Hierarchical Representation of Manufacturing Process Plans using PSL

Dusan SORMAZ^{a,1} and Arkopaul SARKAR^a

^aIndustrial and Systems Engineering, Russ College of Engineering, Ohio University

Abstract. This paper describes the modeling of manufacturing process plans by proposing the extension to the Process Specification Language. First, we briefly describe a hierarchical nature of the process plan model which results from process planning activity. Few levels of hierarchy: part level, machine level, setup level, and process level were defined with the last, process level, being a level of atomic manufacturing processes. After the application-oriented description, we utilize the Process Specification Language (PSL) in order to provide formal definitions of described hierarchy, which is based on the usage of various resources in manufacturing and process planning. We extend the PSL activity to include resources with their roles in atomic processes and then build a hierarchy of PSL activities that correspond to the manufacturing process plans. We complement the hierarchical process plan model with two important considerations, namely, process sequence and ordering constraints and we illustrate how the model can accommodate them. The discussion of the proposed model is supported by an example.

Keywords. Manufacturing Ontology, Process Planning, PSL

1. Introduction

Process planning primarily consists of "determining the most appropriate manufacturing and assembly processes and the sequence in which they should be accomplished to produce a given part or product according to the specifications set forth in the product design." Selection of necessary processes is also the most complex phase of process planning, due to the subjective nature of the act of planning. Traditionally, manufacturing industries depended on the knowledge of craftsmen, artisans, and machinists for planning their production strategy. In variant type of process planning, templates of plans for common design specifications are stored and re-used for generating plans by matching the design requirements with the template specifications. In modern computer-aided process planning systems, generative type planning mainly focuses on capturing manufacturing best practices, standards and machinists' knowledge in reusable formats, which are referred to generate process plan from the design specification. In the area of CNC machining, such as drilling, milling, and turning, these type of knowledge may include the process-specific capability for generating features of particular quality and precedence among tasks, which is to be

¹ Dusan Sormaz, Ohio University, Department of Industrial and Systems Engineering, 1 Ohio University, Stocker Center 284, Athens, OH 45710-2979, USA E-mail: <u>sormaz@ohio.edu</u>. Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

maintained for achieving the desired result, based on inherent constraint in the design and commonly prescribed ordering in the application of processes.

In this paper, we focus on two fundamental aspects of process planning, which are (1) aggregation of processes based on their attributes, such as resources and output of the process, (2) precedence among processes, stemming from the constraints in design specification and applicability of process. The domain-focused brief explanation of process plan representation is given in Section 3. The ontological commitment for the proposed resource-based aggregation and technological precedence between activities are discussed in section 4, based on the theory of manufacturing planning from our previous research work. The formal definitions of sub-types of activity based on the aggregation factors are presented in section 4.1, and process sequence in section 4.2. Then, various precedence constraints are discussed in section 4.3. Before we delve into these topics, we present a brief overview of previous researches on the topic of our interest, including semantic modeling of process flow based on Process Specification Language, which we heavily adopt in our proposed model.

2. Previous Work

The importance of the nonlinear process plan concepts including dynamic and flexible process plans has been identified and considered as a benchmark for process planning and integration by most of the research approaches [1]. Since its inception in 1984, ISO 10303 (an ISO standard for exchange of product manufacturing information aka STEP) has published a number of Application Protocols (AP) covering product knowledge in different domains [2]. The semantic representation of industrial knowledge using ontology and formal languages have garnered popularity, because of the benefits that formal logic-based languages provide in contextual data integration of heterogeneous information and interoperability of models built on such languages though automated reasoning. ONTOStep provides an ontology to map the STEP functions in OWL model [3]. ONTO-PDM is an ontology developed based on STEP PDM to define fine-grained knowledge structure for product assembly, bill-of-material, and material types [4]. Attempts to build ontologies for a manufacturing focused on describing products, manufacturing process, resources and controls which describe the relations among resources and processes. One of the early work on the core terms of manufacturing was conducted by Borgo and Leitão, who proposed a set of such core concepts related to manufacturing based on foundation ontology DOLCE [5].

Open assembly model (OAM) proposed by NIST, defines generic concepts to integrate requirement, functional design, kinematic synthesis, and tolerance analysis along with basic part and assembly information [6]. OAM extends the object-oriented Core Product Model (CPM), also proposed by NIST [1]. MASON ontology proposed by Lemaignan et al. tried to capture the precedence relationship among different manufacturing operations required to produce a part with the desired quality [7]. It also links material, machine, tools, and cost information with every manufacturing operation. Another ontology proposed by Kim et al. describes a number of widely used assembly processes; such as joining, bonding, riveting, welding, and their sub-processes [8]. Ameri et al. developed an ontological model describing manufacturing processes and resources as services, which included a detailed study on the capability of machine and tool [9].

Process Specification Language (PSL), developed by Gruninger, describes such a formal model for representing planning and scheduling problems, especially suited for manufacturing industries [10]. PSL has been standardized as ISO 18629 within TC 184 (Automation systems and integration) SC4 (Industrial Data) and is axiomatized in full first-order logic (FOL) based languages such as initially Knowledge Interchange Format (KIF) and Common Logic (CL). PSL served as a foundational model for a number of researches in shop-floor level decision making, agent-based communication, and orchestration of web services. Furthermore, PSL has been used in prototype implementations by companies such as SAP, Siemens, Rolls-Royce, KBSI, and IBM. Still, PSL has not found its use in knowledge-based generative process planning.

While PSL is originally built for interoperability of process flow information among manufacturers and CIM software, generative process planning involves actual planning before such flow may be construed. In a dynamic shop-floor condition, these plans sometimes need to be re-generated or modified based on real-time conditions. In this type of scenario, canonical models of process plans are not enough as some level of flexibility should be achieved in dynamic process planning with good support of the rules and query system to the knowledge representation language. A semantic model of generative process plan should include the necessary planning criteria and constraint embedded in the plan in lieu of a mere collection of processes universals (*Activity* in PSL terminology). Based on this argument, we are going to present the semantic model for generative process plan in first-order logic, which heavily adopts the process flow related concepts from PSL, with an honest admission that such model may not be directly interpretable in the standard version of PSL.

3. Overview of Manufacturing Planning

Manufacturing planning mainly consists of a series of tasks, performed in a sequence to manufacture a part with the desired quality. However, more often tasks from one step need feedback from previous steps. Production management decision made at every step influences the decision for subsequent steps. This inter-dependency among phases of development is classically tackled by concurrent engineering. The main idea behind concurrent engineering is to integrate various phases of the product life cycle, such as design, manufacturing, assembly, quality assurance, and maintenance; so that the designers can design the product keeping the manufacturability of the product in mind (Design for manufacturing/Design for Assembly). In practice, this demands an integrated system able to analyze the design data and compute the necessary processes, machines, and tools necessary at the design time. Integrated Manufacturing Planning tool (IMPlanner), developed by Sormaz and Khoshnevis (1999) addressed the aforementioned inter-dependency among various planning variables [11], [12].



Figure 1: Manufacturing process plan model represented in three dimensions

IMPlanner uses a three-dimensional object-oriented model to represent the nonlinearity in manufacturing process planning (MPP) information, shown in **Figure 1**. In this model, various stages of non-linear MPP are filed along the aggregation dimension. At each aggregation level, a set of planning variables, such as (features, processes, machines, and tools) are selected to be part of the overall process plan.

At the bottom-most level, each process is identified based on the output it produces. In this model, the most granular form of output of a manufacturing process is a feature, which can be broadly defined as some special areas of interest in a part — "a set of geometric entities (surfaces, edges, and vertices) together with specifications of the bounding relationship between them and which have engineering function and/or provide assembly aid" [13]. In this study, we extend this definition to include features for every kind of products, e.g. thickness of a liquid, fat content of the milk, a grip on a pen, and relief on the sheet metal. For many cases, especially in prismatic parts, features can be conceived as fiat objects, which are part of the product. In other cases, features are simply quality of the product. However, in both cases, features cannot exist independently. From this point of view, features are ubiquitous in every kind of product and manufacturing of a product should primarily concern generating the specified features of that product. Not every feature specification may be produced by a single process and multiple processes required for achieving the target specification for the feature are aggregated at the process level. Then the layers above process level aggregate processes based on common machine, tool and other resources respectively. In this study, we choose machine and fixture to be used as resources for such aggregation. We justify the importance of these resources in process definition in the next section.

Moreover, planning variables at every aggregation level also vary along with variety and time dimension. Variety dimension captures alternative selections of planning variables for a certain manufacturing task, whereas the time dimension captures the temporal dependency among the planning variable values at every aggregation dimension. Such temporal ordering among processes at each aggregation level arises from constraints related to each aggregation factor. In this study, we model two fundamental precedence constraints: feature precedence and process precedence, which are presented in section 4.3.1 and 4.3.2 respectively.

4. Manufacturing Process Plan Representation

In this section, we present the model of the manufacturing process plan based on the ontological argument and present important definitions by extending PSL terminology. First, process hierarchy is described, which is followed by a discussion on process sequences and ordering constraints in the subsequent sections.

4.1. Process Hierarchy

The general meaning of a process in manufacturing is polysemous in nature, as the contexts for the use of the word are often assumed in the purported meaning when we use the word in different situations. For example, a process may address a singular task to be performed as well as an entire series of processes undertaken for manufacturing a product. It is therefore required to classify process based on the difference in contexts. We argue that some combination of machine and fixture is always required whatever type of process is concerned. For example, for metal fabrication industry a CNC type machine (e.g. mill, drill lathe, stamp) along with an appropriate tool works on raw material, which is normally a stock, metal bar or rod, held in a specially designed fixture. The use of the machine is also compulsory in many other types of industrial processes, such as the injection molding machine in casting, and blenders in mixing, however, such statement may not be strictly asserted for the tool as the semantic distinctions between machine and tool is debatable and out of the scope of this document². The fixture in the industrial process is responsible for holding the raw material, or in-process workpiece in the right position and orientation for the machine and the tool to operate on them. For example, the workpiece is placed in the orientation so that the tool path can cut the shape of the feature. The fixture may also be a rack, a conveyor belt, or a vessel when the scope is extended to a wider variety of manufacturing. The ubiquity of some kind of machine and fixture in manufacturing processes may be established by considering the situation for the simplest type the processes, take for example the manual packaging of an artifact, which needs a pair of dexterous hands (the machine) and at least a platform (the fixture) in order to carry out the packaging process.

In PSL, Axiom 16 dictates that an *object* can *participateIn* an *activity* only at those *timepoints* at which both the object *existsAt* and the *activity isOccurringAt*, where an *activity-occurrence* is the *occurrenceOf* a single *activity* [14]. In manufacturing process plan, the collection and order of the process are not about the actual occurrence of the process but behavior specification based on its allocation of resources and specific configuration, which are better suited to be asserted for *activity*. In an effort to enable the allocation of resources in the process plan, Solano et al. conjured up an abstract type called *Resource*, which is used as a reified type to link *Object* (machine, tool, fixture) to instance of *Activity* [15]. During the actualization of such *Activity*, the *Object* instance assumes a role to fulfill the capability and capacity of the *Resource* attached to the *Activity*. In reality, every machine and tool is different from each other

² The ontological distinction between machine and tool is needed before examining the co-dependence of machine and tool in an industrial process, as some processes similar to CNC machine need a mechanized system uses some replaceable tool to carry out the process, whereas in other processes, the machine and tool are tightly integrated into one system such as a furnace.

when their capability is analyzed in detail. Therefore, the essence of the resources is grounded in its capability, which is directly derived from the actual Object, and not from the abstraction of it. The model proposed by Solano et al. still links these actual objects to occurrences [15], whereas the knowledge of the actual objects and their capability is a *priori* for process planning, thus making the use of abstract type *Resource* redundant. Rather we propose a modicum extension to PSL, in which the *participateIn* predicate links object to activity without any predicate requiring the object to *existsAt* the timepoint. In order to classify processes based on the participants of the process, e.g. the machine, tool, and fixture, three subtypes of activity are defined.

The PSL Ontology uses the *subactivity* relation to capture the basic intuitions for the composition of activities. *Subactivity* relationship is a transitive relationship which holds between two activities [14]. Using this property of the subactivity relationship, the process aggregation can be defined using the sub-types of activity as defined in the following axioms for *PartActivity*, *MachineActivity*, and *SetupActivity*, and ultimately provides a grouping of *StepActivities*. The factor of the aggregation is based on the resources and outputs of the process, which is necessary for creating some desirable order in the process plan. For example, one may want to process every step, which can be processed by the same machine, consecutively in order to minimize the machine transfer cost. Similarly, the steps under the same setup may also be grouped in a similar fashion. Such aggregation will provide the necessary grouping constraints on the ordering of the occurrences of the activity. Here we show that the structure generated by applying the given aggregation is a kind of a tree.

StepActivity: Step is an atomic activity, i.e. no other activity can be a *subactivity* of *StepActivity*, in a process plan. An activity is a step activity if the output of a *StepActivity* is a single feature, i.e., no two features can be the output of the same *StepActivity*. This definition is based on the definition of working step in ISO 14649, however, while ISO 14649 is focused on CNC machining, this definition is its extension to other manufacturing processes.

 $\forall a \ Step \ Activity(a) \equiv Activity(a) \land \exists f, p \ achieves(a, part Of(f, p)) \land (\exists a' \ Step \ Activity(a') \land (a \neq a') \rightarrow \exists f' \ achieves(a, part Of(f', p)) \land (f \neq f'))$

The predicate *achieves* is borrowed from the PSL axioms for fluent attached to the occurrence [14]. In PSL ontology, a fluent is defined as the change of the state which takes place due to the occurrence of an activity. When the same concept is used or desired for an activity, it denotes only the intuitive change of state for the corresponding occurrence of that activity, i.e. $\forall a, x \ achieves(a, \phi(x)) \leftrightarrow \exists o, t_b, t_e \ occurrenceOf(o, a) \land beginOf(o, t_b) \land endOf(o, t_e) \land prior(\phi(x), t_b) \land holds(\phi(x), t_e)$, where t_b and t_e are time points. It can be observed from the definition of *StepActivity* that every instance of *StepActivity* is devoted to producing a single feature. We already described the features as fiat objects belonging to some part. Here, we introduce a new predicate *processingStepOf*, which links a StepActivity to that part, containing the feature produced by *StepActivity*. This predicate is introduced with little ontological value but to make the subsequent axioms simpler.

SetupActivity: A SetupActivity is composed of one or more StepActivity, each of which is performed using the same fixture, i.e. no two StepActivity, which are subactivity of the same SetupActivity can use different fixture. Also every StepActivity under a SetupActivity works on features, which all belong to the same part.

 $\begin{array}{l} \forall a_f \ Setup Activity(a_f) \equiv Activity(a_f) \land \exists a \ Step Activity(a) \land \\ subactivity(a, a_f) \to \exists s \ uses Fixture(a, s) \land (\forall s'a' \ \exists p \ Step Activity(a') \land \\ uses Fixture(a', s') \land subactivity(a', a_f) \land processing Step Of(a, p) \land \\ processing Step Of(a', p) \to (s = s')) \end{array}$

, where *usesFixture* is a sub-property of PSL *is_participant_of* and *Fixture* is a sub-type of *Object*. It should be mentioned here that this definition does not imply that all *StepActivities* with the same fixture have to belong to the same *SetupActivity*.

MachineActivity: A *MachineActivity* is a type of activity such that each of its subactivity is processes in the same machine, i.e no two *activities*, which are *subactivity* of the same *MachineActivity*, can be processed on two different machines.

 $\forall a_m MachineActivity(a_m)$

 $= Actvity(a_m) \land \exists a_f StepActivity(a) \land subactivity(a, a_m)$ $\rightarrow \exists m usesMachine(a, m) \land (\forall m'a'_f \exists p StepActivity(a')$ $\land usesMachine(a', m') \land subactivity(a', a_m)$ $\land processingStepOf(a, p) \land processingStepOf(a', p) \rightarrow (m$ = m')

, where *usesMachine* is a subproperty of *isParticipantOf* and *Machine* is a sub-type of *Object*.

PartActivity: A PartActivity is a type of activity such that each of its subactivity makes some change in the workpiece, targeted to make a single part, i.e no two subactivities of the same *PartActivity* can have output features belonging to two different parts.

 $\forall a_p \ PartActivity(a_p) \equiv Activity(A_p) \land \exists a \ Step \ Activity(a) \land subactivity(a, a_p) \\ \rightarrow \exists p \ processing \ Step \ Of(a, p) \land (\forall a'p' \ Step \ Activity(a') \\ \land subactivity(a', a_p) \land processing \ Part \ Of(a', p) \rightarrow (p = p')$

Based on the aggregation factor, the activities at every level of the activity hierarchy can be classified as a type of complex activity, except for the StepActivity, which should always be an atomic or primitive activity. The hierarchy of activities may be formed by classifying the activities at each level as one of the activity types, described above. The aggregation hierarchy, shown in **Figure 2**, is for a simple part with four features. Table 1 shows the machines and tools participating in the instances of *StepActivity* and corresponding features achieved. The part will be completely done if all four of its features are processed successfully, each of which is the output of one of *StepActivity* shown as the leaves of the hierarchy. At the root of this hierarchy is *PartActivity*, which contains every required step as its subactivity. Two instances of its features. Under the *MachineActivity*, different *SetupAct2*-A have same tool direction, still distinct as they are applied to different machines.



Figure 2: A simple part design with four features and corresponding activity aggregation.

Table 1. Machine and fixture assignments for various StepActivities

		-	-	
	StepAct1	StepAct2	StepAct43	StepAct4
usesMachine	Mill1	Mill1	Drill1	Mill1
usesFixture	Fixture1	Fixture2	Fixture1	Fixture2
achieves	Slab1	Slab2	Hole	Slot

The primary purpose of the aggregation in a manufacturing plan is to group steps under the same machine, tool, and other resources, so that the resource allocations, which are expressed at the complex activity level, can be deduced from atomic step level. The reason for such aggregation is that, whenever some resource is changed, there are other non-manufacturing activities required to accomplish that change (for example, if the machine is changed, some transport is necessary). Furthermore, the step activities can be grouped by resources as required. This is possible because PSL declares *subactivity* as transitive. For example, StepAct2 is a subactivity of SetupAct1-B, which is, in turn, a subactivity of MachineAct1. Therefore, StepAct2 uses milling machine Mill1 and fixture Fixture2. On the other hand, one may query which steps can be allocated to Mill1 and find three answers Step1, Step2, and Step4.

4.2. Process Sequence

The successor relationship between two occurrences of activity occurrences is the basis of the occurrence tree in PSL. "The PSL successor relation associates occurrences with each other to represent all temporal orderings of runtime execution of activities whether they conform to a behavior specification or not, and even including orderings that are physically impossible. The relation forms a tree where every occurrence has exactly one successor for each activity, indicating the possibility of that activity happening next, so the branches represent possible execution traces" [16]. Every node of an occurrence tree is a primitive or atomic activity occurrence, which can be safely described by the leaf nodes of the activity tree that are the activities which are not decomposed further. Every occurrence in the tree denotes a single step in a possible routing. Therefore every path from the root to leaf occurrence of the occurrence tree provides a possible routing. For example, the activity tree shown in **Figure 2** has 4 leaves which are instances of *StepActivity*. The corresponding occurrence tree is shown in **Figure 3**, which contains the primitive activities from the activity tree shown in **Figure 2**.



Figure 3: Occurrence tree for the simple part shown in Figure 2.

4.3. Ordering Constraints

It is apparent that without any constraint on the occurrence of the steps, every branch of the occurrence tree is a viable execution trace, or in other words, a possible plan for the part. However, ordering constraints may be applied based on the intrinsic and extrinsic ordering constraints stemming from participating resources to each StepActivity. For example, one may wish to work with a particular machine before going to other machines, or always perform all the steps using the same fixture together. In the presence of such constraints, only some of the branches of an occurrence tree can be regarded as a valid process plan. PSL provides *minPrecedes* and *nextSubocc* relationships for expressing ordering constraints among sub-occurrences of an occurrence.

Apart from these resource-based constraints, two other types of constraints govern the validity of a process plan. The first one stems from precedence imposed by the processing order among features of a part and type of processes to be applied for producing a feature.

4.3.1. Feature Precedence

Features (e.g. Hole, Slot, Pocket, Chamfers, and Bevel) in product design can only be machined in a specific order due to their overlapping spatial location (position) and orientation (vector) [17]. Such ordering among tasks is also common in assembly processes. For



Figure 4: Precedence between assembly processes of a pen

example, the Liaison graph is used to show the interaction between different components in a pen assembly (Figure 4). Clearly, the 'button' cannot be attached to the body until the 'tube' is filled with ink and the 'head' is capped. In this case, each assembly process creates one of the features of the product. In general, this type of precedence constraints is imposed by design specification, such as the dimension of datum reference and tolerances.

As described in section 4.1, features are uniquely tied to the instances of *StepActivity*. Being primitive activities, the occurrences of the *StepActivity* instances constitute various branches of the corresponding occurrence tree. PSL prescribes the use of *nextSubocc* relationship among the subactivities of a complex activity to denote such precedence constraint strictly. During planning, the following axioms impose ordering constraints among the occurrences of the steps (primitive activity) based on the precedence constraints among features. In order to capture the constraints among

the features, which is not itself temporal precedence and derived from the positioning and orientation of those features in the part design, a new predicate *nextFeature* is introduced. This new property can only be held between two features, generating a complete or partial ordering among the features. For example, the partial order among four features of the simple part shown in **Figure 2** is shown in **Figure 5**. It should be noted here that



Figure 5: Feature precedence among the features of simple part

feature precedence is a minimal set of constraints that the product or part requires for their manufacturing, on the other side process precedence may include more constraints (some will be mention later). Based on this feature precedence, a temporal ordering may be applied among planned activities, using the following *minPrecedes* axiom. $\forall o, o_s, o'_s minPrecedes(o_s, o'_s, o)$

 $\rightarrow \exists a, a_s, a'_s, f, f' occurrenceOf(o, a) \land occurrenceOf(o_s, a_s) \\ \land occurrenceOf(o'_s, a'_s) \land subactivity(a_s, a) \land subactivity(a'_s, a) \\ \land achieves(a_s, partOf(p, f) \land achieves(a'_s, partOf(p, f') \\ \land nextFeature(f, f')$

For the activity composition shown in Figure 2, Slab1 is processed by Step1, Slab2 by Step2, Slot by Step4, and Hole by Step3. The constraints among the features can be translated to the order among the corresponding occurrences of these steps by the following facts.

 $occurrenceOf("o_p", "PartAct") \land minPrecedes("o_3", "o_1", "o_p") \land minPrecedes("o_4", "o_2", "o_p") \land minPrecedes("o_4", "o_3", "o_p")$

PSL suggests the relation *legal* to specify an atomic occurrence (*occurrenceOf* a primitive activity) o is an element of the 'legal' occurrence tree. A legal occurrence tree is the subtree of the complete occurrence tree, which captures both possible and impossible branches. For example, *occurrenceOf*(o_4 , "*Step4*") \land *legal*(o_4) \rightarrow *holds*(o_2 , *partOf*("*Slab2*", *p*)) \land *holds*(o_3 , *partOf*("*Hole*", *p*)) \land

prior(o4, partOf("Slot", p)). This set of facts can also be deduced from the following rule by applying the *minPrecedes* axiom and avoiding the fluent predicate. $\forall o_3 \ occurrenceOf(o_4, "Step4") \land legal(o_4) \rightarrow$

 $\exists o, o_2, o_4 occurrence Of(o_2, "Step 2") \land occurrence Of(o_3, "Step 3") \land minPrecedes(o_4, o_2, o) \land minPrecedes(o_4, o_3, o).$

Similar rules can check the ordering constraint on occurrences of Step1, Step2, and Step3. The branches of the occurrence tree, in which every occurrence is *legal* then can be selected as a valid process sequence. The valid sequences are marked with green in the occurrence tree shown in Figure 3.

4.3.2. Process Precedence

Similar to the precedence among features of the part, a valid process plan also needs to follow a particular ordering for the processes applied in order to make one feature meeting its specification. This sort of situation is normal for CNC machining, in which more than one machining step may be required in order to make the product feature in order to meet tolerance speculations such as dimension tolerance and surface finish, however, this may occur in another type of manufacturing depending on the definition of feature adopted. For example, multiple heat treatments may be required to harden a metal bar, a welded joint may require polishing before welding, and paint job may require a coating of primer before the color is applied. The precedence among process stems from the technological limitation of the machine and tool used in a particular manufacturing method, for example, a pilot hole needs to be drilled before drilling a deep hole with a strict straightness requirement in order to avoid the deflection of the drill bit under mechanical stress.

The precedence among processes is best handled if an intermediate object is generated for every step. Following this approach, we may generate intermediate features for the part design whenever more than one process is required to satisfy the specified tolerance requirements. This also corresponds to practice, as such partially completed part may need to be transported from one machine to another (e.g. from a lathe to heat treatment, then to a grinder if grinding operations are required). Those intermediate features become an integral part of the part model for manufacturing planning. Therefore, the extended part model is a union of manufacturing features for the part design and all intermediate features that were identified as necessary during the process selection procedure. In order to complete the extended part manufacturing model, it is necessary to consider the impact of intermediate features on the feature precedence network (FPN).

In order to generate the Extended FPN (EFPN), we will assume that the process selection procedure produced the processes for design features as shown in **Figure 5**. From the figure it is visible that to make Slab we need only milling process (M), while for the other three features we need two processes for each: for Hole we need twist drilling (Step3) and boring (Step3i), for Slot we need rough (Step4) and finish (Step4i) milling operations. The activity hierarchy for the corresponding EFPN is shown in **Figure 6** by adding the intermediate features.

In order to accommodate the intermediate features, we need to extend our original aggregation to include another level of aggregation, which can combine steps having the feature outputs belonging to one feature specification. Every subactivity of this new type of complex activity *FeatureActivity* is targeted to the same *feature specification*. The specification of the feature is distinguished from an actual (physical) feature. The *intermediate features* are a distinct physical feature with different dimensions and properties, whereas the design feature is specified in the product design document. It is expected that only one of the intermediate features will match the specification, and thus may be called the *final feature*. In **Figure 6**, The activity hierarchy shown in **Figure 5** is redrawn by introducing this new complex activity.



Figure 6: Extended FPN of SimplePart and activity hierarchy including FeatureActivity

The axiom of *FeatureActivity* is given below, in which the property *specificationOf*(f_{s} ,f) describes that feature f (possibly intermediate or final) is one of the features created in course of achieving the specification f_s .

 $\begin{array}{l} \forall a_i \ Feature Activity(a_i) \equiv Activity(a_i) \land \exists a \ Step Activity(a) \land \\ subactivity(a, a_i) \rightarrow \exists p, f, f_s \ achieves(a_i, part Of(f, p)) \land \\ specification Of(f_s, f) \land (\forall f', f_s', a' \ Step Activity(a') \land subactivity(a', a_i) \land \\ achieves(a', part Of(f', p)) \land specification Of(f_s', f') \rightarrow (f_s' = f_s)) \end{array}$

The aggregation based on the feature can also be imposed on the occurrence tree using the required precedence among processes for subactivities of a *FeatureActivity*. For example, when applied as subactivity of the same FeatureAct3, twist drilling (Step3i) should precede boring (Step3), which can be stated as $\forall o_b \ occurrenceOf(o_b, "Boring") \land legal(o_b) \rightarrow$

 $\exists o_t, o \ occurrenceOf(o_t, TwistDrilling") \land occurrenceOf(o, "FeatureAct3") \land nextSubocc(o_b, o_t, o)$

5. Conclusion

This paper presents the manufacturing process plan representation in a first-order logic by extending a concept from the Process Specification Language (PSL). The primary contribution of the paper is the formal definition of complex activities based on the type of resources being used and the output of the activity. We have shown how these complex activities can be combined to make a hierarchical aggregation and representation of manufacturing process plans. After the basic model description, it has been augmented to accommodate two types of precedence, namely feature and process precedence, which act as constraints on the flow of the processes. This extension was presented using the PSL occurrence tree concept. The model is illustrated with the help of simple mechanical design (called simple part), which has only four features and corresponding manufacturing steps. However, the proposed model did not go through formal validation using reasoning algorithms and case studies for a variety of situations. Therefore, future work with the model will include completion of formal representation and axioms in first-order logic (FOL), the model application on more realistic examples, and its verification using reasoners to answer typical manufacturing planning questions.

References

- [1] S. J. Fenves, S. Foufou, C. Bock, and R. D. Sriram, "CPM: A Core Model for Product Data 1 Introduction 2 The Core Product Model," pp. 1–14.
- [2] S. Staab and R. Studer, "Handbook on Ontologies," in *Decision Support Systems*, S. Staab and R. Studer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, p. 654.
- [3] R. Barbau, S. Krima, S. Rachuri, A. Narayanan, X. Fiorentini, S. Foufou, and R. D. Sriram, "OntoSTEP: Enriching product model data using ontologies," *Comput. Des.*, vol. 44, no. 6, pp. 575– 590, Jun. 2012.
- [4] H. Panetto, M. Dassisti, and A. Tursi, "ONTO-PDM: Product-driven ONTOlogy for Product Data Management interoperability within manufacturing process environment," *Adv. Eng. Informatics*, vol. 26, no. 2, pp. 334–348, 2012.
- [5] S. Borgo and P. Leitão, "Foundations for a Core Ontology of Manufacturing," in *Integrated Series in Information Systems*, Springer {US}, pp. 751–775.
- [6] M. M. Baysal, U. Roy, R. Sudarsan, R. D. Sriram, and K. W. Lyons, "The Open Assembly Model for the Exchange of Assembly and Tolerance Information: Overview and Example," 2004, vol. 2004, pp. 759–770.
- [7] S. Lemaignan, A. Siadat, J.-Y. Dantan, and A. Semenenko, "MASON: A Proposal For An Ontology Of Manufacturing Domain," in *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence* and Its Applications (DIS'06), 2006, pp. 195–200.
- [8] S. Kim and K. Lee, "An assembly modelling system for dynamic and kinematic analysis," Comput. Des., vol. 21, no. 1, pp. 2–12, Jan. 1989.
- [9] F. Ameri, C. Urbanovsky, and C. McArthur, "A systematic approach to developing ontologies for manufacturing service modeling," CEUR Workshop Proc., vol. 886, pp. 1–14, 2012.
- [10] M. Grüninger, "Using the PSL Ontology," Handb. Ontol., pp. 423-443, 2009.
- [11] D. Sormaz and B. Khoshnevis, "Process Planning Knowledge Representation using an Object-oriented Data Model," Int. J. Comput. Integr. Manuf., vol. 10, no. 1–4, pp. 92–104, 1997.
- [12] D. N. Sormaz, J. Arumugam, and S. Rajaraman, "Integrative process plan model and representation for intelligent distributed manufacturing planning," *International Journal of Production Research*, vol. 42, no. 17. pp. 3397–3417, 2004.
- [13] J. J. Shah and M. T. Rogers, "Functional requirements and conceptual design of the Feature-Based Modelling System," *Comput. Eng. J.*, vol. 5, no. 1, p. 9, 1988.
- [14] C. Schlenoff, J. Lubell, and M. Gruninger, "The Process Specification Language (PSL) overview and version 1.0 specification," no. 6459, 2000.
- [15] L. Solano, P. Rosado, and F. Romero, "An Ontological Approach For Manufacturing Resources Modeling," 21th Int. DAAAM Symp., vol. 21, no. 1, pp. 779–781, 2010.
- [16] C. Bock and M. Gruninger, "PSL: A semantic domain for flow models," *Softw. Syst. Model.*, vol. 4, no. 2, pp. 209–231, 2005.
- [17] D. N. Sormaz and B. Khoshnevis, "Modeling of manufacturing feature interactions for automated process planning," J. Manuf. Syst., vol. 19, no. 1, pp. 28–45, 2000.