U)}$: the maximum value of a process parameter, γ: coefficient, and ε: error.

Step 2: process modeling

'Process modeling' identifies the composition of a manufacturing process. This step enables manufacturers to specify a system boundary and to configure the flows of material and energy and the relationship of the data to be collected.

Production level: a production engineer can specify a set of associated unit-processes in a system boundary, as shown in Fig. 3. A selective unit-process such as 'unit-process 2' can use a separator and an aggregator, as represented by the triangle marks in Fig. 3. In some cases, certain unit-processes or material and energy flows may be irrelevant and then excluded from the system boundary. This process modeling assigns the functions $E_i(x_i)$ of Eq. 1 necessary for the $E_i(x_i)$ specified by the next step, which is 'data collection'. In this example, the numbers of throughput in 'unit-process 2-1, 2-2 and 2-3' can be set as decision variables.

Unit-process level: a process engineer specifies a set of sub-processes for a machining unit-process, as shown in Fig. 4. This unit-process model can be regarded as a "zoomed-in" model of 'unit-process 2' in Fig. 3. The input and output of this unit-process model only includes the electrical power and process parameters (see the bold and italic texts in Fig. 4) and excludes material-relevant flows for this example.

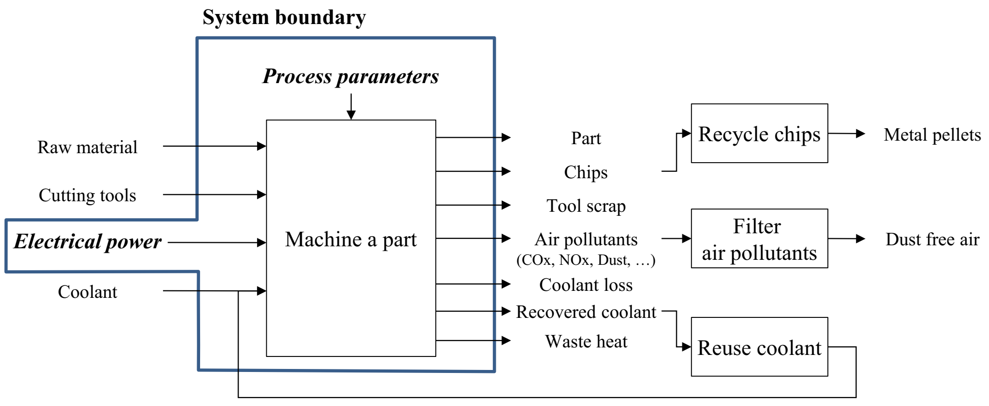


Fig. 4 An example of a machining unit-process model and system boundary

Step 3: data collection

'Data collection' performs collection of the manufacturing data relevant to objective functions, constraints and decision variables. The manufacturing data can be classified into dynamic and static data according to their source. As shop floor data correspond to dynamic data created or used during manufacturing planning and operation, the data can include production/process plan, machine monitoring, inspection and environment information. Inventory data are the static and referential data supplied by external sources and used when shop floor data are not available. The focus of this step is not about data logging from equipment or data collecting from sensors, instead, it is mainly about the insight of data characteristics and relationships among different data, and the identification of data sources.

Production level: the functions $E_i(x_i)$ and the bounds $x_i^{(L)}$ and $x_i^{(U)}$ of Eq. 1 need to be specified by the collection of useful real shop floor data. For example, assuming energy consumption at each unit-process has a piecewise linear dependency on the throughput of the unit-process, associating coefficients are found by the result of this data collection. Equation 3 represents the piecewise linear formula to calculate energy consumption at each unit-process. Then, $E(x)$ of Eq. 1 can be instantiated because the $E(x)$ is the sum of $E_i(x_i)$.

$$E_i(x_i) = \begin{cases} \alpha_1 x_i + \beta_1, \ if \ 0 \le x < x_1, \\ \ldots \\ \alpha_n x_i + \beta_n, \ if \ x_{n-1} \le x \le x_n \end{cases} \quad (3)$$

where, α: gradient coefficient, and β: Y intercept

Unit-process level: the bounds $x_i^{(L)}$ and $x_i^{(U)}$ and the function $P(x)$ of Eq. 2 can be instantiated by the collection of process plan data including a set of process parameters and machine monitoring data including time-series electrical power data. The lower and upper bounds are specified, depending on the combination of a workpiece material, a cutting tool and a machine tool. The $P(x)$ can be obtained through an empirical approach, which conducts experiment on the process plan data and measures their corresponding machine monitoring data.

Step 4: optimization solving

'Optimization solving' applies optimization strategies, methods, and techniques to problem solving. Optimization problems can be classified in terms of the nature of decision variables, equations, the number of objective functions and so on (Rao 2009). For example, an optimization problem can be classified as either Mixed Integer Linear Programming (MILP) or Non-Linear Programming (NLP) in terms of continuous or discrete decision variables (Biegler 2010). However, manufacturers who are not familiar with these optimization techniques may have difficulty of the selection of an appropriate optimization technique and the derivation of optimal solutions. To perform easily this difficult work, SPOT communicates with optimization applications, based on the SPAF models that can be generated by the manufacturers.

Production level: because the functions $E_i(x_i)$ of Eq. 3 form non-convex functions, this problem is a MILP problem. The problem given to the production level, for example, can be solved by a branch and bound technique, which is the most widely-used for solving MILP problems (Clausen 1999). A production engineer can obtain the optimal solution that includes the minimum energy consumption and the number of throughput at each unit-process.

Unit-process level: for example, the problem given to the unit-process level can be solved by a CP technique that solves optimization problems through the representation of the problems and the use of tree search algorithms (Zeballos 2010). A process engineer can obtain the optimal solution that includes the minimum power demand and its corresponding process parameters.

Step 5: validation

'Validation' decides the application of problem solutions and ensures reliability caused by assumptions for the solutions and lack of data and knowledge. Manufacturers should review whether derived solutions are applicable for their production process and unit-processes. For example, the optimal solution derived from an optimization application may not be applicable due to additional constraints or unexpected disturbances that have not been considered in the previous steps. In this case, manufacturers should find alternative solutions that are applicable or redo this optimization procedure with consideration of all the possible constraints and disturbances. The optimization results can also be validated by using historical data or other analysis applications such as simulation systems.

Design consideration

In designing the architecture of SPOT, we should take into account of the design factors required to resolve the five challenges described in "Introduction" section and the optimization procedure presented in "A decision-making procedure for improving sustainability performance" section. We also need to transform these design factors to the requirement specification for tangible development of SPOT. "Design factor" section presents the design factors and "Requirement specification" section describes the requirement specification.

Design factor

We consider the design factors that emphasize *domain-specificity*, *diversity*, *compatibility*, *simplicity* and *reusability* for SPOT architecture development. The meaning of each design factor is explained below:

- *Domain-specificity* an optimization procedure demands abstraction of a problem and its reification to obtain optimal solutions with the use of a DSS. However, reifying the abstractive problem in the DSS is not easy because it requires specific and actual data properties and their corresponding values. Thus, a new DSS should be designed to include the data properties specialized to sustainability performance in manufacturing. The data properties involve manufacturing processes, materials and sustainability indicators.
- *Diversity* it is important to represent manufacturing data without any loss or distortion because the quality of the data influences the quality of optimal solutions. Moreover, when real-time manufacturing data are not available, data from other sources need to be considered, for example, inventory data provided by consortiums such as CO2PE! (Kellens et al. 2012) and Eco-Invent (Frischknecht et al. 2007). A new DSS should support the representation of the manufacturing data as well as the inventory data, i.e., the DSS allows combination of diverse manufacturing data.
- *Simplicity* a DSS should enable manufacturers to formulate and solve an optimization problem easily without intensive programming burden. For this purpose, we separate functions of the DSS into a user interface for optimization modeling and a solver for optimization solving. In addition, a DSS can provide functions that are especially convenient for the process modeling or the data collection steps. A Graphical User Interface (GUI) will support the easy modeling of a manufacturing process.
- *Compatibility* a new DSS that provides an application-neutral modeling environment is needed to enable interoperable model and data exchange. This is one of the reasons we adopt the data structure of SPAF as a representation model because it enables to model a manufacturing process once and use this same model for various optimization applications.
- *Reusability* a DSS should be designed to adopt a modular approach, which partitions the modeling along a logical boundary into a collection of smaller, semi-independent but interrelated model elements. The SPAF approach supports the reuse of existing elements to create a new optimization model. If we have identified and decomposed a set of unit-process, we can reuse these unit-processes in another model. For example, the process element for 'unit-process 2-1' can be reused to model 'unit-processes 2-2 and 2-3', as shown in Fig. 3.

Requirement specification

Figure 5 presents the use case diagram that visualizes the requirement specification transformed from the optimization procedure and the design factors. A user can be a production engineer, a process engineer, an equipment engineer, a facility engineer, an analyzer and an operational manager. Each use case, represented by an ellipse, defines the user's interaction with an in-boundary system. 'Include' implies that the behavior of the included use case is inserted into the behavior of the including use case. 'Extend' specifies that the behavior of a use case may be extended by the behavior of another supplementary use case (OMG 2007).

A user interacts with SPOT by providing inputs and deriving outputs of the first four steps shown in Fig. 2. The current DSS excludes the requirements related to the 'validation' step because this step mainly depends on human knowledge and experience. Each of the use cases also incorporates the five design factors of the previous sub-section. For example,

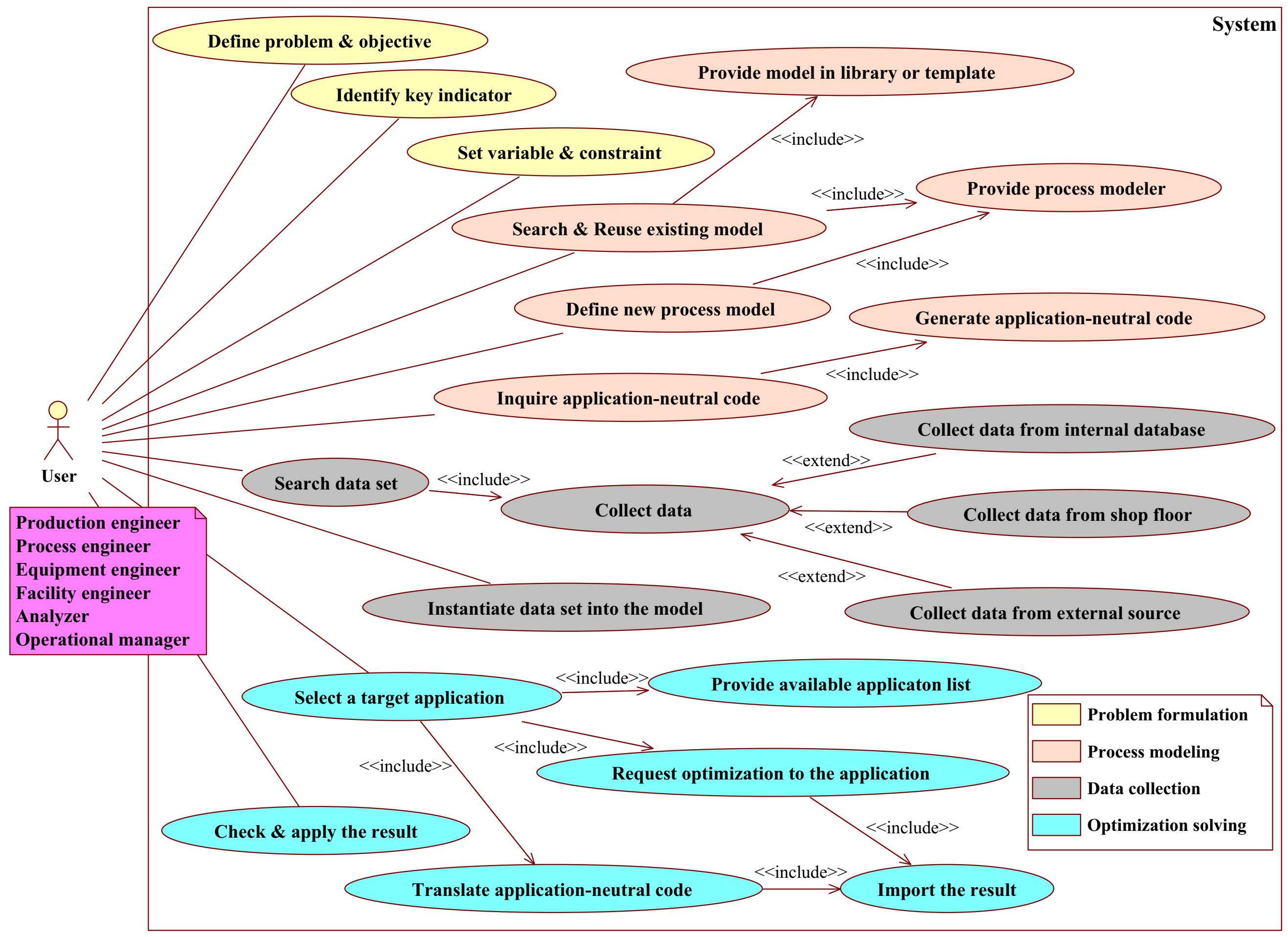


Fig. 5 Use cases of requirement specification for designing SPOT

'search & reuse existing model' is the use case that specifies *reusability* for the reuse of existing model elements. 'Collect data' and its extended cases specify *diversity* for the availability of various data sources. 'Inquire an application-neutral code' and 'generate an application-neutral code' support *compatibility* by generating an application-neutral format.

Architecture design

Based on the requirement specification presented in "Requirement specification" section, we design the architecture that describes the behavior (how activities take place in a logical progression), the function (which functions are responsible for the activities) and the data (which data are necessary for representing an optimization model and solution). The following sub-sections present the behavior, the function, and the data structures, respectively. The design of these structures adopts Unified Modeling Language (UML) to facilitate mutual understanding and commonality.

Behavior structure

Because an activity diagram can specify the execution and command of subordinate behaviors, it is commonly used to identify the behaviors of the system (OMG 2007). Figure 6 visualizes the activity diagram to present the behavior structure that transforms the use cases in Fig. 5 to procedural activities.

The stream on the left includes a set of activities that formulate an optimization problem. In the middle stream, the activities of 'process modeling' support the identification of a set of unit-processes and their relationships. The activities of 'data collection' provide separate channels to collect data from internal and external data sources. The activities of the right stream include the activities for the 'optimization solving' step to generate an application-neutral code that includes process structure, resource flow, data, control parameters, metrics and constraints. There are two data exchange options to request and return optimal solutions: (1) file transaction, and (2) direct data transaction through Application Programming Interfaces (API).

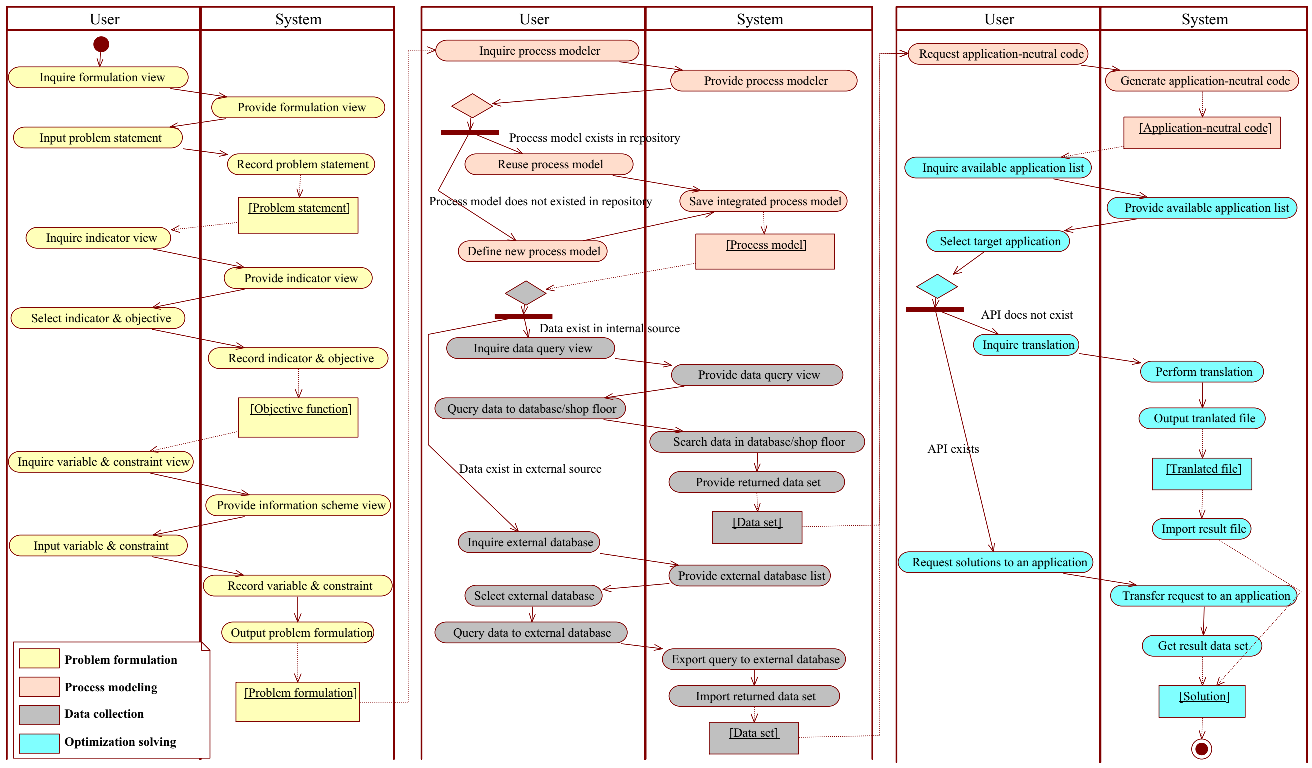

User
System
Inquire formulation view
Provide formulation view
Input problem statement
Record problem statement
[Problem statement]
Inquire indicator view
Provide indicator view
Select indicator & objective
Record indicator & objective
[Objective function]
Inquire variable & constraint view
Provide information scheme view
Input variable & constraint
Record variable & constraint
Output problem formulation
[Problem formulation]
Problem formulation
Process modeling
Data collection
Optimization solving
User
System
Inquire process modeler
Provide process modeler
Process model exists in repository
Reuse process model
Process model does not existed in repository
Save integrated process model
[Process model]
Define new process model
Data exist in internal source
Inquire data query view
Provide data query view
Query data to database/shop floor
Search data in database/shop floor
Data exist in external source
Provide returned data set
[Data set]
Inquire external database
Provide external database list
Select external database
Query data to external database
Export query to external database
Import returned data set
[Data set]
User
System
Request application-neutral code
Generate application-neutral code
[Application-neutral code]
Inquire available application list
Provide available application list
Select target application
API does not exist
Inquire translation
Perform translation
Output tranlated file
[Tranlated file]
API exists
Import result file
Request solutions to an application
Transfer request to an application
Get result data set
[Solution]

Fig. 6 An activity diagram for the behavior design of SPOT

Function structure

Because the activity diagram does not state functions responsible for the behaviors, a component diagram specifies a set of components that take the responsibility (OMG 2007). Figure 7 shows the component diagram that defines the structure and interaction of the functions. The functions are categorized into three packages: workflow engine, user interface, and data interface.

‘Workflow engine’ controls all the activities and interconnections with ‘user interface’, ‘data interface’, and external applications. A user can give inputs and take outputs through viewers provided by ‘user interface’. ‘Data interface’ supplies data through different sub-interfaces: an internal, a shop floor, and an external interface. Such interfaces are designed to implement the *diversity* design factor. The internal interface accesses the data stored within the user’s database. The shop floor interface directly connects with machine interfaces and data acquisition systems. The external interface accesses to external sources. Inventories such as CO2PE! (Kellens et al. 2012) and Eco-invent (Frischknecht et al. 2007) show the potential of interconnection with external sources.

‘Workflow engine’ generates an application-neutral code and translates it to an interoperable code for optimization applications. This realization of the *compatibility* design factor allows one to increase accessibility to various optimization applications and reduce the burden of implementing complex and various optimization techniques. ‘Workflow engine’ can import and parse an output code that encodes optimal solutions. On the other hand, because some optimization applications provide a set of API coded using some programming languages, ‘workflow engine’ can alternatively make use of an application-supplied API to exchange data and results. The following items describe each module shown in ‘workflow engine’ as well as some of the components shown in ‘user interface’ and ‘data interface’.

- Problem formulator: the ‘problem manager’ function oversees the formulation of user-defined problems. It connects with other managers to control proper activities and govern a given problem. A user considers a goal and an initial boundary of a production process through ‘goal and scope definer’. The process boundary will later be specified when the scope and data availability are determined. The user specifies sustainability indicators and an objective function through ‘indicator and function definer’. ‘Variable and constraint definer’ identifies decision variables and constraint variables.
- Process modeler: ‘process modeling manager’ controls the activities of the ‘process modeling’ step. The user defines a production process and its data flow through ‘process model viewer’. This function, together with ‘process model finder’, helps the user to search reusable existing process model components. ‘Process model repository’

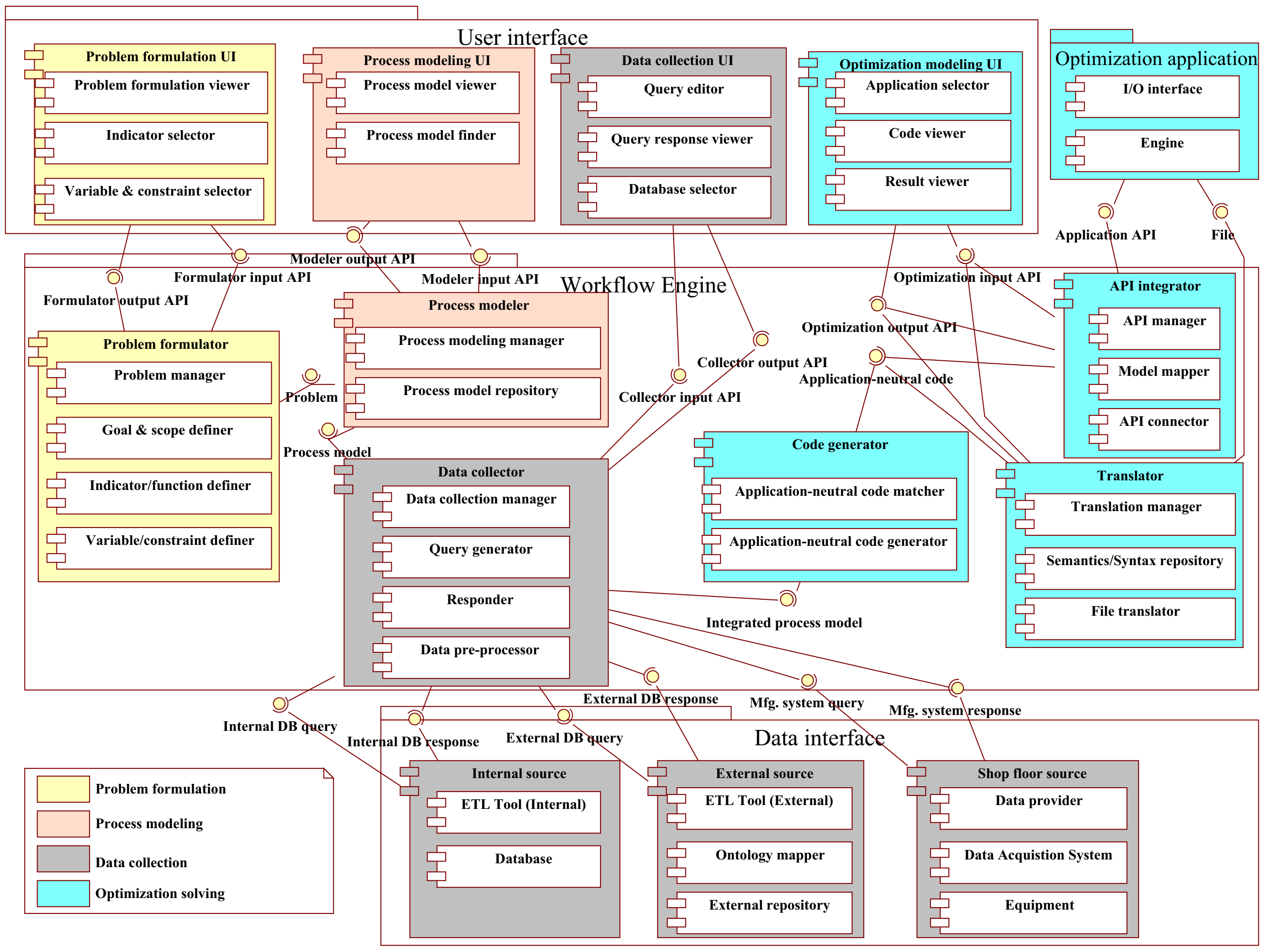


Fig. 7 A component diagram for the function design of SPOT

stores and retrieves process model components registered by the user's former works.

- Data collector: 'data collection manager' controls activities related to the 'data collection' step and configures data connectivity with targeted data sources. After establishing data connectivity, a query-and-response transaction can occur. 'Query generator' reformulates the syntax of user's query as required. It then delivers the reformulated query to an Extract/Transform/Load (ETL) tool installed in the database. The ETL tool delivers the data set that satisfies the query condition to 'responder'. A data set from an external source cannot be used directly because of heterogeneous semantics and syntaxes. 'Ontology mapper' in the 'data interface' resolves this heterogeneity. Correspondences between semantically related entities of the local information sources and the virtual ontology are to be established, known as a matching. The query answering is performed by this matching (Euzenat and Shvaiko 2007). 'Data pre-processor' handles missing and error data to leverage data conformance and reliability. This function can perform data cleaning and data validation. The data cleaning uses reasoning rules to omit unnecessary, erroneous or duplicated data. The data validation checks and quantifies data reliability and consistency. This 'data pre-processor' plays a role of handling data uncertainty (will be discussed in "Discussion" section).
- Code generator: 'application-neutral code matcher' converts the integrated process model (the process model instantiating relevant data) and the problem formulation into instances structured by a data schema (which will be described in "Data structure" section). This function can transform graphical notation to textual code. For example, if a user attaches a separator that decomposes a production flow into several flows, the graphical separator is transformed to a textual separator. 'Application-neutral code generator' creates an application-neutral code in an SPAF form from the data instances.
- API integrator: 'API manager' provides connectivity with the application through API-based data transaction. When a user selects a target application through 'application

selector', 'API manager' assigns the application and invokes 'model mapper'. 'Model mapper' maps the data instances of 'application-neutral code matcher' with a data scheme dedicated to the application. For instance, if a user selects ILOG CPLEX (IBM 2012), the data instances are mapped to the data schema available in ILOG CPLEX. 'API connector' provides a channel between the workflow engine and an optimization application. This connector delivers an optimization model to an optimization application and obtains the solutions directly from the optimization application.

- Translator: 'translation manager' supports file-based data transaction with an optimization application. It also assigns the application and invokes 'file translator' that converts the SPAF file to interpretable files for the target application. 'Semantics and syntax repository' registers and stores the rules of semantics and syntaxes for optimization applications. This repository provides rules that enable conversion of an application-neutral format to an application-specific format. For example, if a user selects ILOG CPLEX, 'translation manager' opens a channel to 'Optimization Programming Language (OPL) file translator'. Then, this file translator generates an OPL file, which can be executed by ILOG CPLEX.

Data structure

Because the behavior structure and the function structure do not cover data representation, a class diagram is created to specify the structure of data objects, their attributes and the relationships among the objects. As shown in Fig. 8, the classes are also clustered into four groups: problem formulation, process modeling, data collection and optimization

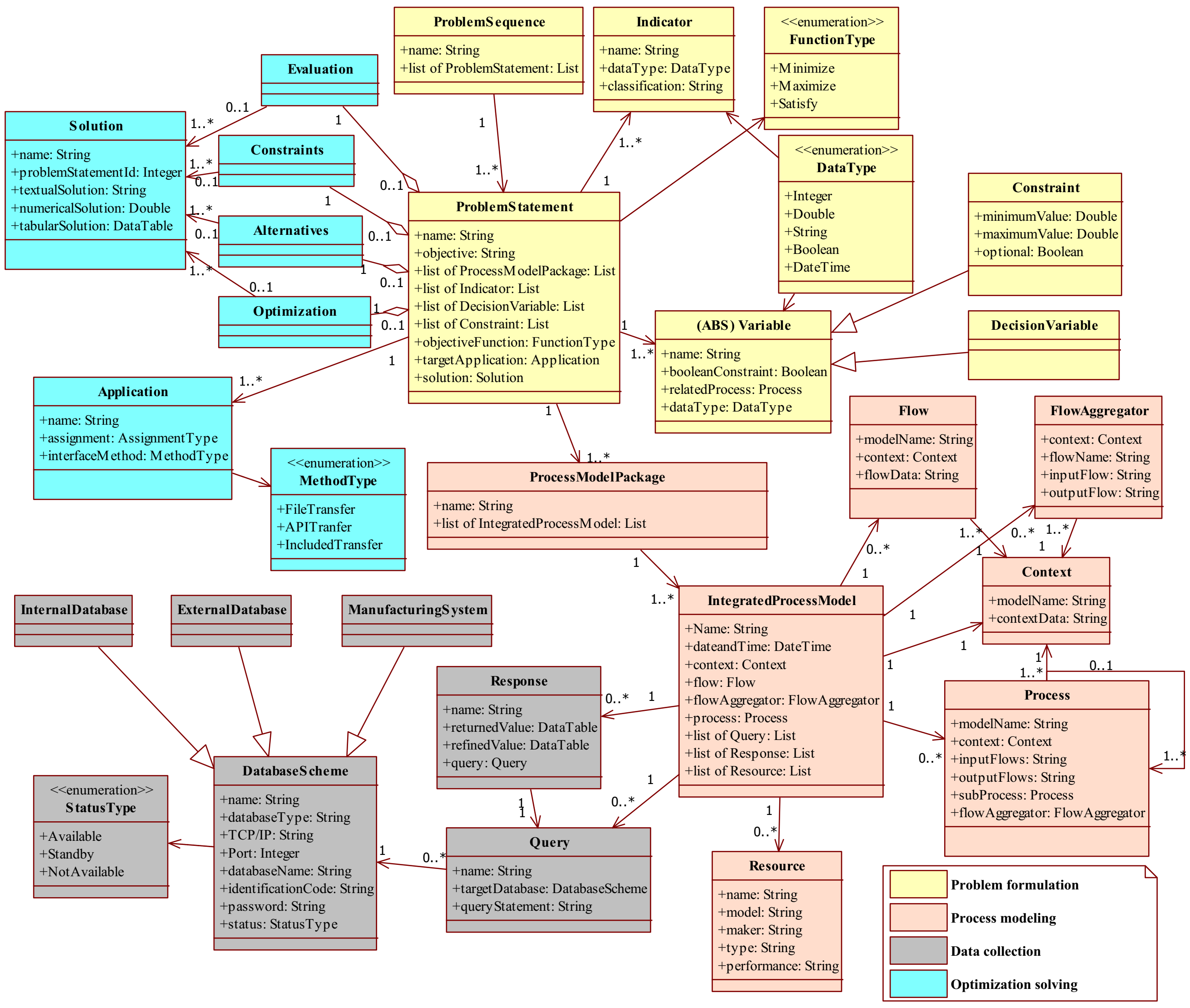


Fig. 8 A class diagram for the data design of SPOT

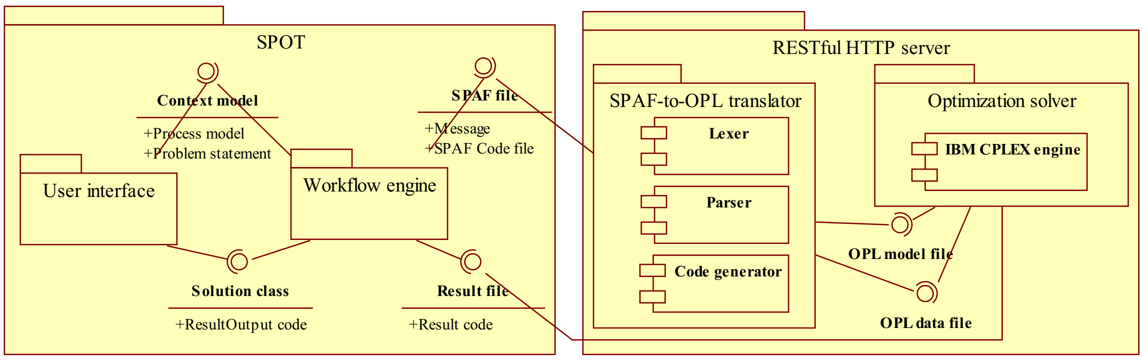


Fig. 9 The roles and interactions of the system components

solving. This class diagram is designed to make consistency with the data structure of SPAF shown in Fig. 1. The 'context', 'flow', 'flow aggregator', and 'process' objects in Fig. 8 correspond to the homonymous objects in Fig. 1. The 'indicator' class in Fig. 8 is used to represent the data content of 'metrics aggregator' in Fig. 1.

The 'problem sequence' class has a set of 'problem statement' objects that play the role of centralized data holders. These objects are also linked to the classes relevant to problem formulation, process models, and solutions. The 'problem statement', 'indicator', and 'variable' objects are instantiated by the activities of the 'problem formulation' step. 'Process model package' is a list of the 'integrated process model' classes that consist of 'context', 'flow', 'flow aggregator', 'process', and 'resource' (Brodsky et al. 2014). 'Query' and 'response' are classes to represent the data related to the 'data collection' step. 'Query' includes the attributes of the query statement and connectivity information for a target data source. 'Response' includes a data set returned from a data source.

The solutions returned from an optimization application can be classified into four abstractive classes in terms of features of the solutions. 'Evaluation' stands for an assessment result within assigned variables and constraints. 'Constraint' is the class that provides the ranges of decision variables and their indicator values. 'Alternative' indicates a list of selective or alternative solutions that satisfy constraints. 'Optimization' indicates the solutions that include optimal values of decision variables and indicators.

Implementation

We develop a prototype system based on the architecture described in "Architecture design" section. Sections "Implementation framework" and "User interface", respectively, present an implementation framework and a user interface for the prototype system.

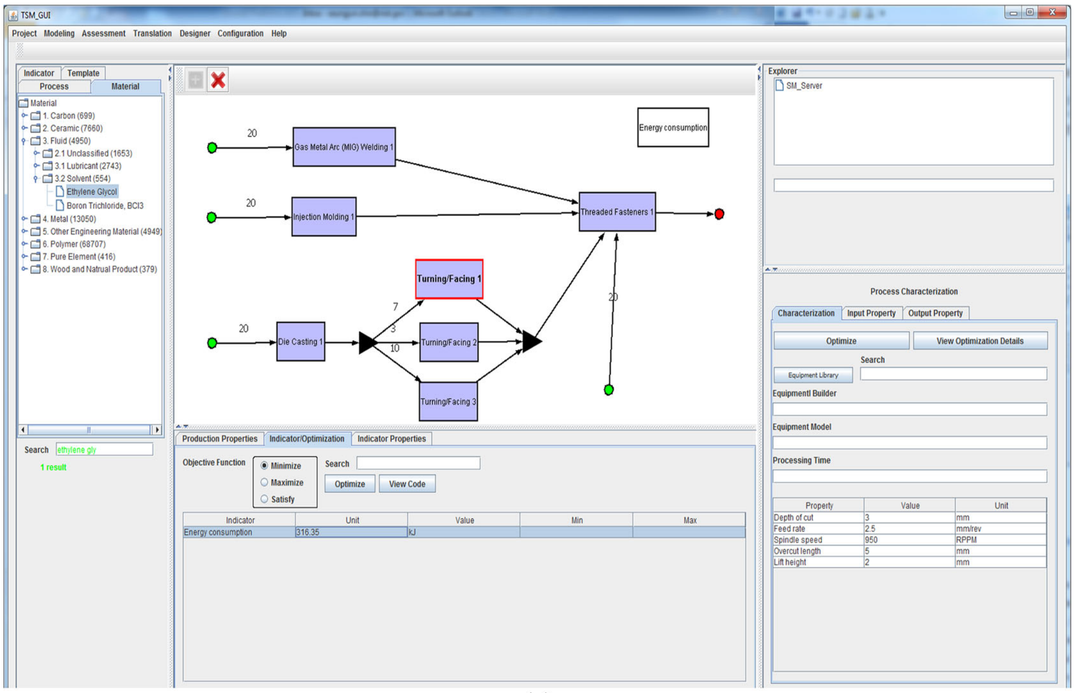
Implementation framework

Figure 9 shows the implementation framework to specify the roles and interactions of the system components. In this paper, this prototype connects with ILOG CPLEX through the file transaction on a web-service. The prototype utilizes Netbeans as Integrated Development Environment, Java as a programming language, and Representational State Transfer as a web-service.

The roles and interactions of the system components are explained as follows: (1) a user designs an optimization model through 'user interface' and requests a service, (2) 'workflow engine' creates a SPAF file, (3) the engine transfers the file to 'web-server', (4) the transfer of the SPAF file invokes internal functions of the web-service, (5) 'lexer' converts a sequence of characters in the SPAF file into a sequence of tokens, (6) 'parser' forms a data structure of the tokens on the basis of the semantics of the SPAF, (7) 'file generator' makes the data structure conform to OPL syntax and semantics, (8) 'CPLEX engine' creates an optimal solution file and returns this file to 'web-server', (9) 'web-server' delivers the file to 'workflow engine', (10) 'workflow engine' transforms the file into the data schema of SPOT (particularly the 'solution' object in Fig. 8), and (11) SPOT displays the optimal results through 'user interface'.

User interface

Figure 10 presents the GUIs of the prototype system. In Fig. 10a, the left panel includes the tree menus that list the hierarchical classifications for unit-processes, materials and sustainability indicators (designed for the implementation of *domain-specificity*). In Fig. 10b, the unit-process tree categorizes unit-processes in terms of energy source, which is a major concern in sustainable manufacturing (*domain-specificity*) (Kumaraguru et al. 2014). The indicator tree provides sustainability indicators to help a user measure sustainability performance (*domain-specificity*) (Joung et al. 2013). A user easily selects unit-processes, materials, and indica-



Material
- 1. Carbon (699)
- 2. Ceramic (7660)
- 3. Fluid (4950)
  - 2.1 Unclassified (1653)
  - 3.1 Lubricant (2743)
  - 3.2 Solvent (554)
    - Ethylene Glycol
    - Boron Trichloride, BCI3
- 4. Metal (13050)
- 5. Other Engineering Material (4949)
- 6. Polymer (68707)
- 7. Pure Element (416)
- 8. Wood and Natrual Product (379)

Search: ethylene gly
1 result

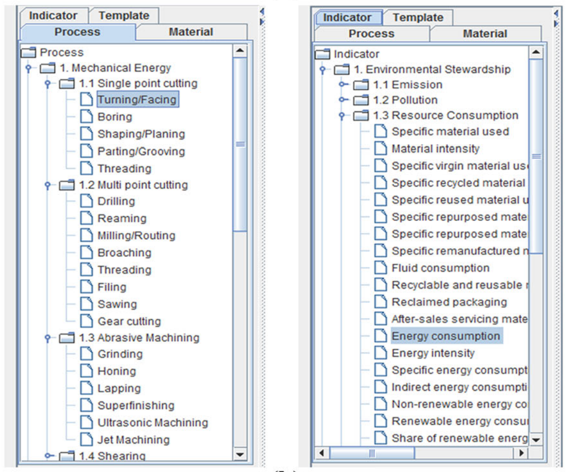
Objective Function: [✓ Minimize] Maximize Satisfy
Search:

Indicator	Unit	Value	Min	Max
Energy consumption	316.35	kJ		

Search:
Equipmentl Builder:
Equipment Model:
Processing Time:

Property	Value	Unit
Depth of cut	3	mm
Feed rate	2.5	mm/rev
Spindle speed	950	RPPM
Overcut length	5	mm
Lift height	2	mm

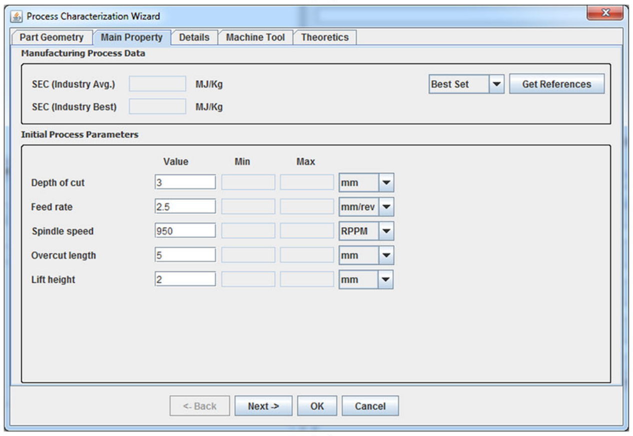
(a)



Process
- 1. Mechanical Energy
  - 1.1 Single point cutting
    - Turning/Facing
    - Boring
    - Shaping/Planing
    - Parting/Grooving
    - Threading
  - 1.2 Multi point cutting
    - Drilling
    - Reaming
    - Milling/Routing
    - Broaching
    - Threading
    - Filing
    - Sawing
    - Gear cutting
  - 1.3 Abrasive Machining
    - Grinding
    - Honing
    - Lapping
    - Superfinishing
    - Ultrasonic Machining
    - Jet Machining
  - 1.4 Shearing

Indicator
- 1. Environmental Stewardship
  - 1.1 Emission
  - 1.2 Pollution
  - 1.3 Resource Consumption
    - Specific material used
    - Material intensity
    - Specific virgin material us
    - Specific recycled material
    - Specific reused material u
    - Specific repurposed mate
    - Specific repurposed mate
    - Specific remanufactured n
    - Fluid consumption
    - Recyclable and reusable r
    - Reclaimed packaging
    - After-sales servicing mate
    - Energy consumption
    - Energy intensity
    - Specific energy consumpt
    - Indirect energy consumpti
    - Non-renewable energy co
    - Renewable energy consu
    - Share of renewable energ

(b)


Process Characterization Wizard
Part Geometry Main Property Details Machine Tool Theoretics
Get References
< Back Next -> OK Cancel

Manufacturing Process Data

SEC (Industry Avg.) MJ/Kg
SEC (Industry Best) MJ/Kg
Best Set

Initial Process Parameters

	Value	Min	Max	
Depth of cut	3			mm
Feed rate	2.5			mm/rev
Spindle speed	950			RPPM
Overcut length	5			mm
Lift height	2			mm

(c)

Fig. 10 Screen shots of SPOT prototype system. **a** A main GUI. **b** Tree viewers for the selection of unit-process and sustainability indicator. **c** A pop-up box for unit-process characterization (e.g., a turning machining process)

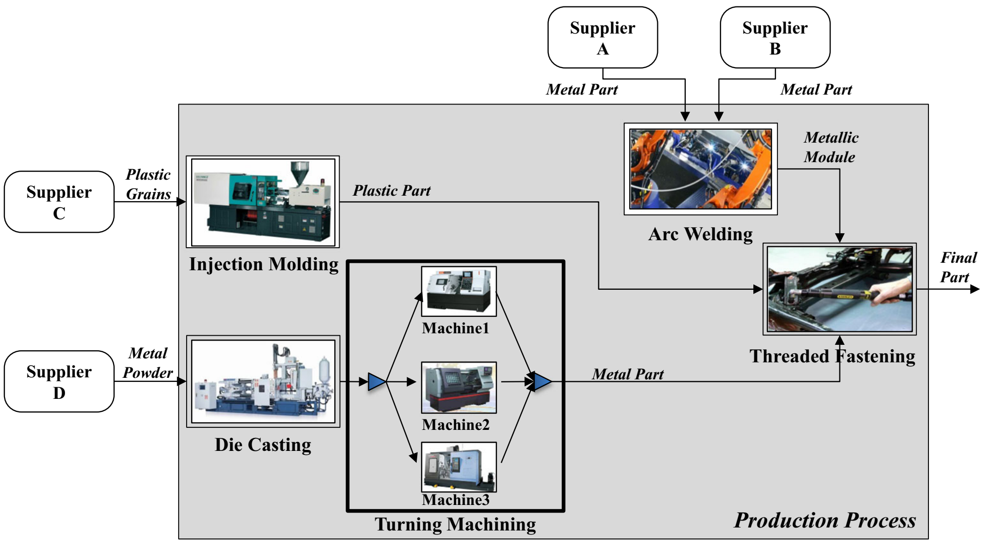


Fig. 11 An example of a production process

tors by drag-and-drop operation (*simplicity*). In addition, the left panel provides a process template tree that saves process model components for reuse (*reusability*).

The upper middle panel provides a process modeler that allows a user to graphically design a production process (*simplicity*). The lower middle panel provides a production property table and an indicator property table. The production property table edits and shows production details such as identification, throughput, and units (*simplicity* and *diversity*). By using the indicator table, the selected indicator can be specified as either a performance indicator or a constraint indicator variable. A user can select an object and ranges of the indicators and constraints.

The lower right panel displays a unit-process characterization table. A user can characterize a unit-process through a characterization wizard, as shown in Fig. 10c. This characterization wizard provides a guidable interface to identify process properties such as parametric design, process parameters, a machine and its auxiliaries (*diversity* and *simplicity*) (Madan et al. 2013). When a unit-process is characterized, a user can check process properties and edit them directly on the viewer (*simplicity*).

Case study

Two case studies are performed to demonstrate the effectiveness of SPOT. "Manufacturing scenario" section describes a manufacturing scenario that includes a production process and its associated assumptions. Sections "Optimization for resource allocation at the production level" and "Optimization for parameter selection at unit-process level", respectively, show minimizing energy consumption by allocation of resources at the production level and selection of process parameters at the unit-process level. The two case studies use assumed data values. "Discussion" section discusses the issues of data uncertainty and size, computations and comparison with existing optimization applications.

Manufacturing scenario

Figure 11 illustrates a production process to manufacture a mechanical part of a drilling tool. A welding process joins two metal parts and an injection molding process produces a plastic part. A die casting process forms a casted part from metal powders and this part passes through one of three machine tools in a turning machining process. These turning machines can manufacture a maximum of 20 parts per hour. A fastening process makes a final part by manual assembly. The production process should yield 20 final parts per hour.

We assume that the energy profiles of the three turning machines have a piecewise linear dependence on hourly throughput. But, these energy profiles have different coefficients of the linear dependency due to different machine capabilities while other unit-processes have constant energy consumption. We also assume that an engineer has no experience in optimization and accesses to an optimization application to solve optimization problems.

Optimization for resource allocation at the production level

The objective of this case study is to minimize energy consumption in the production process by proper allocation of the numbers of parts to each of the three turning machine

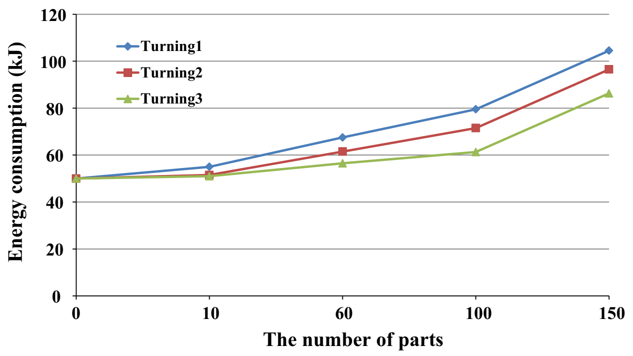


Fig. 12 Examples of energy profiles for the three turning machines

tools. An engineer can formulate an optimization problem, as expressed in Eq. 4:

$$Minimize \quad E(x_1,\ x_2,\ x_3)$$
$$subject\ to \begin{cases} E(x_1,\ x_2,\ x_3) = \sum_{i=1}^{n} E_n(x_n) + C, \\ x_1 + x_2 + x_3 = 20, \\ 0 \leq x_1 \leq 20, \\ 0 \leq x_2 \leq 20, \\ 0 \leq x_3 \leq 20 \end{cases} \tag{4}$$

where, x_n: the numbers of parts to each of the three machine tools $E_n(x_n)$: energy consumption at each machine tool, C: constant energy consumption by other unit-processes.

The engineer designs the production process in Fig. 11 through the 'process modeler' user interface. The engineer needs to collect the data relevant to energy consumption because Eq. 4 needs to be instantiated with real values. Assuming the three turning machines make the energy profiles with regard to the number of parts, as shown in Fig. 12, $E_n(x_n)$ can be instantiated. For example, the energy equation for 'turning machine 1' can be expressed by Eq. 5:

$$E_1(x_1) = \begin{cases} 0.2x_1 + 50 & 0 \leq x_1 \leq 10, \\ 0.25x_1 + 49.5 & 10 \leq x_1 \leq 60, \\ 0.3x_1 + 46.5 & 60 \leq x_1 \leq 100, \\ 0.5x_1 + 26.5 & 100 \leq x_1 \leq 150 \end{cases} \tag{5}$$

The engineer selects ILOG CPLEX as an optimization application through the 'application selector' user interface. Then the prototype system generates a SPAF file. Figure 13 demonstrates a portion of a SPAF file. 'Model flow item{}' indicates the flow model, and 'model flow aggregator itemAggr{}' stands for the flow aggregator model. 'Model process baseEnergyThruMachine { }' represents an abstract function for energy consumption, based on the energy function 'pwlFunction energyFunction' and its constraints.

For example, 'Turning 1' shows a unit-process model that calculates energy consumed by 'turning machine 1'. The profile has a linear gradient (s[0] = 0.2) and an initial energy (initEnergy = 50) during its first segment. 'Model process ManufacturingFloor{ }' includes the metric model and the context model. The 'energyPerHour' variable equals to total energy consumed during the production process. Here the constraint is coded by 'item[threadedOutItem].unitPerHour = 20', and the objective function is coded by 'minimize ManufacturingFloor[ ].energyPerHour', respectively.

The 'OPL translator' component transforms the SPAF file into two OPL files (i.e., a model file and a data file). Figure 14 presents a portion of the OPL model file and Fig. 15 shows a portion of the OPL data file. The mapping rule defined in 'semantics/syntax repository' converts syntax and semantics of the SPAF into those of the OPL. For example, the 'float energyPerHour' variable of the SPAF model is converted to "dexpr float ManufacturingFloor_energyPerHour[..] = …' of the OPL model. The detailed mapping rule (Brodsky et al. 2014) is beyond the scope of this paper.

The CPLEX engine receives the OPL files, derives an optimal solution through the branch-and-bound algorithm and returns the solution to 'web-server'. The optimal solution calculates that minimal energy consumption will be 316.35 kJ for the production process if the numbers of parts are allocated as 7 parts for 'turning machine 1', 3 parts for 'turning machine 2', and 10 parts for 'turning machine 3', as displayed in Fig. 10a.

Optimization for parameter selection at unit-process level

The objective of this case study is to minimize energy consumption of, for example, 'turning machine 1' by selecting appropriate decision variables – feedrate, spindle speed, and cutting depth. The engineer uses an empirical model of power demand that relates the decision variables of 'turning machine 1' to its energy consumption (Shin 2010). Similar to the production case presented in "Optimization for resource allocation at the production level" section, the engineer can formulate and instantiate an optimization problem, as expressed in Eq. 6:

```
model flow item {
   string type = ...;
   float  unitPerHour;
}
model flow aggregator itemAggr {
   ...
   sum(i in inputFlows) item[i].unitPerHour == sum(j in outputFlows) item[j].unitPerHour;
}
model process baseEnergyThruMachine {
                              ...
   forall(i in inputFlows)
      forall(o in outputFlows)
         item[i].unitPerHour == item[o].unitPerHour * inputPerOutput[i];
         ...
   pwlFunction energyFunction = piecewise{s[1] -> b[1]; s[2] -> b[2]; s[3] -> b[3]; s[4]}(minThru, initEnergy);

   float  thru;
   minThru <= thru <= maxThru;
   float energyPerHour = energyFunction(thru);
   forall (i in outputFlows) thru == item[i]a.unitPerHour;
}
model process Turning1 {
   ...
   model  energy =  new baseEnergyThruMachine["turn1"]{
      inputFlows = { turningInItem1 };    outputFlow = turningOutItem1;    inputPerOutput[] = [1];    s[] = [0.2, 0.25, 0.30, 0.5];
      b[] = [10,60,100];    minThru = 0.0;    maxThru = 150;    initEnergy =  50.0
      };
   float energyPerHour = baseEnergyThruMachine[energy].energyPerHour;
   float thru = baseEnergyThruMachine[energy].thru;
}
model process ManufacturingFloor {
   ...
   float energyPerHour = DieCasting[dieCastingProc].energyPerHour + Turning1[turning1Proc].energyPerHour +
                         Turning2[turning2Proc].energyPerHour + Turning3[turning3Proc].energyPerHour +
                         GasMetalArcWelding[gasMetalArcWeldingProc].energyPerHour +
                         InjectionMolding[injectionMoldingProc].energyPerHour;
   item[threadedOutItem].unitPerHour == 20;
}
minimize ManufacturingFloor[].energyPerHour;
```

Fig. 13 A SPAF sample file for the resource allocation problem

$$\begin{aligned} &Minimize \quad P(x_1,\ x_2,\ x_3) \\ &subject\ to \quad \begin{cases} P(x_1,\ x_2,\ x_3) = 2.807 + 0.123x_1 + 0.676x_2 + 0.157x_3 - 0.028x_1^2 + 0.053x_2^2 - 0.008x_3^2 + 0.176x_1x_2 + 0.022x_1x_3 + 0.024x_2x_3, \\ \quad x_1 = \frac{X_1-0.25}{0.02}, \quad x_2 = \frac{X_2-133.25}{33.25}, \quad x_3 = \frac{X_3-2.5}{0.5}, \\ \quad 0.23 \le x_1 \le 0.27, \quad 100 \le x_2 \le 166.5, \quad 2 \le x_3 \le 3 \end{cases} \end{aligned} \tag{6}$$

where, X_1: feedrate (mm/rev), X_2: spindle speed (rad/s) X_3: cutting depth (mm), $P(x)$: the average power demand.

Figure 16 shows a portion of a SPAF file. 'TurningMachineOrders' identifies the ranges of the decision variables and the power demand model. 'TurningMachine' specifies the ranges of the process parameters by allocating real numbers. The SPAF file is also translated to an OPL model file and an OPL data file. The optimization engine derives optimal process parameters through the constraint propagation and search algorithm. The optimal solution calculates that minimal energy consumption will be 178.28 kJ for 'turning machine 1' when the process parameters result in 0.27 mm/rev for *feedrate*, 100 rad/s for *spindle speed*, and 2.6 mm for *cutting depth*.

Discussion

The case study showed the effectiveness of SPOT. This section discusses data uncertainty and size, and computations aspects as well as the difference with an existing optimization application.

```
{string} item_IDS = ...;
string item_type[item_IDS] = ...;
dvar float item_unitPerHour[item_IDS];
…
dexpr float Turning1_energyPerHour[Turning1_index in Turning1_IDS] =
        baseEnergyThruMachine_energyPerHour[Turning1_energy];
dexpr float Turning1_thru[Turning1_index in Turning1_IDS] = baseEnergyThruMachine_thru[Turning1_energy];
dexpr float ManufacturingFloor_energyPerHour[ManufacturingFloor_index in ManufacturingFloor_IDS] =
         DieCasting_energyPerHour[ManufacturingFloor_dieCastingProc] +
         Turning1_energyPerHour[ManufacturingFloor_turning1Proc] +
         Turning2_energyPerHour[ManufacturingFloor_turning2Proc] +
         Turning3_energyPerHour[ManufacturingFloor_turning3Proc] +
         GasMetalArcWelding_energyPerHour[ManufacturingFloor_gasMetalArcWeldingProc] +
         InjectionMolding_energyPerHour[ManufacturingFloor_injectionMoldingProc];
minimize ManufacturingFloor_energyPerHour["P-01"];
subject to {
  forall(id in baseEnergyThruMachine_IDS)
    forall(baseEnergyThruMachine_i in baseEnergyThruMachine_inputFlows[id])
    forall(baseEnergyThruMachine_o in baseEnergyThruMachine_outputFlows[id])
      item_unitPerHour[baseEnergyThruMachine_i] == item_unitPerHour[baseEnergyThruMachine_o] *
      baseEnergyThruMachine_inputPerOutput[<id, baseEnergyThruMachine_i>];
};
subject to {
  forall(id in baseEnergyThruMachine_IDS)
  baseEnergyThruMachine_minThru[id] <= baseEnergyThruMachine_thru[id] <= baseEnergyThruMachine_maxThru[id];
};
subject to {
  forall(id in baseEnergyThruMachine_IDS)
    forall(baseEnergyThruMachine_i in baseEnergyThruMachine_outputFlows[id])
      baseEnergyThruMachine_thru[id] == item_unitPerHour[baseEnergyThruMachine_i];
};
subject to {
  forall(id in ManufacturingFloor_IDS)
  item_unitPerHour[ManufacturingFloor_threadedOutItem] == 20;
};
```

Fig. 14 An example of an OPL model file

```
item_IDS = { "dieIN" "dieOUT" "turnIN1" "turnOUT1" "turnIN2" "turnOUT2" "turnIN3" "turnOUT3" "weld2fast"
             "weldIN" "inj2fast" "injIN" "fastOUT" "fastIN" };
item_type = #["dieIN" : "dieCastingInType", "dieOUT" : "dieCastingOutType" , "turnIN1" : "dieCastingOutType" , "turnOUT1" :
             "turningOutType" , "turnIN2" : "dieCastingOutType" , "turnOUT2" : "turningOutType" , "turnIN3" :
             "dieCastingOutType" , "turnOUT3" : "turningOutType" , "weld2fast" : "weldingOutType" , "weldIN" :
             "turningOutType" , "inj2fast" : "injectionMoldingOutType" , "injIN" : "injectionMoldingInType" , "fastOUT" :
             "finalPartType" , "fastIN" : "addFasteningInType" ]#;
itemAggr_IDS = { "dieAggr" "turnAggr" };
itemAggr_inputFlows = #["dieAggr" : {"dieOUT"}, "turnAggr" : {"turnOUT1", "turnOUT2", "turnOUT3"} ]#;
itemAggr_outputFlows = #["dieAggr" : {"turnIN1", "turnIN2", "turnIN3"}, "turnAggr" : {"weldIN"} ]#;
baseEnergyThruMachine_IDS = { "die" "turn1" "turn2" "turn3" "weld" "inject" "fast" };
…
baseEnergyThruMachine_inputPerOutput = [5.7, 1 , 1 , 1 , 1 , 1 , 1, 1, 1 ];
baseEnergyThruMachine_minThru = #["die" : 0.0, "turn1" : 0.0 , "turn2" : 0.0 , "turn3" : 0.0 , "weld" : 0.0 , "inject" : 0.0 , "fast" : 0.0 ]#;
baseEnergyThruMachine_initEnergy = #["die" : 50.0, "turn1" : 50.0 , "turn2" : 50.0 , "turn3" : 50.0 , "weld" : 50.0 , "inject" : 50.0 , "fast" :
                                   50.0 ]#;
baseEnergyThruMachine_maxThru = #["die" : 150, "turn1" : 150 , "turn2" : 3 , "turn3" : 10 , "weld" : 150 , "inject" : 150 , "fast" : 150 ]#;
baseEnergyThruMachine_s = [0.2, 0.25, 0.30, 0.5, 0.2, 0.25, 0.30, 0.5 , 0.15, 0.2, 0.25, 0.5 , 0.1, 0.11, 0.12, 0.5 , 0.2, 0.25, 0.30, 0.5 , 0.2, 0.25,
                          0.30, 0.5 , 0.2, 0.25, 0.30, 0.5 ];
baseEnergyThruMachine_b = [10, 60, 100, 10, 60, 100 , 10, 60, 100 , 10, 60, 100 , 10, 60, 100 , 10, 60, 100 , 11, 60, 100 ];
...
Turning1_IDS = { "turn1" };
Turning1_turningInItem1 = #["turn1" : "turnIN1"]#;
Turning1_turningOutItem1 = #["turn1" : "turnOUT1"]#;
Turning1_energy =  "turn1";
...
```

Fig. 15 An example of an OPL data file

```
Model process turningMachineOrders {
…
Min_Depth_Cut  <= Depth_Cut <= Max_Depth_Cut;
Min_Spindle_Speed  <= Spindle_Speed <= Max_Spindle_Speed;
Min_Feed_Rate <= Feed_Rate <= Max_Feed_Rate;
…
float P_active[o in Orders] = 2.807 + 0.123*(Feed_Rate[o]) + 0.676*Spindle_Speed[o] + 0.157*(Depth_Cut[o])
                                          - 0.028*(Feed_Rate[o])*(Feed_Rate[o]) + 0.053*Spindle_Speed[o]*Spindle_Speed[o]
                                       - 0.008*(Depth_Cut[o])*(Depth_Cut[o]) + 0.176*(Feed_Rate[o])*Spindle_Speed[o]
                                          + 0.022*(Feed_Rate[o])*(Depth_Cut[o]) + 0.024*(Spindle_Speed[o])*(Depth_Cut[o]);
}
Model process TurningMachine {
…
Min_Depth_Cut  = 2.0;
Max_Depth_Cut = 3.0;
Min_Spindle_Speed = 100.0;
Max_Spindle_Speed = 166.5;
Min_Feed_Rate = 0.23;
Max_Feed_Rate = 0.27;
float Power_Demand = turningMachineOrders[demoTurningMachine].Power_Demand;
…
};
Minimize TurningMachine[].Power_Demand;
```

Fig. 16 A SPAF sample file for the parameter selection problem

- Data uncertainty and size: because manufacturing data normally include erroneous or missing data and the difference between measured values and their true values (Wazed et al. 2009), various degrees of uncertainty exist in the data. Therefore, it is a challenge to handle the data uncertainty because the increase of the data uncertainty decreases the accuracy of optimal solutions. The uncertainty comes from the data, as one of epistemic uncertainties (Roy 2010), can be reduced with uncertainty quantification and data-preprocessing. The uncertainty quantification helps manufacturers characterize the sources of the uncertainty and measure the probability distribution of the data and finally find some controllable solutions. The data pre-processing, which includes data cleaning, normalization, transformation, feature extraction, and selection (Kotsiantis et al. 2006), enables manufacturers to generate the data that decrease the uncertainty by the elimination of noisy and unreliable data and the reduction of large data sets without any loss of important information. We have taken into account the data pre-processing in our architecture design (see the ‘data pre-processor’ component in Fig. 7); however, the current SPOT excludes the implementation of the data pre-processor.
  Another issue is the retrieval and processing of large data because a shop floor obviously generates and uses a large volume of data. Especially, machine monitoring data can have a huge amount of streaming data content. A conventional database environment has a limitation on managing efficiently the large volume of data that are needed to perform fast data retrieval and processing for on-time or real-time optimization. Thus, it is beneficial to use a distributed database like Hadoop Distributed File System (HDFS) (Shvachko et al. 2010) and a programming model such as MapReduce (Dean and Ghemawat 2008). We are developing a HDFS and MapReduce infrastructure to enable on-time prediction and optimization for improving sustainability performance in manufacturing processes (Shin et al. 2014). However, the current SPOT doesn’t integrate this data infrastructure. This should be a future implementation together with the implementation of the data pre-processor.
- Computational aspect of SPOT: in the current implementation, the OPL model and data are submitted to the CPLEX Mixed Integer Linear Programming (MILP) application, which returns an optimal instantiation of all variables to real or integer values if the problem is feasible and bounded. The CPLEX uses the branch and bound algorithm for MILP problems, augmented with a variety of heuristics. The optimization problems can only use linear arithmetic in terms of decision variables. However, because the SPAF-to-OPL translator in Fig. 9 works without this limitation, it could be easily extended with more general applications (e.g., a NLP solver and a CP solver). Also it is important to note that this paper focuses on the easy modeling aspects under the assumption that an optimization application, best suited to the problem, is used to handle the computation. As a future research, we are working on specialized algorithms that can handle non-linearity and stochasticity for manufacturing process problems.
- Difference between SPOT and existing optimization applications: SPOT is a domain-specific DSS designed for manufacturing engineers; whereas existing optimiza-

tion applications such as Optimus, ILOG CPLEX and OptQuest are generic optimization tools designed for Operations Research (OR) experts who will be developing mathematical models with heavily relying on their expertise. As we discussed in “Introduction” section, a key contribution behind SPOT is to raise the level of abstraction so that manufacturing engineers, as opposed to OR experts, would be able to use it.

Conclusion

This paper presented the development of a DSS specifically defined to improve sustainability performance in manufacturing. The system architecture identified the behavior, function and data structures in consideration of the *domain-specificity*, *diversity*, *compatibility*, *simplicity* and *reusability* design factors. We anticipate that SPOT will help manufacturers reduce time and effort for enhancing sustainability performance without heavily relying on optimization expertise.

The modeling approach of this paper can facilitate the development of data analytics, i.e., the science of data analysis to discover new and meaningful information in data (Kohavi et al. 2002), for improving sustainability performance in manufacturing. The adoption of the modular approach by SPAF makes (de-) composition of optimization models easier and the model components can be efficiently reused for various analytics purposes. The application-neutral environment enables interoperable data communication in various analytics applications. Furthermore, data collection connected with a shop floor will embody (near) real-time optimization on a manufacturing process. For this purpose, we are developing a shop floor data interface by the use of MTConnect, which defines a language and structure for open communication with machine tools (AMT 2011).

Some limitations of this paper are that SPOT only shows the possibility of the data connection between SPOT and a shop floor or between SPOT and an external data source. The data used in the two case studies are given from our assumption not from an industry data source. Even though the focus of the paper is to introduce the system tool, not real data analytics, but real manufacturing data definitely make it more interesting. Also SPOT is only integrated with one optimization application at a time. In the future, we will extend the simultaneous connectivity toward multiple data sources for the use of referential inventory and industry data. We will integrate SPOT with a distributed data infrastructure for handling a large volume of manufacturing data. We will also implement the communication of SPOT with other analytics applications such as diagnostics, prognostics and prediction as well as optimization.

Acknowledgements The authors thank Abdullah Alrazgan, a graduate student from George Mason University, for his effort on SPAF compiler development. The work represented here was partially funded through cooperative agreement #70NANB12H277 between George Mason University and NIST.

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

References

Aggarwal, A., Singh, H., Kumar, P., & Singh, M. (2008). Optimizing power consumption for CNC turned parts using response surface methodology and Taguchi’s technique—A comparative analysis. *Journal of Materials Processing Technology*, *200*, 373–384.

Alberti, M., Ciurana, J., Rodriguez, C., & Ozel, T. (2011). Design of a decision support system for machine tool selection based on machine characteristics and performance tests. *Journal of Intelligent Manufacturing*, *22*, 263–277.

AMT. (2011). MTConnect standard part 1—Overview and protocol. The Association for Manufacturing Technology. http://www.mtconnect.org/. Accessed Feb 2014.

Arslan, M., Catay, B., & Budak, E. (2004). A decision support system for machine tool selection. *Journal of Manufacturing Technology Management*, *15*(1), 101–109.

Battaia, O., Dolgui, A., Guschinsky, N., & Levin, G. (2012). A decision support system for design of mass production machining lines composed of stations with rotary or mobile table. *Robotics and Computer-Integrated Manufacturing*, *28*, 672–680.

Bhushan, R. (2013). Optimization of cutting parameters for minimizing power consumption and maximizing tool life during machining of Al alloy SiC particle composites. *Journal of Cleaner Production*, *39*, 242–254.

Biegler, L. (2010). *Nonlinear programming: Concepts, algorithms, and applications to chemical processes* (Vol. 10). SIAM-Society for Industrial and Applied Mathematics.

Brodsky, A., Shao, G., & Riddick, F. (2014). Process analytics formalism for decision guidance in sustainable manufacturing. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0892-9.

Clausen, J. (1999). Branch and bound algorithms—Principles and examples. University of Copenhagen. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.5.7475&rep=rep1&type=pdf. Accessed 19 Nov 2014

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. New York: Wiley.

Diaz, N., Helu, M., & Dornfeld, D. (2010). Design and operation strategies for green machine tool development. In Proceedings of MTTRF 2010 annual meeting.

Diaz, N., & Dornfeld, D. (2012). Cost and energy consumption optimization of product manufacture in a flexible manufacturing system. In Proceedings of the 19th CIRP conference on life cycle engineering (pp. 411–416).

Dimopoulos, C., & Zalzala, A. (2000). Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons. *IEEE Transactions on Evolutionary Computation*, *4*(2), 93–113.

Euzenat, J., & Shvaiko, P. (2007). *Ontology mapping*. Heidelberg: Springer.

Fang, K., Uhan, N., Zhao, F., & Sutherland, J. (2013). Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research*, *206*(1), 115–145.

Frischknecht, R., Jungbluth, N., Althaus, H. J., Doka, G., Dones, R., Hischier, R., Hellweg, S., Nemecek, T., Rebitzer, G., & Spielmann, M. (2007). Eco-Invent: Overview and methodology. Swiss Centre for Life Cycle Inventories. http://www.ecoinvent.org/fileadmin/documents/en/01_OverviewAndMethodology.pdf. Accessed Dec 2013.

Giachetti, R. (1998). A decision support system for material and manufacturing process selection. *Journal of Intelligent Manufacturing*, *9*, 265–276.

Hanafi, I., Khamlichi, A., Cabrera, F., & Almansa, E. (2012). Optimization of cutting conditions for sustainable machining of PEEK-CF30 using TiN tools. *Journal of Cleaner Production*, *33*, 1–9.

IBM. (2012). Introducing IBM ILOG CPLEX Optimization Studio V12.2. IBM. http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r2/index.jsp?topic=%2Filog.odms.ide.help%2FContent%2FOptimization%2FDocumentation%2FOPL_Studio%2F_pubskel%2Fglobals%2Feclipse_and_xplatform%2Fps_opl307.html. Accessed Aug 2013.

Iqbal, A., Zhang, H., Kong, L., & Hussain, G. (2014). A rule-based system for trade-off among energy consumption, tool life and productivity in machining process. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-013-0851-x.

Jia, S., Tang, R., & Lv, J. (2014). Machining activity extraction and energy attributes inheritance method to support intelligent energy estimation of machining process. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0894-7.

Joung, C., Carrell, J., Sakar, P., & Feng, S. (2013). Categorization of indicators for sustainable manufacturing. *Ecological Indicators*, *24*, 148–157.

Kellens, K., Dewulf, W., Overcash, M., Hauschild, M., & Duflou, J. (2012). Methodology for systematic analysis and improvement of manufacturing unit process life-cycle inventory (UPLCI) CO2PE! Initiative part1: Methodology description. *International Journal of Life Cycle Assessment*, *17*, 69–78.

Kohavi, R., Rothleder, N., & Simoudis, E. (2002). Emerging trends in business analytics. *Communications of the ACM*, *45*(8), 45–48.

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Data preprocessing for supervised leaning. *International Journal of Computer Science*, *1*(2), 111–117.

Kumaraguru, S., Rachuri, S., & Lechevalier, D. (2014). Faceted classification of manufacturing processes for sustainability performance evaluation. *The International Journal of Advanced Manufacturing Technology*, *75*(9–12), 1309–1320.

Kuram, E., Ozcelik, B., Bayramoglu, M., Demirbas, E., & Simsek, B. (2013). Optimization of cutting fluids and cutting parameters during end milling by using D-optimal design of experiments. *Journal of Cleaner Production*, *42*, 159–166.

Le, C., & Pang, C. (2013). An energy data-driven decision support system for high-performance manufacturing industries. *International Journal of Automation and Logistics*, *1*(1), 61–79.

Lei, Q., Wang, H., & Song, Y. (2014). Hybrid knowledge model of process planning and its green extension. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0928-1.

Leng, J., Jiang, P., & Ding, K. (2014). Implementing of a three-phase integrated decision support model for parts machining outsourcing. *International Journal of Production Research*, *52*(12), 3614–3636.

Li, C., Tang, Y., Cui, L., & Li, P. (2013a). A quantitative approach to analyze carbon emissions of CNC-based machining systems. *Journal of Intelligent Manufacturing*.

Li, L., Liu, F., Chen, B., & Li, C. (2013b). Multi-objective optimization of cutting parameters in sculptured parts machining based on neural network. *Journal of Intelligent Manufacturing*.

Madan, J., Mani, M., & Lyons, K. (2013). Characterizing energy consumption of the injection molding process. In Proceedings of the ASME 2013 international manufacturing science and engineering conference.

OECD. (2009). Sustainable manufacturing and eco-innovation—Framework, practices and measurement. http://www.oecd.org/innovation/inno/43423689.pdf. Organization for Economic Cooperation and Development. Accessed Jan 2014.

OMG. (2007). OMG unified modeling language -superstructure. Object Management Group. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. Accessed Feb 2014.

Rao, S. (2009). *Engineering optimization: Theory and practice*. New York: Wiley.

Roy, C. (2010). A complete framework for verification, validation, and uncertainty quantification in scientific comuting. In Proceedings of 48th AIAA aerospace sciences meeting.

Shao, G., Brodsky, A., Shin, S., & Kim, D. (2014). Decision guidance methodology for sustainable manufacturing using process analytics formalism. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0995-3.

Shin, S. (2010). Development of a framework for green productivity enhancement and its application to machining system. Ph. D. Dissertation. POSTECH.

Shin, S., Woo, J., & Rachuri, S. (2014). Predictive analytics model for power consumption in manufacturing. *Procedia CIRP*, *15*, 153–158.

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The hadoop distributed file system. In Proceedings of IEEE mass storage Sys & Tech.

Taha, Z., & Rostam, S. (2012). A hybrid fuzzy AHP-PROMETHEE decision support system for machine tool selection in flexible manufacturing cell. *Journal of Intelligent Manufacturing*, *23*, 2137–2149.

Vinodh, S., Jayakrishna, K., Kumar, V., & Dutta, R. (2014). Development of decision support system for sustainability evaluation: A case study. *Clean Technologies and Environmental Policy*, *16*(1), 163–174.

Vitanov, V., Tjahjono, B., & Marghalany, I. (2007). A decision support tool to facilitate the design of cellular manufacturing layouts. *Computers & Industrial Engineering*, *52*, 380–403.

Wazed, M., Ahmed, S., & Yusoff, N. (2009). Uncertainty factors in real manufacturing environment. *Australian Journal of Basic and Applied Sciences*, *3*(2), 342–351.

Winter, M., Li, W., Kara, S., & Herrmann, C. (2013). Determining optimal process parameters to increase the eco-efficiency of grinding processes. *Journal of Cleaner Production*, *66*(1), 644–654.

Yam, R., Tse, P., Li, L., & Tu, P. (2001). Intelligent predictive decision support system for condition-based maintenance. *International Journal of Advanced Manufacturing Technology*, *17*, 383–391.

Yurdakul, M., Arslan, E., Tansel, Y., & Turkbas, O. (2014). A decision support system for selection of net-shape primary manufacturing processes. *International Journal of Production Research*, *52*(5), 1528–1541.

Zeballos, L. (2010). A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, *26*, 725–743.

Zhu, Q., Lujia, F., Mayyas, A., Omar, M., Al-Hammadi, Y., & Saleh, S. (2014). Production energy optimization using low dynamic programming, a decision support tool for sustainable manufacturing. *Journal of Cleaner Production*. doi:10.1016/j.jclepro.2014.02.066.