Integration Framework of Process Planning based on Resource Independent Operation Summary to Support Collaborative Manufacturing

Boonserm Kulvatunyou*, Richard A. Wysk**, Hyunbo Cho***, Albert Jones*

*MSI Division, National Institute of Standard and Technology
Gaithersburg, MD 20899, U.S.A.
** Department of Industrial and Manufacturing Engineering, Pennsylvania State University
University Park, PA 16802, U.S.A.
*** Department of Industrial Engineering, Pohang University of Science and Technology
Pohang 790-784, Korea

Abstract

In today's global manufacturing environment, manufacturing functions are distributed as never before. Design, engineering, fabrication, and assembly of new products are done routinely in many different enterprises scattered around the world. Successful business transactions require the sharing of design and engineering data on an unprecedented scale. This paper describes a framework that facilitates the collaboration of engineering tasks, particularly process planning and analysis, to support such globalized manufacturing activities. The information models of data and the software components that integrate those information models are described. The integration framework uses an Integrated Product and Process Data (IPPD) representation called a Resource Independent Operation Summary (RIOS) to facilitate the communication of business and manufacturing requirements. Hierarchical process modeling, process planning decomposition and an augmented AND/OR directed graph are used in this representation. The Resource Specific Process Planning (RSPP) module assigns required equipment and tools, selects process parameters, and determines manufacturing costs based on two-level hierarchical RIOS data. The shop floor knowledge (resource and process knowledge) and a hybrid approach (heuristic and linear programming) to linearize the AND/OR graph provide the basis for the planning. Finally, a prototype system is developed and demonstrated with an exemplary part. Java and XML (Extensible Markup Language) are used to ensure software and information portability.

Keywords: Process planning, Collaborative manufacturing, Resource independent operation summary, Resource specific process planning, AND/OR graph linearization

1. Introduction

Virtual manufacturing adds “rapidity” or “agility” of time and “globalization” of space to the dimension of classical manufacturing. The manufacturers who adopt the concept of virtual manufacturing are able to respond quickly to customers with various types of product demands. The Internet and recent Information Technology developments like the Extensible Markup Language (XML) provide the opportunity for customers and manufacturers in the supply chain to share product, process, and production-related data across the enterprise. Using these new technologies, the designer, the planner, and producer can collaborate to reduce the product-development life cycle and it associated costs.

Until recently, process planning was considered a promising candidate for enhancing the adaptability and flexibility of manufacturing systems. The primary concerns about process planning have been its role in the automation and integration of design, shop floor control, and business functions in the face of the increasing, dynamic nature of manufacturing. For example, designers now prepare a product design and attempt to validate its specification and manufacturability using process planning software. Since this software uses engineering as well as design data, this effort requires the sharing data electronically on an unprecedented scale.

This paper describes a process planning integration framework that facilitates the sharing of engineering data for collaborative manufacturing. The framework starts with the designer’s input, called a Resource Independent Operation Summary (RIOS) to facilitate the communication of product, process, and production requirements. The RIOS is structured hierarchically into two levels and represented as an augmented AND/OR directed graph. A Resource-Specific Process Planning (RSPP) module is then responsible for interfacing with the shop floor knowledge base to assign required equipment and tools, to select process parameters, and to determine manufacturing costs based on RIOS data. The RSPP module is implemented and demonstrated with a sample part in the context of a request-for-quote (RFQ) scenario.

The rest of the paper is organized as follows. Chapter 2 describes related research. An overview of the integration framework is presented in Chapter 3. Each component illustrated in Chapter 3 is then detailed in Chapter 4, 5, and 6 including the RIOS data specification, the manufacturing resource model, the process knowledge model of the manufacturing site, and the RSPP. An exemplary planning effort using the described framework is presented in Chapter 7. Finally, the conclusions and contributions of this work are discussed.

2. Related Work

The process planning begins uses engineering drawings, specifications, and parts or material lists as inputs. It identifies the machining processes, resources, and cutting parameters necessary to convert the raw materials into finished products using three different approaches – variant, generative, and semi-generative. The collection of software applications and databases needed to execute these functions is called a Computer Aided Process Planning (CAPP) system. In the variant approach, a standard process plan is retrieved from a database and edited if necessary. In the generative, approach, the system creates a new process plan from the input data and extracted manufacturing features. In the semi-generative approach, these two approaches are combined [2].

A number of distributed process planning and process engineering tools has been reported. CyberCut [15] is an internet-based design tool that is integrated with a design-rule checker for machining feasibility. The client logs on using a feature-based WebCAD design interface. The designer starts by defining the workpiece and adding negative features. The design is then submitted for process planning and fabrication, which is currently done manually. This integration avoids the part specification exchange by providing the client with a remote, accessible design interface. However, the clients usually prefer to their own CAD modeling package. In addition, constraining the design to negative features typically limits the designer creativity.

Design Space Colonization (DSC) is an agent-based enterprise integration for the cable production service [3]. The system consists of several agents that exchange data and negotiate with each other. Important agents of this system are the process broker agent and the process capability agents. The process broker agent, who receives the part design from the design agent, locates appropriate, candidate, process capability agents and communicates the required information. The process capability agents represent outside manufacturers who have different process capabilities. The communication in this integration was under (STandard for Exchange of Product data) STEP umbrella. (NEED A REFERENCE TO STEP)

Candadai *et al.* [1] developed an automated vendor selection for the manufacturing of microwave modules. The AIM project at Lockheed Martin [13] has developed several interfaces and forms to establish partners electronically and to implement using STEP a request-for-quote service to facilitate manufacturing activities. So far, this project has achieved only business and legal services using standard Internet protocols, such as (Hyper Text Transfer Protocol) HTTP, (Send Mail Transfer Protocol) SMTP, and (Network News Transfer Protocol) NNTP. The research currently is in the process of setting up the infrastructure, including the supplier-information scheme and associated evaluation, and selection procedures. In addition, the project is also developing an agent-based supplier evaluation and monitoring system based on that infrastructure.

3. Process Planning for Collaborative Manufacturing

3.1. Conceptual Framework

We have decomposed the process planning function for machined parts into seven problems. The first three problems are resource-independent and can be addressed in the conceptual and detailed product design stages. The last four steps, which require information from the first three steps, require resource-specific data; they are performed at the manufacturing vendor site. The paper describes the integration issues of the last four steps.

1) Resolve repeatability requirements, which constrain the grouping of processes to those that must be done without refixturing the workpiece.
2) Resolve geometric tolerance requirements, which constrain the processing sequence to maintain the repeatability and accuracy (e.g., process reference surface first).
3) Resolve precedence constraints based on economic and technological considerations of machining – for instance, performing roughing operations before finishing operations will reduce tool chattering.
4) Analyze producibility. This step involves aggregate level analysis in order to determine and quickly respond with whether the shop has sufficient manufacturing capability to satisfy the requirements specified in the RIOS.
5) Select available resources on which the processes are available to satisfy technological requirements and utilization.

6) Select available processes, and precedence constraint among those processes, to meet shape requirements and accuracy constraints. The result is a unidirectional chain of process operations [18]. For example, when a hole needs to be bored or reamed to obtain required accuracy, it must be drilled first.
7) Generate final linearized plan that specifies exact sequence of processes and resources based on some performance criterion such as to minimize cost and/or time.

The framework used to integrate design and manufacturing activities and implement the RSPP module with the RIOS is illustrated in Figure 1. The process planning system is required to respond automatically to a RFQ from the design house. In addition to the actual RFQ, the client sends the product data as well as the RIOS. The RIOS is represented by a set of process requirements and alternatives organized into a set of hierarchical, AND/OR directed graphs with references to part geometry. These serve as input into the RSPP module, which parses them into object-based entities. Using two repositories, a Manufacturing Resource (MR) model and a Process Knowledge (PK) model, the RSPP module conducts a shop-level producibility analysis. If this analysis shows that the best attainable manufacturing capabilities cannot satisfy the process requirements, then a rejection quote is returned. Otherwise, a detailed planning activity proceeds. It should be noted that the MR and PK models are vendor-specific manufacturing data.

During the detailed planning stage, all possible equipment, processes, and costs are assigned to each manufacturing operation in the RIOS. The PK and MR models are used to make these assignments. Since alternative processes and equipment are possible, the next step is to select the best from among those alternatives and then compute the total, expected, production cost. Chapters 4 to 6 detail each component illustrated in Figure 1.

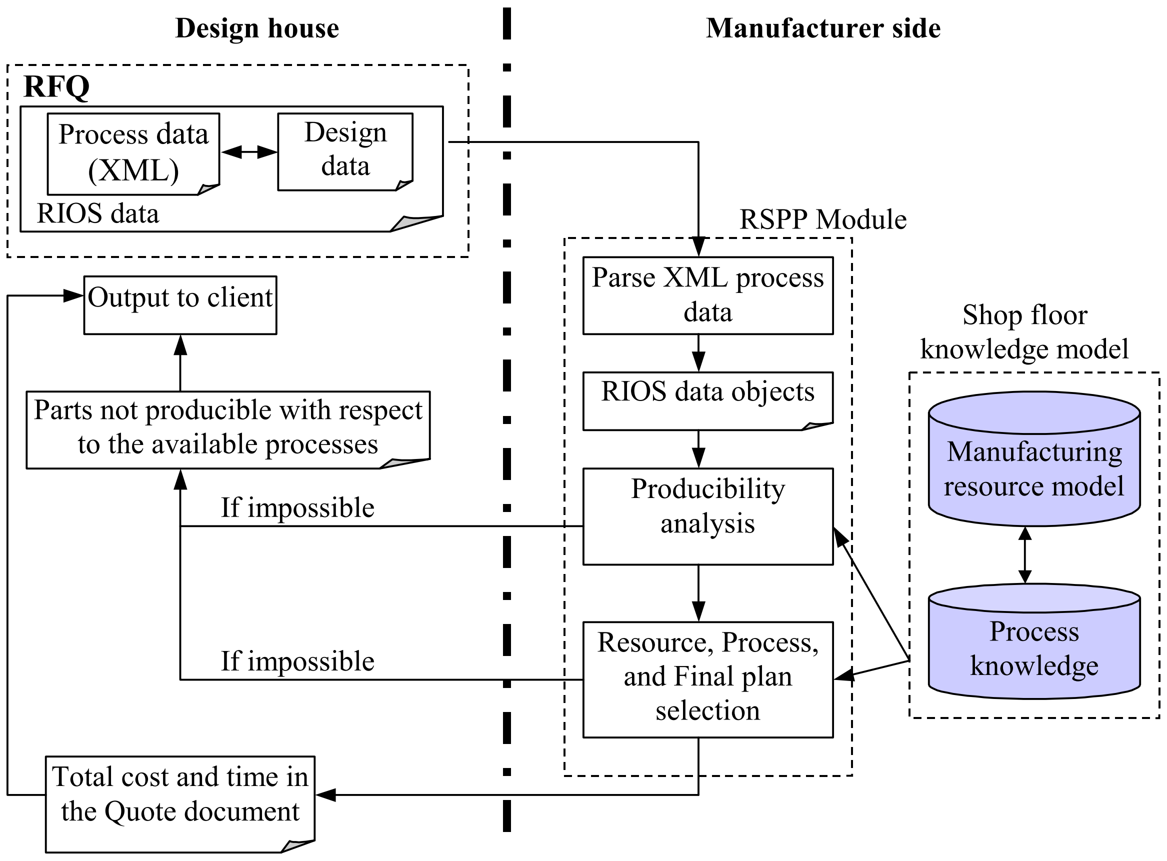


Figure 1: Overview of process planning for collaborative manufacturing

3.2. Implementation Framework

This section describes the process-knowledge service architecture, which the RSPP module utilizes to perform its tasks. Figure 2 shows that the RSPP module functions as a control center utilizing the service provided by the MR model (named as `resourceModel` package in the figure) through the PK model (named as `processknowledge` package) for the process planning. The `processknowledge` package is an interface specification, which decouples the RSPP module from the actual process-knowledge implementation, which is done by the MR model. Thus, the RSPP module can interoperate across process-knowledge implementations that conform to the interface specification. This in turn enables third-party vendors and users to supply process knowledge that might be provided by the original equipment manufacturer (OEM) along with the purchased equipment.

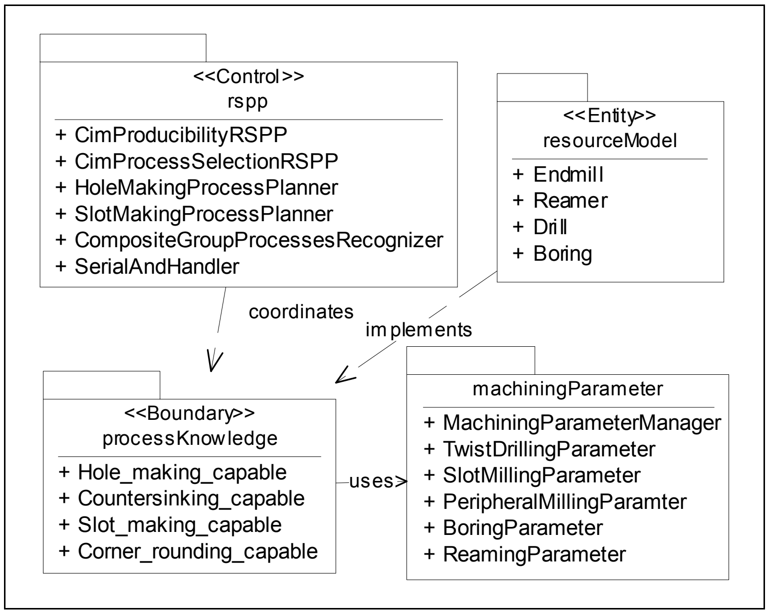


Figure 2: UML package diagram of the RSPP module implementation

In the service architecture, each manufacturing resource provides process knowledge through a set of interfaces that capture its capabilities including shape producing capability, process accuracy, process selection and sequencing, and machining parameter selection. Conceptually, an interface indicates a service or services provided by the object implementing the interface. Theoretically, an interface defines a common set of specifications for any object that commits to provide a service or services. Thus, a set of interfaces defines a service architecture, which consists of service specifications, service providers, and clients. The service architecture provides any service-provider object conforming to the service specification seamless plug-in to the client module, which utilizes the service. In this architecture, the manufacturing resources are service providers since they implement the interfaces, and the RSPP module is the client.

The UML class diagram in Figure 3 demonstrates how the service architecture works with the MR model. Three interfaces (indicated by the boundary class stereotype) are shown in the figure including the `Hole_making_capable`, `Slot_making_capable`, and `Countersinking_capable`. The `Drill` class implements the `Hole_making_capable` and the `Countersinking_capable`. This means that `Drill` and its subtypes have a hole- and countersink-shape-producing capabilities and are committed to provide the hole-making and countersinking services. Consequently, `Drill` must implement the functions defined in the interface specification. For example, the functions `verifyHoleDiaTolPlus(tolVal)` and `verifyHoleDiaTolMinus(tolVal)` indicate that the object implementing the `Hole_making_capable` interface must provides a service, which determines whether it can achieve the specified diametric tolerance (given by the `tolVal` input parameter).

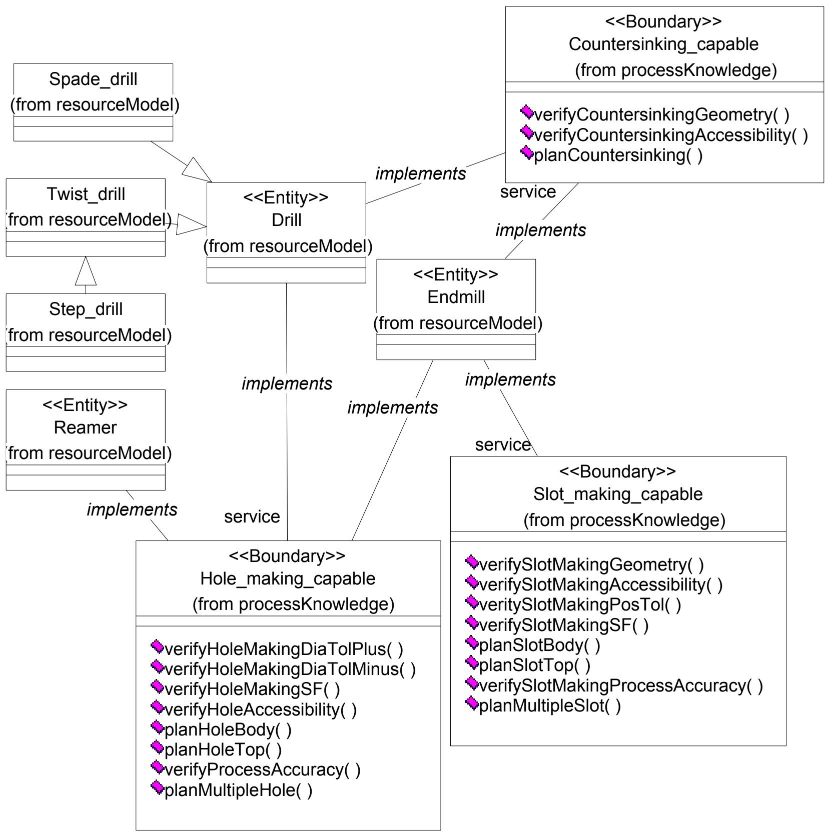


Figure 3: UML class diagram showing the process knowledge interfaces

Since the UML specification does not have specific symbol for the interface, the class stereotype is used as a notation to differentiate the class and interface. The `entity` stereotype is normally used to represent class that is purposely created to hold static information (i.e., data structure and class logic). The `boundary` stereotype typically (1) handles communication between system surroundings and the inside of the system, and (2) provides the interface to a user or another system [14]. Consequently, the `boundary` prototype corresponds to the interface and the verb at the center of the association connection describes the relationship – for examples, the `Drill` implements the `Hole_making_capable`. The text at the end of the association connection indicates the role in the relationship – for example, the `Hole_making_capable` represents a service. The arrow connection represents the subtype relationship.

4. Resource Independent Operation Summary

The RIOS contains order management data, production requirements data, and a two-level process-plan graph. The order management data includes a general client identity and order-tracking data. The production requirement data includes order size, delivery date, and material requirements. These data are not the focus of this paper; it focuses on the process-plan graph, which consists of manufacturing specifications. The upper level graph is called an Operation Level Graph (OLG) and the lower level graph is called a Process Level Graph (PLG). The OLG is a directed AND/OR graph, in which each node describes a type of operation, equipment, work-holding requirements, and a pointer to associated PLG. An operation is viewed as an aggregation of processes where the product specification necessitates that they are executed without refixturing (e.g., due to extremely tight repeatability requirement). The PLG is an *augmented* directed AND/OR graph, in which each node contains process capability requirements such as type of process, required accuracy, and associated geometric entities, which come from the product data. Together, the RIOS forms an integrated product and process definition.

These graphs can represent a variety of plan alternatives. The AND connections lead to sequencing alternatives of which there are two subtypes: serial and parallel. If the serial AND circumscribes two or more nodes, they must be done in sequence yet in any order. If the parallel AND connects two or more nodes, then they must be done but it is possible to execute them simultaneously. The OR connections lead to processing alternatives, in which only one node within the connection shall be executed. It is used, for example, when two or more types of operations can achieve the same requirement. A GROUP connection is also used in the PLG whenever it is possible to combine the tasks into a single execution. This can happen only when the volumetric entities of the tasks within the GROUP are adjacent in the product model – for example, it is possible to combine a hole processing and a counterbore processing into a step drilling.

Typical process plans are generated with specific resources on the shop floor. We regard the RIOS as resource independent, because it (1) is based on process requirements only, (2) is generated from universal-level process knowledge [1], (3) incorporates the principles of process-plan decomposition, and (4) it allows representation of alternative operations and processes. The following paragraphs describe these characteristics in more detail.

The RIOS specifies processing requirements only; it does not state directly a set of tasks to be executed to meet those requirements. However, these requirements could certainly be used to generate such set. The universal-level process knowledge refers to knowledge and practices in common use among professional process engineers and found in manufacturing handbooks [11]. For example, parameters like overall shape-producing capability, workpiece materials, overall geometry, and production size can indicate the preferred type of manufacturing processes – casting, machining, stamping, and so on. Typically, these parameters are determined at the conceptual design stage, where it is impossible to identify specific manufacturing equipment. The use of conceptual design data and universal level-process knowledge to select process requirements can be found at the Manufacturing Advisory Service web site [17]. An exemplary construction of RIOS data is presented in Section 7.1.

5. Shop Floor Knowledge

We now describe the MR and PK models, which contain the vendor-specific shop floor knowledge base that is required for the RSPP module to generate a Resource Specific Plan (RSP) from the RIOS.

5.1. Manufacturing Resource (MR) Model

The MR model characterizes the vendor-specific shop floor resources that could be used to fabricate the requested product. Generally, the MR model must represent the technological and economical specifications for the resources. The technological specifications of machine tool may include table size, repeatability, and capacity. The economical specifications of machine tool may include the utilization cost, set-up cost, tool life, maintenance schedule and operating status [10]. Figure 4 illustrates the top-level schema for a typical set of manufacturing resource entities (extended from [6]). The relation between `Milling_machine` and `Machine_tool` represents an '`is-a`' relationship. The relation between `Milling_machine` and `Tool_magazine` represents an '`owns-a`' relationship. The relation between `Milling_machine` and `Milling_spindle` is also an '`own-a`' relationship; however, a `Milling_machine` may own several `Milling_spindle`'s. All other attributes associated with an entity, which is of simple data type (not of entity type), are said to have '`has-a`' relationships with the entity. We note that the term 'entity' refers to the generic definition, while the term 'instance' refers to a particular realization of an entity.

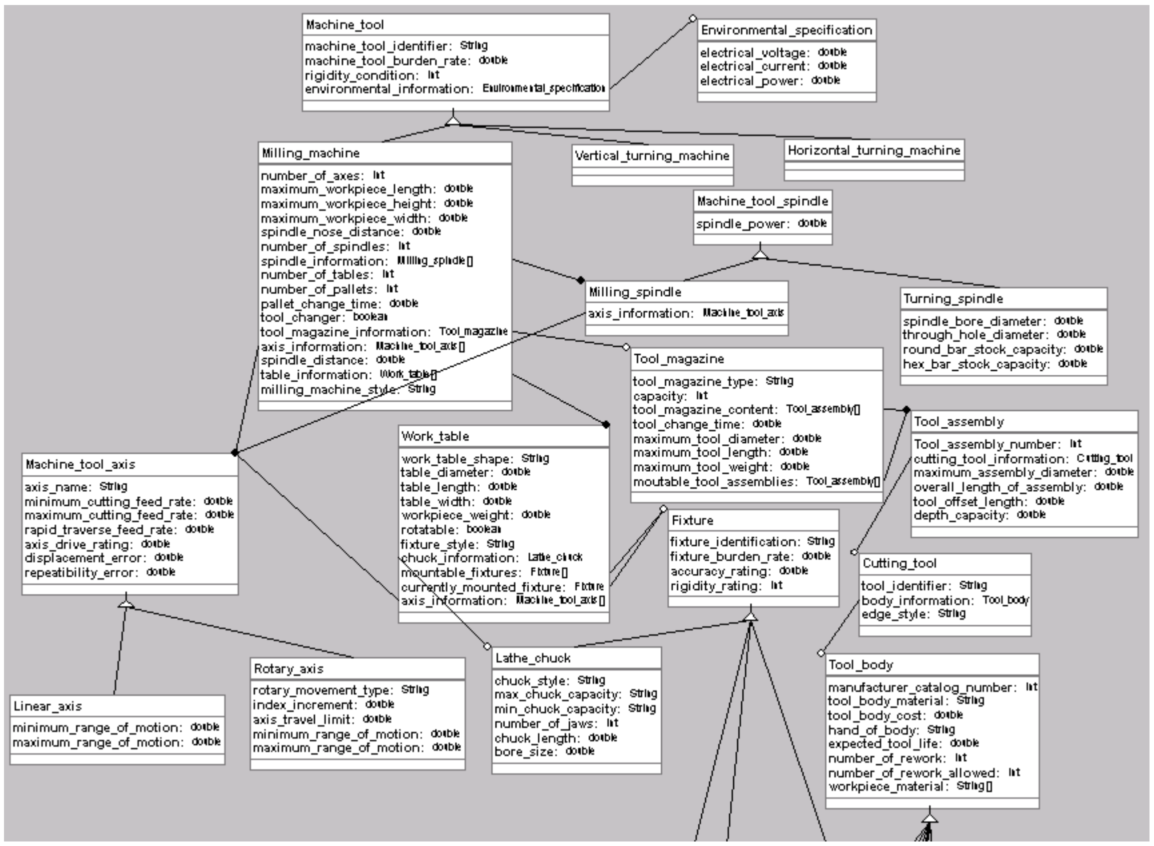

Machine_tool
machine_tool_identifier: String
machine_tool_burden_rate: double
rigidity_condition: int
environmental_information: Environmental_specification
Environmental_specification
electrical_voltage: double
electrical_current: double
electrical_power: double
Milling_machine
number_of_axes: int
maximum_workpiece_length: double
maximum_workpiece_height: double
maximum_workpiece_width: double
spindle_nose_distance: double
number_of_spindles: int
spindle_information: Milling_spindle[]
number_of_tables: int
number_of_pallets: int
pallet_change_time: double
tool_changer: boolean
tool_magazine_information: Tool_magazine
axis_information: Machine_tool_axis[]
spindle_distance: double
table_information: Work_table[]
milling_machine_style: String
Vertical_turning_machine
Horizontal_turning_machine
Machine_tool_spindle
spindle_power: double
Milling_spindle
axis_information: Machine_tool_axis
Turning_spindle
spindle_bore_diameter: double
through_hole_diameter: double
round_bar_stock_capacity: double
hex_bar_stock_capacity: double
Tool_magazine
tool_magazine_type: String
capacity: int
tool_magazine_content: Tool_assembly[]
tool_change_time: double
maximum_tool_diameter: double
maximum_tool_length: double
maximum_tool_weight: double
moutable_tool_assemblies: Tool_assembly[]
Tool_assembly
Tool_assembly_number: int
cutting_tool_information: Cutting_tool
maximum_assembly_diameter: double
overall_length_of_assembly: double
tool_offset_length: double
depth_capacity: double
Machine_tool_axis
axis_name: String
minimum_cutting_feed_rate: double
maximum_cutting_feed_rate: double
rapid_traverse_feed_rate: double
axis_drive_rating: double
displacement_error: double
repeatibility_error: double
Work_table
work_table_shape: String
table_diameter: double
table_length: double
table_width: double
workpiece_weight: double
rotatable: boolean
fixture_style: String
chuck_information: Lathe_chuck
mountable_fixtures: Fixture[]
currently_mounted_fixture: Fixture
axis_information: Machine_tool_axis[]
Fixture
fixture_identification: String
fixture_burden_rate: double
accuracy_rating: double
rigidity_rating: int
Cutting_tool
tool_identifier: String
body_information: Tool_body
edge_style: String
Linear_axis
minimum_range_of_motion: double
maximum_range_of_motion: double
Rotary_axis
rotary_movement_type: String
index_increment: double
axis_travel_limit: double
minimum_range_of_motion: double
maximum_range_of_motion: double
Lathe_chuck
chuck_style: String
max_chuck_capacity: String
min_chuck_capacity: String
number_of_jaws: int
chuck_length: double
bore_size: double
Tool_body
manufacturer_catalog_number: int
tool_body_material: String
tool_body_cost: double
hand_of_body: String
expected_tool_life: double
number_of_rework: int
number_of_rework_allowed: int
workpiece_material: String[]

Figure 4: Top-level entities in the manufacturing resource schema

There are number of possible relationships between manufacturing resources. To distinguish among these relationships, we developed a set of predicates that formally define the semantics of each relationship as shown in Table 1. In addition, each manufacturing resource provides process knowledge associated with it by implementing a set of interfaces as described in Section 3.2. Table 1 also includes predicates that define the semantics of relationships between resources and interfaces. These predicates are necessary to make inferences about the resource capabilities associated with the interfaces and ownership relations. They also enable the top-down producibility analysis as well as resource and process selections mechanism. We note that the Resource is the highest-level object in the MR model’s universe of discourse. Thus, every object in the MR schema is a subtype of the Resource.

Table 1: Predicates defining the semantics of resource relationships

Predicate	Description
Resource(x1), Resource(x2), Resource(x3), Resource(x4), Resource(x5)	x1, x2, x3, x4, and x5 are instances of the Resource object.
Attribute(a1), Attribute(a2)	a1 as well as a2 is an Attribute.
HasAttr(x1, a1)	Resource x1 has attribute a1.
HasAttrVal(x1, a1, c1)	Attribute a1 of resource x1 has value c1.
Is(x1, x2)	Resource x1 is a subtype of resource x2 ('is-a' relationship).
HasAttrVal(x1, a1, c1) → HasAttr(x1, a1)	If attribute a1 of resource x1 has a value of c1, then it is true that resource x1 has attribute a1.
Owns(x1, x4)	Resource x1 owns resource x4 ('owns-a' relationship).
Owns(x1, x4) and HasAttrVal(x4, a2, c2) → HasAttrVal (x1, a2, c2)	If resource x1 owns resource x4 and attribute a2 of resource x4 has value c2, then resource x1 also has attribute a2 of value c2. This implies that an owner of a resource posses its capability.
Is(x1, x2) and HasAttr(x2, a1) → HasAttr(x1, a1)	If resource x1 is a subtype of resource x2, then it inherits the attribute of x2 (object inheritance).
Is(x1, x2) and Is(x2, x3) → Is(x1, x3)	The 'is-a' relationship is transitive.
Owns(x1, x4) and Owns(x4, x5) → Owns(x1, x5)	The 'owns-a' relationship is transitive.
Interface(y1), Interface(y2)	y1 and y2 are interfaces.
Is(x1, x2) and Implements(x2, y1) → Implements(x1, y1)	If resource x1 is a subtype of resource x2 which implements interface y1, then resource x1 also implements interface y1.
Implements(x2, y1) ↔ ProvidesService(x2, y1)	If resource x2 implements interface y1, then it is implied that resource x2 provides service y1. The reverse is also true. Note this is to say that Implements means ProvidesService.
Owns(x1, x3) and Implements(x3, y2) → ProvidesService(x1, y2)	If resource x1 owns resource x3 and x3 implements interface y2, then it implies that owner x1 also provides service y2.

5.2. Process Knowledge (PK) Model

Process knowledge captures capabilities and utilizations of each resource including shape producing capability, process accuracy, process selection and sequencing, and machining parameter selection.

5.2.1. Shape Producing Capability Knowledge

The shape-producing capability indicates the types of RIOS processes the equipment can produce. In this research, a resource indicates its shape producing capabilities by implementing interfaces as illustrated in Section 3.2.

5.2.2. Process Accuracy Knowledge

The process-accuracy indicates the level of precision capability of the process. In particular, one may use process bounds to represent process accuracy knowledge. For example, the process bounds used for the hole-making process and the slot-making process may follow the research results from [7] and [18], respectively. These bounds give ranges for the attainable accuracy of these processes.

Ferreira *et al.* [4] suggested that the ownership relations signify the capabilities of the owner. For example, if a milling machine owns a 0.5 cm twist drill (i.e., the twist drill can be mounted on the milling machine and perform its task) then it can drill a hole with diameter 0.5 cm. The relationship suggests that the process knowledge as well as resource knowledge may be classified into *individual facts* and *relational facts*. A relational fact is the data resulting from the combination of resources. In a flexible manufacturing system, process accuracy is typically affected by relations between resources because the same tool mounting on different machines can result in different process accuracy. For example, relational facts between a particular milling machine and a particular carbide drill bit may be the diametric accuracy capability, the positioning accuracy, among thers. The relational facts help build the knowledge base of the flexible manufacturing system. The facts are characterized along with the ownership relations between resources. The individual fact is opposite. It is not changed when resources are used in combination with other resources. An exemplary individual fact about process is its economical utilization.

5.2.3. Process Selection and Sequencing

The process selection and sequencing capture the unidirectional chain of processes and the economical utilization of processes. The unidirectional chain of processes, which is derived from the process accuracy requirement, may be captured in a decision tree along with the process accuracy as described in [2]. In this research, the unidirectional chain of process is captured as a self-contained knowledge of each process. For example, if it is determined that a `Reamer`, which implements `Hole_making_capable`, is required to achieve the accuracy requirements, then the `Reamer` itself provides the unidirectional chain of processes (e.g., drilling first, semi-finished reaming, and then finished reaming). The economical utilization of process can be determined directly from the cost and/or time. However, this can be exhaustive. In this research, the economical utilization of process is performed in two stages. First, a heuristic is used to indicate the preferred process (as well as resource) for each node in the process-plan graph; then the cost/time rational is used at the final plan generation to linearize the graph.

5.2.4. Machinability Data

Machinability data, typically from a reference handbook, specifies properties of materials with respect to specific cutting and forming processes. For example, recommended machining speeds, cutting depths, feed rate, and coolant choice will be specified when using particular cutters to machine particular types of materials. In some cases, characterizations of thermal effects, surface finishes, and chaff/chip formations are included. The machinability data can be declarative (e.g., machining data handbook), procedural (machining parameter optimization) [18], a mix of declarative and procedural [16], based on the experience of machinists, or supplied with the equipment by the manufacturer.

6. Resource-Specific Process Planning

This chapter describes the core software components implemented in the RSPP with a special emphasis on the object-oriented architecture that binds them together. These software components are the *XML parser*, the *XML object packages*, and the process planning-related classes. These components provide a basis for the integration of process planning between a client side and a server side. Typically, each manufacturing partner implements these components by extending (subclassing) the utility classes and customizing a set of defined procedures using its own shop floor knowledge. The discussion in this section follows the subcomponents of the RSPP module illustrated in Figure 1.

6.1. XML RIOS Data Objects

Because the RIOS data from the design house are encoded in XML, an XML parser is necessary. A parser, which converts XML data elements into self-contained objects, allows software components to exploit object-oriented features such as object abstraction and object polymorphism. Such parser framework (available at [5]) allows the RSPP module to be reusable. To use this parsing framework, we defined Java classes for each type of data element in the information models. Figure 5 shows that all the classes are organized into XML Object Packages with respect to the RIOS information schemas including the graph -- operation, process, and support. The operation and the process schemas define the process-plan schemas for the operation- and the process-level graphs. The support schemas define low-level entities for the operation and process requirements (such as measurement items).

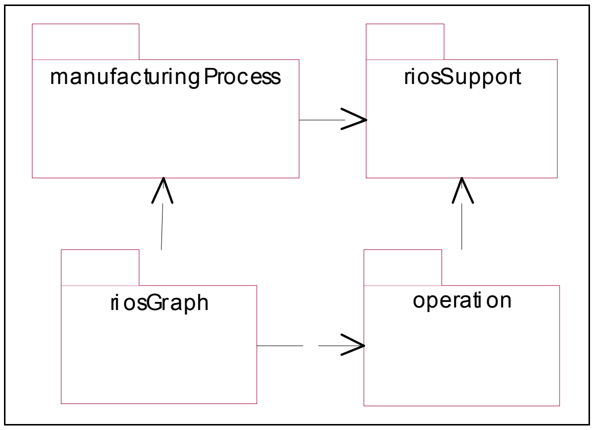


Figure 5: UML package diagram of XML object packages

Figure 6 shows a small portion of class diagram within the `RIOSGraph` package -- where all of the graph- related elements are defined. The `traverseGraph` method was defined only once in the `DirecteGraph` class for all `DirectedGraph` subtypes (object abstraction). Alternatively, the subclass can override or add more definition to this behavior (object polymorphism).

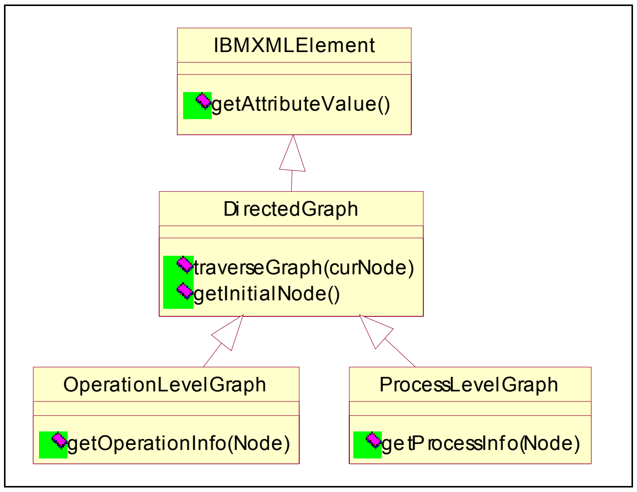


Figure 6: UML class diagram showing examples of XML object definitions

6.2. Producibility Analysis

The producibility analysis conducts an aggregate analysis of a vendor's manufacturing capabilities for a quick response to a RFQ from the design house. The class diagram in Figure 7 shows the object architecture used for the producibility analysis. The `ProducibilityRSPP` class utilizes the MR and PK models described earlier to evaluate manufacturability and to respond to the RFQ. It handles the aggregate pass of manufacturability analysis. Nevertheless, manufacturing partners must supply their own process knowledge and decision-making procedures through the abstract methods defined within the class. They do this by subclassing the class and implementing those abstract methods (marked by the 'key' icons). A particular manufacturing partner may subclass the class as `CimProducibilityRSPP` and define the associated abstract methods. The producibility analysis is executed in two levels: the operation level and the process level. The operation level analysis considers the operation type and work holding, while the process level analysis consider the process type, size, and best attainable accuracy.

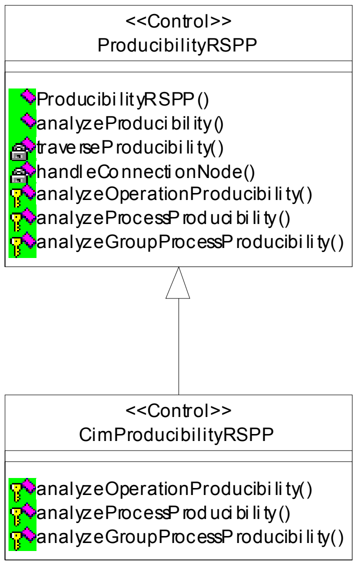

<<Control>>
ProducibilityRSPP
ProducibilityRSPP()
analyzeProducibility()
traverseProducibility()
handleConnectionNode()
analyzeOperationProducibility()
analyzeProcessProducibility()
analyzeGroupProcessProducibility()
<<Control>>
CimProducibilityRSPP
analyzeOperationProducibility()
analyzeProcessProducibility()
analyzeGroupProcessProducibility()

Figure 7: UML class diagram for producibility analysis

The operation-level procedures must be encoded in the `analyzeOperationProducibility()` method where the `CimProducibilityRSPP` class inherits from the `ProducibilityRSPP` class. The `traverseProducibility()` method will invoke this method on each node of the OLG. Similarly, the process-level producibility procedures must be encoded in the `analyzeProcessProducibility()` method. The `traverseProducibility()` method will invoke this method on each node of the PLG.

A set of predicates included in Table 2 formally describes the functional requirements for producibility analysis in the abstract level. The predicates defined in Table 1 are used in this table. The semantics of these predicates must be captured within the `analyzeOperationProducibility()` and the `analyzeProcessProducibility()`.

Table 2: Producibility analysis predicates

Predicate	Description
∃xOperationType(x) → ∃y[Machine_tool(y) and isCapableOf(y,x)]	If there exists a requirement for operation `x`, there must exist machine tool `y` that is capable of performing operation `x`.
∃xWorkpiece(x) → ∃y∃z[Machine(y) and Fixture(z) and Owns(y,z) and CanHold(y,z,x)]	If there exists a requirement for workpiece `x`, there must exist a machine y that owns fixture `z`. The combination of machine `y` and fixture `z` must be able to hold workpiece `x`.
Process(x)	`X` is any type of material removal process.
GeometricRqmt(x)	A set of geometric requirements of process `x`.
TopologicalRqmt(x)	A set of topological requirements of process `x`.
ProcessAccuracyRqmt(x)	A set of process accuracy requirement of process `x`.
∃xProcess(x) → ∃y∃z[Machine_tool(y) and Tool_body(z) and Owns(y,z) and satisfiesGeometricRqmt (y,z,GeometricRqmet(x)) and satisfiesProcessAccuracyRqmt (y, z, TopologicalRqmt(x)) and satisfiesProcessAccuracyRqmt (y,z,ProcessAccuracyRqmt(x))]	If there exists a requirement for process `x`, there must exist machine tool `y` that owns tool body `z`. The combination of machine tool y and tool body `z` must satisfy the geometric requirements, topological requirements, and process accuracy requirements of process `x`.

6.3. Resource Selection

Resources are selected by matching the resource specifications with respect to the requirements in the OLG. The first two predicates in Table 2 capture this semantics. The class diagram in Figure 8 shows the object architecture for the resource selection (as well as process selection described in the next section). Similar to that described in Section 6.2, the abstract functions, which require vendor specific implementations, are defined in the `ProcessSelectionRSPP` and must be implemented in the subclass (e.g., the `CIMProcessSelectionRSPP` class). The function `getFeasibleMachines()` is executed on each node of the OLG to provide a list of possible machines for the process selection step. The list may be ordered with respect to a set of conflict resolution strategies. Examples of such strategies are minimum resource utilization and minimum utilization cost. These strategies may be ranked differently by each vendor. In addition, this ranking may change over time. For instance, equipment with lower resource utilization may be ranked higher during heavier production periods to produce more and meet customer demand. On the other hand, equipment with lower utilization cost may be ranked higher during lighter production periods.

6.4. Process Selection

A partial list of the abstract functions used for the process selection is listed in Figure 8. For instance, the `assignProcessPlanAndCost()` is invoked on every process node on the process-level graph, and the `assignGroupProcessPlanAndCost()` is invoked on every GROUP connection. The manufacturing vendor provides the implementations of these functions through the `ProcessSelectionRSPP` subclass (e.g., `CimProcessSelectionRSPP`) using its shop floor knowledge.

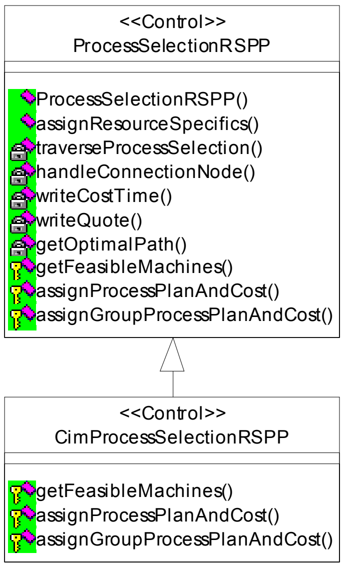


Figure 8: UML class diagram for resource, process, and final plan selections

The implementation in this research creates helper classes including planners, handlers, and recognizers to provide specific functionalities for the process selection. A planner class, aggregates a set of services to achieve a specific planning task, is associated with each process type. Handler and recognizer classes are created to facilitate the AND/OR graph linearization. The process-selection procedure calls appropriate planners, handlers, and recognizers with respect to node types and the process requirements specified in each node of the PLG. A collaboration diagram in Figure

9 illustrates examples of helper classes including the `HoleMakingProcessPlanner`, `SlotMakingProcessPlanner`, `SerialAndHandler`, and `CompositeGroupProcessRecoginzer`. In this diagram, the interactions between classes are illustrated by request-for-assistance messages passing between them. For example, the `CimProcessSelection` ask the `HomeMakingProcessPlanner` to plan hole (message #1) when a `Hole_making_process` is encountered in the PLG. Note that the planner classes utilize the process knowledge through the interface specifications (e.g., `Hole_making_capable`), which are implemented by the manufacturing resources.

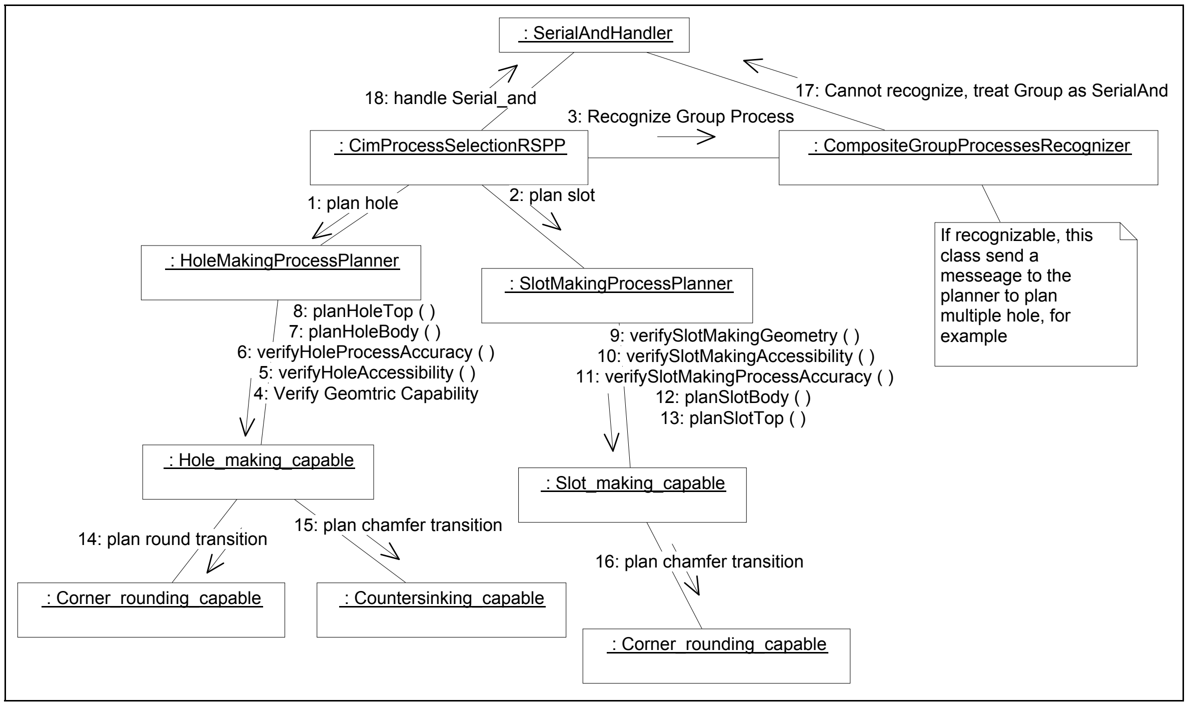

: SerialAndHandler
18: handle Serial_and
17: Cannot recognize, treat Group as SerialAnd
3: Recognize Group Process
: CimProcessSelectionRSPP
: CompositeGroupProcessesRecognizer
1: plan hole
2: plan slot
If recognizable, this class send a messeage to the planner to plan multiple hole, for example
: HoleMakingProcessPlanner
: SlotMakingProcessPlanner
8: planHoleTop ( )
7: planHoleBody ( )
6: verifyHoleProcessAccuracy ( )
5: verifyHoleAccessibility ( )
4: Verify Geomtric Capability
9: verifySlotMakingGeometry ( )
10: verifySlotMakingAccessibility ( )
11: verifySlotMakingProcessAccuracy ( )
12: planSlotBody ( )
13: planSlotTop ( )
: Hole_making_capable
: Slot_making_capable
14: plan round transition
15: plan chamfer transition
16: plan chamfer transition
: Corner_rounding_capable
: Countersinking_capable
: Corner_rounding_capable

Figure 9: UML collaboration diagram between control classes and boundary classes

The process selection traverses each process node in the PLG to select a process or processes and assign a cost. The analysis examines the process capability of each machine with respect to the ordered list of feasible machines received from the resource selection. The `CimProcessSelectionRSPP` coordinates the analysis through planners (e.g., message #1 and #2 in Figure 9), which are subordinate controllers. The planners in turn issue messages to request process-capability verification from the PK interfaces. They use the results to select appropriate processes and resources -- a conflict-resolution or a vendor-specified order of preference is used when multiple candidate processes are available. For example, an ordering of twist drill, end mill, reamer, and boring may be specified whenever `Hole_making_capable` is invoked. This means that the twist drill will always be selected first (because of its cost and time efficiency), if it is capable of achieving the process requirements. Suppose a reamer is selected for a hole, the planner then makes a planning-service request -- a call to the `planHoleBody()` method would be made to generate the necessary processing sequences and associated costs -- from the `Hole_making_capable` interface of the reamer. The interfaces may send messages among themselves to request auxiliary services. For instance, the `Hole_making_capable` may need to send a request to the `Countersinking_capable` to plan for chamfer on the top.

When a `Serial_and` connection is encountered, the `CimProcessSelectionRSPP` delegates the task to the `SerialAndHandler`. The `SerialAndHandler` tasks are to find out if an intersection exists and to linearize the connection -- an intersection between a pair of processes occurs whenever the surfaces generated by the processes share at least a common edge. Since there might be an intersection between the processes within the `Serial_and` connection, the information related to the removal volume may be changed after the linearization.. When this happens, the related volume-removal information must be modified; this can be a complicated task, which depends upon how the processes are linearized. For example, the volume-removal information associated with successive `Hole_making_process` operations is obtained by compensating the body depth. Figure 10 illustrates this example. Finding the optimal linearization of the `Serial_and` connection can be exhaustive, so is typically rule-based. For example, a simulation study [18] showed that it is more cost effective to make a step hole by starting from the largest diameter. However, other considerations may affect the rule. For instance, the most cost-effective sequence may reduce the accuracy or it may subject to power or geometric constraint.

The `CimProcessSelectionRSPP` also asks the `CompositeGroupProcessRecognizer` to determine if multiple processes within a GROUP can be achieved in a single sequence. A graph-based pattern-matching algorithm can be used to handle this task. If a pattern matches, the recognizer asks the respective planner to determine whether resources exist to achieve the desired requirements and to plan accordingly (e.g., the `planMultipleHole()` method in the `HoleMakingProcessPlanner` class). If the recognizer determines that there is no matching pattern or there is no applicable resource for the matched pattern, the GROUP connection is treated as a `Serial_and` connection, which sends the control back to the `SerialAndHandler`.

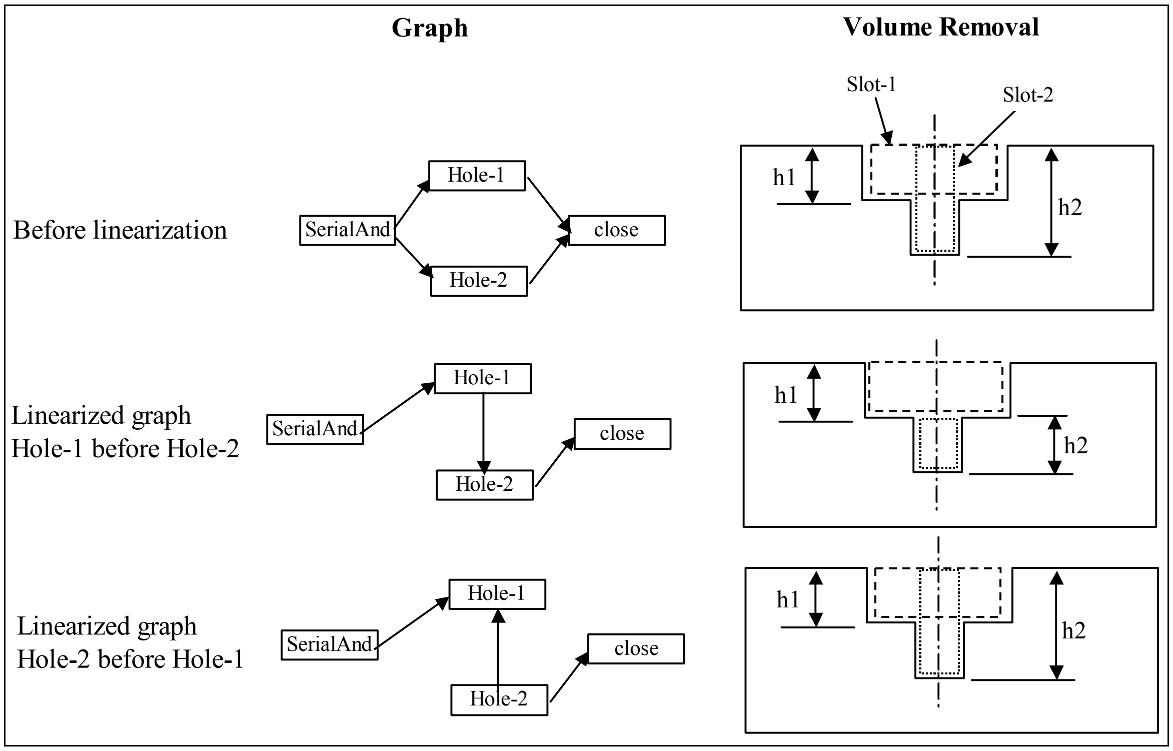
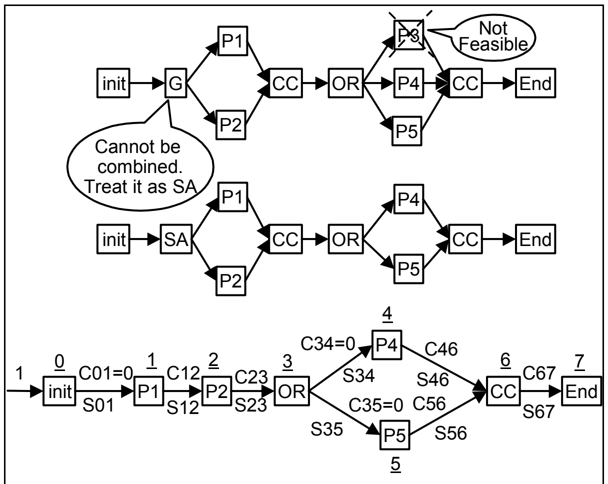


Figure 10: Effect of the graph linearization to the volume removal information

6.5. Final Plan Selection

The final plan selection includes the procedure to linearize the process-plan graph. The procedure for this is readily provided in `getOptiomalPath()` function within the `ProcessSelectionRSPP` class. When the process selection finishes traversing the PLG, only the OR connections remain. At this step, the graph forms the basis for a network-flow problem, which can be formulated and solved using linear programming -- Figure 11 illustrates the problem formulation. The total cost of each PLG is assigned to the associated node in the operation level. The OLG is then linearized to obtain the total cost.



$$\textit{Objective function} \quad \textit{Min} \quad \sum_{i=0}^{n-1}\sum_{j=1}^{n} S_{ij}C_{ij}$$

$$\textit{Subject to} \qquad S_{01} = 1$$

$$\sum_{i=0}^{n-1} S_{ij} - \sum_{k=1}^{n} S_{jk} = 0 \qquad \textit{all } j = 1 \textit{ to } n$$

$$\textit{where } S_{ij} = \text{flow from node i to j}$$

$$C_{ij} = \begin{cases} \text{cost from node i to j} = \text{cost of node i.} \\ \text{cost of Connection node} = 0. \end{cases}$$

Figure 11: Graph linearization

7. Planning Example using RIOS Data

7.1. Example RIOS Data

Figure 12 illustrates a 2.5D prismatic part with a step slot and with all dimensions in inches. Since the design is 2.5D and Aluminum has an excellent formability, extrusion seems to be a good candidate process. However, the production size, which is 1000 pieces, might be too small. Therefore, a material-removal operation should be considered as well. It is universally true that these two types of operations have a capability to accommodate the workpiece size and accuracy specification. While casting might be another candidate, mold cost and the finishing requirement exclude it from consideration.

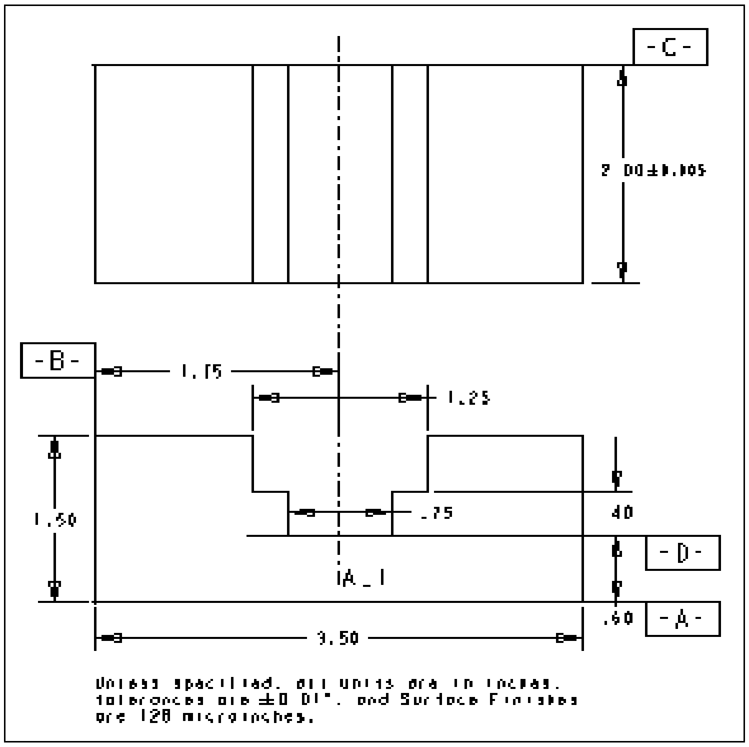

-C-
2.00±0.005
-B-
1.75
1.25
1.50
.75
.40
-D-
A_1
.60
-A-
3.50
Unless specified, all units are in inches.
tolerances are ±0.01", and Surface Finishes
are 128 microinches.

Figure 12: Example Part

The OLG is shown in Figure 13, which consists of one material removal operation and one extrusion operation. Since one setup is sufficient for both topological and process-accuracy constraints, only one removal operation is needed. The tolerance chart for material removal operation in Figure 14 shows that there is one tolerance stacking in equation (2) causing `M01` and `M02` to have a smaller allowance. However, if `M01` and `M02` are equally allocated with 0.01" allowance (where `C02` = 0.02"), this tolerance is still achieved easily by a milling process. Similar analysis can be performed for the extrusion operation (`OP2`), but we have not included the results.

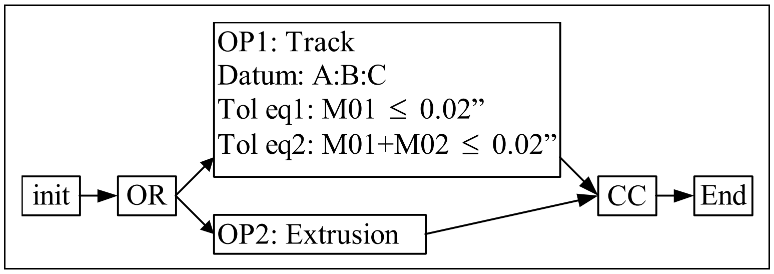


Figure 13: Operation level graph

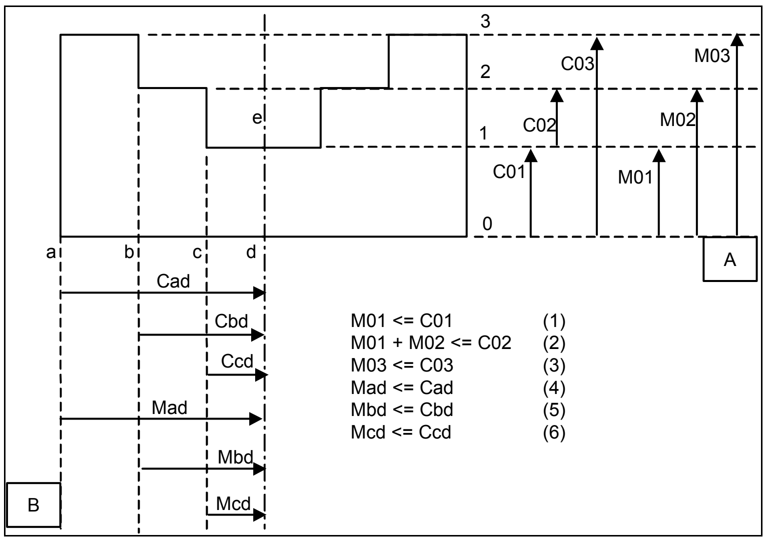


Figure 14: Tolerance analysis chart

Figure 15 shows the PLG of `OP1`, namely `PLG1`. `PLG1` consists of two `Slot_making_process` nodes within a `GROUP` connection. In this particular instance, the `GROUP` connection indicates that the two processes can be executed in any order and can be combined into a single sequence using composite tool geometry. Note, as indicated earlier, that the removal volumes of the processes within the `GROUP` connection are adjacent.

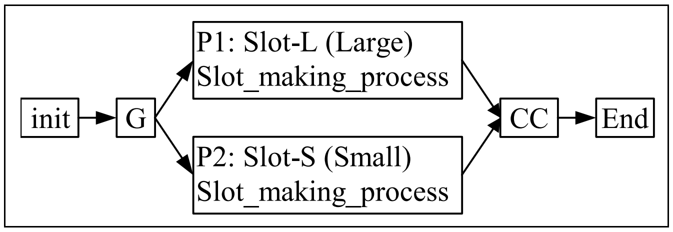


Figure 15: Process level graph for operation OP1, PLG1

7.2. Producibility Evaluation and Process Selection

This section illustrates the shop floor knowledge and the RSPP module working together through the fabrication laboratory at Penn State University, CIM Lab, and the example described in Section 7.1 with the focus on `OP1`. CIM Lab has two Haas vertical milling machines -- Haas VF-OE (3-axis) and Haas VF-3B (5-axis). A number of fixtures and tools are associated with these two machines; these tools and fixtures are discussed as necessary.

The example in Figure 16 illustrates an operation-level producibility analysis. The relational capability between the fixture and Haas VF-OE must comply with the workpiece holding size and the fixture requirements. In this example, the workpiece width (W), the workpiece length (L), the vise's maximum open distance (VW), and the machine's maximum workpiece width (C), must satisfy one of the following two conditions: [W < VW and L < C] or [W < C and L < VW]. In addition, the stacking up of workpiece height (H), the vise's bottom thickness (VT), and the machine's maximum workpiece height (A) must satisfy H + VT < A. In this example, both of the machines satisfy the constraints in `OP1`. Hence, both are forwarded to the process-level producibility analysis. This analysis traverses through each process node in the PLG graph to determine whether at least one of the machines possesses process capabilities that satisfy the process requirements associated with the process nodes. An example process level producibility analysis of `PLG1`, particularly `P1`, is given in detail in Table 3. The resource and process selections occur in a similar way with an exception that the connection nodes are linearized and the processes and cost/time estimations are assigned to each node. The final plan is then selected and a quote is sent to the design house. If the design is not producible, the manufacturing vendor may reply with the RIOS graph, in which each non-manufacturable node is marked.

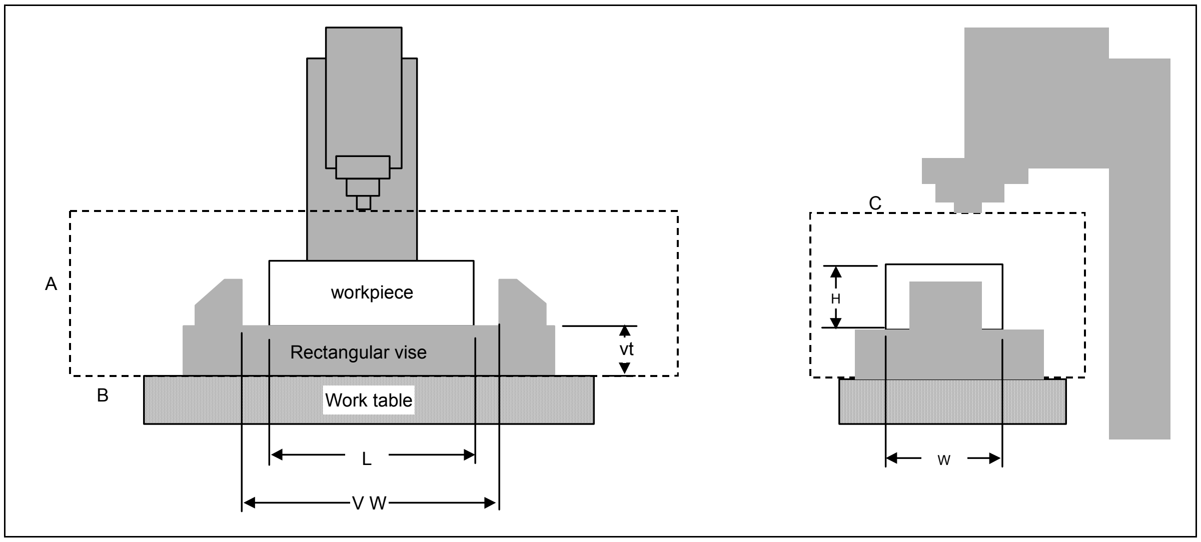


Figure 16: Example of operation level producibility analysis, the work holding requirement

Table 3: Example data of PLG1 and its process level producibility analysis

Data name	Value	Producibility Analysis Discussion
Process_level_graph	id = Door_Track_PLG1, init_node_id = node1.1	Call analyzeProcessProducibility( )
Init_node	node_id = node1.1	Go to next node
Group_node	node_id = node1.2, prev_node_id = node1.1	Call analyzeGroupProcessProducibility( ). This group of processes is recognized as a step slot, however, there is no step mill in the inventory that fits the geometric requirements. The two processes are processed separately.
Manufacturing_process_node	node_id = node1.3, prev_node_id = node1.2, process_name = Slot-L	Suppose current production status indicates that lower cost criteria has higher priority. In this case, the Hass VF-OE is preferable to the Hass VF-3B.
Slot_making_process	Slot-L	If a tool complies with all of the requirements then the node is producible; otherwise, it is not producible. Suppose an endmill of 0.5" diameter is being evaluated.
>Processing_material	Aluminum	The tool is checked if the material is within the roster of materials, which this tool is capable of removing.
>slot_shape_information	Rectangular_slot	Call verifySlotGeometricCapacity( ) where tool diameter and the depth capacity are checked against these shape constraints. In this example, the endmill's diameter and depth capacity are 0.5" (< 1.25" width) and 2.0" (> 0.5" depth), respectively. Therefore, it passes the geometric capacity verification. Call verifySlotProcessAccuracy( ) where all the process accuracy requirements are checked. In this example the endmill dimensional capability is +/- 0.0005 and positional tolerance is up to 0.001 and the surface finish is up to 30 microinches. The tolerance of body_width and body_depth (in the worst case where M02 is allocated 0.001, and M01 has more than enough to satisfy C02) are larger than the dimensional capability. The surface finish capability also satisfies the SF_side and SF_bottom. This is also true with the positioning tolerance; therefore, this endmill satisfies the process accuracy requirements.
>body_width	Toleranced_length_measure 1.25"+/- 0.01 w.r.t. Datum A_1	
>body_depth_of_size	Length_measure: 0.5"	
>positioning	Toleranced_length_measure 1.75" +/- 0.01 w.r.t Datum B	
>orientation_tolerance	None	
>bottom_information	Flat_slot_bottom_condition(No bottom_transition)	
>>body_depth_tolerance_plus	Length_measure: M02/2	
>>body_depth_tolerance_minus	Length_measure: M02/2	
>>resulting_bottom_face	Surface_id = 52	
>>SF_bottom	Surface_finish = 128 microinches	
>resulting_side_faces	Surface_id = 42, 54	
>Slot_side_form_tolerance	None	
>SF_side	Surface_finish = 128 microinches	
>placement_surface	Surface_id = 40	Call verifySlotTopologicalCapability( ) where the tool offset length is checked if it is long enough to accommodate the access distance plus the slot depth. In addition, the offset length is also checked against the workholding assembly whether there is enough space for the tool after the workpiece and fixture assembly. The endmill offset length is 3.3125". In the operation level example, there is still more than enough (21.5") space available after the workpiece and fixture assembly. The placement_surface and ends_information are also checked if any required pre-condition or post-condition can be met for the tool to remove the material and produce the required end condition. In this case, the slot has both ends through and the beginning_surface data below indicates no blocking, therefore no plunging or post process is required.
>access_distance	Basic_length_measure: 0 w.r.t Datum AA (opposite to Datum A)	
>ends_information	Through_end_condition (without transition)	
>top_transition	None	Request service from Countersinking_capable or Corner_rounding_capable resources as necessary.
>sweeping_information	Sweeping_path	The sweeping_information is used for cost calculation purpose.
>>sweeping_type	Linear_path	
>>beginning_surface	Surface_id = 23	
>>sweeping_distance	2.0	
>>Manufacturing_process_node	node_id = node1.4, prev_node_id = node1.2, process_name = Slot-S	This node is also a Slot_making_process node. Therefore, the producibility analysis follows the same procedure as described above.
Slot_making_process	Slot-S	The detail of this process is similar to the Slot-L process and will be omitted from this table.
Connection_close_node	node_id = node1.5, prev_node_id = node1.3 node1.4	Go to next node
End_node	node_id = node1.6, prev_node_id = node1.5	This process level graph is producible; thus, the OP1 is producible.

8. Conclusion

This paper described an integration framework for process planning to facilitate collaborative product development activities such as design and manufacturing. The framework uses an Integrated Product and Process Data (IPPD) representation called a Resource Independent Operation Summary (RIOS) to facilitate the communication of business, product, process, and production requirements. Hierarchical process modeling, process planning decomposition, and AND/OR directed graphs form the basis for this representation. We described the shop floor knowledge architecture and requirements to support the planning task from the RIOS input. A hybrid approach, based on a heuristic and linear programming, was used to linearize the AND/OR graph of the RIOS data. Finally, we demonstrated the integration framework for shop floor knowledge and process planning task based on the object-oriented and service architectures. These architectures allow the software to be reusable and plug-and-play compatible across the enterprise -- particularly, the RSPP module can be reused and the process knowledge can be supplied to the RSPP module from the original equipment manufacturer. Reusability and flexibility provide potential cost savings to the implementations of business and engineering integrations. We believe that this framework will be especially valuable in supply chains that make complex products such as cars, planes, and computer where the outsourcing design and engineering functions have become a growing practice.

Acknowledgment
This work was funded by National Institute of Standard and Technology, contract No. NIST SB1341-01-W-0383.

Product Disclaimer
Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply that these products are necessarily the best available for the purpose

References

1. Candadai, A., Champati, S., Jeffrey, W. H., Ioannis, M., and Ramachandran, V. (September, 1994). 101-109, Information needs in agile manufacturing, Proceedings of the ASME Database Symposium, Minneapolis, MN.
2. Chang, T.C. and Wysk, R. A. (1985). An Introduction to Automated Process Planning Systems, Prentice Hall, Englewood Cliffs, NJ.

3. Cutkosky, M. R. and Leifer, L. J., http://cdr.stanford.edu/DSC/ (September 1997). Design Space Colonization.

4. Ferreira, J. C. E., Steele, J., Wysk, R. A., and Pasi, D. A., A schema for flexible control in manufacturing systems. International Journal of Advanced Manufacturing Technology, 18 (6), 410-421.

5. IBM Alphawork Internet Web Site. Available Online via http://www.alphaworks.ibm.com [accessed December 2001].

6. Jurrens, K. K., Fowler, J. E., Algeo, M., and Elizabeth A. (July 1995). Modeling of manufacturing resource information: Requirements specification. NISTIR Document No. 5707, National Institute of Standards and Technology.

7. Khoshnevis, B. and Tan, W. (August 1995). A process planning system for hole making. Proceedings of IEEE International Symposium on Assembly and Task Planning, Pittsburgh, Pennsylvania.

8. Knutilla, A., Schlenoff, C., Ray, S., Polyak, S. T., Tate, A., Cheah, S.C. and Anderson, R. C. (May 1998). Process specification language: An analysis of existing representations. NISTIR Document No. 6160, National Institute of Standards and Technology.

9. Kulvatunyou, B. (2001). A Resource Independent Process Representation for Enterprise-based Engineering Integration, Ph.D. Thesis, Pennsylvania State University.

10. Kulvatunyou, B., and Wysk, R.A., (July 2000). A functional approach to enterprise-based engineering integration, Journal of Manufacturing System, 19 (3), 156-171.

11. Metcut Research Associates Inc. (1980). Machining Data Handbook 3rd Edition, v.1 and v.2, Cincinnati, OH.

12. Shah, J. J. and Mantyla, M. (1995). Parametric and Feature-Based CAD/CAM, John Wiley & Sons, NY.

13. Sriram, R. and Candadai, A. (March 1996), Agile infrastructure for manufacturing systems (AIMS): A pilot program, Fifth National Agility Forum Conference, Boston, MA.

14. Terry, Q. (1998). Visual Modeling with Rational Rose and UML, Addison Wesley, Reading, MA.

15. Wang, F. and Wright, P. K. (September 13-16, 1998). Web-based CAD tools for a networked manufacturing service. 1-9, ASME Design Engineering Technical Conferences, Atlanta, GA, (ASME Document No. DETC98/CIE-5517).

16. Wang, H. P. (1986). Intelligent Reasoning For Process Planning, Ph.D. Thesis, The Penn State University.

17. Wright, P. (2000) Manufacturing Advisory Service. Integrated Manufacturing Lab. University of California, Berkeley. Available online via http://cybe-rcut.berkeley.edu/mas2/index.html [accessed August 2000].

18. Wysk, R. A (1977). An Automated Process Planning and Selection Program: APPAS, Ph.D. Thesis, Purdue University.