PROCESS SPECIFICATION LANGUAGE: An Analysis of Existing Representations

Amy Knutilla
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD
E-mail: amy.knutilla@nist.gov

Craig Schlenoff
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD
E-mail: craig.schlenoff@nist.gov

Steven Ray
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD
E-mail: steve.ray@nist.gov

Stephen T. Polyak
Department of Artificial Intelligence
The University of Edinburgh, UK
E-mail: Steve_Polyak@ed.ac.uk

Austin Tate
Artificial Intelligence Applications Institute
The University of Edinburgh, UK
E-mail: a.tate@ed.ac.uk

Shu Chiun Cheah
Department of Computer Science
and Institute for Systems Research
The University of Maryland, College Park, MD
E-mail: scheah@cs.umd.edu

Richard C. Anderson
Department of Engineering Management
George Washington University, Washington,DC
E-mail: anderson@seas.gwu.edu

Abstract

The goal of the NIST Process Specification Language (PSL) project is to investigate and arrive at a neutral, unifying representation of process information to enable sharing of process data among manufacturing engineering and business applications. This paper focuses on the second phase of the project, the analysis of existing process representations to determine how well existing process representation methodologies support the requirements for specifying processes found in Phase One. This analysis will provide an objective basis from which to develop a comprehensive language and will promote the leveraging of existing work.

Table of Contents

1. Introduction

1.1 The Process Specification Language (PSL) Project[1]

Many manufacturing engineering and business software applications use process information, including production scheduling, manufacturing process planning, workflow, business process reengineering, simulation, process realization process modeling, and project management. The problem is that all of these applications represent process information in their own internal representations, which makes communication among them, a growing need for industry, nearly impossible without an application-specific translator. The goal of the PSL project is to create a process specification language that is common to all manufacturing applications, generic enough to be de-coupled from any given application, and robust enough to be able to represent the necessary process information for any given application. This representation would facilitate process data sharing among various applications because they would all "speak the same language", either as a second language or their native language.[2]

The project has five major phases:

1. **Requirements gathering**. This phase, completed September 1996 [Schlenoff *et al*. 96], involved the identification of the requirements necessary for modeling manufacturing processes by analyzing process-centered manufacturing software applications such as production scheduling, manufacturing process planning, and workflow to determine if there exists a common set of requirements for specifying processes. For example, a process description should include notions of sequence, data requirements, resources, duration and time, location, abstraction, etc.
2. **Existing process representation analysis**. In this phase, completed April 1997, various process representations, methodologies and languages were analyzed to determine how well they represent the requirements found in Phase One. The intent of this analysis was to provide an objective basis from which to develop a comprehensive language and to leverage existing efforts in the area of process representation.
3. **Develop initial PSL scenarios, semantics, syntax, and presentation(s)**. This phase involves the, 1) creation or identification of appropriate scenarios relevant to the PSL objectives, 2) definition of the conceptual (semantic) concepts that will be modeled in

[1] This project is funded by NIST's Systems Integration for Manufacturing Applications (SIMA) Program. Initiated in 1994 under the federal government's High Performance Computing and Communications effort, SIMA is addressing manufacturing systems integration problems through applications of information technologies and development of standards-based solutions. With technical activities in all of the NIST's laboratories covering a broad spectrum of engineering and manufacturing domains, SIMA is making information interpretable among systems and people within and across networked enterprises.

[2] For a detailed description of the PSL project and its background, the reader is referred to [Schlenoff *et al*. 96] and http://www.nist.gov/psl.

the PSL, 3) specification of one or many appropriate syntaxes, depending on the chosen implementation(s), and 4) development of one or many presentations (notations). Because the language is independent of any predetermined notation, it becomes possible to use multiple alternative notations to convey the same information, thus enabling multiple “views”. The initial specification of the semantics, syntaxes, and presentations will be defined further in an iterative approach as the project progresses.

4. **Pilot Implementation and Validation.** During this phase, several application-related process models will be constructed within a pre-defined scenario, translation software developed and process information exchanged using the PSL, to test its robustness and completeness. An iterative approach will be followed until the specification becomes stable.

5. **Submission as a Candidate Standard**. At this point, the validated, documented language will be submitted to the appropriate organization as a candidate international standard.

During each phase, a series of interactions with related communities have been and will continue to be vigorously pursued. These include workshops, formal and informal collaborations, active use of Internet-based tools for collaborative research and development, and attendance at standards meetings. Such interactions assure both technical feedback and the commitment to the specification as it evolves. This external collaboration is the key to ensure that the language will be complete and robust.

1.2 Phase 1: Requirements for Process Specification

The completion of the first phase resulted in a comprehensive set of requirements for specifying process that were grouped into four major categories [Schlenoff *et al*. 96].

- **Core**: The most basic, essential requirements inherent to all processes. To represent process, it is either critical that these requirements be included, or these requirements are so common that every application either explicitly or implicitly uses them. While all processes contain core requirements, the core requirements provide the basis for representing only the simplest of processes, e.g., resource, task.
- **Outer Core**: The pervasive, but not essential, requirements for describing processes common to most applications, e.g., temporal constraints, resource grouping, alternative tasks.
- **Extensions**: The groupings of related requirements, common to some, but not all, applications that together provide an added functionality. Although the requirements listed within the extensions are not inherently necessary for representing processes, they are useful during implementation to provide their respective functionality. They are included here because the PSL must be able to represent information that will ultimately allow this functionality. The six extensions are Administrative/Business, Planning/Scheduling/Quality/Analysis, Real-Time/Dynamic, Process Intent, Aggregate Resources/Processes, and Stochastics/Statistics.

- **Application Specific**. The requirements only relevant within specific applications, e.g., dynamic rescheduling for the production scheduling environment.

A full description of this phase and the defined set of process specification requirements can be found at [Schlenoff *et al*. 96][1]. These process specification requirements provided the context for analyzing existing process representations.

1.3 Phase 2: Analysis of Existing Process Representations

Twenty-six representations (i.e., languages, methodologies, tools, standards, etc., used to specify processes) were identified as candidates for analysis by the PSL team.[2] With help from other experts, these representations were studied and analyzed with respect to the requirements identified in the first phase of the PSL project.

The original objectives for analyzing existing approaches for representing process included:

- gain an improved understanding of existing approaches for representing process
- identify how process specification requirements are represented within existing approaches
- determine the strengths and limitations of existing approaches
- identify the existing representations or combination of representations that provide the best coverage of all process specification requirements
- understand and define what types of representations (e.g., object-oriented) provide the best coverage of all requirements

As the analysis progressed, additional benefits to conducting this type of analysis arose:

- determine the completeness of process specification requirements identified in the first phase
- refine the technical approach for developing a process specification language
- identify the need as well as the candidates for PSL semantic concepts and their definitions
- provide a basis for developing mappings between PSL and existing presentations

[1] Also available at http://www.nist.gov/psl/.
[2] The PSL team that participated in the identification of representations is an informal collaboration of researchers including representatives from NIST, University of Maryland, College Park, George Washington University, the Artificial Intelligence Applications Institute at the University of Edinburgh, and Knowledge Based Systems, Inc.

This paper describes this analysis. Section 2.0 provides brief descriptions of the representations studied and provides references for more detailed discussion. Section 3.0 describes the approach to the analysis. Section 4.0 discusses the results of the analysis. Section 5.0 provides a summary. The full matrix containing the analysis can be found on-line at http://www.nist.gov/psl/. The matrix has also been printed in the appendix.

2. Existing Process Representations[1]

During the second phase of the PSL project, twenty-six process representations were analyzed to determine their applicability for representing the set of process requirements found in the first phase of the project. The term "representation" is used as an all-encompassing term to include languages, methodologies, tools, standards, etc. In general, a representation is an approach to specifying process models. This may include semantic definitions, methods, and/or syntax (which may be textual, graphical, or both). In addition to these twenty-six representations, five supporting representations were also identified. These five representations, although not analyzed directly, were found to play a supporting role in the other representations that were analyzed. In many cases, these twenty-six representations integrated the concepts and constructs of these supporting representations to represent information requirements that the supporting representations captured especially well.

These representations are not intended to be an exhaustive list of every process representation currently available. It does, however, represent a sample of representations that provide some insight into different ways of representing process information.

2.1 Representations Under Investigation

Included in this section is a brief description of and references for all representations. [2]

ACT

ACT [Wilkens & Meyers 95] was created at the SRI International Artificial Intelligence Center as part of research into systems that select and execute appropriate actions for achieving goals in dynamic and uncertain environments. Traditionally, plan generation and reactive execution have been considered as separate activities, with few attempts to integrate them within a single system. The ACT formalism is a language for representing the knowledge required to support both the generation of complex plans and reactive

[1] No approval or endorsement of any commercial product in this paper by the National Institute of Standards and Technology is intended or implied. This paper was prepared in part by United States Government employees as part of their official duties and is, therefore, a work of the U. S. Government and not subject to copyright.

[2] Two of these representations, PAct (Parts and Actions) and EPFL's petri net representations, were only minimally analyzed because of lack of available expertise and literature, and are therefore not discussed in this section.

execution of those plans in dynamic environments. ACT has been used as the interchange language in an implemented system that links a previously implemented planner (System for Interactive Planning and Execution Monitoring (SIPE-2) [Georgeff & Ingrand 89]) with a previously implemented executor (Procedural Reasoning System (PRS) [Wilkens 84]).

ACT is intended to serve as a general-purpose representation language that could be used to share knowledge between many different execution and planning systems. The representational and computational adequacy of ACT has been validated by implementing the Cypress system [Wilkens and Myers 95], which uses ACT as an interlingua to enable runtime interactions between planning and execution subsystems. ACT focuses on a practical, yet sufficiently expressive representation that can address a variety of needs. Sample domains that ACT has been used in include controlling an indoors mobile robot and military operations planning.

The ACT formalism is a domain-independent language for representing the kinds of knowledge about activity used by both plan generation and reactive execution systems. The basic unit of representation is an Act, which can be used to encode both plan fragments and standard operating procedures (SOPs). An Act describes a set of actions that can be taken to fulfill some designated purpose under certain conditions. The purpose could be either to satisfy a goal or to respond to some event in the world. The purpose and applicability criteria for an Act are formulated using a fixed set of environmental conditions. Action specifications are called the plot, and consist of a partially ordered set of actions and sub-goals. The environmental conditions and plots are specified using goal expressions, each of which consists of one of a predefined set of meta-predicates applied to a logical formula. The meta-predicates permit the specification of many different modes of activity, including goals of achievement, maintenance, and testing. ACT can be used to build a very strong model of the relationships between actions, temporal requirements, and resources. It has been shown to have expressive and computational adequacy in several applications. Specific manufacturing elements would need to be added as extensions to support these domain-specific requirements.

A Language for Process Specification (ALPS)

ALPS [Catron & Ray 91, Ray 92] was designed to serve as a generic model to support process plans used within the discrete-process manufacturing industry. The need for such a generic model became apparent in the context of a series of projects initiated at the National Institute of Standards and Technology during the late 1980’s addressing various aspects of Computer Integrated Manufacturing (CIM). One of the key elements for factory integration was consensus on the structure of the underlying information used and generated by component systems. One such body of information is the process plan.

ALPS is based on a directed graph notation to indicate the temporal relationships between process tasks. Directed graphs were chosen because they provide the required attributes of expression: simplicity, clarity, basic precedence, alternative sequences, parallelism, and abstraction. The design goals of ALPS included the support for task decomposition, parallel tasks, synchronization of tasks, alternative tasks, sequences, resource allocation, critical (uninterruptable) task sequences, and information manipulation operatives.

AP213

AP213 [ISO 92, ISO 95a] is an application protocol (AP) within STEP (STandard for the Exchange of Product model data) [ISO 92] whose scope is the exchange, archiving, and sharing of computer-interpretable numerical control process plans for machines parts, focusing on the sharing of data between dissimilar Computer-Aided Process Planning (CAPP) systems. STEP is an international standard (International Standards Organization (ISO) 10303) for the exchange of product information. Application protocols describe the use of a predefined group of resource constructs to satisfy the scope and information requirements for specifying the intended use of information within an application.

AP213 specifies the data contained within a process plan as opposed to the data necessary to perform process planning functions. Included within AP213 are the relationships that exist between the different process plan data items as well as those that exist between these data items and the product definition data. Within the scope of this AP are: information from the planning activity that is contained in the NC process plan for machined parts; work instructions for the tasks required to manufacture the part such as references to resources required to perform the work; the sequence of work instructions and relationship of the work to part geometry; information required to support NC programming of processes specified in the process plan; information required to support in-process inspection specified in the process plan; shop floor information specified in the process plan. AP213 builds off of STEP Part 49 (see Part 49 description on page 17).

AP213 is captured in the EXPRESS language [ISO 93]. EXPRESS is a formal data specification language that provides the mechanism for the normative description of information while allowing a complete description of the data and constraints applicable to that information. EXPRESS permits the definition of resource constructs from data elements, constraints, relationships, rules, and functions.

Behavior Diagrams

Behavior Diagrams [Ballard 89, Alford 90], developed by Mack Alford of Ascent Logic Corporation, are the primary notations for describing system functionality within RDD-100® systems engineering software. They combine the features of Functional Flow Block Diagrams and Data Flow Diagrams, capturing data flows (functional interfaces) as well as control transitions and sequences. These diagrams are used to describe the functional behavior of a system design with a time sequence of functions indicating functional inputs and outputs, strict precedence relationships, control flow and data flow, as well as completion conditions for purposes of time-based simulation and analysis.

Behavior diagramming provides a number of basic constructs and features that can be used to model functional behavior. Most of these make use of one or more *Control Flow Nodes*, which are logical symbols that indicate:

- *selection* of one or more alternate paths,
- iteration and looping of a function or sequence of functions,
- concurrency, or simultaneous activation of a number of functions, and
- replication (i.e., simultaneous or parallel instances) of a function.

As is the case with many other process-related representations, Behavior Diagrams are hierarchically decomposable. The elements of the diagrams (particularly function blocks) can also be linked to related textual documents that contain detailed information about such things as performance requirements, requirements traceability, and so on.

Behavior Diagrams were designed, and best suited, for the behavioral modeling of physical artifacts (i.e., the operational behaviors of well-defined physical systems) as oppose to capturing the more abstract and complex programmatic elements (such as non-machine, human activities like design) that are seen in the product realization process modeling and workflow domains.

Core Plan Representation (CPR)

The DARPA-sponsored Object Model Working Group (OMWG) is currently developing a “core plan representation” (CPR) [Pease & Carrico 97] which is aimed at supporting the representational needs of many types of planning systems. The OMWG’s stated goal is:

> “...to leverage common functionality and facilitate the reuse and sharing of information between a variety of planning and control systems” [Pease & Carrico 97]

CPR has utilized ARPI (Advanced Research Planning Agency (ARPA) Rome Planning Initiative) work on Knowledge Representation Source Language (KRSL) [Lehrer 93], the Planning Ontology Construction Group (POCG) [Tate 96a], the O-Plan project [Currie & Tate 91], and the <I-N-OVA> representation (see below). CPR is composed of a set of basic plan concepts that have been assembled into a refined design framework. The initial minimal set of concepts included Action Resource, Actor, and Objective. This set was then expanded with more entities (e.g. Plan, TimePoint etc.), defined with individual properties (e.g. an Actor has an Objectives slot, etc.), and structured with stated relations (e.g. a Plan contains Actions, Actions contain TimePoints, etc.)

CPR’s intended application might involve the Joint Task Force Advanced Technology Demonstration (JTF-ATD) and Joint Forces Air Component Commander (JFACC) programs that are two DARPA joint-force military planning applications. The OMWG has also identified the possibility of applying CPR to non-military applications as well.

Entity Relationship (E-R)

The Entity-Relationship model was originally proposed by P. Chen in 1976 [Chen 76, Chen 81] as a way to unify the network and relational database views. This model allows one to model information and data that may be conceptualized as having components that are inter-related. Moreover, the relations among components are captured as well as their respective components and attributes. The E-R model also gave rise to the Entity-Relationship diagram, which is its graphical representation.

In the E-R model, data components are represented as entities. These entities may have relationships with other entities. Furthermore, the mapping cardinalities of these relationships may be one-to-one, one-to-many, many-to-one, and many-to-many. Entities can be denoted weak or strong. A weak entity is one whose existence is solely dependent on a strong entity that has a relationship with the weak entity. This construct is particularly useful for consistency maintenance of a database. Attributes may be associated with each entity, usually representing the properties of the component represented by the entity. Furthermore, these attributes provide one with a mechanism to identify each of the entities uniquely.

Since it was first introduced, the E-R model has been widely used in information systems design, especially in systems involving relational databases.

Functional Flow Block Diagrams (FFBD)

Functional Flow Block Diagrams (FFBDs) [Grady 93, Scotti 94, DSMC 86] are a fundamental representational tool within the systems engineering community. It is used to define and illustrate graphically the functions that must be performed by a system as well as the sequential relationships among the functions. They are used in functional analysis to gain, in an organized way, insight into what a system (or system element) is required to do and in what sequence it must be done. Systems engineers have traditionally used FFBDs to provide a graphical view of system behavior as sequences, selections, and concurrences of functions. An FFBD is perceived as an analog of actual system operation where a function is performed by a set of system resources that are not yet fully specified. As such, the activities of a process model and the functions of an FFBD are very closely related constructs.

The primary benefit of FFBDs is their ability to support analyses of the process flow or behavior of a system with respect to time. However, while they do show all the possible sequences of system behavior, they ignore data flows, including those that trigger a transition from one behavior state to another. As well, FFBDs are like IDEF0 diagrams [Wisnosky & Batteau 90] in that they are not intended to show the time duration of activities/functions, nor do they convey the time between functions. Equally importantly, from a process modeling perspective, FFBDs are not, by definition, resource- or equipment- oriented. That is, they identify "what" must happen and do not assume a particular answer to "how" a function will be performed.

Gantt Charts

Named for its developer, Henry Laurence Gantt, the Gantt chart (also known as a bar chart) [Avallone & Baumeister 87, PMI 96] provides a graphic display of schedule-related information, including the relative and absolute durations and start/finish of activities. They can be used to schedule resources as well as activities. In the typical Gantt chart, activities or other project elements are listed down the left side and dates or other suitable time intervals are shown across the top. In the area formed between these two axes, individual activities or project elements are shown as bars. Each bar has a length corresponding to the duration of its corresponding activity, and is placed on the diagram in a location consistent with its designated start or finish time. When used in conjunction with PERT [Avallone & Baumeister 87], any delayed critical-path activities can be highlighted by some kind of differently shaded or colored bars.

Gantt charts are a simple but effective means to convey schedule-related information graphically. They allow a recognition of the tasks that must be performed sequentially and those that may be performed in parallel. They also provide an easily understandable description of the workflow throughout an entire project.

Generalized Activity Network

The Generalized Activity Network (GAN), as defined by Elmaghraby [Elmaghraby 77], is an example of an activity-on-arc representation (see discussion on PERT charts). Activities are represented as arcs, each denoting a particular transition between some initial and final state. These states are shown as nodes.

Activity arcs in a GAN are treated as vectors having at least three associated parameters: the probability that the activity will occur given that its precedent node has been realized; the duration of the activity (a random variable); and, the cost of the activity as a function of activity duration and defined resource usage. In the standard nomenclature for GANs, an activity has receiver and emitter logical conditions associated with its initial and final states, which are graphically displayed as nodes split into two corresponding halves. One half of each node accepts the receiver connections of one activity in a chain, while the other half connects to the emitter of another activity.

One noteworthy element of the graphical presentation of GANs that may be useful in the development of a process specification language is its use of the activity-on-arc technique. This allows the possibility of allowing arc lengths to correspond to activity duration, in a similar fashion to that in Gantt charting. Also important is its explicit graphical representation of start/finish conditions and states for activities. [Lyons *et al*. 95]

Hierarchical Task Networks (HTN)

HTN [Erol *et al*. 94] was developed by AI researchers as a representation mechanism for AI planning. Variations of it have been used by many researchers in the planning community. Despite its long existence, it was not until recently that HTN was formalized. AI researchers from the University of Maryland carried out the formalization work. A task network is a collection of tasks that need to be carried out, together with constraints on the order in which tasks can be performed, the way variables are instantiated, and what conditions must be true before or after each task is performed. The collection of tasks may be all primitives in the sense that they cause a simple state transition to the world, or they may contain compound tasks that could be decomposed during planning to primitive tasks. Essentially, an HTN may represent a plan ranging from a partial plan to a fully instantiated plan.

In HTN, a task can be a primitive task, a compound task, or a goal. A primitive task corresponds to some operation that can be readily done and can change the state of the world. A compound task is one that is decomposable to other compound tasks or primitive tasks. And, a goal is something that needs to be achieved by assigning it a compound task or a primitive task. The decomposition of compound tasks is accomplished by a method. A method is a construct associating a compound task to a task network consisting of sub-tasks of the compound task. Conceivably, one can have multiple methods for each compound task. Finally, an operator is a construct that consists of a primitive task, pre-conditions, and post-conditions. It describes the conditions under which the task is executable and those that result from executing the task.

IDEF0

IDEF0 [Wisnosky & Batteau 90] is a standard for functional modeling derived from the Structured Analysis and Design Technique (SADT)™ [SofTech 81], a well-established cell modeling technique for system analysis and a graphical language for communication of the results, developed for the United States Air Force by Douglas T. Ross and SofTech, Inc. An extension of the representation scheme known as Functional Decomposition Diagramming (FDD), IDEF techniques emerged in the mid-1970s as part of the United States Navy ICAM (Integrated Computer-Aided Manufacturing) initiative to increase manufacturing productivity. [CSDC 94]

IDEF0 is one of the original three IDEF methodologies (IDEF0, IDEF1, and IDEF2), and is used to model systems from the functional/organizational perspective. Today, IDEF0 is being used in both the public and private sectors for the modeling of a wide range of enterprises and application domains, and has been formally standardized in *Federal Information Processing Standards 183* [FIPS183 93]. FIPS 183 describes the IDEF0 modeling language (semantics and syntax) and associated rules and techniques for developing structured graphical representations of a system or enterprise.

As described in an analysis of the strengths and weaknesses of IDEF0 by Knowledge Based Systems, Incorporated: “The primary strength of IDEF0 is that the method has proven effective in detailing the system activities for function modeling, the original structured analysis communication goal for IDEF0. Activities can be described by their inputs, outputs, controls, and mechanisms (ICOMs). Additionally, the description of the activities of a system can be easily refined into greater and greater detail until the model is as descriptive as necessary for the decision-making task at hand. In fact, one of the observed problems with IDEF0 models is that they often are so concise that they are understandable only if the reader is a domain expert or has participated in the model development.”.[1]

IDEF3

Another member of the IDEF “family,” IDEF3, or, “Process Flow and Object State Description Capture Method,” was developed under the Information Integration for Concurrent Engineering (IICE) project, funded by the U. S. Air Force. [Mayer *et al.* 92] Knowledge Based Systems, Inc., is the prime contractor for IICE and developer of IDEF3.[2] IDEF3 is designed to capture the behavioral aspects of an existing or proposed system, including descriptions of precedence (activity sequence) and causality relationships, in a structured way. This has the intended effect of enabling a domain expert to intuitively express his knowledge of the operation of a particular system or organization. The resulting descriptions that are captured by the IDEF3 can then be used to facilitate the construction of analytical and design models. The methodology stops short of providing the ability to construct predictive simulation-based *models*; rather, it is a method to obtain structured *descriptions* of what a system actually can or will do in practice.

The IDEF3 methodology is capable of providing different user views of temporal precedence and causality relationships via two main diagram types, or “description modes.” One, the Process Flow Network (PFN), provides a process-centered view of a system, while the other, the Object State Transition Network (OSTN), allows an object-centered view. Both of these complementary representations employ the same basic diagrammatic notation scheme, featuring series of boxes (either square or oblong), circles, and interconnecting arcs. Textual “elaboration forms” are also attached to each of the graphical icons, providing additional information. The meanings and usage (i.e.,

[1] Knowledge Based Systems, Inc., “IDEF0 Function Modeling Method,” http://www.kbsi.com/idef/idef0.html, April 1997.
[2] Knowledge Based Systems, Inc., “IDEF3 Process Description Capture Method,” http://www.kbsi.com/idef/idef3.html, April 1997.

semantics and syntax) of these graphical entities is dependent upon which of the two description modes is being viewed.[1]

<I-N-OVA> Constraint Model

<I-N-OVA> [Tate 96a, Tate 96b, Tate 97] is a constraint model of tasks, plans, processes, and activities which adopts the perspective that all of these sources are "constraints on behavior". This model can be used as an ontology for shared representations amongst various operations in the planning and execution process including: knowledge acquisition, formal analysis, user communication, and system manipulation.

The acronym, <I-N-OVA>, stands for: Issues, Nodes, Ordering, Variable, and Auxiliary constraints. Issues and nodes are also expressed as constraints and can be thought of as implied constraints and activity constraints, respectively. The inclusion of "issues" in the specification of a plan or process is unique and allows the "state" of the planning process to be captured and communicated throughout the life cycle of a plan. Tate relates these various constraint types together by stating:

> "Planning is the taking of planning decisions (I) which selects the activities to perform (N) which creates, modifies or uses the plan objects or products (V) in the correct time (O) within the authority, resources and other constraints specified (A)."[2]

<I-N-OVA> is not a representation language like some of the other candidates discussed in this paper (e.g. ACT, O-Plan TF; see below). Rather, it is a conceptual model that can provide an underpinning to languages that describe activities, plans and processes. O-Plan's widely used domain description language, Task Formulation (TF) (see below), can be seen as an implementation that rests upon the more general <I-N-OVA> model. The different types of constraints in the <I-N-OVA> model reflect the different types of components in an O-Plan agent (issue controller, issue handlers, and plug-in constraint managers) [Tate *et al.* 96].

Knowledge Interchange Format (KIF)

KIF [Genesereth & Fikes 92, Patil *et al*. 92] is a computer-oriented language aimed at knowledge sharing among disparate programs developed by the Interlingua Working Group of the DARPA Knowledge Sharing Effort. Disparate applications refer to programs that are written by different programmers at different times in different languages. In the DARPA knowledge sharing effort, one of the important research topics was to figure out an "interlingua" (neutral exchange language) for knowledge

---

[1] Terri Lydiard, AIAI, University of Edinburgh, "Using IDEF3 To Capture The Air Campaign Planning Process," http://www.aiai.ed.ac.uk/~arpi.
[2] See <I-N-OVA> rationale at http://www.aiai.ed.ac.uk/~oplan/inova.html

interchange. KIF was born under research in this area. At the beginning stage of design, KIF needed to have a formally defined declarative semantics, sufficient expressive power, and a structure that enables semi-automatic translation into and out of conventional knowledge representation language. Furthermore, KIF was needed to decrease the number of translators per knowledge base to one: local language to KIF and back.

KIF has a tree-like, structured syntax as well as a corresponding linear, ASCII, list-based syntax. Intuitively, KIF terms denote objects in the universe of discourse, and every sentence is either true or false. One may define objects and the relations and functions pertaining to the objects and the world. KIF provides the user with a set of basic objects, which are described by its standard vocabulary on numbers, lists, sets, functions, and relations. Thus, all KIF users must, at least, abide by the KIF conceptualization of these objects, but they are free to define other worlds of discourse otherwise. Besides the basic objects, KIF also allows expressions of meta-linguistic knowledge as well as provides supports for representations necessary for non-monotonic reasonings.

O-Plan Task Formulation

The O-Plan (Open Planning Architecture) Project [Currie & Tate 91, Tate *et al*. 96] is exploring issues of coordinated command, planning and control. The objective of the O-Plan Project at the Artificial Intelligence Applications Institute (AIAI) and the University of Edinburgh is to develop an architecture within which different agents have command (task assignment), planning, and execution monitoring roles. O-Plan is a domain independent planning system. The agents in this system require the input of a domain representation in order to complete their respective tasks. Task Formalism (TF) is used to provide this detailed knowledge. TF is a language that is used to convey a detailed description of permissible actions or operations within an application area, including information about how constraints imposed on the use of these actions should be satisfied, and their effects on the domain if the actions are used. It was originally developed for the NONLIN planner [Tate 77] in 1975 and has been extended and refined for use in O-Plan.

Task Formalism is used to give an overall hierarchical description of an application area by specifying the possible activities within the application domain and describing how those activities can be “expanded” into sets of sub-activities with a wide range of constraints imposed. Plans are generated by choosing suitable expansions for activities in the plan (i.e. refining those activities as in HTN planning) and including the sets of more detailed sub-activities described by the chosen expansions. Ordering and variable constraints are then satisfied to ensure that asserted effects of some actions satisfy, and continue to satisfy, conditions on the use of other actions. Other temporal and resource constraints are also included in the descriptions. These descriptions of actions form the main structure within TF, the schema. Schemas are also used in a completely uniform manner to describe tasks set to the planning system, in the same formalism. Other TF structures hold global information of various sorts and heuristic information about preferences for choices to be made during planning. TF can be used to represent complex knowledge about a domain. This "rich" knowledge includes action effects and conditions, hierarchical relationships, temporal requirements, authority, resource needs, etc. Its constraint-based approach provides a strong, extensible approach to domain representation. O-Plan TF is a specific language for planning and lacks some of the generality provided by a conceptual model such as <I-N-OVA> on which it is based.[1]

OZONE

OZONE [Smith & Becker 97] is a toolkit for configuring constraint-based scheduling systems developed at The Robotics Institute, Carnegie Mellon University. A central component of OZONE is its scheduling ontology, which defines a reusable and extensible base of concepts for describing and representing scheduling problems, domains and constraints.

There is commonality in scheduling system requirements and design at several levels across application domains. Many of the concepts and constraints in the problem domain could be considered to be reusable and extensible. The OZONE ontology provides a framework for analyzing the information requirements of a given target domain, and a structural foundation for constructing an appropriate domain model. Through direct association of software component capabilities with concepts in the ontology, the ontology promotes rapid configuration of executable systems and allows concentration of modeling efforts on those idiosyncratic aspects of the target domain. The OZONE ontology and toolkit represent a synthesis of extensive prior work in developing constraint-based scheduling models for a range of applications in manufacturing, space and transportation logistics.

OZONE adopts an activity-centered modeling viewpoint. There are five basic concepts of the ontology - Demand, Activity, Resource, Product, and Constraint. The ontology also defines specific inter-relationships and properties for these entities. Scheduling is defined as a process of feasibly synchronizing the use of resources by activities to satisfy demands over time, and application problems are described in terms of this abstract domain model. OZONE has a powerful architecture that permits a domain modeler to focus on those items that are special for a specific instance. The use of constraint managers assists in rapid identification of aspects to consider. While the work on OZONE reflects years of experience in the scheduling field, the ontology is still relatively new.

PAR2

PAR2 (Product-Activity-Resource Model for Realization of Electro-Mechanical Assemblies: Version 2) was developed by M.R. Duffey and J.R. Dixon in the late 1980s [Duffey 93] at the Mechanical Design Automation Laboratory of the University of

[1] O-Plan Task Formalism (TF) Manual, http://www.aiai.ed.ac.uk/release.

Massachusetts as a proof-of-concept object-based implementation designed to explore representational issues related to the interdependencies of product, process, and resource representations within the domain of electro-mechanical design. Additionally, PAR2 was designed to explore the modeling of cash flow uncertainty for the product realization process for electro-mechanical assemblies using activity network simulation.

PAR2's process representation includes an activity-on-arc representation based on the Generalized Activity Network (GAN) model of Salah Elmaghraby that allows stochastic branching and other features that enable iteration.

According to Duffey, "the implementation of PAR2 uses crude but effective object-based representations and an associated simulation engine written in common lisp. It included a (now extinct) SunView graphical interface for 1) hierarchical decomposition of product instance, activity class, and resource class representations, 2) relational matrices for product-activity and activity-resource relationships, 3) a Gantt-type chart of activity subgroups that could be manipulated to explore effects of overlapping/concurrency alternatives, and 4) a cash flow diagram dynamically created during the process simulation."

Duffey contends that "PAR2 is not meant to be a robust language, but was a doctoral research experiment. It is quite ad-hoc in its modeling of stochastic networks and lacks rigorous grounding in mathematical formalisms." Nonetheless, it features a number of important elements that can be of use in future modeling of the relationships between products, processes, resources, and even requirements.

Part 49

Part 49 (Process structure and properties) is an integrated generic resource of STEP (Standard for the Exchange of Product model data) [ISO 92, ISO 95b]. STEP is an international standard (International Standards Organization (ISO) 10303) for the exchange of product information. An integrated generic resource is a group of context-independent resource constructs used as the basis for future information. Part 49 includes the information necessary to specify the actions or potential actions to realize a process. This includes the relationships between the actions or potential actions in the process and the relationships between the processes that are used to realize a product. This part does not specify any particular process, but defines the elements to exchange process information. This part is applicable to all types of process definitions that can be represented in a discrete manner.

The constructs define the structure for specifying: relationships between processes, when a process is used, the properties of a process, the resources required for the process, the properties of the resource, the representation of process, the representation of the resource, and the relationship of the process to the product. Together, these constructs can be combined to create a process plan.

Part 49 is captured in the EXPRESS language [ISO 93]. EXPRESS is a formal data specification language that provides the mechanism for the normative description of information while allowing a complete description of the data and constraints applicable to that information. EXPRESS permits the definition of resource constructs from data elements, constraints, relationships, rules, and functions.

PERT Networks

PERT (Program Evaluation and Review Technique) [Avallone & Baumeister 87] was developed in the late 1950s to manage scheduling for the Polaris missile project. Today, PERT and its numerous descendant methodologies are widely used, especially in the project management and product development domains. According to the *Project Management Body of Knowledge* (PMBOK) [PMI 96], PERT is "an event-oriented network analysis technique used to estimate project duration when there is a high degree of uncertainty with the individual activity duration estimates."

Part of the technique is the application of the Critical Path Method (CPM), which attempts to complete projects in a minimum time by finding the particular sequence of events for which a delay in any one event in the sequence will cause a delay in overall project completion. For such a critical path, a critical event is defined as one that has no scheduling "slack;" that is, it has identical earliest possible start and latest possible start times and identical earliest possible finish and latest possible finish times. These determinations are based upon as many as three time estimates that are assigned to each activity in the PERT network: optimistic (early) completion time; most likely (average) completion time; and longest (pessimistic) completion time. However, most of the time only one time is given and it is assigned to be deterministic.

In modern practice, PERT is represented graphically by some form of project network diagram displaying a schematic of a project's activities and the logical relationships between them. According to PMBOK, the term "PERT Chart" is actually a misnomer, stating that "The project network diagram is often incorrectly called a PERT chart. A PERT chart is a specific type of project network diagram that is seldom used today."

Petri Nets

A Petri Net [Reisig 92, Peterson 81] is a graphical language that is appropriate for modeling systems with concurrency. [1] Petri Nets have been under development since the beginning of the 60's, when Carl Adam Petri defined the language in his PhD thesis (Kommunikation mit Automaten). The language was created to represent a net-like, mathematical tool for the study of communication with automata such that the concept of concurrently occurring events could be expressed. It was the first time a general theory

[1] Petri Nets, http://www.pisa.intecs.it/products/PnNICE/petrinet.html, and Petri Net WWW Pages, http://www.daimi.aau.dk/~petrinet/

for discrete, parallel systems was formulated. Petri Nets are a suitable model for a wide variety of applications. Successful areas are, for example, modeling and analysis of concurrent and parallel systems, communication protocols, performance evaluation, fault-tolerant systems, and manufacturing control systems. Since it was first introduced, Petri Nets have been modified and extended by various researchers to allow for more powerful modeling capabilities. The variations include Timed Petri Nets, Stochastic Petri Nets, Predicate/Transition Nets, Colored Petri Nets, Object Petri Nets, Compact Petri Nets, Role-based Extended Petri Net Models, Hierarchical Petri Nets, and Queueing Petri Nets.

A Petri Net is a particular kind of directed graph with an initial state called initial marking. The underlying graph of a Petri Net is a directed, bipartite graph consisting of two kinds of nodes, called places and transitions. Arcs represent connections between nodes. An arc can only connect from a place to a transition or from a transition to a place. Connections between two nodes that are of the same kind are not allowed. In graphical representation, places are drawn as circles and transitions as bars or boxes. A marking (state) is an assignment of tokens to the places of the Net. A transition is enabled if each place connected to the transition input arc (input place), contains at least one token. The firing of an enabled transition removes a token from each input place and deposits a token on each place connected with its output arcs (output place). At any given time instance, the distribution of tokens on places defined the current state of the Petri Net; thus, the modeled system. Petri Nets also allow the determination of reachability (if a reachable/obtainable from a given state) and deadlocking (if a state could be reached where the process can not proceed).

Process Flow Representation (PFR)

PFR[1] was designed as an extensible, computer- and human-readable language for describing semiconductor processing. It was created at the Massachusetts Institute of Technology to be used with the Computer-Aided Fabrication Environment (CAFE). [Boning et al. 92, McIlrath *et al*. 92] PFR was developed to explore some ideas about process modeling, design synthesis, and manufacturing.

The PFR language is a text language with a parenthesis grammar adapted from Lisp and is intended to be read and possibly interpreted (executed) by various programs for the purpose of simulation, manufacturing, or analysis. In most cases today, such programs run in CAFE, which is used to operate the MIT semiconductor fabrication facilities and includes an object oriented database. PFR includes a turing-complete programming language (adapted loosely from scheme). It also includes an extension language for accessing the environment of the executing program (e.g., the CAFE database).

[1] Process Flow Representation (PFR), http://www-mtl.mit.edu/CIDM/SemiAnnual/02.PFR.html, February 27, 1997.

Process Interchange Format (PIF) Core

Critical in Business Process Reengineering or Enterprise Integration is the ability to share and link heterogeneous process models. The goal of the PIF (Process Interchange Format) project [Lee *et al.*. 96][1] is to support the exchange of business process models across different formats and schemas. The project pursues this goal by developing PIF (a common translation language that serves as a bridge among heterogeneous process representations), translators between PIF and local process representations, and a mechanism for extending PIF to accommodate different expressive needs in a modular way (Partially Shared Views).

At the heart of PIF is a core set of classes. Some of these classes are described in the following excerpt:

> In PIF, everything is an ENTITY; that is, every PIF construct is a specialization of ENTITY. There are four types of ENTITY: ACTIVITY, OBJECT, TIMEPOINT, and RELATION. These four types are derived from the definition of process in PIF: a process is a set of ACTIVITIES that stand in certain RELATIONS to one another and to OBJECTS over TIMEPOINTS [Lee *et al*. 96].

It is the feeling of the PIF group that the Core be reduced to a bare minimum of concepts (as described above) to enable translation among those who cannot agree on anything else. Another characteristic of the Core is that it contains modules that build on one another. This way, groups with different expressive needs can share a subset of the modules, rather than the whole monolithic set of constructs. As a result of this, the Core is reduced to the minimum that is necessary to translate the simplest process descriptions and yet has built-in constructs for “hanging off” modules that extend the core in various ways.

A PIF process description consists of a set of frame definitions that are typically contained in a file. Each frame definition refers to an entity instance and is typed, and they form a class hierarchy. A frame definition has a particular set of attributes defined for it. Each of these attributes describes some aspect of the entity. The syntax of PIF adopts that of KIF (Knowledge Interchange Format). KIF is a language that has been developed by Interlingua Working Group, under the DARPA Knowledge Sharing Initiative to facilitate knowledge sharing.

Quirk Model

W. J. Quirk and R. Gilbert originally developed the Quirk Model [Motus & Rodd 94] in 1977 at the Atomic Energy Research Establishment in Harwell.[2] In its original form, it is a specification methodology for the design of distributed computer control systems. This has since been extended for the specification of real-time software. Real time in this sense

---

[1] PIF Process Interchange Format and Framework, http://soa.cba.hawaii.edu/pif/

[2] Quirk Model, http://faith.swan.ac.uk/~eegoodw/quirk.html

is taken as what is sometimes referred to as “hard” real time, i.e. all timing constraints placed upon the system must be met under all circumstances. The main concern in this model is the temporal aspect of the problem rather than the logical aspect of it. The model provides a graphical representation. The basic purpose of the model is to show the timing constraints on each of the processes in the system represented so that one can analyze the overall system performance and determine the upper and lower bounds on processing time.

A system is described in terms of sequential processes connected by channels. A channel is a link that serves as a pathway for communication/data-flow between processes. Two types of processes can be represented here: common processes and selector processes. In its graphical representation, common processes are represented as circles while selector processes are rectangles. Timing constraint parameters can be associated with each of the processes. This is denoted by a list of two parameters, min and max, which correspond to lower bound and upper bound of processing time respectively. Channels are drawn as directed links from one process to another. A channel can be asynchronous or null. A null channel (labeled with n) carries synchronization signals to a process while an asynchronous channel (labeled with a) carries information. Channels are linked to ports on the processes. Processes in the system may be concurrent or sequential. Common processes execute as data/information comes in through a channel. On the other hand, selector processes only execute when a synchronization signal comes in.

Visual Process Modeling Language (VPML)

VPML is the underlying language for the ProSLCE™ Process Editor and the Process Simulator. ProSLCSE™ is a software package developed by ISSI (International Software Systems, Inc.)[1]. Its goal is to help a company perform process engineering, which is defined as a long-term, whole system approach to help identify, analyze, and improve a business’s key processes. The hopeful result is a streamlined organization able to respond quickly and effectively to the changing business environment. This will entail a totally-integrated, process-driven environment composed of tools and services to support process capture, analysis, refinement, enforcement/enactment, and improvement.

VPML is a graphical language for defining process. Within VPML, a process is defined as a set of partially ordered steps toward meeting a goal. The components of a process diagram are activities, products, resources, and the connections between them. Activities represent work that is performed in a process and is the central focus of VPML. Products represent items (information) that are used, created, modified, and transferred among activities in a process. Resources are real-world resources that are required to perform an activity. Connections are used to establish relationships between constructs, pass product information between activities, and coordinate the scheduling of activities

[1] Visual Process Modeling Language, http://www.issi.com/proslcse-3.5i/vpml.html, August 22, 1997.

2.2 Basic Knowledge/Supporting Representations

Besides the representations discussed above, five supporting representations were identified. These representations, although not analyzed directly, were found to play a supporting role in the other twenty-six representations that were analyzed. In many cases, these twenty-six representations integrated the concepts and constructs of these supporting representations to represent information requirements that the supporting representations captured especially well. For this reason, these supporting representations were only analyzed with respect to the representations into which they were incorporated and not included in the matrix analysis.

AND/OR Graphs

An AND/OR graph [vanderBrug & Minker 75, Moses 71] is a representation of a problem together with all the possible situations and options involved in solving this problem. In other words, one can think of it as representing a problem that is divided into sub-problems. These sub-problems may, in turn, be further divided. AND/OR graphs had been developed as a general mathematical tool that could be applied to various Artificial Intelligence problems such as planning and game playing. AND/OR graphs provide a flavor of task decomposition (divide and conquer), and thus, have inspired many traditional Artificial Intelligence planning methods.

An AND/OR graph consists of a set of nodes and a set of directed links. There are two kinds of links: the AND link and the OR link. Each node represents a problem to be solved, or a goal to be achieved. Directed links connect each node to nodes representing its sub-goals or sub-problems. If a set of links originating from a node N are made to be AND links, then all the sub-problems connected to by those links need to be solved before N is considered solved. Otherwise, the links are OR links that denote that at least one of the sub-problems need to be solved. AND links are denoted graphically by drawing an arc across them, and OR links are simply left alone. If one were to denote readily solved problems by nodes which are colored black, an AND/OR graph may potentially represent a solution to the main problem. If one can find a path leading from the main problem node to a set of black nodes, provided that the siblings of any AND links in the path must also be in the path, then the path represents a solution to the problem.

Data Flow Diagrams (DFD)

Data flow Diagrams (DFDs) [Grady 93, Scotti 94] are another staple of the system engineer’s toolbox, and have also found extensive usage in many other fields including software and information design. They are primarily used as a tool for performing structured systems analyses to explore the relationships between processes and the data that they transform or create.

DFDs represent the flow of data throughout a process or between processes, depicting a system from a *data perspective* (of those who use the data), as opposed to a control perspective (of those who act upon the data), or a resource perspective (what is needed by whom and why, to do what). DFDs include specifications of the boundaries of a system, sources and destinations of data, flows of data, transformation processes, and stores of data for later use. DFDs are a graphical/diagrammatic representation with processes rendered as circles, or "bubbles." Directed arcs indicate that data move from one process to another with the name of the data labeling the arc. For each process element (bubble) in a DFD, an engineer must identify all of the inputs required, all of the outputs that the bubble must produce, and the data stores the bubble must access.

A Data Flow Diagram consists of arrangements of the following graphical entities[1]:

- Directed Arcs: capturing information flows, or the "movement" of information within the system;
- Circles/Bubbles: representing transforms, the transformation or processing of information from one physical form to another;
- Straight Lines: denoting the storage of information (e.g., data file or data base);
- Boxes: indicating either the "sources" where information comes from or the "sinks" where information goes or terminates.

DFDs are also hierarchically decomposable - each DFD at a given level can be seen to "explodes" a process bubble from a preceding level.

Directed Graphs (Digraphs)

Directed Graph [Lewis *et al*. 81, Schneider & Bruell 92] is a mathematical structure that has been widely used in many Computer Science related areas. It allows for theoretical modeling of certain kinds of computing machineries, such as Finite State Automata and Finite State machines. Besides that, it has also been frequently used to model flow of information/data within a finite number component system (e.g. network communications within a system of networked computers). As a matter of fact, due to its generality as a mathematical structure, it does not have specific semantics that limit its use to certain purposes. Directed Graphs have inspired a lot of modeling structures in various fields not limited to Computer Science, such as manufacturing and work flow management.

A Directed Graph structure consists of a set of nodes and a set of directed links. Each directed link is identified by a head node, the node pointed to by the link, and a tail node, the node from which the link originates. For each pair of nodes (A & B), there can only be one link whose tail node is A and head node, B. Furthermore, one is allowed to assign weights to each of the links.

[1] The Elements of a Data Flow Diagram, http://www.cs.pdx.edu/beta/SE_Course/Design/elements.html

State Transition Diagrams (STD)

State transition diagrams [Harel 87, Rumbaugh *et al*. 91] are one of the commonly used techniques in the software engineering community to model the dynamics of systems. One person who has had much success coming up with a formalization of this representation technique is David Harel. The representation is based on the concepts of finite state machines that are closely related to finite state automata. STD provides a graphical presentation that makes it a good visualization tool during the analysis phase of software development.

An STD consists of a set of states and a set of transitions. Actions that do not change the state of the system may be associated to a state. Each transition must be associated with an event. Furthermore, it may be associated with a set of attributes, conditions, and actions to be performed during the transition. All the states, except the start state and the final state, must be the result of an event, and upon its exit, an event must occur. Every transition must be an output of some state and the input to another state. Moreover, all transitions leaving a state must correspond to different events. Some other features of an STD include its allowance for generalizations of states and events, its ability to represent concurrent process synchronization, internal actions within a state, event sending, and automatic transitions.

Tree Structures

Tree structures [Moses 71, Sedgewick 90, Shapiro 92] are among the most commonly used data structures in Computer Science. Variations of tree structures have been found very useful in many algorithms including sorting, searching, and indexing. Among the variations are binary trees, the heap structure, B-trees, KD-trees, Quad-trees, and red-black trees. Tree structures are also used in programming languages to specify certain semantics of a language. Furthermore, compilers make use of such tree structures in parsing sentences in a program. In AI planning and game playing, tree structures are used to represent possible outcomes of some states, e.g. minimax trees and decision trees.

A tree structure consists of a set of nodes, a set of links, and a root node. Cycles are not allowed in a tree structure. The root node is a node which does not have any parent nodes. A node N is the parent of another node M if N and M are connected by a link and M is connected to the root node by less links than N is. Nodes that do not have any child nodes are called leaf nodes of the tree. By definition, each node may have multiple child nodes, but only one parent node.

3. Analysis of Process Representations

3.1 Comparing Requirements to Representations

In the initial phase of this work, a set of requirements was gathered by inspecting a number of applications that utilize process knowledge [Schlenoff *et al.* 96, Gruninger *et al.* 97]. These requirements were categorized into core, outer core, extension, and application-specific groupings. The first two categories of requirements drove most of the analysis work that was done in Phase 2. The "core" reflected those requirements that the PSL group concluded were either essential, critical, or typically common for all of the identified uses of process knowledge. The "outer core" contained requirements that were considered to be "pervasive" but not necessarily essential. The approach for Phase 2 was to identify existing representations that were believed to address some reasonable subset of these requirements. In this context, the term "representation" is used in a general sense to include languages, methodologies, standards, tools, etc. PSL working group members that were familiar with possible representations offered them as candidate sources for analysis. These candidates were then analyzed by providing both an evaluation of how well the representation addressed each of the requirements and a description of the constructs and features used.

All of the evaluations were then gathered into a cross-comparison matrix to view the results side-by-side. This approach helped to derive an overall picture of how existing representations met various process requirements in a variety of ways. It was thought that candidates that were "strong" in certain areas might suggest good representational approaches to consider for the Phase 3 development.

3.2 Matrix Rating

The evaluation of a representation was composed of: a set of "ratings", one for each requirement; a listing of constructs or structures from the representation which supported the rating; and a set of comments to further describe why a requirement received a particular rating. The possible values for each requirement rating were:

- completely satisfies
- partially satisfies
- cannot satisfy
- uncertain

It is obvious that this is a very large-grained measure and is subject to a number of perspectives on what it "means" to satisfy a requirement. A more fine-grained scale could have been used but it was concluded that it would be even harder to explain why a representation received a "kind of does satisfy" versus a "kind of doesn't satisfy", for example. In most cases a single member of the PSL working team or an outside expert who was familiar with the representation performed the analysis and the comments section was used to record the various perspectives as well as changes for a particular rating suggested by additional reviewers.

An issue that made interpretation of the results more difficult involved the varying perspectives on how to rate the individual entries. Some members performing the analysis made the judgment that a requirement could only be completely met if there was a specific construct that was specifically designed to meet a given requirement. So, for example, if a representation used a frame-based syntax and did not have a slot specifically designed for “deadline management”, then it did not “completely satisfy” the requirement. Another perspective was to look at the available constructs of a representation and to determine if there was a way in which the requirement could be expressed (e.g. deadlines can be achieved through an association of an activity status and a specific time point). A number of cross-check analyses were performed between different analysts who confirmed this difference. This led to a skewing of any attempted comparisons (based solely on the ratings) between analyses performed by different analysts. It is important to stress here though that the intent of the analysis was not to rate which representation was better overall, but to act as a generator of candidate ideas for Phase 3. The constructs identified and the comments made were very helpful for this purpose. These results demonstrated that much of the identified existing work is relevant and will be helpful inputs for meeting the required project objectives.

3.3 On-line Matrix

A number of operational issues relating to the administration and coordination of this approach were identified, including the need to:

- accept new analyses
- share results with the geographically-dispersed PSL working group and other interested researchers in the community
- see cross-comparisons between two or more analyses by different people
- view the entire results for a single representation
- comment on or change the rating for a specific entry

Addressing these issues involved the provision of a structure for a standard, centralized, and accessible collection of the analyses. This structure was provided via a dynamic, World-Wide-Web-based (WWW) tool that was implemented on the NIST PSL WWW server.[1] All participants then used the on-line matrix to submit their evaluations as well as to view, comment on, and maintain this data. Since a number of the contributors were operating on heterogeneous platforms and machines, it was decided that the WWW would be the best environment for collaboration.

4. Findings

4.1 Approach to Analyzing the Process Representation Matrix

Upon completion of the matrix, the challenge was to further organize the vast amount of information that had been collected to draw conclusions on the types of approaches useful in representing certain kinds of process specification requirements. To simplify the task of analyzing 26 approaches to over 50 process specification requirements, the set of requirements were categorized according to both their common characteristics and the commonality of representation approaches used to address the requirements.

The core and outer core PSL requirements were grouped into the following categories.

[1] The World-Wide-Web-based tool, developed by Shu Chiun Cheah at the University of Maryland, College Park consisted of a data repository, a web interface, and a set of CGI (Common Gateway Interface) scripts for handling transactions. All these components resided and ran on a web server. Although this later turned out to be rather inefficient when the data set got bigger, it enabled a temporary homogeneous user interface for data collection. The data repositories were simply files stored in a UNIX file system, containing ratings, comments, and construct descriptions. The CGI scripts, written in Perl, provided services allowing the user to begin the analysis of a new representation, to enter ratings, to enter textual comments and descriptions of constructs, and to make changes to existing data. This was all done with an interactive web interface that prompted inputs from the user. The CGI scripts handled reading and writing data to and from the data repository, were responsible for the organization of the data into files and directories, and facilitated the web interface to the data repository. Two different views of the data were provided by some of the scripts: the user was allowed to view the whole matrix as well as individual representations in table form on the web.

Resource Representation and Characteristics

- resource representation
- resource capability and characteristics
- product representation
- product characteristics
- resource grouping

Task/Process Representation and Basic Characteristics

- task representation
- task characteristics
- grouping of tasks

Resource/Task Characteristics

- task executor
- resource requirements for a task
- level of effort
- cost data
- ad hoc notes

Precedence/Sequences

- simple precedence
- simple sequences
- alternative tasks
- concurrent tasks
- conditional tasks
- iterative loops
- parallel tasks
- serial tasks

Constraints

- general constraints
- pre- and post- conditions
- state existence constraints
- temporal constraints

Date/Time

- date and time
- task duration

The matrix was analyzed across all representations to identify the constructs or features used for modeling each category of requirements. The descriptions of approaches were further reviewed to determine common representation categories. These constructs and features were then grouped into various “approaches” to help characterize the different ways that satisfaction of requirements were achieved. These results are summarized in Table 1. This table captures the high-level findings of the analysis of the on-line matrix, which documents the detailed, uninterpreted data for all representations versus requirements for process specification. Table 1 shows requirement categories in columns and construct categories in rows. For each column, the entries show whether the types of constructs are used for the particular requirement category. In this way, constructs can be associated with requirements at a high level. More detailed discussion is found in Section 4.2.

Table 1: Summary of Requirement Categories versus Representation Approaches

Constructs	Resource Rep'n	Task/Process Rep'n	Resource/Task Characteristics	Precedence /Sequences	Constraints	Date/Time
Object-Based	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
Constraint-based	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
Graph-based	-	-	-	xxxxxx	xxxxxx	xxxxxx
Relationships	-	-	-	xxxxxx	xxxxxx	xxxxxx
Conditions	-	-	-	xxxxxx	xxxxxx	-
Inference	-	-	-	xxxxxx	-	-
Task Decom-position	-	xxxxxx	xxxxxx	-	-	-
Annotations	-	xxxxxx	-	-	-	-
Activity-Resource Matrix	-	-	xxxxxx	-	-	-
Task-Network	-	-	-	-	xxxxxx	-
Classes (Groups)	-	xxxxxx	-	-	-	-
Timer	-	-	-	-	-	xxxxxx
Time Intervals /Timepoints	-	-	-	-	-	xxxxxx

4.2 Interpretation of Findings

General categories of approaches were identified for the requirements. Often, requirements were addressed through a combination of approaches. For example, a representation may use both simple, pre-defined attributes and task decomposition to address the requirement for task characteristics. Following is a discussion of general approaches for each requirement category. Note that while these explanations may refer to specific representations as examples, all representations that use an approach may not be stated. It is best to refer to the matrix containing the analysis to understand how specific representations address the process specification requirements.

Task/Process Representation

Object-based. A task is an object that has a name, and is linked to other objects via relationships and associations. Relationships can be constraints. (Units of Behavior have description, facts and objects to describe them - IDEF3) The objects and the links can be pre-defined or user-defined. Tasks as objects, or entities, can have associated attributes. These entities and attributes are either pre-defined (like Part 49), user-defined, or some combination of both.

Constraint-based. Tasks are constraints on behavior. (Node constraint - Name of Activity <I-N-OVA>)

Task Decomposition. Tasks are entities that are further described by decomposing each task. (ALPS, HTN, IDEF0)

Annotations. Tasks are further described via annotations. (O-Plan TF)

Groups of Tasks

Hierarchy / Decomposition. The most common method of grouping tasks is via a hierarchy of task flows (e.g., an ACT's plot consists of a directed graph of nodes that represents a group of actions, IDEF3 calls a hierarchy of Process Flow Diagrams a scenario, HTN calls them task networks, O-Plan TF – Action Schemas. ALPS, PIF, IDEF0 use sub-plans or task decomposition.

Classes. The other common related approach is to group classes of tasks (Part 49, AP213, PAR2) or have sets or lists of tasks (KIF).

Resource Representation and Basic Characteristics

Object-based. In this approach, resource classes, or simply entities, may be defined, which contain attributes that describe the resource. Furthermore, the resource/product characteristics can be defined as separate objects, which can then be associated to the resource objects. Among the representations that present this approach are Entity Relationship, AP213, IDEF3, O-Plan TF, JTF CPR, KIF, Petri Nets, and PAR2. ACT makes use of this approach as well except that the resource characteristics requirement is not supported. Some representations using this approach provide explicit constructs, which one may think of as some pre-defined class hierarchies, to describe resources. Hence, "resource" becomes a distinct concept within these representations. These include Part 49, ALPS, VPML, AP213, and OZONE. The resource groupings requirement is handled in three different ways. First, more specific resources can be derived from high-level resource classes. Thus, groupings by categories can be achieved through these high level classes. O-Plan TF, ACT, and PAR2 use such approach. Another approach involves the definition of a resource group, object/class, or relation. KIF and Entity Relationship use this approach. Part 49, ALPS, and OZONE have such objects/entities defined explicitly as resource groups. A third way is presented by Colored Petri Nets in which tokens that represent resources that are to be grouped under the same category are colored the same. This approach does not provide explicit information on the categories.

Constraint-based. This approach represents resources and characteristics by applying certain constraints on objects that are resources. <I-N-OVA> and IDEF3 present this approach. Some representations, like HTN, PIF Core V11, and IDEF0, represent resources within functional constructs or activities. Resources are things that will be used under certain circumstances, e.g., instead of “Resource A,” the existence of resource A is evident in something like “Use Resource A.” Within this approach, only PIF Core V.1.1 is able to support characteristics by allowing attributes to be added to the resources. It supports grouping indirectly as resources are referred to as in a group by activities when being used. So, one is able to infer a resource’s group/category that way.

Product Representation

Object-based. These include Entity Relationship, AP213, IDEF3, JTF CPR, KIF, Petri Nets, and PAR2. These representations provide ways of defining/creating general purpose entities and links/relationships, to describe products and their characteristics. Some representations have entities and attributes pre-defined explicitly for products. These include PFR, Part 49, and VPML.

Constraint-based. Another approach for representing products is shown in <I-N-OVA> in which various constraints are used to capture the information pertaining to products. O-Plan TF, HTN, and IDEF3 both make use of constraints/conditions to describe product characteristics.

Input and results of functional constructs. Finally, there is the approach in which products are the effects/output of some functional block and the input to some other functional block. This can be seen in O-Plan TF. Also, in IDEF0, PACT, and Behavior Diagrams. FFBDs has the same idea, but it simply conveys the concept of product flow. This kind of approach describes products in terms of the functional units in the representations rather than describing the product as a stand-alone object. Products, in this case, are intermediate products that exist between functional units, or are results of such functional units.

Resource/Task Combined Characteristics and Associations

This category of requirements includes specialized characteristics of tasks and/or resources that imply some association between resource and task (i.e., the role of a particular resource for a particular task is task executor).

Object-based. All these requirements are objects associated via links, relationships, and associations that can be pre-defined (e.g., ConsumableResource - JTF-CPR) or user-defined. Also, action objects can contain resource objects (JTF-CPR). Specialized requirements like cost data are handles as property/value or attribute/value pairs. These attributes or properties could be pre-defined (Part 49) or user-defined.

Constraint-based. These requirements were addressed as conditions or constraints on the task or resource. These constraints could be properties of the entity and could be pre-defined (Act) or user-defined.

Activity-Resource Matrix. PAR2 uses an activity-resource matrix that allows hierarchical decomposition.

Precedence

Object-based. This is accomplished through the use of explicit entities/attributes. In the case of Part 49, an attribute is used to specify a number describing where the activity falls in a sequence. For example, the second activity in a sequence would have the number 2 in the appropriate attribute, which implies it must be preceded by activity number 1.

Constraint-based. Constraints, e.g., temporal constraints, can be used to capture precedence.

Graph-based. Graph-based involves the use of a pictorial representation to convey the precedence relationships. Although there is certainly an underlying representation, the usual way of conveying precedence is by modifying graphical objects. Precedence is usually shown though process flow diagrams, acyclic graphs, and directed arcs.

General Relationships. General relationships include things like *affects* (causal) and *expands* (decomposition) relations. In a sense, they are the miscellaneous relations that are not as prevalent as the conditions and constraints (which can also be considered relationships).

Conditions. With respect to precedence, preconditions are the most prevalent types of conditions. Although conditions are related to constraints, they are usually very distinct in the representations under investigation.

Sequences

Object-based. This is accomplished through the use of explicit entities/attributes. Almost all of the representations that have explicit constructs are EXPRESS-based (with the exception of VPML). Although the representation is explicit, it is limited (what is defined is all you can do). In the case of sequences, there would need to be (and is, in most representations under study) an explicit construct to handle alternative tasks, concurrent tasks, iterative loops, conditional tasks, parallel tasks, and serial tasks. The attributes within these entities define what characteristics these entities have (usually a name, description, relating task, and related task).

Constraint-based. Constraints are a very robust way of representing sequences. In one sense, they are generic enough to be able to represent just about anything if defined at a high enough level. On the other hand, they can be specialized to be as detailed as desired. There are a number of ways constraints are used to represent sequences, including timing constraints, ordering constraints, disjunctive constraints, constraints on execution, schema filters (use_only_if), precedence and control flow constraints. All of these constraints can be combined in some fashion to handle all of the requirements under sequences.

Graph-based. Probably the most common mechanism to show all types of ordering. As mentioned above, although there is certainly some type of underlying representation, most of the editing is done using graphical means. Within this, logic gates seem to be a common way of representing alternative tasks, concurrent tasks, conditional tasks, and parallel tasks. Some types of graph-based representations are process flow diagrams, directed graphs that use conditional arcs/branching, arrows, linearly connected places and transitions, and channels.

General Relationships. There are a number of general relationships that can be used to describe the sequences of activities. Interval relationships can be used to show where activities lie with respect to one another, temporal relationships can show time spacing between activities (this is by far the most common), or even the lack of a relationship can show a parallel task.

Conditions. There is a fine line between conditions and constraints. Conditions can include things such as conditional branching (arcs), general if_then_else statements, conditions to determine exit criteria for a node, test metapredicates (rules), for_each iteration conditions, and pre- and post- conditions. Again, all of these can be combined in some fashion to handle all of the requirements under sequences.

Inference. One representation, PFR, allows you to infer task relationships by looking at which wafers are to be processed at which time.

Time

Relationships. ACT, for example, deals with timing issues by having temporal elements (including time windows to capture duration) relative to other temporal elements. Another variation (O-Plan TF) can have metric temporal relationships relative to an initial or zero baseline time as well.

Constraint-based. With<I-N-OVA>, metric temporal constraints are used to relate defined time points to absolute (or actual) date/time references, and ordering constraints can give relationships between time points. With HTN, task duration can be specified by constraint. IDEF3 captures duration as a UOB fact or constraint.

Object-based. HTN associates dates and times with methods, while Entity-Relationship represents them simply as attributes. Duration is captured as an explicit attribute/slot/entity by Behavior Diagrams, OZONE, Entity-Relationship, and PERT. Part 49 can also represent duration by defining a specific action_property entity, while AP213 could include it as part of an activity description. KIF would define duration as a function of a task.

Timer. Absolute references to specific dates and times are set up and recorded according to a "timer," usually as part of a simulation package or module (VPML, Behavior Diagrams, ACT). Timed Petri nets handle task duration by associating delays with either places or transitions.

Time Points and Time Intervals. OZONE and PIF capture date and time information with defined time points and time intervals. PIF,<I-N-OVA>, and JTF-CPR derive duration from defined timepoints for activity start and end times. O-Plan TF also derives task duration from metric timepoints, but all are relative to a baseline (zero) time.

Graphical. Gantt Charts show date and time information for activity start and finish times by displaying activities as bars shown within a time scale. Duration is simply correlated to the length of each activity bar against the time scale.

Constraints

Inherently Constraint-based. Many of the representation schemes examined (O-Plan TF, <I-N-OVA>, IDEF3, IDEF0, and OZONE) handle constraints "naturally", as these representation schemes are fundamentally constraint-based. This may seem a bit of a cyclical or redundant definition/explanation, but there seems no better way to explain it succinctly - constraints are the stock-in-trade media for these representations. Various types of pre-defined or user-defined condition types are typically available for specifying particular kinds of constraints.

Object-based. JTF-CPR handles constraints as objects, while PIF, Entity-Relationship, Behavior Diagrams, and PAR2 features constraints as attributes or slots. Behavior Diagrams can handle state existence constraints by defining “State Items” that can serve as “messages” between tasks - the condition of the message will dictate the state of the “receiving” task. KIF can handle temporal constraints by defining start/ end point objects.

Relationships. KIF implements pre-and post-processing constraints through the definition of relations. VPML and PIF handle temporal constraints through the various finish/start relationships between tasks.

Task-Network Constraint. HTN can capture constraints by creating a constraint formula for a task network, while Generalized Activity Networks, ACT, and Functional Flow Block Diagrams employ defined constraints for channeling control flow in the form of nodes (including logical gates, etc.).

Graphical. IDEF0 uses specialized constraint arrows to show the constraint-based relationships that exist between functions/tasks/activities.

Conditional. Many of the representations feature a conditional approach to handling pre/post processing and state existence constraints, regardless of which of the above implementation approaches they otherwise belong to. In other words, there is a technique for representing these kinds of constraints that is common to several of the approaches, be they object-based, constraint-based, etc. Many of the representations (PIF,<I-N-OVA>, HTN, Behavior Diagrams, Petri Nets, OZONE, O-Plan TF, ACT) all use some kind of true/false or other similar pre/post conditional arrangements as the basis for setting a particular set of pre/post or state constraints on a task. An example is a Petri Net, where the absence or presence of tokens in an activity’s input places determines the state of the activity upon its execution.

4.3 Conclusions of the Analysis

This study has revealed many different ways of meeting the prescribed process representation requirements. One challenge to drawing any broad conclusions from the analysis is that some representations are general approaches (e.g., Petri Nets), while others are more specialized, domain and application-focused (e.g., Part 49). By their very nature, the general approaches are more widely applicable and flexible, but require the building of additional, specialized constructs to capture some of the specific requirements. Overall results show that object-based and constraint-based approaches, in their most general form, meet all classification of requirements. However, once these approaches are instantiated and specialized (as in the case of Part 49’s representation of the object-based approach), their ability to capture process requirements becomes limited by the way they are represented. This conclusion can be misleading, however, as it is difficult to make distinctions between the various approaches because they can be seen as representing the same thing in only slightly different ways. For example, other approaches like “graph-based”, “relationships”, and “conditions” could all be viewed as alternative ways of expressing constructs in the “constraint-based” approach. This analysis and its associated observations will serve to steer the PSL project during its study of suitable constructs when building specific language presentations (syntaxes), as well as to prioritize which existing languages will be mapped to the semantic model currently being defined. The overall results from this analysis effort were presented in the April 1997 PSL Roundtable discussion [Schlenoff *et al*. 97]. A subset of the analyses that included the DARPA/Rome Laboratory Planning Initiative (ARPI) work[1] has been summarized as well [Polyak *et al* 97].

5. Summary

The original intended result of this phase of the PSL project was essentially the identification of one or more existing process representations that could be adapted for the PSL. The process of the analysis itself, however, produced a different set of results. First, the resulting in-depth understanding of the many existing approaches to representing processes provided further definition of what was required to exchange process information. Nearly all representations studied focused on the syntax of process specification rather than the semantics, or meanings, of terms. This may be sufficient when process information exchange is occurring within a single domain (e.g., process planning). However, exchange of process models among different domains creates situations where the same terms can have different meanings. A process specification that is developed for exchange must have an unambiguous set of semantic terms. For example, a concept like *work-in-progress* must have a clear, unambiguous definition in an exchange language (e.g., PSL) so that information associated with equivalent terms, such as *material* in one application and *workpiece* in another application, after being mapped to *work-in-progress*, can be exchanged. The discussion prompted by the analysis discussed in this paper focused the PSL project on defining semantic concepts inherent in process models.

The in-depth review of current process representations also provided a basic understanding of the many existing notations and presentations. This will be useful in future phases of PSL development when it is necessary to develop mappings between PSL and existing presentations of process models.

An exciting outcome of this phase of the project was the increased involvement of many related researchers. On-line email discussions involving dozens of interested parties are rapidly resolving the issues surrounding the development of an unambiguous core for the PSL. This widespread involvement is an indication of the growing momentum in the development of a neutral representation for process exchange that could result in a worldwide process specification standard.

---

[1] The representations described in [Polyak *et al.* 97] are: ACT, CPR, <I-N-OVA>, OZONE, PIF, Part 49, O-Plan TF.

Acknowledgment

The authors wish to thank key contributors to the process representation analysis, including: Florence Tissot, Knowledge Based Systems, Inc.; Adam Pease, Teknowledge, Inc.; Dimitris Kiritsis, Swiss Federal Institute of Technology at Lausanne (EPFL), Switzerland; Michael McIlrath, MIT; Michael Duffey, George Washington University; Shaw Feng, NIST.

References

[Alford 90] Mack Alford, "Strengthening the Systems Engineering Process", Ascent Logic Corporation, San Jose, CA, 1990.

[Avallone & Baumeister 87] E. Avallone and T. Baumeister III, Marks' Standard Handbook for Mechanical Engineers Ninth Edition, McGraw-Hill Inc., 1987.

[Ballard 89] Carl W. Ballard, "Basics of Behavior Diagrams", Ascent Logic Corporation, San Jose, CA, 1989.

[Boning *et al*. 92] D. S. Boning, M. B. McIlrath, P. Penfield, Jr., and E. M. Sachs, "A General Semiconductor Process Modeling Framework", *IEEE Transactions Semiconductor Manufacturing*, pp. 266-280, November, 1992.

[Nebel *et al*. 92] Bernhard Nebel, Charles Rich, and William Swartout, Ed., Principles of Knowledge Representation and Reasoning, Cambridge, MA, Morgan Kaufmann, 1992.

[Catron & Ray 91] Bryan A. Catron and Steven R. Ray, "ALPS: A Language for Process Specification", *Int. J. Computer Integrated Manufacturing*, 1991, Vol. 4, No. 2, 105-113.

[Chen 76] P. P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No 1, March 1976, p 9-36.

[Chen 81] Peter P. Chen (Ed.), "Entity-Relationship Approach to Information Modeling and Analysis", *Proceedings of the Second International Conference on Entity-Relationship Approach*, Washington, D.C., October 12 - 14, 1981. North-Holland Publishers, Amsterdam & New York City, 1981

[CSDC 94] Computer Systems Development Corporation, Flexible Computer Integrated Manufacturing (FCIM) Reverse Engineering/Re-Engineering Process Improvement, Phase 1 Report, CSDC Document Number: TR-FCM-0004-01-01, U.S. Army CECOM Contract Number DAAB07-93-D-B009, May 1994.

[Currie & Tate 91] K. Currie and A. Tate, "O-Plan: The Open Planning Architecture", *Artificial Intelligence*, 52:49-86, 1991.

[DSMC 96] Systems Engineering Management Guide, Defense Systems Management College (DSMC), December 1986.

[Duffey 93] Michael R. Duffey, "Managing the Product Realization Process: A Model for Aggregate Cost and Time-to-Market Evaluation," with J.R. Dixon, *Concurrent Engineering: Research and Applications*, London, UK no. 1, vol. 1, 1993.

[Elmaghraby 77] S. E. Elmaghraby, Activity Networks: Project Planning and Control by Network Models, John Wiley & Sons, New York: 1977.

[Erol *et al*. 94] Kutluhan Erol, James Hendler, Dana S. Nau, "Semantics for Hierarchical Task Network Planning", Technical Report, Computer Science Dept., ISR, UMIACS, University of Maryland, College Park, March 1994.

[FIPS183 93] Integration Definition for Function Modeling (IDEF0). Federal Information Processing Standards (FIPS) Publication 183, National Institute of Standards and Technology, December 21, 1993.

[Genesereth & Fikes 92] M. Genesereth and R. Fikes, "Knowledge Interchange Format Version 3.0 -- Reference Manual", Logic Group Report Logic-92-1, Stanford University, 1992.

[Georgeff & Ingrand 89] M. P. Georgeff and F. F. Ingrand, Decision-Making in an embedded reasoning system, In proceedings of the 1989 International Joint Conference on Artificial Intelligence, American Association for Artificial Intelligence, Menlo Park, CA, 1989.

[Grady 93] Jeffrey O. Grady, System Requirements Analysis, New York: McGraw-Hill, 1993.

[Gruninger *et al*. 97] M. Gruninger, C. Schlenoff, A. Knutilla, S. Ray, "Using Process Requirements as the Basis for the Creation and Evaluation of Process Ontologies for Enterprise Modeling", *ACM SIGGROUP Bulletin Special Issue on Enterprise Modelling*, Vol. 18, No. 3, 1997.

[Harel 87] D. Harel, "State Charts: A Visual Formalism for Complex Systems", *Science of Computer Programming*, Vol. 8, p. 231 - 274 (1987).

[ISO 92] ISO, "Product data representation and exchange: Part 1: Overview and fundamental principles", ISO Standard 10303-1, 1992.

[ISO 93] ISO, "Product data representation and exchange: Part 11: EXPRESS language reference manual", ISO Standard 10303-11, 1993.

[ISO 95a] ISO, "Product data representation and exchange: Part 213: Application Protocol: Numerical Control process plans for machined parts", ISO Standard 10303-213, 1995.

[ISO 95b] ISO, Product data representation and exchange: Part 49: Integrated generic resources: Process structure and properties, ISO Standard 10303-49, 1995.

[Lee *et al*. 96] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, and G. Yost, "The PIF Process Interchange Format and Framework Version 1.1", Technical Report Working Paper No. 194, MIT Center for Coordination Science, 1996.

[Lehrer 93] N. Lehrer, ARPI KRSL Reference Manual 2.0.2, Technical Report February, ISX Corporation, 1993.

[Lewis *et al*. 81] Harry R. Lewis and Christos H. Papadimitriou, <u>Elements of the theory of computation</u>, Prentice Hall, 1981.

[Lyons *et al*. 95] Kevin W. Lyons, Michael R. Duffey, and Richard C. Anderson, "Product Realization Process Modeling: A study of requirements, methods and research issues", NISTIR 5745, National Institute of Standards and Technology, Gaithersburg, MD, June 1995.

[Mayer *et al.* 92] R. Mayer *et al.*, "IDEF3 Process Description Capture Method Report", Technical Report AL-TR-1992-0057 for

Armstrong Laboratory Contract No. F33615-90-C-0012, May 1992.

[McIlrath *et al*. 92] M. B. McIlrath, D. E. Troxel, M. L. Heytens, P. Penfield, Jr., D. S. Boning, and R. Jayavant, "CAFÉ – The MIT Computer-Aided Fabrication Environment", *IEEE Transactions: Computers, Hybrids, and Manufacturing Technology*, pp. 353-360, June 1992.

[Moses 71] J. Moses, "Symbolic Integration: The Stormy Decade", *CACM*, 14(8), 548-560 (1971).

[Motus & Rodd 94] L. Motus and M. G. Rodd, Timing Analysis of Real-time Software, Pergamon Press, Oxford, UK, 1994

[Patil *et al*. 92] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches, "The DARPA Knowledge Sharing Effort: Progress Report", In [Nebel *et al*. 92], pp. 777-788, 1992.

[Pease & Carrico 97] R. A. Pease and T. Carrico, "The JTF ATD Core Plan Representation: A Progress Report", *Proceedings of the AAAI Spring Symposium on Ontological Engineering*, to appear, 1997.

[Peterson 81] J. Peterson, Petri Net Theory and Modeling Systems, Prentice Hall, NJ, 1981.

[PMI 96] PMI Standards Committee, William R. Duncan, Director of Standards, A Guide to the Project Management Body of Knowledge, Project Management Institute, Upper Darby, PA, 1996. Also available on-line http://www.pmi.org/pmi/publictn/pmboktoc1.htm

[Polyak & Tate 97] S. T. Polyak and A. Tate, "Analysis of Candidate PSL Process/Plan Representations", AIAI-PR-66, Artificial Intelligence Applications Institute (AIAI), 80 South Bridge, EH1 1HN, Edinburgh, United Kingdom, 1997.

[Ray 92] S. R. Ray, "Using the ALPS Process Plan Model", *Proceedings of the ASME Manufacturing International Conference*, Dallas, TX, 1992.

[Reisig 92]	W. Reisig, A Primer on Petri Net Design, Springer-Verlag, New York, 1992
[Rumbaugh *et al*. 91]	J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice Hall, 1991.
[Schlenoff *et al*. 96]	C. Schlenoff, A. Knutilla, S. Ray, "Unified Process Specification Language: Requirements for Modeling Process", NISTIR 5910, National Institute of Standards and Technology, Gaithersburg, MD, 1996.
[Schlenoff *et al*. 97]	C. Schlenoff, A. Knutilla, S, Ray, "Proceedings of the Process Specification Language (PSL) Roundtable", NISTIR 6081, National Institute of Standards and Technology, Gaithersburg, MD, 1997.
[Schneider & Bruell 92]	G. Michael Schneider and Steven C. Bruell, Concepts in Data Structures & Software Development, West Publishing Company, 1992.
[Scotti 94]	Richard Scotti, Lecture Notes for Engineering Management 281 *(Systems Engineering and Management)*, The George Washington University, Washington, DC, 1994.
[Sedgewick 90]	R. Sedgewick, Algorithms in C, Addison Wesley, 1990.
[Shapiro 92]	S. Shapiro, Encyclopedia of AI, 1992.
[Smith & Becker 97]	S. F. Smith and M. Becker, "An Ontology for Constructing Scheduling Systems", *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, Stanford, CA, March, 1997, AAAI Press.
[SofTech 81]	SofTech, Inc., "Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II: Volume IV - Function Modeling Manual (IDEF0)," Air Force Materials Laboratory, Wright-Patterson AFB, OH, (DTIC-B062457), June 1981.

[Tate 77] A. Tate, “Generating Project Networks,” Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77), pp. 888-893, Cambridge, MA, 1997.

[Tate 96a] A. Tate, “Representing Plans as a Set of Constraints - the <I-N-OVA> Model”, *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, Edinburgh, May 1996.

[Tate 96b] A. Tate, “Towards a Plan Ontology”, *AI*IA Notiziqe (Quarterly Publication of the Associazione Italiana per l’Intelligenza Artificiale), Special Issue on Aspects of Planning Research*, 9(1): 19-26, March 1996.

[Tate 96c] A. Tate, editor. <u>Advanced Planning Technology: Technological Advancements of the ARPA/Rome Laboratory Planning Initiative</u>, Menlo Park, CA, AAAI Press, 1996.

[Tate 97] A. Tate, “Mixed Initiative Interaction in O-Plan”, in *Proceedings of the AAAI Spring Symposium on Computer Models for Mixed Initiative Interaction*, Standford, CA, March 1997.

[Tate et al. 96] A.Tate, B. Drabble, and J. Dalton, “O-Plan: A Knowledge-based Planner and its Application to Logistics”, in [Tate 96c], pp. 259-266, 1996.

[vanderBrug & Minker 75] G. B. vanderBrug and J. Minker, “State Space, Problem-Reduction, and Theorem-Proving --- Some Relationships,” CACM 19(2), 107-115 (1975).

[Wilkens 84] D. E. Wilkens, “Domain-independent planning: representation and plan generation,” Artificial Intelligence (22), 1984.

[Wilkens & Myers 95] D. E. Wilkens and K. L. Myers, “A Common Knowledge Representation for Plan Generation and Reactive Execution,” Journal of Logic and Computation 5(6), pp. 269-301, 1995.

[Wilkens & Meyers 95] D. E. Wilkins and K. L. Myers, “A Common Knowledge Representation for Plan Generation and Reactive

Execution", SRI International Artificial Intelligence Center, 8 July 1994, *Journal of Logic and Computation*, 1995.

[Wisnosky & Batteau 90] Dennis E. Wisnosky and Allen W. Batteau, "IDEF in Principle and Practice," Gateway, May/June 1990, pp 8-11.

Appendix: Representations versus Requirements for Specifying Processes

This appendix contains the information found in the on-line matrix (http://www.nist.gov/psl/) developed through the PSL project. Note that these analyses are based on the opinions and knowledge of the individual participants and are, therefore, subjective. The value of these tables and the matrix is the information resulting about the types of constructs and features used to specify process characteristics. No approval or endorsement of any commercial product is intended or implied.

ACT

Requirements	ACT	Descriptions
ad hoc Notes	**Completely**	An individual ACT's environment conditions contain a comment slot where text can be inserted. This may include a file reference to drawings, etc. Also, there is a property slot that holds property/value pairs that can be used-defined.
Cost Data	**Partially**	ACT permits a reusable resource model that could be utilized to represent some aspects of cost. Also, cost can be treated as a property/value pairing in the properties slot.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Cannot**	No construct/feature specified.
Resource	**Completely**	Resources can be logically represented for use in ACTs.
Resource Requirements for a Task	**Completely**	A simple association of resources and an ACT is established with the resource slot.
Simple Groupings	**Completely**	An ACT's plot consists of a directed graph of nodes that represents a grouping of actions.
Simple Resource Capability/Characteristics	**Partially**	A rough approximation of characteristics of a resource can be inferred by the typed system used in ACT. (i.e., an Airplane.1 has different characteristics than a Boat.2).
Simple Sequences	**Completely**	An ACT's plot consists of a directed graph. The nodes in this graph represent actions and the links represent a partial ordering that can provide simple sequences.
Simple Task Representation and Characteristics	**Completely**	An individual ACT's environment conditions contain a comment slot where text about the actions can be inserted. Also, there is a property slot for user-defined property/value pairing that can also annotate characteristics.
Task Duration	**Completely**	Time-constraints impose any of the 13 Allen relations between actions. In addition to these constraints, time windows can be setup for start/end, duration, etc.
Task Executor	**Completely**	While there isn't an explicit slot for task executor, this property/value could be inserted into the properties slot.
Extensibility	**Completely**	The ACT properties slot provides a mechanism to allow additional user-defined information to be added to the representation.
Resource Allocation/deallocation for one or many tasks	**Completely**	An ACT USE-RESOURCE statement is used to provide a representation for resource allocation/deallocation.
Simple Precedence	**Completely**	Various constraints can be placed on the precedence orderings of actions. Temporal constraints can be used to create specific time windows, preconditions can be used to express situational constraints that must be satisfied in order to apply the act.

Requirements	ACT	Descriptions
Composition/Decomposition	**Completely**	ACTs can be arranged in a hierarchical fashion that links through the Cue gating slot in a plot. In fact, an ACT is an abstraction of a set of actions and those actions may be abstractions of other ACTs.
Incompleteness/Vagueness	**Completely**	ACTs can be defined for a task at various levels. These elements may be incomplete or vague. More detailed ACTs can be used to further elaborate a model where necessary.
Alternative Task	**Completely**	ACTs support alternative tasks in a variety of ways. A set of ACTs may share the same cue environment conditions offering alternative choices. Within a plot, conditional arcs can offer disjunctive paths.
Associated Illustrations and Drawings	**Completely**	Property/value slots can be assigned to ACTs that contain filename pointers to graphical files, etc.
Complex Groups of Tasks	**Completely**	An ACT is essentially a complex grouping of tasks (in a plot). ACTs are also connected together via Achieve, Achieve-by, etc.
Complex Resource Characteristics	**Partially**	Since ACT uses typed variables, resources could be grouped as instances of certain classes. Characteristics of a class could then be inferred.
Complex Sequences	**Completely**	ACTs (and plot nodes) can be ordered in complex relationships that support a variety of conditional, temporal possibilities.
Complex Task Representation and Parameters	**Completely**	Plots provide a very detailed expression of how a task (or grouping of actions) may be completed.
Concurrent Tasks	**Completely**	Concurrency is possible with parallel nodes in plots that can split/join the network.
Conditional Tasks	**Completely**	ACTs (and plot nodes) can use test metapredicates to provide conditional processes.
Confidence Levels	**Partially**	While there isn't direct support for this within ACT itself, there is a subsystem in the implementation (Gister-CL) that can reason about uncertain information about the world and actions.
Constraints	**Completely**	A variety of constraints can be placed on an ACT that can control things like applicability, temporal limits, etc.
Multiple Duration(s)	**Cannot**	No construct/feature specified.
Date(s) and Time(s)	**Partially**	The temporal elements (windows, durations, etc.) are all relative points to other temporal elements in the representation.
Implicit/Explicit Resource Association	**Partially**	This can partially be achieved implicitly. For instance, whenever we wish to say that if you use x, you must have y as well. (USE-RESOURCE (x y))
Iterative Loops	**Completely**	Looping is possible by linking a plot node back to an ancestor node in the graph. Test metapredicates control the number of times.
Manual vs. Automated Tasks	**Completely**	Precondition gating slots can filter which ACT is applicable.
Manufacturing Product Quantity	**Cannot**	This is not part of ACT.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	Parallel tasks can be defined using parallel plot nodes.
Parameters and Variables	**Completely**	ACT has a typed variable system that can be bound and rebound as needed.
Pre- and Post-processing Constraints	**Completely**	Preconditions and effects provide both of these.
Queues, Stacks, Lists	**Partially**	ACTs support lists of items.

Requirements	ACT	Descriptions
Resource Categorization and Grouping	**Completely**	Logical categorization and grouping can be done because resource can be considered to belong to a class of resource. (e.g. airplane.1 is an airplane, etc.)<
Resource Location	**Completely**	On pg. 23 of the cited paper, there is an example ACT that tracks resource locations.
Resource/Task Combined Characteristics	**Completely**	Multiple ACTs can be defined with different gating conditions and effects that can be used to express this requirement.
Serial Tasks	**Completely**	Serial ordering of tasks is supported.
State Existence Constraints	**Completely**	The test metapredicate can be used to evaluate state existence.
State Representations	**Completely**	State representations are central to the ACT representation.
Temporal Constraints	**Completely**	A rich set of temporal constraints can be used to cover all 13 relations.
Uncertainty/Variability/Tolerance	**Partially**	There are many ways that ACTs can express tolerance or variability of values. (For example, you can define an earliest/latest starting time, etc.)
Ability to Insert or Attach a Highlight(milestones)	**Partially**	This can be roughly approximated by adding property/value entries that are user-defined as milestones.
Complex Precedence	**Completely**	ACT provides a rich set of gating conditions and plot node orderings.
Convey the Ancestry or Class of a Task	**Completely**	An ACT's plot is essentially a specialization of the overall task.
Deadline Management	**Completely**	Time windows can express a variety of deadlines (e.g. x must happen before time1, etc.)
Dispatching	**Partially**	There isn't an explicit mechanism that is designed for this purpose, but looping, rebinding of variables and a test metapredicate should be sufficient for partial.
Eligible Resources	**Completely**	The same mechanism used to describe location of a resource can be used to create custom eligibility needs.
Exception Handling and Recovery	**Completely**	This is a central concern for PRS (which uses ACT). Conditional actions provide means to describe recovery procedures.
Information Exchange Between Tasks	**Completely**	Information is exchanged via variable bindings.
Mathematical and Logical Operations	**Completely**	ACT supports FOL as its representation system.
Support for Task/Process Templates	**Completely**	ACTs are essentially a process templates become further detailed by other ACTs.
Support for Simultaneously Maintained Associations of Mult Lev of Abstraction	**Completely**	Information can be associated with an ACT that is appropriate for that ACT's relative level in the process representation.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	Parallel nodes in ACT plots' serve to synch parallel task sequences where necessary.

A Language for Process Specification (ALPS)

Requirements	ALPS	Descriptions
ad hoc Notes	**Partially**	Ad hoc comments are supported for an entire plan. Since ALPS supports decomposition of plans, an entire plan could be the decomposition of a single task in a higher plan.
Cost Data	**Partially**	Any attribute associated with a task (including cost), is supported via a general association of a task with any number of typed attribute-value pairs
Level of Effort	**Partially**	Same support as for cost data
Product Characteristics	**Cannot**	No explicit support of product information
Resource	**Completely**	Can support the representation of resource type, instance, or capability
Resource Requirements for a Task	**Partially**	Cannot explicitly support resource quantity. ALPS assume a task requires "one of" a stated resource (either by type, instance or capability). It can support the notion of alternative resources
Simple Groupings	**Completely**	Supported using task decomposition
Simple Resource Capability/Characteristics	**Completely**	Explicitly supported, with capability lists, capability names, and descriptions, associated with resource types.
Simple Sequences	**Completely**	Directed graph representation
Simple Task Representation and Characteristics	**Partially**	Supported using task decomposition to characterize a task. Alternatively, uses the notion of "work element" to reference a member of an external library of tasks.
Task Duration	**Completely**	Explicitly supported.
Task Executor	**Completely**	Explicitly supported.
Extensibility	**Cannot**	
Resource Allocation/deallocation for one or many tasks	**Completely**	Explicitly supports resource allocation for individual or groups of tasks, including the notion of resource preemptability across several tasks.
Simple Precedence	**Completely**	Directed graph approach.
Composition/Decomposition	**Partially**	Supports the compositional notion of abstraction - i.e. a higher level description of a set of tasks which themselves are described in more detail (hierarchical task decomposition). Does not formally support ambiguity/vagueness, or partial specification.
Incompleteness/Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Completely**	Explicit support for 1, M, or all task alternatives, where M is between 1 and all.
Associated Illustrations and Drawings	**Partially**	Supports it only by defining a variable associated with an illustration or drawing.
Complex Groups of Tasks	**Partially**	Tasks can be grouped in any manner desired. You cannot, however, have the same task grouped from multiple perspectives at the same time.
Complex Resource Characteristics	**Completely**	Resources can be characterized by capability; resource instances can be members of "eligible resource sets" for possible allocation to a given task (one resource chosen from each set). Finally, resources are associated with a resource type. This allows ALPS to allocate resources to tasks by instance, type, or capability.
Complex Sequences	**Completely**	ALPS specifically supports alternatives, concurrent, and parallel tasks, as described in the requirement document. (In fact, this is where this requirement came from).

Requirements	ALPS	Descriptions
Complex Task Representation and Parameters	**Partially**	Uninterruptability is supported. The ability of describing task limits is not specifically supported, other than the value of any control variable may be an expression (possibly a predicate, but it hasn't been implemented, and thus would depend on the underlying arithmetic and symbolic manipulation language (currently TCL)).
Concurrent Tasks	**Completely**	Yes. See complex sequences.
Conditional Tasks	**Completely**	Yes. See complex sequences.
Confidence Levels	**Cannot**	
Constraints	**Cannot**	
Multiple Duration(s)	**Partially**	Could generate this behavior only by defining ad hoc variables to contain these values.
Date(s) and Time(s)	**Partially**	No explicit support for dates/times. One could use an arbitrary attribute to contain the information.
Implicit/Explicit Resource Association	**Cannot**	
Iterative Loops	**Completely**	Yes, by using the SPLIT and JOIN nodes in reverse order, you can easily create iterative loops (even nested iterative loops).
Manual vs. Automated Tasks	**Cannot**	
Manufacturing Product Quantity	**Partially**	Can create a variable to be used within a plan, passed among plans, etc. but it is not explicitly supported.
Material Constraints	**Cannot**	
Parallel Tasks	**Completely**	Yes. See complex groups of tasks.
Parameters and Variables	**Completely**	Explicitly supported. Contains input, output, and in/out parameters to a plan, plus dynamic variable binding within a plan.
Pre- and Post-processing Constraints	**Cannot**	Only supports explicit task invocations.
Queues, Stacks, Lists	**Cannot**	
Resource Categorization and Grouping	**Partially**	Resource grouping by capability, type is supported. Not by location.
Resource Location	**Cannot**	
Resource/Task Combined Characteristics	**Partially**	Preemptability of a resource in connection with a task is supported. Other task/resource properties, such as described in the document, are not.
Serial Tasks	**Completely**	
State Existence Constraints	**Cannot**	
State Representations	**Cannot**	
Temporal Constraints	**Partially**	...other than the simplest - task precedence.
Uncertainty/Variability/Tolerance	**Cannot**	
Ability to Insert or Attach a Highlight(milestones)	**Cannot**	
Complex Precedence	**Partially**	Cannot support partially ordered graphs, but can support conditional precedence. Cannot explicitly support arbitrary constraints on precedence.
Convey the Ancestry or Class of a Task	**Partially**	I am not interpreting this as task decomposition, which ALPS can support. I interpret this as supporting the expression of the class to which a given task belongs, such as with task templates. ALPS does support the notion of a "work element" which some might interpret as the task template.
Deadline Management	**Cannot**	No explicit support for this.
Dispatching	**Cannot**	No explicit support.

Requirements	ALPS	Descriptions
Eligible Resources	**Completely**	Use the Eligible Resource Set entity to contain eligible resources by instance, type (class) or capability.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Completely**	Support the use of variables that are global in scope within a given plan, plus parameters for passing of information between tasks in different plans.
Mathematical and Logical Operations	**Partially**	ALPS is designed to operate with an embedded mathematical and logical engine. Thus, it doesn't explicitly define the behavior, but does prescribe the requirements for math and logic operations.
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Mult Lev of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	Uses various kinds of semaphores (rendezvous, lock, unlock...) to accomplish a variety of synchronization scenarios.
Business Practices, Rules, Constraints	**Cannot**	No construct/feature specified.
Configuration Management Information and Processes	**Cannot**	No construct/feature specified.
Customer-driven Process Specification and Constraints	**Cannot**	No construct/feature specified.
Forecast and Customer Orders	**Cannot**	No construct/feature specified.
Priorty Attributes	**Cannot**	No construct/feature specified.
Representation of the Origin of Task(s)	**Cannot**	No construct/feature specified.
Analysis Characteristics	**Cannot**	No construct/feature specified.
Critical Task	**Cannot**	No construct/feature specified.
Predictive and Time-dependent Resource Availability	**Cannot**	No construct/feature specified.
Prescriptive Task Behavior	**Cannot**	No construct/feature specified.
Task/Process Performance Measurement	**Cannot**	No construct/feature specified.
Co-existence of Plans and Resolution of Conflicts	**Cannot**	No construct/feature specified.
Dynamic Model Modification	**Cannot**	No construct/feature specified.
Optimization	**Cannot**	No construct/feature specified.
Resource/System/Process Monitoring and Feedback	**Partially**	Can support the acquisition of sensor feedback through variables.
Support for Validation of the Entire Process Plan	**Cannot**	No construct/feature specified.
Tracking of Changes in the System	**Cannot**	No construct/feature specified.
What-if Analysis	**Cannot**	No construct/feature specified.
Resource Amount and Availability	**Cannot**	No construct/feature specified.
Resource Interruptions	**Cannot**	No construct/feature specified.
Process Yield	**Cannot**	No construct/feature specified.

Requirements	ALPS	Descriptions
Dynamic Model Modification	**Cannot**	No construct/feature specified.
Event Signaling and Notification	**Completely**	Achieved through the use of semaphores.
Resource Behavior During Processing Time	**Cannot**	No construct/feature specified.
Resource/System/Process Monitoring and Feedback	**Cannot**	No construct/feature specified.
Tracking of Changes in the System	**Cannot**	No construct/feature specified.
Track In-progress Goods	**Cannot**	No construct/feature specified.
Decision Rationale	**Cannot**	No construct/feature specified.
Intentional Dimension of Processes, or Goals	**Cannot**	No construct/feature specified.
Relationship between Task and Goal and Resource and Goal	**Cannot**	No construct/feature specified.
Task/Process Purpose	**Cannot**	No construct/feature specified.
Value-added Attributes	**Partially**	ALPS supports arbitrary variables to be associated with any task
Access to Past and Present Decision Rationales	**Cannot**	No construct/feature specified.
Characteristics of Groups of Resources	**Partially**	Only insofar as eligible resources can be grouped in association with a task, and grouped by capability and through a resource class taxonomy.
Implicit Task Association	**Cannot**	No construct/feature specified.
Parallelism	**Cannot**	No construct/feature specified.
Descriptive Manufacturing/Performance Variability	**Cannot**	No construct/feature specified.
Prob of Down Times	**Cannot**	No construct/feature specified.
Stochastic Properties	**Cannot**	No construct/feature specified.
Uncertainty of Sequences	**Cannot**	No construct/feature specified.
Account for Randomness	**Cannot**	No construct/feature specified.
Stochastic Functionality	**Cannot**	No construct/feature specified.
Prod Scheduling - production type data	**Cannot**	No construct/feature specified.
Prod Scheduling - dynamic rescheduling	**Cannot**	No construct/feature specified.
Process Planning – clamping surfaces	**Cannot**	No construct/feature specified.
Process Planning – datums and offsets	**Cannot**	No construct/feature specified.
Process Planning – features to be machined	**Cannot**	No construct/feature specified.
Process Planning – production type date	**Cannot**	No construct/feature specified.
Simulation - queue entry and exit rates	**Cannot**	No construct/feature specified.
Enterprise Eng. and Bus Process Re-Eng. - conceptual entities	**Cannot**	No construct/feature specified.
Workflow - manual vs. automatable tasks	**Cannot**	No construct/feature specified.

Requirements	ALPS	Descriptions
Workflow - invoked tool capability	**Cannot**	No construct/feature specified.
Workflow - support specifications of task structure (control flow)	**Cannot**	No construct/feature specified.
Project Management - work breakdown ids	**Cannot**	No construct/feature specified.

AP213

Requirements	AP213	Descriptions
ad hoc Notes	**Partially**	Notes on the process plan can be entered in the Entity Auxiliary_header_information. Notes on any activity can be entered in the Attribute Description of the Entity Activity.
Cost Data	**Cannot**	None. Costing data is out of the scope of AP 213.
Level of Effort	**Cannot**	None.
Product Characteristics	**Partially**	The entity Part_version captures the high-level description of the product.
Resource	**Completely**	The AP213 model captures resources of machines, fixtures, fixture assemblies, tools, tool assemblies, and their spatial relationships.
Resource Requirements for a Task	**Completely**	Machine, tool, and fixture requirements for an operation are supported.
Simple Groupings	**Completely**	Activity_group and Activity are used together for grouping tasks in AP213.
Simple Resource Capability / Characteristics	**Completely**	High-level description of machining resource is modeled in AP213, e.g., machine and tool description and parameters are captured in model.
Simple Sequences	**Completely**	Each activity is numbered. Activities are linearly sequenced.
Simple Task Representation and Characteristics	**Completely**	Attributes name and description of Entity Activity are used for the high-level description of a task.
Task Duration	**Partially**	Task duration can be stated in the Description of Activity.
Task Executor	**Partially**	Executor information can be stated in Description of Activity.
Extensibility	**Partially**	The entity Ancillary_action is a catch-all for all other possible machining activities.
Resource Allocation/deallocation for one or many tasks	**Partially**	AP 213 includes the allocation of resource but not deallocation.
Simple Precedence	**Completely**	The precedence is defined by the activity number.
Composition / Decomposition	**Completely**	Entities Activity and Activity_group and assertions is_an and is_composed_of are used for composition and decomposition of tasks.
Incompleteness / Vagueness	**Cannot**	AP213 does not capture resource availability information.
Alternative Task	**Completely**	Entity Alternate_activity can be used to capture alternative tasks.
Associated Illustrations and Drawings	**Completely**	Drawing entity and its assertions provide pointers that point to part, tool, and fixture drawing information.
Complex Groups of Tasks	**Partially**	Activity entity points to machine and tools, not the other direction. (However, complex grouping of tasks can be perform in software.)
Complex Resource Characteristics	**Partially**	Resource-related entities, such as machine, tool_assembly, fixture_assembly, etc.
Complex Sequences	**Partially**	Activity and Alternate_activity entities.

Requirements	AP213	Descriptions
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Cannot**	No construct/feature specified.
Conditional Tasks	**Cannot**	No construct/feature specified.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Cannot**	No construct/feature specified.
Multiple Duration(s)	**Cannot**	No construct/feature specified.
Date(s) and Time(s)	**Cannot**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Cannot**	No construct/feature specified.
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Cannot**	No construct/feature specified.
Parameters and Variables	**Partially**	Process parameter entities are specified in AP213.
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Partially**	AP213 specifies task-resource requirement relationship.
Serial Tasks	**Completely**	Activity entity has a number attribute, which specifies that tasks should be in sequence.
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Cannot**	No construct/feature specified.
Uncertainty/Variability/ Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight(milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	Tasks hierarchy can be specified using Activity_group and Activity entities in the ARM.
Convey the Ancestry or Class of a Task	**Completely**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.

Requirements	AP213	Descriptions
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

Behavior Diagrams

Requirements	Behavior Diagrams	Descriptions
ad hoc Notes	**Completely**	Attached Text
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Completely**	Every function/ activity within the diagram can have input and output items associated with them. These are intermediate products of the whole process, as the diagrams are hierarchically decomposable.
Resource	**Cannot**	No construct/feature specified.
Resource Requirements for a Task	**Cannot**	No construct/feature specified.
Simple Groupings	**Completely**	Time-Items and R-nets can be used to group several discrete activities.
Simple Resource Capability / Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Completely**	No construct/feature specified.
Simple Task Representation and Characteristics	**Completely**	Each activity is named according to its function - what the activity is supposed to do/produce. Further description can be supplied in attached text.
Task Duration	**Completely**	Time Duration is a standard attribute that can be populated for any activity.
Task Executor	**Not sure**	No construct/feature specified.
Extensibility	**Completely**	Slots for additional attributes can be defined; time items can be used as placeholders for future definition of parts of the network.
Resource Allocation/deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Completely**	ordering of connected functions in the diagram specifies precedence constraints, subject to defined conditions and logic nodes.

Requirements	Behavior Diagrams	Descriptions
Composition / Decomposition	**Completely**	R-Nets can be used as summary tasks. Functions can be decomposed into more primitive functions. Time-Items can be decomposed down until reaching Discrete Items at lowest level.
Incompleteness / Vagueness	**Not sure**	No construct/feature specified.
Alternative Task	**Completely**	Logic Gates/ conditional branching
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Completely**	Fully supported by logic gates/nodes: Concurrency Node (and), Selection Node (or), Iteration, Loops, Exit Loops.
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Completely**	Concurrency Nodes (and gates)
Conditional Tasks	**Completely**	Each function can have multiple exit or completion criteria defined. Depending on the conditions at the time a function would be completed, certain exit paths will be followed.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Partially**	Each functions can have entrance and exit criteria defined. All types of constraints are covered, except for resource constraints. While resource constraints are not directly supported, they could be "fudged" by defining an input item that is actually a resource.
Multiple Duration(s)	**Completely**	Various Probability distributions can be used to capture multiple possible durations.
Date(s) and Time(s)	**Completely**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Completely**	Iteration is used to represent *a specified number* of repetitive sequences or functions. A feedback path connects iteration nodes at either end of the repetitive behavior. The number of times the behavior is to be repeated is specified by definition of a "domain set." Loops are like iterations but repeats infinitely. Exit loops iterate until a specified exit condition is satisfied.
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	Concurrency nodes (when simultaneous), or multiple exit paths from a function.
Parameters and Variables	**Not sure**	This might be possible. Would involve further investigation and experiment.
Pre- and Post-processing Constraints	**Completely**	Entrance and exit conditions are fully supported.
Queues, Stacks, Lists	**Not sure**	No construct/feature specified.

Requirements	Behavior Diagrams	Descriptions
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Not sure**	While resources are not explicitly represented, their effects on functions could possibly be simulated by defining particular entrance/exit conditions, or by the definition of input items.
Serial Tasks	**Completely**	Functions can be lined up one after the other to form a serial process.
State Existence Constraints	**Completely**	A Function can send a message embodied as a "State Item." Various states can be recorded, depending on the state of the "transmitting" function. This State Item can then be used to control or influence subsequent functions, as an input item.
State Representations	**Not sure**	It may be possible, although definitely not in terms of resource state.
Temporal Constraints	**Completely**	No construct/feature specified.
Uncertainty/Variability/ Tolerance	**Completely**	Probability distributions for durations and exit conditions.
Ability to Insert or Attach a Highlight(milestones)	**Completely**	Dummy (duration zero) tasks are allowed.
Complex Precedence	**Completely**	Yes on all counts. See previous descriptions.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Not sure**	No construct/feature specified.
Dispatching	**Not sure**	Construct for this specific feature not present, but might be able to come up with something that simulates it using the constraint features.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Completely**	See descriptions of constraints and complex sequences, etc. Exit looping, conditional features, multiple exit conditions, etc. should allow this.
Information Exchange Between Tasks	**Completely**	This is a key feature of Behavior Diagrams. While functional (precedence/control) flow moves downward in the diagram, "Items" (which can represent I/O products, messages, data, information, state conditions, etc.) flow between functions horizontally.
Mathematical and Logical Operations	**Completely**	Logic Nodes (and, or, looping, etc.) are present. Also, criteria/conditions for multiple exit paths can be expressed mathematically.
Support for Task/Process Templates	**Not sure**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Not sure**	Items can be set up to flow across/between different levels of the decomposition, but I'm not sure how the representation would satisfy this particular requirement.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	Concurrency nodes can be used to insure that two sub-processes start together. Once started, messages or other items can connect functions further along in the respective parallel processes that can be used to insure that things proceed in a coordinated way.

EPFL's Petri net Representation[1]

Requirements	EPFL	Descriptions
ad hoc Notes	**Not sure**	No construct/feature specified.
Cost Data	**Partially**	No construct/feature specified.
Level of Effort	**Not sure**	No construct/feature specified.
Product Characteristics	**Not sure**	No construct/feature specified.
Resource	**Not sure**	No construct/feature specified.
Resource Requirements for a Task	**Partially**	No construct/feature specified.
Simple Groupings	**Completely**	No construct/feature specified.
Simple Resource Capability/Characteristics	**Not sure**	No construct/feature specified.
Simple Sequences	**Completely**	No construct/feature specified.
Simple Task Representation and Characteristics	**Completely**	No construct/feature specified.
Task Duration	**Completely**	No construct/feature specified.
Task Executor	**Not sure**	No construct/feature specified.
Extensibility	**Not sure**	No construct/feature specified.
Resource Allocation/deallocation for one or many tasks	**Not sure**	No construct/feature specified.
Simple Precedence	**Completely**	No construct/feature specified.
Composition/Decomposition	**Not sure**	No construct/feature specified.
Incompleteness/Vagueness	**Not sure**	No construct/feature specified.
Alternative Task	**Completely**	No construct/feature specified.
Associated Illustrations and Drawings	**Not sure**	No construct/feature specified.
Complex Groups of Tasks	**Not sure**	No construct/feature specified.
Complex Resource Characteristics	**Not sure**	No construct/feature specified.
Complex Sequences	**Completely**	No construct/feature specified.
Complex Task Representation and Parameters	**Not sure**	No construct/feature specified.
Concurrent Tasks	**Completely**	No construct/feature specified.
Conditional Tasks	**Not sure**	No construct/feature specified.
Confidence Levels	**Not sure**	No construct/feature specified.
Constraints	**Partially**	No construct/feature specified.
Multiple Duration(s)	**Not sure**	No construct/feature specified.
Date(s) and Time(s)	**Not sure**	No construct/feature specified.
Implicit/Explicit Resource Association	**Not sure**	No construct/feature specified.
Iterative Loops	**Not sure**	No construct/feature specified.
Manual vs. Automated Tasks	**Not sure**	No construct/feature specified.
Manufacturing Product Quantity	**Not sure**	No construct/feature specified.
Material Constraints	**Not sure**	No construct/feature specified.
Parallel Tasks	**Partially**	No construct/feature specified.

[1] PAct (Parts and Actions) and EPFL's petri net representations, were only minimally analyzed because of lack of expertise and literature available at the time of analysis, therefore, there were many "not sure" ratings.

Requirements	EPFL	Descriptions
Parameters and Variables	**Not sure**	No construct/feature specified.
Pre- and Post-processing Constraints	**Not sure**	No construct/feature specified.
Queues, Stacks, Lists	**Not sure**	No construct/feature specified.
Resource Categorization and Grouping	**Not sure**	No construct/feature specified.
Resource Location	**Not sure**	No construct/feature specified.
Resource/Task Combined Characteristics	**Not sure**	No construct/feature specified.
Serial Tasks	**Not sure**	No construct/feature specified.
State Existence Constraints	**Not sure**	No construct/feature specified.
State Representations	**Not sure**	No construct/feature specified.
Temporal Constraints	**Completely**	No construct/feature specified.
Uncertainty/Variability/Tolerance	**Not sure**	No construct/feature specified.
Ability to Insert or Attach a Highlight(milestones)	**Not sure**	No construct/feature specified.
Complex Precedence	**Not sure**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Not sure**	No construct/feature specified.
Deadline Management	**Not sure**	No construct/feature specified.
Dispatching	**Not sure**	No construct/feature specified.
Eligible Resources	**Not sure**	No construct/feature specified.
Exception Handling and Recovery	**Not sure**	No construct/feature specified.
Information Exchange Between Tasks	**Not sure**	No construct/feature specified.
Mathematical and Logical Operations	**Not sure**	No construct/feature specified.
Support for Task/Process Templates	**Not sure**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Not sure**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Partially**	No construct/feature specified.

Entity-Relationship

Requirements	Entity-Relationship	Descriptions
ad hoc Notes	**Cannot**	
Cost Data	**Partially**	A task could be represented as an entity of which the cost data are attributes.
Level of Effort	**Partially**	Resources can be represented as attributes of a task. Their respective levels of effort would simply be the values of these attributes.
Product Characteristics	**Partially**	A product could be represented as an entity whose characteristics are simply its attributes.
Resource	**Partially**	Resources can be represented as entities. For example, "people" is represented by an entity labeled "people." Similarly, "milling machine" can be an entity labeled "milling machine."

Requirements	Entity-Relationship	Descriptions
Resource Requirements for a Task	**Partially**	A resource can be associated with a task by drawing a "relationship" between these two entities.
Simple Groupings	**Completely**	Tasks may be grouped together using the "relationship" construct if we represent tasks as entities.
Simple Resource Capability / Characteristics	**Partially**	By representing resources as entities, we can specify the characteristics and capabilities of these resources as the attributes of the entities.
Simple Sequences	**Partially**	Tasks as entities; then, we can draw "executed before" relationships from one entity to another, thus, forming a simple sequences.
Simple Task Representation and Characteristics	**Partially**	Similar to resources, the characteristics of a task may be represented as the attributes of the task entity.
Task Duration	**Partially**	Can be represented as attributes of a task.
Task Executor	**Partially**	A task executor can be represented as a separate entity with a relationship drawn to some task entities.
Extensibility	**Completely**	One can certainly add more entities, relationships, and attributes to the existing data structures.
Resource Allocation / deallocation for one or many tasks	**Cannot**	
Simple Precedence	**Partially**	Precedence is just another type of relationship between task entities. The distinction can be made by labeling the relationship links differently.
Composition / Decomposition	**Partially**	One can make use of the "isa" relationship to specify generalization/specialization of a task or a resource. This is similar to composition/decomposition in the sense that it provides one with the construct of representing a hierarchical structure of information.
Incompleteness / Vagueness	**Partially**	One can assume that an entity that is not linked with any “isa” relationships is good by itself. And, if in the future, more details are to be added, they can be added using the "isa" relationship. Thus, these additions become the breakdowns of the task into further details. However, there is no way of saying that these breakdowns are complete or not.
Alternative Task	**Partially**	Alternative tasks can be represented as entities, each of which has a relationship link to an entity which is labeled as the function that all of them are meant to be able to perform.
Associated Illustrations and Drawings	**Cannot**	
Complex Groups of Tasks	**Partially**	A group of tasks that are related in some way may be tied together by relationship links. However, there is no construct for drawing relationship to a group of tasks without having to draw link to each of the entities in that group.
Complex Resource Characteristics	**Partially**	By representing resources as entities, their characteristics may be represented as attributes.
Complex Sequences	**Partially**	These can all be represented by drawing the appropriate relationship links among the tasks involved.
Complex Task Representation and Parameters	**Partially**	Entities with attributes can represent tasks and their respective parameters and characteristics.

Requirements	Entity-Relationship	Descriptions
Concurrent Tasks	**Partially**	Concurrent tasks can be represented as entities which are linked by some sort of "begin at same time" relationship.
Conditional Tasks	**Cannot**	
Confidence Levels	**Partially**	Confidence levels of a task may be represented as attributes of that task entity.
Constraints	**Partially**	Tasks or resources can be represented as entities, and the constraints can then be the attributes of those entities.
Multiple Duration(s)	**Partially**	Multiple durations associated to a task or a resource may be represented as attributes to the task/resource entity. One can simply have attributes such as "duration 1," "duration 2," and so on.
Date(s) and Time(s)	**Partially**	Similar to multiple durations, these can be represented as attributes as well.
Implicit/Explicit Resource Association	**Partially**	The association may be represented as relationship link between the resource entities involved.
Iterative Loops	**Cannot**	
Manual vs. Automated Tasks	**Partially**	One could have manual vs. automated as a flag attribute for the task entity involved.
Manufacturing Product Quantity	**Partially**	The products can be represented as entities, and the quantities as the attributes.
Material Constraints	**Partially**	Materials as entities, constraints as attributes.
Parallel Tasks	**Partially**	With tasks represented as entities, we can designate some sort of "occurrence time" attribute for each of the task. With no time constraint relationship linked between these tasks, they may occur at any time, as specified in the attribute.
Parameters and Variables	**Partially**	Attributes to entities are certainly placeholders for values. Furthermore, updates may be performed on these attribute values at any time.
Pre- and Post-processing Constraints	**Partially**	These can be specified as attributes.
Queues, Stacks, Lists	**Partially**	Queues may be represented by having a "front flag" attribute and an "end flag" attribute for each entity. Thus, the entity at the front would have a true for "front flag." Likewise for the end entity. The entities may be linked together with "next" relationship links. Stacks, the same way with a "top flag" and a "bottom flag." Lists, the same way without any flags.
Resource Categorization and Grouping	**Partially**	This can be achieved by having an entity for the particular characteristic of interest, and all the resources that share this characteristic will have entities linked to this characteristic entity via some relationship links.
Resource Location	**Partially**	Location of a resource may be specified as an attribute of that resource entity.
Resource/Task Combined Characteristics	**Partially**	Such characteristics may be specified as the attributes to the relationship links connecting tasks to their associated resources.
Serial Tasks	**Partially**	Serial tasks may be linked, one after another, by some sort of "performed after" relationship links.
State Existence Constraints	**Partially**	This can be represented as an attribute of a task.

Requirements	Entity-Relationship	Descriptions
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Partially**	Can be represented using attributes of entities.
Uncertainty / Variability / Tolerance	**Partially**	This can also be represented as an entity's attributes.
Ability to Insert or Attach a Highlight (milestones)	**Partially**	This may be accomplished by having a highlight flag as an attribute to the entities to be highlighted.
Complex Precedence	**Partially**	The tasks to which the precedence constraints are applied to can be linked with relationship links whose attributes specify the details of the constraints.
Convey the Ancestry or Class of a Task	**Completely**	"ISA" relationship links can represent specialization/generalization of tasks. Furthermore, the attributes of the higher-level tasks(entities) are automatically inherited by the specialized tasks.
Deadline Management	**Partially**	Deadlines can be written as attributes of an entity or a relation. Thus, the user of E-R model is able to consider any sort of predetermined deadline when making any decisions.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Partially**	The attributes of entities and relations are basically data stores to which the user can enter values. However, the values need to be shown in a separate table rather than on the E-R diagram itself.
Support for Simultaneously Maintained Associations of Mult Lev of Abstraction	**Completely**	Associations of information with a task are accomplished by linking the task entity with other entities representing the information (e.g. resource) with relations. At each level of the "ISA" generalization structure, the entities are allowed to be linked to other entities by relations. Thus, information can be associated at multiple levels.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.
Business Practices, Rules, Constraints	**Cannot**	No construct/feature specified.
Configuration Management Information and Processes	**Cannot**	No construct/feature specified.
Customer-driven Process Specification and Constraints	**Cannot**	No construct/feature specified.
Forecast and Customer Orders	**Partially**	Orders may be represented as entities in an E-R model. Their attributes can represent any information related to the orders.

Requirements	Entity-Relationship	Descriptions
Priorty Attributes	**Partially**	These could be represented as the attributes of the respective task entities.
Representation of the Origin of Task(s)	**Cannot**	No construct/feature specified.
Analysis Characteristics	**Partially**	Analysis results may be represented separate entities. The attributes of these entities are, then, the different characteristics of the analysis.
Critical Task	**Partially**	Critical task entity can have a "critical task flag" attribute that indicates such characteristic.
Predictive and Time-dependent Resource Availability	**Cannot**	No construct/feature specified.
Prescriptive Task Behavior	**Cannot**	No construct/feature specified.
Task/Process Performance Measurement	**Cannot**	No construct/feature specified.
Co-existence of Plans and Resolution of Conflicts	**Cannot**	No construct/feature specified.
Dynamic Model Modification	**Cannot**	No construct/feature specified.
Optimization	**Cannot**	No construct/feature specified.
Resource/System/Process Monitoring and Feedback	**Cannot**	No construct/feature specified.
Support for Validation of the Entire Process Plan	**Cannot**	No construct/feature specified.
Tracking of Changes in the System	**Cannot**	No construct/feature specified.
What-if Analysis	**Cannot**	No construct/feature specified.
Resource Amount and Availability	**Partially**	With resources represented as entities, the amount and availability of a resource can be specified in the attributes of the resource entity.
Resource Interruptions	**Cannot**	No construct/feature specified.
Process Yield	**Cannot**	No construct/feature specified.
Dynamic Model Modification	**Cannot**	No construct/feature specified.
Event Signaling and Notification	**Cannot**	No construct/feature specified.
Resource Behavior During Processing Time	**Cannot**	No construct/feature specified.
Resource/System/Process Monitoring and Feedback	**Cannot**	No construct/feature specified.
Tracking of Changes in the System	**Cannot**	No construct/feature specified.
Track In-progress Goods	**Cannot**	No construct/feature specified.

Requirements	Entity-Relationship	Descriptions
Decision Rationale	**Cannot**	No construct/feature specified.
Intentional Dimension of Processes, or Goals	**Cannot**	No construct/feature specified.
Relationship between Task and Goal and Resource and Goal	**Cannot**	No construct/feature specified.
Task/Process Purpose	**Cannot**	No construct/feature specified.
Value-added Attributes	**Cannot**	No construct/feature specified.
Access to Past and Present Decision Rationales	**Cannot**	No construct/feature specified.
Characteristics of Groups of Resources	**Cannot**	No construct/feature specified.
Implicit Task Association	**Partially**	Tasks may be associated to one another through relationship links.
Parallelism	**Cannot**	No construct/feature specified.
Descriptive Manufacturing/Performance Variability	**Cannot**	No construct/feature specified.
Probability of Down Times	**Partially**	A "probability of down times" attribute can be added to the resource entity for which such information needs to be specified.
Stochastic Properties	**Cannot**	No construct/feature specified.
Uncertainty of Sequences	**Cannot**	No construct/feature specified.
Account for Randomness	**Cannot**	No construct/feature specified.
Stochastic Functionality	**Cannot**	No construct/feature specified.
Prod Scheduling - production type data	**Cannot**	No construct/feature specified.
Prod Scheduling - dynamic rescheduling	**Cannot**	No construct/feature specified.
Process Planning – clamping surfaces	**Cannot**	No construct/feature specified.
Process Planning – datums and offsets	**Cannot**	No construct/feature specified.
Process Planning – features to be machined	**Cannot**	No construct/feature specified.
Process Planning – production type date	**Cannot**	No construct/feature specified.
Simulation - queue entry and exit rates	**Cannot**	No construct/feature specified.
Enterprise Eng. and Bus Process Re-eng - conceptual entities	**Cannot**	No construct/feature specified.

Requirements	Entity-Relationship	Descriptions
Workflow - manual vs. automatable tasks	**Partially**	The task entities may have a flag attribute signifying whether the task is manual or automatable.
Workflow - invoked tool capability	**Cannot**	No construct/feature specified.
Workflow - support specifications of task structure (control flow)	**Cannot**	No construct/feature specified.
Project Management - work breakdown ids	**Partially**	The ids can be an attribute of some entity acquiring such an id.

Functional Flow Block Diagrams (FFBD)

Requirements	FFBD	Descriptions
ad hoc Notes	**Completely**	Attached text
Cost Data	**Cannot**	Cannot represent cost data in current form - could possibly be made to carry costs if modified, but would also require the addition of an explicit representation of duration and resources.
Level of Effort	**Cannot**	FFBDs represent functional/ activity flows - no constructs for resources at present time.
Product Characteristics	**Partially**	Directed Arcs between function blocks could be used to represent product flow (each function would have inputs and outputs that are intermediate products).
Resource	**Cannot**	Would require modifications & enhancements to capture resource info. (might be included in attached text in practice).
Resource Requirements for a Task	**Cannot**	No construct/feature specified.
Simple Groupings	**Completely**	Hierarchical Decomposition. At the highest level of abstraction, an entire process can be specified as one single all-encompassing functional block. Lower-level sets of activities/functions can always be summarized by a simpler set of higher level tasks.
Simple Resource Capability/Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Completely**	Arrows between function blocks denote logical sequencing
Simple Task Representation and Characteristics	**Completely**	Function Blocks represent discrete tasks. Each block carries a name to denote what its function is (what it "does").
Task Duration	**Partially**	FFBDs were not intended to carry explicit duration times, but it would be very easy to modify them so they could.
Task Executor	**Cannot**	FFBDs are traditionally used at a point in the design process where specific resources and executors have not yet been defined or assigned to specific functions.
Extensibility	**Completely**	Associated Text

Requirements	FFBD	Descriptions
Resource Allocation/deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Completely**	Directed Arcs (arrows)
Composition/Decomposition	**Completely**	Hierarchical Decomposition. For FFBDs, this applies to functions and the flows between them. These are described in increasing detail at lower levels.
Incompleteness/Vagueness	**Completely**	FFBDs have been traditionally used to describe product & process functionality, and operate within a framework of uncertainty.
Alternative Task	**Completely**	Logic Gates. Contingent or alternative courses of action (activities/functions) can be specified by using a simple or inclusive or gate to switch activity flow.
Associated Illustrations and Drawings	**Cannot**	FFBDs (as traditionally used) capture what a process need to do and the sequence by which to do it, but do not assume a particular answer to "how" a function will be performed.
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Partially**	Logic Gates. Alternative, serial, and parallel tasks are fully supported. However, concurrent tasks are not explicitly supported. FFBDs, as typically used, do not include the kind of timing constraints required. Could easily be modified to enforce concurrency.
Complex Task Representation and Parameters	**Not sure**	Some information might wind up on the associated text entries, but there is no explicit mechanism in the representation for satisfying this requirement.
Concurrent Tasks	**Partially**	tasks can be parallel, but FFBDs would need modification in order to force concurrency.
Conditional Tasks	**Completely**	Each branch (arrow) diverging from an "or" gate can be annotated with the conditions that would cause the process to flow along it.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Partially**	Diagrams capture precedence/ control-flow constraints, but no other types of constraints explicitly supported.
Multiple Duration(s)	**Cannot**	No construct/feature specified.
Date(s) and Time(s)	**Cannot**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Completely**	An output arrow from a function block can point back to previously completed functions, including itself.
Manual vs. Automated Tasks	**Cannot**	FFBDs do not capture how a task is to be implemented.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	(see complex sequences)
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Not sure**	FFBDs could be made to satisfy this requirement, but I have not yet seen it done.

Requirements	FFBD	Descriptions
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	(see constraints).
State Existence Constraints	**Not sure**	Constraints that force various operational modes for a process could possibly be captured by denoting conditions at the exits of logic gates.
State Representations	**Not sure**	(see state existence constraints)
Temporal Constraints	**Partially**	FFBDs can capture the relative timing of activities with respect to one another (precedence), but do not allow for enforcing that activities occur at a specific, absolute time (e.g., 3:05pm on Thursday).
Uncertainty / Variability / Tolerance	**Partially**	Logical "or" gates portray uncertainty or variability in process flow, but no other mechanisms for representing tolerance, etc. is observed.
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Completely**	Conditional alternative paths can be specified, including iteration. Some of these could be defined so that they activate on a contingency basis if a failure or anomaly occurs. Such situations would need to be designed into the process from the start - not run-time.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Partially**	FFBDs capture logical operations of AND, OR, IOR. The diagrams, in their current state of evolution, are not executable, and thus do not perform actual calculations or operations. It is not a run-time representation. It merely captures what is known about required tasks and the logical flows that must occur between them.
Support for Task/Process Templates	**Not sure**	The prescribed FFBD methodology does not include a mechanism for creating or using templates, but couldn't just about anything be made into some kind of template?

Requirements	FFBD	Descriptions
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.
Business Practices, Rules, Constraints	**Cannot**	No construct/feature specified.
Configuration Management Information and Processes	**Cannot**	No construct/feature specified.
Customer-driven Process Specification and Constraints	**Cannot**	No construct/feature specified.
Forecast and Customer Orders	**Cannot**	No construct/feature specified.
Priority Attributes	**Cannot**	No construct/feature specified.
Representation of the Origin of Task(s)	**Cannot**	No construct/feature specified.
Analysis Characteristics	**Cannot**	No construct/feature specified.
Critical Task	**Cannot**	No construct/feature specified.
Predictive and Time-dependent Resource Availability	**Cannot**	No construct/feature specified.
Prescriptive Task Behavior	**Cannot**	No construct/feature specified.
Task/Process Performance Measurement	**Cannot**	No construct/feature specified.
Co-existence of Plans and Resolution of Conflicts	**Cannot**	No construct/feature specified.
Dynamic Model Modification	**Cannot**	In current form/implementation, FFBDs are not run-time. They help in designing a process, but is not capable of simulating the process' execution.
Optimization	**Cannot**	No construct/feature specified.
Resource/System/Process Monitoring and Feedback	**Cannot**	No construct/feature specified.
Support for Validation of the Entire Process Plan	**Cannot**	No construct/feature specified.
Tracking of Changes in the System	**Cannot**	No construct/feature specified.

Requirements	FFBD	Descriptions
What-if Analysis	**Cannot**	No construct/feature specified.
Resource Amount and Availability	**Cannot**	No construct/feature specified.
Resource Interruptions	**Cannot**	No construct/feature specified.
Process Yield	**Cannot**	No construct/feature specified.
Dynamic Model Modification	**Cannot**	No construct/feature specified.
Event Signaling and Notification	**Cannot**	No construct/feature specified.
Resource Behavior During Processing Time	**Cannot**	No construct/feature specified.
Resource/System/Process Monitoring and Feedback	**Cannot**	No construct/feature specified.
Tracking of Changes in the System	**Cannot**	No construct/feature specified.
Track In-progress Goods	**Cannot**	No construct/feature specified.
Decision Rationale	**Not sure**	Different output branches from a logic gate can be annotated to include reasons/scenarios governing why a particular path might be taken; however, the representation does not track real-time process execution - it cannot track the *results* of decisions, nor the reasoning to support having made one.
Intentional Dimension of Processes, or Goals	**Completely**	Each function block is defined as a particular function that needs to be carried out within the process. Each one is then an expression of a functional requirement for the process. More detailed parametric requirements can be attached via text annotation.
Relationship between Task and Goal and Resource and Goal	**Not sure**	Eventually, all of the tasks/functions in an FFBD are assigned, or allocated to a variety of resources. Thus, the possibility exists to satisfy this requirement, but FFBDs in their current form do not involve resource representations.
Task/Process Purpose	**Partially**	Hierarchical decomposition of functions within the diagrams allows the visualization of how a detailed task fits in with the overall (highest-level) process objectives. Complete details of the exact interfaces between the functions are not explicitly captured in an FFBD - they are traditionally captured in an accompanying N-squared diagram, or the like.
Value-added Attributes	**Cannot**	No construct/feature specified.
Access to Past and Present Decision Rationales	**Cannot**	No construct/feature specified.
Characteristics of Groups of Resources	**Cannot**	No construct/feature specified.
Implicit Task Association	**Cannot**	No construct/feature specified.
Parallelism	**Cannot**	No construct/feature specified.
Descriptive Manufacturing/Performance Variability	**Cannot**	No construct/feature specified.

Requirements	FFBD	Descriptions
Probability of Down Times	**Cannot**	No construct/feature specified.
Stochastic Properties	**Cannot**	No construct/feature specified.
Uncertainty of Sequences	**Cannot**	No construct/feature specified.
Account for Randomness	**Cannot**	No construct/feature specified.
Stochastic Functionality	**Cannot**	No construct/feature specified.
Prod Scheduling - production type data	**Cannot**	No construct/feature specified.
Prod Scheduling - dynamic rescheduling	**Cannot**	No construct/feature specified.
Process Planning – clamping surfaces	**Cannot**	No construct/feature specified.
Process Planning – datums and offsets	**Cannot**	No construct/feature specified.
Process Planning – features to be machined	**Cannot**	No construct/feature specified.
Process Planning – production type date	**Cannot**	No construct/feature specified.
Simulation - queue entry and exit rates	**Cannot**	No construct/feature specified.
Enterprise Eng. and Bus Process Re-eng - conceptual entities	**Cannot**	No construct/feature specified.
Workflow - manual vs. automatable tasks	**Cannot**	No construct/feature specified.
Workflow - invoked tool capability	**Cannot**	No construct/feature specified.
Workflow - support specifications of task structure (control flow)	**Completely**	Arrows and logic gates specify control flow. Although probabilistic branching is not directly supported, it could be easily added.
Project Management - work breakdown ids	**Not sure**	No construct/feature specified.

Gantt Charts

Requirements	Gantt Charts	Descriptions
ad hoc Notes	**Cannot**	No construct/feature specified.
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Cannot**	No construct/feature specified.
Resource	**Cannot**	No construct/feature specified.

Requirements	Gantt Charts	Descriptions
Resource Requirements for a Task	**Cannot**	No construct/feature specified.
Simple Groupings	**Partially**	only diagrammatic representation of sequence
Simple Resource Capability/Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Partially**	diagrammatic only
Simple Task Representation and Characteristics	**Cannot**	No construct/feature specified.
Task Duration	**Partially**	No construct/feature specified.
Task Executor	**Cannot**	No construct/feature specified.
Extensibility	**Cannot**	No construct/feature specified.
Resource Allocation / deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Partially**	diagrammatic only
Composition/Decomposition	**Cannot**	No construct/feature specified.
Incompleteness/Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Cannot**	No construct/feature specified.
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Cannot**	No construct/feature specified.
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Cannot**	No construct/feature specified.
Conditional Tasks	**Cannot**	No construct/feature specified.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Cannot**	No construct/feature specified.
Multiple Duration(s)	**Cannot**	No construct/feature specified.
Date(s) and Time(s)	**Partially**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Cannot**	No construct/feature specified.
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Cannot**	No construct/feature specified.
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Partially**	No construct/feature specified.
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Cannot**	No construct/feature specified.
Uncertainty/Variability/Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight(milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.

Requirements	Gantt Charts	Descriptions
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

Generalized Activity Network (GAN)

Requirements	GAN	Descriptions
ad hoc Notes	**Cannot**	No construct/feature specified.
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Cannot**	No construct/feature specified.
Resource	**Cannot**	No construct/feature specified.
Resource Requirements for a Task	**Cannot**	No construct/feature specified.
Simple Groupings	**Cannot**	No construct/feature specified.
Simple Resource Capability/Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Completely**	Essentially a superset of PERT sequence capabilities. see other aspects of sequencing capabilities in outer core write-up
Simple Task Representation and Characteristics	**Cannot**	No construct/feature specified.
Task Duration	**Completely**	No construct/feature specified.
Task Executor	**Cannot**	No construct/feature specified.
Extensibility	**Cannot**	Not in original definition of GAN. Capability may have been included in later implementations.
Resource Allocation/deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Completely**	No construct/feature specified.
Composition/Decomposition	**Cannot**	No construct/feature specified.
Incompleteness/Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Cannot**	No construct/feature specified.
Associated Illustrations and Drawings	**Partially**	GANs use explicit activity-on-arc diagrams, but lack any "how to perform" diagrammatic help.
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Completely**	6 possible node conditions enable complex sequencing logic
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Completely**	"and" node conditions in activity-on-arc representation
Conditional Tasks	**Completely**	conditional branching constructs at nodes
Confidence Levels	**Completely**	probabilistic activity realizations and durations
Constraints	**Completely**	node conditions enable temporal, pre and post, and state existence constraints

Requirements	GAN	Descriptions
Multiple Duration(s)	**Partially**	probabilistic activity durations
Date(s) and Time(s)	**Cannot**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Completely**	conditional branching in cyclic graph segment
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	"and" node conditions, etc.
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	node conditions in activity-on-arc representation
State Existence Constraints	**Partially**	node conditions in activity-on-arc representation
State Representations	**Partially**	node conditions in activity-on-arc representation
Temporal Constraints	**Partially**	node conditions in activity-on-arc representation
Uncertainty / Variability / Tolerance	**Partially**	time tolerances only using random activity durations
Ability to Insert or Attach a Highlight(milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Partially**	node conditions in activity-on-arc representation
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Partially**	node conditions in activity-on-arc representation
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

Hierarchical Task Network (HTN)

Requirements	HTN	Descriptions
ad hoc Notes	**Cannot**	No construct/feature specified.
Cost Data	**Partially**	Cost can be thought of as a constraint; thus, it can be specified in the constraint formula.
Level of Effort	**Partially**	Can be specified as constraints.
Product Characteristics	**Partially**	Can be specified in the constraints as well as the conditions parts in the operators.
Resource	**Partially**	Methods can be used to represent appropriate resources.
Resource Requirements for a Task	**Partially**	A method can be used to associate a task to various resources.
Simple Groupings	**Completely**	A task network is, by itself, a group of tasks, sub-tasks to achieve a certain goal.
Simple Resource Capability / Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Completely**	A task network is specified by a set of tasks followed by a constraint formula. To represent linear, time-sequential sequences, one can simply specify the order in which the tasks are to be executed in the constrain formula. e.g. (n1 < n2) and (n2 < n3) would specify the sequence n1 - n2 - n3 in which n1, n2, and n3 are all tasks.
Simple Task Representation and Characteristics	**Completely**	A task network can represent the sub-tasks that make up the current, the constraints that apply to the sub-tasks, conditions that need to be true before and after the task, etc.
Task Duration	**Partially**	Can be specified in constraint.
Task Executor	**Partially**	Can be specified within the tasks and compound tasks.
Extensibility	**Completely**	One can certainly add more tasks into a task network.
Resource Allocation/deallocation for one or many tasks	**Partially**	The use of a method for a certain task can be thought of as allocating the resource included in the method to the task. When the task is completed, the resource can be thought of as de-allocated.
Simple Precedence	**Completely**	Within a constraint formula, one can specify exactly which group of tasks needs to precede some other groups of tasks.
Composition / Decomposition	**Completely**	A compound task network is essentially a high level description of a task. With decomposition, one can then find out more details of the task.
Incompleteness / Vagueness	**Partially**	Method. When no further detail regarding a task is available, we can have a method that decomposes into "unknown" sub-tasks. These “unknowns” can later be decomposed into sub-tasks that make sense when the appropriate information is available.
Alternative Task	**Completely**	One can certainly have different task networks that achieve the same goal. These different task networks are the alternative tasks for achieving the goal function.
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.
Complex Groups of Tasks	**Partially**	Methods could be used to group tasks as well as resources allocated to them. A task network can also group tasks related to achieving a certain task. The restriction, however, is that, for example, two tasks sharing the same resource but have nothing else in common may not be grouped under HTN.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.

Requirements	HTN	Descriptions
Complex Sequences	**Partially**	HTN cannot explicitly represent concurrent tasks. There is no explicit construct for synchronizing the begin time of multiple tasks.
Complex Task Representation and Parameters	**Partially**	Parameters are represented by having variables within a task network. The constraint formula allows one to specify the capabilities, behavior, restrictions, etc. associated with a task. The made up of a task is explicitly represented by decomposition.
Concurrent Tasks	**Cannot**	No construct/feature specified.
Conditional Tasks	**Completely**	One can certainly specify the conditions within the constraint formula of a task network. The conditions can be the state of the world, the execution of other tasks, etc.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Partially**	Constraints can be specified in the constraint formula of a task network.
Multiple Duration(s)	**Partially**	Durations may be specified in each of the methods, and multiple methods may be used to accomplish a certain task; thus, multiple durations may be represented.
Date(s) and Time(s)	**Partially**	One can associate dates and times with methods. When there are multiple dates and times associated to a task/resource, we can simply have multiple methods, each of which with the respective dates and times.
Implicit/Explicit Resource Association	**Partially**	Methods can associate resources with tasks, and within the constraint formula, one can specify what other resources are needed when the current resource is used to accomplish a certain task, and so on.
Iterative Loops	**Cannot**	No construct/feature specified.
Manual vs. Automated Tasks	**Partially**	One can certainly say that a certain task is to be accomplished by a human, or machine A, etc.
Manufacturing Product Quantity	**Partially**	This could be represented as a task, eg. produce n products. The number, n, could be a variable in the task.
Material Constraints	**Partially**	See the annotation in Constraints, above.
Parallel Tasks	**Cannot**	No construct/feature specified.
Parameters and Variables	**Completely**	Task networks can contain variables and parameters since each of the tasks could contain variables. One is certainly allowed to change the bindings of these variables at any point of time.
Pre- and Post-processing Constraints	**Completely**	Within an operator, one can specify the pre- and post-conditions of executing the task. The pre- and post-processing constraints can go here.
Queues, Stacks, Lists	**Partially**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	A task network can contain a series of tasks to be performed in a particular order specified in the constraint.
State Existence Constraints	**Partially**	The pre-condition part in an operator could contain such state information.
State Representations	**Partially**	The current state of the world is reflected in the constraint formula of a task network. If a process is to be represented as a combination of states, one could use multiple different task networks, each of which may have different state information in its constraint formula.

Requirements	HTN	Descriptions
Temporal Constraints	**Partially**	Can be specified in constrain formula.
Uncertainty / Variability / Tolerance	**Partially**	One may specify this in the constraint formula.
Ability to Insert or Attach a Highlight (milestones)	**Partially**	Since HTN represents a process as a task network, one can certainly highlight each of the individual subtasks in the network. This can be done by specifying, in those subtasks that are to be highlighted, their importance to the process, or by having an extra variable to accommodate a flag of some sort.
Complex Precedence	**Partially**	Such constraints may be specified within the constraint formula of a compound task network.
Convey the Ancestry or Class of a Task	**Partially**	Even though decomposition provides representation of hierarchy of tasks, generalization, and specialization, there is no guarantee of inheritance of characteristics of tasks through decomposition to each of the subtasks.
Deadline Management	**Partially**	A task network provides ways to specify deadlines of each of the subtasks; however, the management needs to be performed by external programs which utilize the HTN representation.
Dispatching	**Partially**	The items could be represented as methods while the rules and guidelines for releasing these items are represented in the constraint formula in these methods. The process of dispatching, however, will need to be performed by some program.
Eligible Resources	**Partially**	Resources are represented as methods. Their eligibility for being selected are specified in the constraints.
Exception Handling and Recovery	**Partially**	A task network can certainly be decomposed into several different task networks, each of which is capable of achieving the goal task. These networks may serve as fallbacks for the planner; however, HTN does not have explicit constructs that specify which ones to use if some other ones fail.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Partially**	One may use mathematical and logical operators while writing the conditions as well as the constraints in HTN. However, there really isn't any construct that performs the operations.
Support for Task/Process Templates	**Partially**	It is certainly reasonable to think of a task network, whose variables are not yet bound, as a template. One can reuse a task network in multiple problems with different variable bindings.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Partially**	This can be represented using methods with decomposition of tasks.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

IDEF0

Requirements	IDEF0	Descriptions
ad hoc Notes	**Completely**	associated text and glossary
Cost Data	**Partially**	Via control arrow or as data flow on input and output arrows
Level of Effort	**Cannot**	Amount of resource needed not covered

Requirements	IDEF0	Descriptions
Product Characteristics	**Partially**	Via input/output arrows, can show products created, modified or used during a function.
Resource	**Completely**	Mechanism arrow is a person or device that carries out the function.
Resource Requirements for a Task	**Completely**	Via control and mechanism arrows
Simple Groupings	**Completely**	Via decomposition
Simple Resource Capability / Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Cannot**	Sequence is often implied, but a function's "position" is determined by the input constraints.
Simple Task Representation and Characteristics	**Completely**	Boxes represent activities, actions, processes or operations, and arrows represent Input, Control, Mechanism and Output constraints on these activities.
Task Duration	**Cannot**	No construct/feature specified.
Task Executor	**Completely**	Mechanism arrow represents person or device that carries out a function.
Extensibility	**Partially**	Can extend anything in terms of additional ICOMs (Input, Control, Output, Mechanism arrows)
Resource Allocation / deallocation for one or many tasks	**Cannot**	Can only show requirements for a functions
Simple Precedence	**Cannot**	Shows "precedence" only in terms of constraints
Composition / Decomposition	**Partially**	Composition/Decomposition handled with Subfunctions (submodules) of single parent modules
Incompleteness / Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Completely**	Via output and control arrows
Associated Illustrations and Drawings	**Not sure**	IDEF0 representation includes "text" and "glossary".
Complex Groups of Tasks	**Cannot**	Can only group tasks via decomposition
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Cannot**	IDEF0 represents activities and relationships independent of sequence and timing
Complex Task Representation and Parameters	**Completely**	Via function boxes and ICOMs
Concurrent Tasks	**Partially**	Arrows may branch and join. Cannot associate timing.
Conditional Tasks	**Completely**	Via input and control arrows
Confidence Levels	**Cannot**	
Constraints	**Partially**	Temporal constraints not included
Multiple Duration(s)	**Cannot**	Temporal aspects of process not represented
Date(s) and Time(s)	**Cannot**	
Implicit/Explicit Resource Association	**Not sure**	This could probably be done implicitly (and partially satisfy requirement) through mechanisms and controls, and decompositions.
Iterative Loops	**Partially**	This can be done via output and input arrows. Temporal aspects of iterations are not represented

Requirements	IDEF0	Descriptions
Manual vs. Automated Tasks	**Partially**	All functions of a process can be represented.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	Constraints on functions (tasks) are represented.
Parallel Tasks	**Cannot**	Temporal aspects of sequences not represented.
Parameters and Variables	**Partially**	The output of a function could be a value which could be an input requirements of another function.
Pre- and Post-processing Constraints	**Completely**	Input, control, and mechanism arrows are for representing pre- and post-processing constraints.
Queues, Stacks, Lists	**Cannot**	Concepts can be used to represent any type of object or type of object, although not in any kind of detail.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Cannot**	Cannot represent temporal sequences. Tasks may appear to be serial in that the output of one is required as input to another, but serial task representation is not explicit.
State Existence Constraints	**Completely**	The Control arrow could represent state existence constraints.
State Representations	**Partially**	State changes for functions, but not resources, can be represented via input and output arrows.
Temporal Constraints	**Cannot**	No construct/feature specified.
Uncertainty/Variability/Tolerance	**Partially**	Tolerances for an activity could be represented by control arrows.
Ability to Insert or Attach a Highlight(milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	Temporal sequences not represented.
Convey the Ancestry or Class of a Task	**Completely**	With decomposition, ICOMs can be maintained from parent to child.
Deadline Management	**Cannot**	Temporal aspects of process not addressed. The deadline management function could be modeled.
Dispatching	**Cannot**	Dispatching functions and rules could be represented, but real-time, temporal aspects cannot be represented.
Eligible Resources	**Partially**	Via output and mechanism arrows. A function whose output is "eligible resources" could provide mechanism for another functions.
Exception Handling and Recovery	**Partially**	Output of functions can indicate the exception that is input and controls of "exception handling" functions.
Information Exchange Between Tasks	**Completely**	Via output and input arrows.
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Partially**	While this cannot be done explicitly, elements can be reused.

Requirements	IDEF0	Descriptions
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	Decomposition and tunneling allows ICOMs to be associated, or not association with various levels of functions.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

IDEF3

Requirements	IDEF3	Descriptions
ad hoc Notes	**Partially**	Facts and constraints on model elements or description of model elements. IDEF3 supports the concept of fact and constraint. Notes can be captured as facts or constraints or in the description of the model element they apply to. Process flow diagrams, and scenarios also have facts, constraints, and a description. Models have a model summary, a purpose and a context.
Cost Data	**Partially**	Notes Facts Constraints. IDEF3 does not explicitly support the notion of cost but allows users to specify notes, facts, or constraints on any model element in a model.
Level of Effort	**Partially**	Notes, Constraints, Facts. IDEF3 enables the representation of facts, constraints, and notes that can be used to specify the level of effort needed to accomplish a task.
Product Characteristics	**Partially**	Constraints and facts on objects and description of objects. In IDEF3, a product can be represented as a special type of object. Characteristics of the product can be specify using constraints and/or facts on the object or the description field of the object.
Resource	**Partially**	Objects and Object types. IDEF3 supports the concept of objects. Hence, resources can be represented as objects of the type 'resource'.
Resource Requirements for a Task	**Completely**	Association of objects with tasks (UOBs in IDEF3). In IDEF3, objects can be associated with a UOB to indicate their participation to that UOB. Objects associated with a UOB can be given a role such as: agent, created, destroyed, affected, etc.
Simple Groupings	**Completely**	Scenarios and Process Flow Diagrams. A process flow diagram enables users to describe a sequence of tasks. The tasks in the sequence are related through temporal relationships. A scenario is a set of process flow diagrams (PFDs) that describe a process or plan. Typically, a PFD represents a high level description of the plan. Each task in that PFD can have one or more PFDs associated with it that detail the task further. The hierarchy of PFD constitutes the scenario.
Simple Resource Capability / Characteristics	**Partially**	Facts and Constraints on objects. The characteristics and capability of a resource can be expressed using facts and constraints associated with the object representing the resource
Simple Sequences	**Completely**	Process Flow Diagrams.

Requirements	IDEF3	Descriptions
Simple Task Representation and Characteristics	**Completely**	UOBs (Units Of Behavior) are used in IDEF3 to represent events, tasks, activities, situations, etc. Note that a UOB describes a type of task, not a specific task that occurred at a particular point in space and time. UOBs have a description, facts, constraints, and objects that are used to describe them. They can be further described by associating process flow diagrams with them.
Task Duration	**Partially**	Facts and constraints on a UOB. The time it takes to complete a task can be captured in a fact or constraint on the UOB.
Task Executor	**Completely**	Associate an object with role agent or executor on a UOB. Objects can be associated with UOBs and can have a role defined on them.
Extensibility	**Cannot**	The only way for users to add information to a model is by using the predefined facts and constraints constructs.
Resource Allocation / deallocation for one or many tasks	**Partially**	Association of objects with UOBs with appropriate roles and facts and constraints. Facts and constraints can be used to record how resources are allocated to tasks.
Simple Precedence	**Completely**	Process Flow Diagrams.
Composition / Decomposition	**Completely**	Decomposition on UOBs. UOBs can be further described by associating process flow diagrams to them. Note that this requirement is very ambiguous, as it seems to confuse abstraction with the token/type distinction. IDEF3 does not support the representation of instance level tasks.
Incompleteness / Vagueness	**Completely**	IDEF3 supports the representation of both process descriptions and process models. Process descriptions, by definition, can be incomplete.
Alternative Task	**Partially**	Facts and constraints.
Associated Illustrations and Drawings	**Completely**	Source, facts, constraints, and descriptions. IDEF3 supports the concept of source. A source enables users to describe any material that was used to individuate a model element.
Complex Groups of Tasks	**Completely**	Process Flow Diagrams and Scenarios.
Complex Resource Characteristics	**Partially**	Facts and constraints on objects.
Complex Sequences	**Completely**	Junctions enables to specify that some tasks must be performed in parallel. Junctions have a logic associated with them to enable users to specify whether tasks must be performed concurrently, tasks are mutually exclusive, etc.
Complex Task Representation and Parameters	**Completely**	Facts, constraints, referents ("call and wait" and "call and continue"). Referents can be used in IDEF3 to indicate that a task must be interrupted and that a task will trigger the beginning of another task.
Concurrent Tasks	**Completely**	An AND junction indicates that the following tasks are performed in parallel. The junction can be specified as synchronous to indicate that the tasks must all start at the same time.
Conditional Tasks	**Completely**	OR and XOR junctions.
Confidence Levels	**Partially**	Confidence levels can be expressed using facts and constraints.
Constraints	**Completely**	IDEF3 constraints allow for the capture of any type of constraints. However, some special types of constraints (e.g., temporal) can be captured in a more structured way using appropriate constructs provided by the method.
Multiple Duration(s)	**Partially**	Facts can be used to express estimated, actual, and average duration. Note that IDEF3 does not support the representation of task instances (i.e., actual events).
Date(s) and Time(s)	**Cannot**	
Implicit/Explicit Resource Association	**Completely**	Facts and constraints on objects. Resources in IDEF3 are captured using the 'Object' construct. Facts and constraints can be used to capture these kinds of dependencies.

Requirements	IDEF3	Descriptions
Iterative Loops	**Completely**	Junctions and 'go to' referents can be used to capture loops. Conditions for exiting the loop can be captured using facts and constraints on junctions.
Manual vs. Automated Tasks	**Partially**	Facts, constraints, or description.
Manufacturing Product Quantity	**Partially**	Facts and constraints.
Material Constraints	**Partially**	Facts and constraints.
Parallel Tasks	**Completely**	'AND' junctions enables modelers to specify that some tasks are performed in parallel. The junction can be synchronous or asynchronous to indicate whether the tasks start at the same time.
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Completely**	Constraints
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Partially**	Facts and constraints on objects.
Resource/Task Combined Characteristics	**Partially**	Facts and constraints.
Serial Tasks	**Completely**	Precedence links in process flow diagrams.
State Existence Constraints	**Completely**	State transition diagrams and state transition conditions.
State Representations	**Completely**	State transition diagrams.
Temporal Constraints	**Completely**	Process flow diagrams.
Uncertainty /Variability/Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Completely**	Using a combination of junctions enables the representation of complex temporal constraints between tasks.
Convey the Ancestry or Class of a Task	**Cannot**	This requirement is satisfied in the integrated IDEF3/5 method.
Deadline Management	**Partially**	Could be represented as a task itself that determines what path is taken is a process flow diagram.
Dispatching	**Partially**	Can be captured using facts and constraints.
Eligible Resources	**Completely**	Association of resource objects with UOBs.
Exception Handling and Recovery	**Partially**	Facts and constraints and referents.
Information Exchange Between Tasks	**Completely**	Object flow links represents the flow of an object from one task to another.
Mathematical and Logical Operations	**Partially**	Facts, constraints, notes.
Support for Task/Process Templates	**Completely**	Pool items that can be used in process flow diagrams.

Requirements	IDEF3	Descriptions
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	UOB decomposition hierarchy. Mult. Level of Abst. is supported mainly for decomposing tasks into subtasks.
Synchronization of Multiple, Parallel Task Sequences	**Partially**	Facts and constraints and junctions.

<I-N-OVA>

Requirements	<I-N-OVA>	Descriptions
ad hoc Notes	**Completely**	A - Misc-Annotation constraint.
Cost Data	**Completely**	A - Misc constraint in global <I-N-OVA model if not specific to a given process or plan, or in plan's <I-N-OVA representation if it is specific to that.
Level of Effort	**Completely**	A - Resource (or A-Resource-Agent) constraint.
Product Characteristics	**Completely**	V - entity/variable constraint.
Resource	**Completely**	A - object used in resource constraint.
Resource Requirements for a Task	**Completely**	A - Resource constraint.
Simple Groupings	**Completely**	N - include activity constraint.
Simple Resource Capability / Characteristics	**Completely**	V - global <I-N-OVA entity/variable constraint for object to be used as a resource.
Simple Sequences	**Completely**	O - Ordering constraint on time point associated with begin or end of any activity.
Simple Task Representation and Characteristics	**Completely**	N - Name of activity.
Task Duration	**Completely**	O - Metric temporal constraint between time points associated with begin and end of an activity.
Task Executor	**Completely**	A - Resource-Agent constraint. This allows for a specific "performer" of an activity.
Extensibility	**Completely**	A - Open framework for adding any information in the form of a constraint or annotation.
Resource Allocation / deallocation for one or many tasks	**Completely**	A - resource constraints are expressive enough to support this.
Simple Precedence	**Completely**	O - Ordering constraints.

Requirements	<I-N-OVA>	Descriptions
Composition / Decomposition	**Completely**	A,N - Constraints of various types (in particular A-World State constraints) may be modeled at any abstraction level. Activity decompositions (Include activity constraints in process or activity description library) (N). Missing constraints just imply a wider allowed space of behavior. The <I-N-OVA model is specifically designed to allow for incompleteness and uncertainty in process and activity descriptions. The <I-N-OVA model is specifically designed to allow for incompleteness and uncertainty in process and activity descriptions. Specific constraints would need to have uncertainty in their formulation and expression <I-N-OVA makes no commitment to this.
Incompleteness / Vagueness	**Completely**	Missing constraints just imply a wider allowed space of behavior. The <I-N-OVA model is specifically designed to allow for incompleteness and uncertainty in process and activity descriptions.
Alternative Task	**Completely**	Disjunctive constraints may be included in the <I-N-OVA model in any place - and this is not limited to disjunctions within any one specific constraint type or sub-type. An other node can also represent conditional activities.
Associated Illustrations and Drawings	**Completely**	A - Associated information and annotations may be stated as "annotation constraints" or more generally "Miscellaneous constraints".
Complex Groups of Tasks	**Completely**	N - other nodes that contain sub-plans can be used to group a task for a common purpose (i.e. the detailed expression of an activity).
Complex Resource Characteristics	**Completely**	A - Resource constraints or Agent constraints can describe these characteristics.
Complex Sequences	**Completely**	O - ordering constraints can describe a variety of necessary relationships.
Complex Task Representation and Parameters	**Completely**	N - Nodes that include activities can take into account concepts such as applicability, performance limits, resource usage, number of constraints on its conditions, suitable parameter bindings, etc.
Concurrent Tasks	**Completely**	O - activities can be constrained to have "concurrent" execution.
Conditional Tasks	**Completely**	N - other nodes may also represent a conditional "if then else" within the plan.
Confidence Levels	**Cannot**	
Constraints	**Completely**	<I-N-OVA views a plan as a set of constraints.
Multiple Duration(s)	**Partially**	Thinking more about your examples, there is probably only partial support. predicted duration / worst case duration Yes. Time windows are defined with a min/max and projected value. This would result in a best, worst, and most likely durations for a specific implementation of a task. If you do have a case where you know that a task might take "around an hour" or "around 2 hours" (e.g. if you use machine A or machine B to accomplish the task), then you'd go with my first example of an "or-split". average duration I was originally thinking of "average" in the context of a predicted value, but obviously they mean two different things. I'd say no to this one. actual duration Again, I was thinking about actual duration in terms of, "Task A will actually take 1hr (at most/at least/probably)" as opposed to Task A will last from timepoint.1 to timepoint.2. TF is not used in recording the execution time of a task, so no to this one as well.
Date(s) and Time(s)	**Completely**	O - metric temporal constraints can relate a given time point to an actual time or calendar reference.
Implicit/Explicit Resource Association	**Completely**	A,N - Resource constraints can explicitly be attached to an activity. A node that contains sub-plans implicitly constrains resource usage though its sub-constraints.
Iterative Loops	**Completely**	N - other nodes can represent an encapsulation of iteration or for-each.

Requirements	<I-N-OVA>	Descriptions
Manual vs. Automated Tasks	**Completely**	A - Misc. constraints can be created to characterize specialized attribute requirements.
Manufacturing Product Quantity	**Completely**	A - Resource constraints can be used to control the maximum allowable amount of the resource.
Material Constraints	**Completely**	A - resource constraints can be used to describe specialized characteristics. "always" constraints can be used to declare unchanging global information.
Parallel Tasks	**Completely**	O - ordering constraints can describe activities that occur in parallel.
Parameters and Variables	**Completely**	V - entity/variable constraints can be used to manage "place holders" that can take on a range of values.
Pre- and Post-processing Constraints	**Completely**	O - input and output temporal constraints are used to describe what should hold immediately before or after a given timepoint.
Queues, Stacks, Lists	**Partially**	<I-N-OVA does not have an explicit representation for data structures such as queues or stacks.
Resource Categorization and Grouping	**Completely**	It is anticipated that a representation language that expresses the <I-N-OVA model will use a sorted first order logic.
Resource Location	**Completely**	A, V - A-Resource constraints can add information such as location, entity/variables can be used to update a location attribute.
Resource/Task Combined Characteristics	**Completely**	O,N - This requirement can be met by creating alternate "include activity" nodes that utilize the same resources, but may have different input temporal constraints.
Serial Tasks	**Completely**	O - ordering constraints are used to declare activities in serial.
State Existence Constraints	**Completely**	O - input temporal constraints specify those things that are required to hold before a given time point (which may be attached to an activity).
State Representations	**Completely**	A - World State constraints act on the plan state representation.
Temporal Constraints	**Completely**	O - Temporal modeling is performed by using time points and ordering constraints.
Uncertainty/Variability/Tolerance	**Completely**	The <I-N-OVA model is specifically designed to allow for incompleteness and uncertainty in process and activity descriptions. Specific constraints would need to have uncertainty in their formulation and expression <I-N-OVA makes no commitment to this.
Ability to Insert or Attach a Highlight (milestones)	**Partially**	A - Misc or Annotation constraints can be attached to nodes to give them "milestone significance".
Complex Precedence	**Completely**	O - Ordering constraints can be generally specified to establish node precedence.
Convey the Ancestry or Class of a Task	**Completely**	N - other node constraints can be used to encapsulate specialized sub-plans.
Deadline Management	**Completely**	O - Ordering constraints are used to arrange activities within specified temporal constraints.
Dispatching	**Completely**	O - Input temporal constraints can be placed on activities that release represent releasing items for production.
Eligible Resources	**Completely**	A - Resource constraints for an activity describe a sorted requirement for resource usage.
Exception Handling and Recovery	**Completely**	O - input and output temporal constraints can be used to specify what should hold before and after a time point (therefore an activity).
Information Exchange Between Tasks	**Completely**	V - Information is shared between nodes through entity/variable constraints.

Requirements	<I-N-OVA>	Descriptions
Mathematical and Logical Operations	**Completely**	The expressions in <I-N-OVA are considered to be based in first order logic that will allow for logical and mathematical manipulation.
Support for Task/Process Templates	**Completely**	N - other nodes and include activity nodes are linked in a "generic process template" that is applicable for use assuming the constraints are satisfied.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	A - Constraints can be attached at any level of a node hierarchy that would be appropriate for that model.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	O - Temporal constraints can be attached to activities that make the hard requirement that begin/end timepoints are equal.

JTF - Core Plan Representation (CPR)

Requirements	JTF-CPR	Descriptions
ad hoc Notes	**Completely**	Annotation object is contained in PlanObject superclass.
Cost Data	**Cannot**	
Level of Effort	**Completely**	Contained in the CPR specialization objects of ConsumableResource
Product Characteristics	**Partially**	Work products can been given as the underspecified object DomainObject.
Resource	**Completely**	Resource object or its specializations
Resource Requirements for a Task	**Completely**	Action objects (tasks) may contain Resource objects
Simple Groupings	**Completely**	Actions may contain sub-Actions
Simple Resource Capability / Characteristics	**Partially**	A suggested set of specializations to Resource is provided including Consumable, Reusable, SynchronouslyReusable, ExactCapacity and NonSharable.
Simple Sequences	**Completely**	Constraints may be assigned to Actions that enforce parallelism or serialism.
Simple Task Representation and Characteristics	**Cannot**	No construct/feature specified.
Task Duration	**Completely**	Actions have start and end times
Task Executor	**Completely**	Actions have associated Actors
Extensibility	**Cannot**	No construct/feature specified.
Resource Allocation/deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Cannot**	No construct/feature specified.
Composition / Decomposition	**Completely**	Actions, Plans, and Actors may all have sub-entities
Incompleteness / Vagueness	**Completely**	There is no implied enforcement of completeness. Uncertainty and Imprecision (fuzzy logic) constructs are included.
Alternative Task	**Partially**	Actions can be given arbitrary constraints but there is no specified construct to describe one as an alternative to another.
Associated Illustrations and Drawings	**Completely**	Arbitrary Annotations may be linked to any plan object

Requirements	JTF-CPR	Descriptions
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Partially**	A hierarchy of resource types is provided
Complex Sequences	**Partially**	Arbitrary types of constraints may be given to specify parallelism or serialism.
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Completely**	Actions may be constrained to run concurrently or may be unconstrained allowing concurrent execution if possible.
Conditional Tasks	**Completely**	Actions may have constraints on execution. Assumptions may also be included which trigger new Actions if the assumptions are violated.
Confidence Levels	**Completely**	All low level data may be tagged with Uncertainty or Imprecision measures. High level objects like Entity or Action may be encapsulated in an UncertainEntity object which has an associated uncertainty or imprecision
Constraints	**Partially**	Examples are given for temporal and pre- and post-condition constraints but the Constraint object is relatively underspecified.
Multiple Duration(s)	**Partially**	Action may be specialized to contain other durations but the base class only contains start and end.
Date(s) and Time(s)	**Completely**	CPR includes TemporalPoint a specialization of which is the OMG universal time object that has both time and date.
Implicit/Explicit Resource Association	**Partially**	A Resource may contain a Constraint that specified dependency on another Resource.
Iterative Loops	**Cannot**	No construct/feature specified.
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Partially**	DomainObjects with associated quantity may be specified as products of Actions.
Material Constraints	**Partially**	Constraints may state ranges about arbitrary attributes of an Entity.
Parallel Tasks	**Partially**	Arbitrary types of constraints may be given to specify parallelism or serialism.
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Partially**	Examples are given for temporal and pre- and post-condition constraints but the Constraint object is relatively underspecified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	Resources may have subResources but only hierarchical arrangements are currently allowed
Resource Location	**Completely**	Resources may be constrained to have a particular SpatialPoint
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Partially**	Arbitrary types of constraints may be given to specify parallelism or serialism.
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Completely**	Actions have associated TimePoints which constrain their execution
Uncertainty/Variability/ Tolerance	**Completely**	Uncertainty and Imprecision (fuzzy logic) constructs are included and may be specified for any object including TimePoints.

Requirements	JTF-CPR	Descriptions
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

Knowledge Interchange Format (KIF)

Requirements	KIF	Descriptions
ad hoc Notes	**Partially**	Text documentation may be represented using quote. Its association with some plan component may be represented by defining an object whose name says that it's a note of the component, and whose term is the quote (the actual documentation).
Cost Data	**Partially**	Costs associated to a resource or a task can be hardcoded into a function.
Level of Effort	**Partially**	KIF supports numbers, which can represent the amount of a resource needed.
Product Characteristics	**Partially**	A product may be represented by defining an object, which has the characteristics of the product as its definition.
Resource	**Partially**	resources may be represented as objects.
Resource Requirements for a Task	**Partially**	The resources required for a task can be made returned by a function in the form of a set/list/object.
Simple Groupings	**Completely**	Tasks may be grouped in terms of sets or lists. Or, one may simply define a new object for task groupings.
Simple Resource Capability/Characteristics	**Partially**	New objects can be defined to describe resource capabilities/characteristics. These objects may then be associated to the resources they describe by defining new relations.
Simple Sequences	**Partially**	Time linear, sequential sequences can be grouped in lists.

Requirements	KIF	Descriptions
Simple Task Representation and Characteristics	**Partially**	tasks can be defined as objects
Task Duration	**Partially**	Duration can be defined as a function of the task
Task Executor	**Partially**	Task executors can be defined as objects.
Extensibility	**Completely**	These constructs allow further information to be added to the existing data.
Resource Allocation/deallocation for one or many tasks	**Partially**	Resources allocated to a task may be put in a list. Thus, deallocation can be represented by removing the resource from the list.
Simple Precedence	**Partially**	We can define a binary relation for tasks, which returns true if one task is to precede another.
Composition/Decomposition	**Partially**	information at various levels can be defined as separate objects. Later on, a function can be defined to return the information given the respective level.
Incompleteness/Vagueness	**Partially**	it is certainly possible to make definitions which contain unspecified information.
Alternative Task	**Partially**	these alternative tasks can be defined as objects or functions. One may use a certain naming convention so that it is clear that these separate functions/objects are alternatives for the same job.
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.
Complex Groups of Tasks	**Partially**	task groupings can be represented as sets/lists. Lists probably is more appropriate.
Complex Resource Characteristics	**Partially**	By defining an object for each resource, we will be able to provide as much detail as we want for the resource.
Complex Sequences	**Partially**	All the sequencing types can be represented. See their respective cells for constructs.
Complex Task Representation and Parameters	**Partially**	Tasks or groups of tasks can be defined as objects using logical sentences and quantity sentences.
Concurrent Tasks	**Partially**	a relation can be defined to return true when given concurrent tasks. Besides that, we can also specify,within the object definitions, that the starting point of the execution times much be the same.
Conditional Tasks	**Completely**	These constructs can specify the conditions under which a certain task, represented as an object are to be executed.
Confidence Levels	**Partially**	Confidence levels can be represented as numbers.
Constraints	**Partially**	All constraints can be represented
Multiple Duration(s)	**Partially**	"event begin" and "event end" can be defined as objects. These objects can then be included in the definition of tasks or resources to specify multiple durations.
Date(s) and Time(s)	**Partially**	Dates and times can be represented as lists of numbers. Better yet, we can define them as objects.
Implicit/Explicit Resource Association	**Partially**	the dependency can be represented as a relation defined over resources.
Iterative Loops	**Partially**	KIF allows recursive definitions. Thus, iterative loops can be converted into recursion.
Manual vs. Automated Tasks	**Partially**	can be defined as a relation over tasks.
Manufacturing Product Quantity	**Partially**	The quantity can be represented as numbers.
Material Constraints	**Partially**	This can be represented by defining a function over materials, which, in turn, are defined as objects.

Requirements	KIF	Descriptions
Parallel Tasks	**Partially**	A relation can be defined over tasks. This relation may return a true if the given tasks are parallel, and false otherwise.
Parameters and Variables	**Completely**	Variables in KIF are words preceded by a ? or a @.
Pre- and Post-processing Constraints	**Partially**	Within a definition, one can specify conditions with cond or if. The pre-post-cond constraints can be represented by defining them as new relations.
Queues, Stacks, Lists	**Partially**	Lists are readily defined in KIF. Queues and stacks can be defined in terms of lists by defining the necessary functions, relations, and objects.
Resource Categorization and Grouping	**Partially**	A new object can be defined to represent the categorization and grouping. The resources may be grouped by means of a list or a set. Their common characteristics may be specified within the definition under cond or if.
Resource Location	**Partially**	Locations can be defined as objects. Then, relations can be defined to relate resources to these locations.
Resource/Task Combined Characteristics	**Partially**	Resources and tasks may be combined by defining them as a new object. Then, some functions can be defined over this new object to return the characteristics of this task/resource combination in some form of logical sentences.
Serial Tasks	**Partially**	Serial tasks can be put in a list. To be clearer, this list can be defined as an object.
State Existence Constraints	**Partially**	The requirements can be put within the cond statement that allows the execution of the task only if the requirements are satisfied. The requirements can be some sort of relation that is defined over some objects that define states.
State Representations	**Partially**	States can be represented as objects. The combination of the states can be put in a list. Thus, the list would describe a process in terms of some states.
Temporal Constraints	**Partially**	Start point and end point can be defined as objects. These objects can, in turn, be used within the definition of the task or resource objects.
Uncertainty / Variability / Tolerance	**Partially**	Uncertainty etc. can be represented as numbers. A function can also be defined over tasks to return such information.
Ability to Insert or Attach a Highlight(milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Partially**	Precedence can be represented by defining a relation over tasks. The conditions are specified within the definition.
Convey the Ancestry or Class of a Task	**Partially**	Specialization relationships, ancestry relationships, etc. can be defined using defrelation over tasks. Tasks at different levels can be defined as different objects, with the lower level ones defined using higher-level objects and some more information. Inheritance of characteristics can be represented as rules such as "if characteristic A is in a higher-level task, then there exists A in all lower-level tasks."
Deadline Management	**Partially**	Decision-making can be represented as functions. Deadlines can be considered during decision making by means of cond statements, or if statements.
Dispatching	**Partially**	KIF is capable of logic programming; thus, rules and guidelines for dispatching can certainly be represented.
Eligible Resources	**Partially**	Again, rules can be represented. Furthermore, relations may be defined to relate resources to these rules that determine the resources' eligibility.
Exception Handling and Recovery	**Partially**	One can define functions that return corrective actions in the form of a list or a quote. Exceptions or error can be specified as some kind of conditions within the definition of the function.

Requirements	KIF	Descriptions
Information Exchange Between Tasks	**Partially**	Functions can be defined for tasks, the parameters will represent information flowing into the task and the returned parameters will be information flowing out of the task. Furthermore, a list with embedded lists can be used to represent a chain of tasks through which information flows.
Mathematical and Logical Operations	**Completely**	KIF has all of these built in.
Support for Task/Process Templates	**Partially**	Functions can contain variables. Thus, it is, in itself, a template.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Partially**	A relation can be defined to associate a resource with a task. Multiple objects or relations can be defined to represent multiple levels of abstraction for such association.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	Functions that perform certain things upon receiving events can be defined. However, it is up to the application that makes use of KIF to synchronize executions of any sort of functions or relations.

O-Plan Task Formalism

Requirements	O-Plan TF	Descriptions
ad hoc Notes	**Completely**	Notes via comments and "tf_info" items. Individual plan items can contain "annotation-constraints". Extended documentation for schemas can be achieved by linking "info" attribute/value pairs with filenames of associated drawings, etc.
Cost Data	**Partially**	O-Plan TF can be used to describe an action that consumes a resource (e.g. money, in the case of cost). Uncertainty costs, variability, etc. is incorporated by the use of upper/lower bounds on numerical values.
Level of Effort	**Completely**	O-Plan TF has a rich set of resource elements that can describe the units, types, and number of resource items that are required by an action.
Product Characteristics	**Partially**	O-Plan TF can be used to model a class of resources that are "producible" when an action is applied. This "produced" item can be an intermediate product that is used to supply a condition for another action.
Resource	**Completely**	O-Plan TF can be used to describe resources and resource types.
Resource Requirements for a Task	**Completely**	O-Plan TF resource statements can quantify an action's usage of a resource.
Simple Groupings	**Completely**	Action schemas can define partially ordered sub-actions and action schemas can be arranged hierarchically through the use of "expands" action patterns.
Simple Resource Capability / Characteristics	**Completely**	O-Plan TF can give resource characteristics that can be used to select the appropriate resource for a task. (e.g. attributing "wolf-proof" characteristics to "bricks" in a sample domain.)
Simple Sequences	**Completely**	O-Plan TF has a number of ways to express temporal relationships. "At" links actions to a specific timepoint. "Duration" specifies a range. TF can also express "delay_between" as a means to specify a latency period between the end and begin of two actions.
Simple Task Representation and Characteristics	**Completely**	Simple high-level descriptions can be attached via the schema annotations that were described in the annotations.

Requirements	O-Plan TF	Descriptions
Task Duration	**Completely**	As per Tate (22-Nov): In O-Plan a user can express duration in metric time points against a reference basis of zero time. (e.g. day 45 12:00:00 for example for noon on day 45 of a project.)
Task Executor	**Completely**	O-Plan TF can select modeled resources to be associated with an instantiated action. (e.g. selecting vehicles in pacifica sample domain). TF can also be used to directly model the "contracting" relationship using [un]supervised conditions.
Extensibility	**Completely**	O-Plan "other-constraints" can be used to record additional information.
Resource Allocation / deallocation for one or many tasks	**Completely**	O-Plan TF can be used to model assignment and release of resources.
Simple Precedence	**Completely**	O-Plan TF conditions, effects, and expands can be used to form interschema relationships while orderings are used to define intraschema sub-action relationships.
Composition / Decomposition	**Completely**	Schemas arranged in a hierarchical fashion can abstract the details of various plan expansions. TF can be arranged into plan levels/phases that allows O-Plan to control how far to plan (incompleteness). More than 1 schema can be appropriate (ambiguity).
Incompleteness / Vagueness	**Completely**	Schemas arranged in a hierarchical fashion can abstract the details of various plan expansions. TF can be arranged into plan levels/phases that allows O-Plan to control how far to plan (incompleteness). More than 1 schema can be appropriate (ambiguity).
Alternative Task	**Completely**	More than one TF schema may be appropriate for a plan node expansion.
Associated Illustrations and Drawings	**Completely**	Textual items (comments) can be attached to O-Plan TF items and extended documentation for the domain can be achieved by linking tf_info attribute/value pairs with filenames of associated drawings, etc.
Complex Groups of Tasks	**Completely**	O-Plan TF can describe an explicit grouping of actions (e.g. install services). TF can also address constraints related to the overall group. (e.g. describing how much resource an action and its expansions can consume.)
Complex Resource Characteristics	**Completely**	Resources can have a "specific" type that affects how the planning system may use the resource. (movable_objects vs. objects, etc.)
Complex Sequences	**Completely**	O-Plan TF schemas can explicitly represent complex sequences as well as express the elements necessary to create more ordering relationships during generative planning.
Complex Task Representation and Parameters	**Completely**	Action schemas can take into account concepts such as applicability (only_use_if), performance limits (time windows, resource consumption), and a number of constraints on its conditions, suitable parameter bindings, etc.
Concurrent Tasks	**Completely**	20-Nov-96 via Tate: "Two actions can be constrained to have the same begin and end times by giving a zero duration link between their begin points and the same zero duration link between their end points."
Conditional Tasks	**Completely**	TF schema filters (only_use_if) control the applicability of a specific schema.
Confidence Levels	**Cannot**	O-Plan TF does not have a means to express certainty degrees.
Constraints	**Completely**	O-Plan has a rich set of constraint types to limit the plan behavior (this includes actions and resource usage).

Requirements	O-Plan TF	Descriptions
Multiple Duration(s)	**Partially**	Thinking more about your examples, there is probably only partial support. predicted duration / worst case duration Yes. Time windows are defined with a min/max and projected value. This would result in a best, worst, and most likely durations for a specific implementation of a task. If you do have a case where you know that a task might take "around an hour" or "around 2 hours" (e.g. if you use machine A or machine B to accomplish the task), then you'd go with my first example of an "or-split". average duration I was originally thinking of "average" in the context of a predicted value, but obviously they mean two different things. I'd say no to this one. actual duration Again, I was thinking about actual duration in terms of, "Task A will actually take 1hr (at most/at least/probably)" as opposed to Task A will last from timepoint.1 to timepoint.2. TF is not used in recording the execution time of a task, so no to this one as well.
Date(s) and Time(s)	**Completely**	O-Plan TF can be used to express relative temporal relationships that are tied to an initial zero date/time.
Implicit / Explicit Resource Association	**Cannot**	There are no dependency relationships between resource types in O-Plan TF.
Iterative Loops	**Cannot**	While the use of an "iterate" or "foreach" node type is planned, TF version 2.3 does not contain this functionality. (Now in O-Plan version 3.1 January 1997.)
Manual vs. Automated Tasks	**Completely**	Separate action schemas can be designed with constraints on agent binding types. If a schema is instanitated with an agent binding of type "machine" there will be a certain seq. whereas the type "human" schema would be different.
Manufacturing Product Quantity	**Completely**	The amount of product to be produced can be expressed as an achieve condition in a task schema and the action schemas can be designed to "produce" the resource based on constraints.
Material Constraints	**Partially**	Materials can be qualified through the use of resource types and "always" assertions. (e.g. bricks are wolf-proof, etc.)
Parallel Tasks	**Completely**	O-Plan actions are arranged in a partially ordered fashion that can represent parallel tasks.
Parameters and Variables	**Completely**	O-Plan plan state variables can be used to bind values to various aspects of the plan.
Pre- and Post-processing Constraints	**Completely**	This is achieved through the use of O-Plan conditions (pre) and effects (post).
Queues, Stacks, Lists	**Partially**	O-Plan TF utilizes "sets" but does not have specific data structures such as queues or stacks.
Resource Categorization and Grouping	**Completely**	Logical resource grouping is created by using specific resource types.
Resource Location	**Completely**	The "pacifica" TF sample shows how resource location can be represented using an "{at OBJ} = LOC".
Resource/Task Combined Characteristics	**Completely**	The simplest way to address this requirement is to create alternate action schemas that utilize different resources and can also thereby have different time constraints.
Serial Tasks	**Completely**	O-Plan TF can be used to impose a total ordering between actions where necessary.
State Existence Constraints	**Completely**	This requirement can be expressed in detail by selecting an appropriate condition type in O-Plan TF.
State Representations	**Completely**	O-Plan uses a state-based approach for plan domain representations (i.e. conditions and effects relative to a world state)

Requirements	O-Plan TF	Descriptions
Temporal Constraints	**Completely**	Time "windows" can be expressed for actions in O-Plan TF.
Uncertainty/Variability/Tolerance	**Completely**	Numerical variables can be represented via Min/Max pairs and a "computed" value that must lie within this range. This allows for tolerance and variability of a value.
Ability to Insert or Attach a Highlight (milestones)	**Partially**	As per Tate: O-Plan can support the attachment of milestones or statements (effects) about some point in the plan. But the ability to "highlight" or annotate some area of the plan is outside of what TF is trying to do.
Complex Precedence	**Completely**	O-Plan action orderings can be specified within an action schema or implied through the conditions and effects.
Convey the Ancestry or Class of a Task	**Completely**	The "expands" entry in an action schema denotes how it extends a higher level action.
Deadline Management	**Completely**	O-Plan can handle tasks with relative time constraints, durations, etc.
Dispatching	**Completely**	The preconditions of an action can be utilized as a mechanism for stating dispatching rules.
Eligible Resources	**Completely**	In O-Plan TF, conditions on using resources can be defined that meet this requirement.
Exception Handling and Recovery	**Partially**	Alternative schemas (and orderings) can be chosen to satisfy a task when a suggested course of action fails.
Information Exchange Between Tasks	**Completely**	Information is "passed" between actions via plan state variables.
Mathematical and Logical Operations	**Completely**	O-Plan TF can be used to express the necessary mathematical and logical operations for this requirement.
Support for Task/Process Templates	**Completely**	via Tate (22-Nov): All Task Formalism schemas are "generic processes" or "task descriptions" that meet this requirement.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	Constraints can be attached at any level of an action hierarchy that would be appropriate for that schema.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	See concurrent tasks.

OZONE

Requirements	OZONE	Descriptions
ad hoc Notes	**Not sure**	No construct/feature specified.
Cost Data	**Partially**	There is no explicit cost property for resources or tasks in OZONE, but some aspects of cost can be treated as a property that is a function of the domain (i.e. the same was as LAND or SPEED are noted in the paper).
Level of Effort	**Completely**	Demands can be defined that explicitly represent the quantity required. Activity RESOURCE-REQUIREMENTS impose resource usage/consumption constraints for the activity to execute.
Product Characteristics	**Completely**	OZONE uses a distinct concept definition for a product. Intermediate product information and work item characteristics can be attached directly to a product.
Resource	**Completely**	A resource is a distinct concept definition in OZONE. A variety of resource types are supported.
Resource Requirements for a Task	**Completely**	An activity can be defined with relationships to resources that it requires.
Simple Groupings	**Completely**	OZONE supports the grouping of tasks in a variety of ways. Tasks (activities) can be grouped into those that fulfill a demand, produce a product, or are involved in a hierarchical ordering.
Simple Resource Capability/Characteristics	**Completely**	A variety of capabilities/characteristics can be assigned to a resource via properties. (e.g. capacity, amount of set-up time needed, etc.)
Simple Sequences	**Completely**	OZONE contains INTERVAL-RELATIONS that can easily handle simple linear sequencing.
Simple Task Representation and Characteristics	**Not sure**	No construct/feature specified.
Task Duration	**Completely**	OZONE activities contain a "duration" property for this purpose.
Task Executor	**Completely**	A task executor can be modeled as a required resource for the activity.
Extensibility	**Completely**	OZONE puts forward a concept of model specialization. Elements can be added that specialize the representation for a target domain.
Resource Allocation/deallocation for one or many tasks	**Completely**	Resources provide Allocate-Capacity and Deallocate-Capacity capabilities and Activities provide reserve-resources and free-resources capabilities.
Simple Precedence	**Completely**	Various constraints can be defined to regulate precedence relationships of activities.
Composition/Decomposition	**Completely**	Compositional relationships can be defined via sub-activity relationships that form hierarchical networks of activities.
Incompleteness/Vagueness	**Partially**	To a degree, it can be stated that a constraint-based approach permits a model to be incomplete and vague on everything, except those items that are necessary to meet requirements. (e.g. Schedule these tasks in any order you like, but just make sure C is after B, etc.) What is described in the requirement though is more of a runtime test condition.
Alternative Task	**Completely**	Two activities can be defined that have the same effects. The scheduler can then select an alternative that satisfies the requirement.
Associated Illustrations and Drawings	**Not sure**	No construct/feature specified.
Complex Groups of Tasks	**Completely**	OZONE supports complex grouping of tasks. For example, a set of tasks can be grouped that meet the requirements for a specific demand, a set of tasks that produce a work item can be attached to the specific product as well.

Requirements	OZONE	Descriptions
Complex Resource Characteristics	**Completely**	OZONE provides a variety of ways to assign characteristics to resources. For example: associating state information with a resource, physical properties (range, speed), capacity models, etc.
Complex Sequences	**Completely**	Complex ordering relationships can be defined via INTERVAL-RELATIONS. (e.g. BEFORE, SAME-END, CONTAINS, etc.)
Complex Task Representation and Parameters	**Completely**	An activity can be defined with a complex set of properties. OZONE activities support an explicit set of parameters that can influence the representation of the task.
Concurrent Tasks	**Completely**	An activity can contain temporal relationships to other activities. If a relationship of same-start and same-end is defined then the two activities are constrained to be concurrent.
Conditional Tasks	**Partially**	At a high level, we can say that an activity is conditional because its execution is dependent on outstanding demands. However, there does not seem to be an explicit conditional structure.
Confidence Levels	**Cannot**	
Constraints	**Completely**	OZONE presumes an underlying constraint-based solution framework.
Multiple Duration(s)	**Partially**	While there is support for multiple durations (e.g. duration of an activity, duration of setup-time for a resource, etc.), a specific requirement of multiple durations for the overall activity does not seem possible.
Date(s) and Time(s)	**Completely**	OZONE uses various date/time relationships and assumes the existence of TIME-POINTS, and TIME-INTERVALS.
Implicit/Explicit Resource Association	**Completely**	Various levels of implicit/explicit resource associations can be made (i.e. sub-resources for aggregation, dynamic compatibility between 2 resource assignments, etc.)
Iterative Loops	**Cannot**	OZONE does not appear to support iteration or looping constructs.
Manual vs. Automated Tasks	**Completely**	OZONE does not make an explicit distinction of this type, but it would seem possible to create two activities, one that represented the manual task and one that represented the automatic task and any "differing" would be defined by each respective activity.
Manufacturing Product Quantity	**Completely**	An explicit slot for specifying product quantity is part of a demand in OZONE.
Material Constraints	**Completely**	OZONE has an explicit slot for material constraints as part of a demand (i.e. the type of material to be used).
Parallel Tasks	**Completely**	Nodes in OZONE's networks of activities can be ordered in parallel.
Parameters and Variables	**Completely**	The OZONE ontology has parameters (e.g. an activity accepts a quantity from demand) and variables (e.g. recording changes in state).
Pre- and Post-processing Constraints	**Completely**	A variety of pre and post processing constraints apply to activities. (e.g. (pre) state existence (post) duration before next activity, etc.)
Queues, Stacks, Lists	**Partially**	Lists of elements only.
Resource Categorization and Grouping	**Completely**	OZONE supports a rich set of categories and groupings of resources based on their usage, atomicity, capacity, etc.
Resource Location	**Completely**	OZONE has an explicit slot in a demand for the ORIGIN and DESTINATION for a material.
Resource/Task Combined Characteristics	**Completely**	Combined activity/resource characteristics are utilized in evaluating static and dynamic compatibility constraints.
Serial Tasks	**Completely**	Simple serial assignment falls under a "before" interval.
State Existence Constraints	**Completely**	Activities and resources can be in a given state and requirements about state existence can be applied.
State Representations	**Completely**	Activities and resources can be represented as being in certain states.

Requirements	OZONE	Descriptions
Temporal Constraints	**Completely**	A variety of constraints: absolute-time-constraint, relative-time-constraint (interval-relations, duration-constraints)
Uncertainty/Variability/Tolerance	**Partially**	Various upper/lower bounded values support variability and tolerance of assignment values, but probabilistic uncertainty is not supported.
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Completely**	Duration-Constraints, interval-relations, state requirements, and aspects of demand management all combine to provide complex precedence mechanisms.
Convey the Ancestry or Class of a Task	**Completely**	Class ancestry in OZONE is expressed through its extension mechanism of model specialization.
Deadline Management	**Completely**	Deadline management is possible via RELEASE-DATE, DUE-DATE properties of a demand.
Dispatching	**Completely**	Dispatching is encompassed in the demand-product combined capabilities. Work item generation is linked to explicit elements of demand.
Eligible Resources	**Completely**	OZONE maintains the "eligibility" of resources and also provides other USAGE-RESTRICTIONS that can allow a richer model of restrictions (e.g. UNAVAILABILITY-INTERVALS reflect time periods where a resource is not eligible, etc.)
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Completely**	OZONE supports parameters passing to exchange information between various elements (e.g. demand information is passed to an activity, etc.)
Mathematical and Logical Operations	**Completely**	Constraint expressions use mathematical and logical constructs in OZONE.
Support for Task/Process Templates	**Completely**	The ontological element "activity" is a template for what a task should be. The various properties are expected to be filled in and new slots can be added to extend this base concept.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	Constraints can be added at any level of abstraction to further define the requirements on the target space. In the example listed, you would require 5 people (resources). Next you may add a constraint on those resources (special ability). Next you may add a very specific constraint (who they are), etc.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	Multiple activities can be synchronized when parallel via INTERVAL-RELATIONS.

Parts and Actions (PAct)[1]

Requirements	PAct	Descriptions
ad hoc Notes	**Not sure**	No construct/feature specified.
Cost Data	**Not sure**	No construct/feature specified.
Level of Effort	**Not sure**	No construct/feature specified.
Product Characteristics	**Completely**	PAct focuses on describing the state of a part at all times. The construct is a graphical box.
Resource	**Not sure**	No construct/feature specified.
Resource Requirements for a Task	**Cannot**	Only resources that will become part of a product appear to show up in PAct diagrams. No manufacturing equipment, for example, appears.
Simple Groupings	**Completely**	Can be grouped by agent, or simply combined into a higher-level node.
Simple Resource Capability/Characteristics	**Not sure**	No construct/feature specified.
Simple Sequences	**Not sure**	No construct/feature specified.
Simple Task Representation and Characteristics	**Completely**	Represented by a circle in the graph-based notation.
Task Duration	**Not sure**	No construct/feature specified.
Task Executor	**Completely**	This is the principal strength of PAct. Each task, or group of tasks, is associated with one or more "agents" who "engage in a value added flow" [see Control of Parts, by Stephen Holmes Kendall, Ph.D. Thesis, MIT, Department of Architecture, 1990]. The role can be executor, or responsible, or contractor. When multiple agents are related to a single task, only one agent can control a part at a time.
Extensibility	**Not sure**	No construct/feature specified.
Resource Allocation/deallocation for one or many tasks	**Not sure**	No construct/feature specified.
Simple Precedence	**Completely**	Achieved through a part liaison line.
Composition/Decomposition	**Completely**	Achieved by expansion of a box (part) or circle (operation) into an entire sub-graph of operations and parts.
Incompleteness/Vagueness	**Not sure**	No construct/feature specified.
Alternative Task	**Not sure**	No construct/feature specified.
Associated Illustrations and Drawings	**Not sure**	No construct/feature specified.
Complex Groups of Tasks	**Not sure**	No construct/feature specified.
Complex Resource Characteristics	**Not sure**	No construct/feature specified.
Complex Sequences	**Not sure**	No construct/feature specified.
Complex Task Representation and Parameters	**Not sure**	No construct/feature specified.
Concurrent Tasks	**Not sure**	No construct/feature specified.
Conditional Tasks	**Not sure**	No construct/feature specified.
Confidence Levels	**Not sure**	No construct/feature specified.
Constraints	**Not sure**	No construct/feature specified.

[1] PAct (Parts and Actions) and EPFL's petri net representations, were only minimally analyzed because of lack of expertise and literature available at the time of analysis, therefore, there were many "not sure" ratings.

Requirements	PAct	Descriptions
Multiple Duration(s)	**Not sure**	No construct/feature specified.
Date(s) and Time(s)	**Not sure**	No construct/feature specified.
Implicit/Explicit Resource Association	**Not sure**	No construct/feature specified.
Iterative Loops	**Not sure**	No construct/feature specified.
Manual vs. Automated Tasks	**Completely**	Easily accomplished since the executing and/or specifying agent is explicitly shown.
Manufacturing Product Quantity	**Not sure**	No construct/feature specified.
Material Constraints	**Not sure**	No construct/feature specified.
Parallel Tasks	**Completely**	All tasks not shown as having a precedence relationship are implicitly parallel
Parameters and Variables	**Not sure**	No construct/feature specified.
Pre- and Post-processing Constraints	**Not sure**	No construct/feature specified.
Queues, Stacks, Lists	**Not sure**	No construct/feature specified.
Resource Categorization and Grouping	**Not sure**	No construct/feature specified.
Resource Location	**Not sure**	No construct/feature specified.
Resource/Task Combined Characteristics	**Not sure**	No construct/feature specified.
Serial Tasks	**Completely**	Precedence relationship between tasks.
State Existence Constraints	**Not sure**	No construct/feature specified.
State Representations	**Not sure**	No construct/feature specified.
Temporal Constraints	**Not sure**	No construct/feature specified.
Uncertainty / Variability / Tolerance	**Not sure**	No construct/feature specified.
Ability to Insert or Attach a Highlight (milestones)	**Not sure**	No construct/feature specified.
Complex Precedence	**Not sure**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Not sure**	No construct/feature specified.
Deadline Management	**Not sure**	No construct/feature specified.
Dispatching	**Not sure**	No construct/feature specified.
Eligible Resources	**Not sure**	No construct/feature specified.
Exception Handling and Recovery	**Not sure**	No construct/feature specified.
Information Exchange Between Tasks	**Not sure**	No construct/feature specified.
Mathematical and Logical Operations	**Not sure**	No construct/feature specified.
Support for Task/Process Templates	**Not sure**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Not sure**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Not sure**	No construct/feature specified.

PAR2

Requirements	PAR2	Descriptions
ad hoc Notes	**Completely**	object attribute in activity, resource, and product representations
Cost Data	**Partially**	extensive cost attributes for activity, resource objects, and mechanisms for consolidating and analyzing cost data in network
Level of Effort	**Partially**	scalar attribute for allocated resource objects
Product Characteristics	**Partially**	Hierarchical feature representation of products using object class/instance. emphasis on mechanical parts/assemblies of limited complexity
Resource	**Partially**	Extensible library of resource classes for people, machine objects.
Resource Requirements for a Task	**Completely**	activity-resource matrix which allows hierarchical decomposition
Simple Groupings	**Completely**	Activity groupings by class membership. pre-defined "templates" of activity groupings for detailed design, prototyping, mfg. processes, etc. which can be hierarchically decomposed.
Simple Resource Capability/Characteristics	**Completely**	No construct/feature specified.
Simple Sequences	**Completely**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Simple Task Representation and Characteristics	**Completely**	object attribute in activity, resource, and product representations
Task Duration	**Completely**	No construct/feature specified.
Task Executor	**Completely**	hierarchically decomposable activity-resource matrix
Extensibility	**Completely**	extensible lisp-based objects and methods
Resource Allocation/deallocation for one or many tasks	**Partially**	dynamic resource tracking during process simulation, but limited mechanism for resolving resource conflicts
Simple Precedence	**Completely**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Composition/Decomposition	**Completely**	Fairly sophisticated decomposition capabilities for product, activity, and resource objects. Network can be decomposed not only hierarchically, but keyed by relationships in product-activity and activity-resource matrices.
Incompleteness/Vagueness	**Partially**	Incompleteness can be specified, but not dynamically during process simulation.
Alternative Task	**Partially**	using conditional branching with extensible calls to object data structure
Associated Illustrations and Drawings	**Partially**	Diagrammatic subset of Generalized Activity Network. Also can show tree hierarchies for product, activity, resource objects
Complex Groups of Tasks	**Completely**	Using object inheritance mechanism, both for pre-defined attributes (e.g., activities performed by a given resource or specific to a given product element), by "templates," and other mechanisms.
Complex Resource Characteristics	**Partially**	Specific resource attributes include ability of a resource to provide multiple functions, etc.
Complex Sequences	**Completely**	GAN-based sequencing includes iteration capabilities, etc.
Complex Task Representation and Parameters	**Partially**	some ability to use task attributes for dynamic alteration of network flow, extensible

Requirements	PAR2	Descriptions
Concurrent Tasks	**Completely**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Conditional Tasks	**Partially**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Confidence Levels	**Completely**	probabilistic activity durations, branching
Constraints	**Partially**	both network logic constraints (GAN subset) and using states of product and resource attributes
Multiple Duration(s)	**Completely**	representation of probabilistic activity durations, and subjective input of worst/nominal/best parameters used to construct distribution
Date(s) and Time(s)	**Completely**	Dynamic tracking of date/time used after simulation for process analysis.
Implicit/Explicit Resource Association	**Partially**	implicit dependency in activity-resource matrix
Iterative Loops	**Completely**	GAN-based iterative looping with dynamic changes to branching probabilities at nodes
Manual vs. Automated Tasks	**Partially**	implicit in activity-resource matrix
Manufacturing Product Quantity	**Cannot**	
Material Constraints	**Cannot**	
Parallel Tasks	**Partially**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Parameters and Variables	**Completely**	Highly flexible but computationally inefficient manipulation of product/activity/resource class attributes during simulation and/or process enactment.
Pre- and Post-processing Constraints	**Partially**	No construct/feature specified.
Queues, Stacks, Lists	**Partially**	No construct/feature specified.
Resource Categorization and Grouping	**Completely**	resource multiple class inheritance and links in activity-resource matrix
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Partially**	activity-resource matrix keyed by object attributes
Serial Tasks	**Completely**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
State Existence Constraints	**Partially**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
State Representations	**Partially**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Temporal Constraints	**Partially**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Uncertainty/Variability/Tolerance	**Partially**	Temporal uncertainty in probabalistic duration and branching attributes
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Partially**	based on Generalized Activity Network (GAN) representation (see entry for GAN)
Convey the Ancestry or Class of a Task	**Completely**	object attributes in activity representations
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.

Requirements	PAR2	Descriptions
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	
Mathematical and Logical Operations	**Completely**	GAN network logic with extensible calls to the object structures for branching decisions, etc.
Support for Task/Process Templates	**Completely**	Pre-defined process templates for different design, testing, prototyping activities.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	

Part 49

Requirements	Part 49	Description
ad hoc Notes	**Partially**	most entities have a name and description attribute which have few restrictions
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Partially**	through the product_definition entity - this information would be accessed through a different Part of STEP
Resource	**Completely**	through the action_resource and resource entities this information would be accessed through a different Part of STEP
Resource Requirements for a Task	**Completely**	through the action_resource_requirements and requirement_for_action_resource entities the action_resource_requirement specifies a requirements of a resource for the performance of an action. It can specify either a particular type of resource or a characteristic possessed by a resource. The requirement_for_action_resource specifies the resources which can satisfy the requirement(s). No quantity is explicitly included.
Simple Groupings	**Partially**	through the action_method_to_select_from eneity this specifies the number of action_methods that are available to choose from. A context can be applied by using the context_dependent_action_ method_relationship entity instead of the action_method_relationship entity.
Simple Resource Capability/Characteristics	**Completely**	Through the resource_property entity this is a characteristic of a resource. NOTE: This is the same construct used for complex resource characteristics in the outer core.
Simple Sequences	**Completely**	through the sequential_method entity each set of action_methods are completed in a certain order
Simple Task Representation and Characteristics	**Completely**	through the action, action_method, product_definition_process, and product_property_process entities an action can be defined by how it contributes to the creation of a product (product_definition_process) or in a more general sense by what it is expected to produce irrespective of what product it is used for (property_property_process)
Task Duration	**Completely**	through the action_property entity, you can associate any characteristic with an action entity

Requirements	Part 49	Description
Task Executor	**Cannot**	except if you assume a task executor is only a type of resource with no unique properties
Extensibility	**Completely**	Through the creation of an AP that expands on Part 49. Once this AP is created, it is not extensible.
Resource Allocation/deallocation for one or many tasks	**Cannot**	only resource requirements, not allocation
Simple Precedence	**Partially**	through the serial_action_method and sequential_action_method entities the order of the operations can be specified but specific details such as information requirements scan not
Composition / Decomposition	**Cannot**	No construct/feature specified.
Incompleteness/Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Completely**	through the replacement_relationship entity an action_relationship that specifies that a specific action may replace an existing action
Associated Illustrations and Drawings	**Completely**	through the action_method_with_specification_reference and action_method_ with_specification_method_constrained entities an action_method_with_specification_reference is a subtype of an action_method which specifies a related document an action_method_with_specification_reference_constrained is a subtype of an action_method_with_specification_reference that specifies portions of a document or a constraint on the whole document
Complex Groups of Tasks	**Completely**	through the action_method_to_select_from entity this just specifies the number of action_methods that are available to choose from. A context can be applied by using the context_dependent _action_method_relationship instead of the action_method_relationship
Complex Resource Characteristics	**Completely**	Through the resource_property, resource_property_representation, and resource_ property_relationship entities represents a characteristic of a resource. this description may include the behavior, capability, or performance measures that are pertinent to the process or the actions to effect a process which the resource is used. A way of representing (realizing) the property is also included.
Complex Sequences	**Partially**	through the sequential_method, concurrent_action_method, context_dependent_ action_method, and serial_action_method entities
Complex Task Representation and Parameters	**Completely**	Through the action_property and action_property_relationship entities description of the behavior, capabilities, or performance measures of some property (aspect) of the action along with some way of representing (realizing) the property.
Concurrent Tasks	**Completely**	Through the concurrent_action_method entity the individual action_method in this collection shall be completed during completion of the action_method with the greatest duration (no start-to-start, finish-to-finish, etc.)
Conditional Tasks	**Completely**	Through the context_dependent_action_relationship and context_ dependent_action_method_relationship entities an association between two action(_methods)_relationships that specifies a context for the completion of the action(_method). It uses the context_dependent_relationship_condition to specify the context and/or condition.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Cannot**	temporal - not really (only through concurrent, serial, and sequential actions) material - no existence - no
Multiple Duration(s)	**Cannot**	no aspects of time are explicitly represented

Requirements	Part 49	Description
Date(s) and Time(s)	**Cannot**	
Implicit/Explicit Resource Association	**Partially**	through teh requirements_for_action_resource only explicit associations which are represented as a set of possible resources
Iterative Loops	**Cannot**	can possibly use the context_dependent_action_method_relationship entity but I have doubts that this will work
Manual vs. Automated Tasks	**Cannot**	only in the generic description of the action
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Partially**	although there is no explicit "parallel task" entity, Part 49 allows you to specify two separate tasks that are not related to each other
Parameters and Variables	**Partially**	EXPRESS - the language Part 49 is written in, can handle them
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Partially**	through the requirement_for_action_resource entity there is an attribute in this entity called 'resources' which points to a set of action_resources that can satisfy the requirement(s) of the action
Resource Location	**Not sure**	information might be accessible through another Part of STEP but unsure
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	through the serial_action_method entity the individual action_methods shall be complete when the collection of action_methods is complete
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Cannot**	No construct/feature specified.
Uncertainty / Variability / Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight (milestones)	**Partially**	through the action entity you can insert an additional action which just described the milestone since task durations are not represented in Part 49
Complex Precedence	**Partially**	through teh serial_action_method and the sequential_action_method entities the order of the operations can be specified but specific details such as information requirements can not
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Completely**	through the requirements_for_action_resource entity there is an attribute called "resources' which list the possible resources which can fulfill the requirements for an action
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Completely**	EXPRESS (the language that Part 49 is written in) can do mathematical and logical operations using DERIVEd attributes
Support for Task/Process Templates	**Cannot**	No construct/feature specified.

Requirements	Part 49	Description
Support for Simultaneously Maintained Associations of Mult Lev of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

PERT Networks (assuming standard PERT networks, not probabilistic PERT, GERT, etc. variations)

Requirements	PERT Networks	Descriptions
ad hoc Notes	**Cannot**	No construct/feature specified.
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Cannot**	No construct/feature specified.
Resource	**Cannot**	No construct/feature specified.
Resource Requirements for a Task	**Cannot**	No construct/feature specified.
Simple Groupings	**Cannot**	No construct/feature specified.
Simple Resource Capability/Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Completely**	No construct/feature specified.
Simple Task Representation and Characteristics	**Cannot**	No construct/feature specified.
Task Duration	**Partially**	deterministic or 3-parameter approx. to beta distribution typical of most PERT variants
Task Executor	**Cannot**	No construct/feature specified.
Extensibility	**Cannot**	No construct/feature specified.
Resource Allocation/deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Completely**	precedence in simple directed acyclic graph
Composition/Decomposition	**Cannot**	No construct/feature specified.
Incompleteness/Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Cannot**	No construct/feature specified.
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Cannot**	No construct/feature specified.
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Partially**	parallel tasks in directed acyclic graph
Conditional Tasks	**Cannot**	No construct/feature specified.
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Cannot**	No construct/feature specified.
Multiple Duration(s)	**Cannot**	No construct/feature specified.
Date(s) and Time(s)	**Cannot**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.

Requirements	PERT Networks	Descriptions
Iterative Loops	**Cannot**	No construct/feature specified.
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Cannot**	No construct/feature specified.
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Partially**	No construct/feature specified.
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Partially**	No construct/feature specified.
Uncertainty/Variability/Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Cannot**	No construct/feature specified.
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Cannot**	No construct/feature specified.

Petri Nets

Requirements	Petri Nets	Descriptions
ad hoc Notes	**Partially**	you can add any notes you want to a transition or place, it is not restricted
Cost Data	**Partially**	you can add any notes you want to a transition or place, it is not restricted
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Partially**	colored petri nets. a product can be associated with a specific type of token and that token can have characteristics
Resource	**Partially**	tokens in a coloured petri net. a specific color or type of token in a coloured petri net can represent resources

Requirements	Petri Nets	Descriptions
Resource Requirements for a Task	**Partially**	in a colored petri net, all tokens that are need for a transition to fire. if different resources are represented by different colors, a transition may state that it needs a green, red, and yellow token to fire. These are the resource requirements for the task (transition).
Simple Groupings	**Completely**	an entire petri net can be considered a process plan (a simple grouping of tasks) and can be referenced by other petri nets
Simple Resource Capability/Characteristics	**Partially**	colored petri nets. a resource can be associated with a specific type of token and that token can have its respective characteristics
Simple Sequences	**Completely**	combination of transitions and places. you can have a simple sequence by linearly having a place, transition, place, transitions, etc.
Simple Task Representation and Characteristics	**Completely**	using an transition box. tasks are called transitions and are represented by a box
Task Duration	**Completely**	with a timed petri net, you can associate delays with either places or transitions
Task Executor	**Cannot**	No construct/feature specified.
Extensibility	**Completely**	petri nets have been extended by many people including: stochastic petri nets colored petri nets predicated petri nets timed petri nets etc.
Resource Allocation/deallocation for one or many tasks	**Partially**	when a transition is occurring, not only transition can use the tokens in which it needed to fire. If these token represents resources, resource allocation is accomplished.
Simple Precedence	**Partially**	you can represent a transition (task) that must be done before another transition by having the first transition output a token that the second transition needs in order to fire
Composition / Decomposition	**Completely**	by having nested and expandable petri nets
Incompleteness / Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Completely**	by having multiple output transitions from any given state with only one token in that state
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.
Complex Groups of Tasks	**Cannot**	task can be grouped into a system using a petri net but that's it (this is more simple than complex)
Complex Resource Characteristics	**Partially**	if a token represents a resource, you can associate anything you want with that token
Complex Sequences	**Completely**	see conditional tasks, alternative tasks, parallel tasks, concurrent tasks, serial tasks
Complex Task Representation and Parameters	**Partially**	assuming that a transition represents a task, one can associate anything they want with the transition
Concurrent Tasks	**Partially**	only if you use timed petri nets and include the same delays
Conditional Tasks	**Partially**	interpreted petri nets. in an interpreted petri net, one can state a condition that must be true for a transition to fire
Confidence Levels	**Cannot**	No construct/feature specified.
Constraints	**Partially**	see various constraints listed separately
Multiple Duration(s)	**Cannot**	timed petri nets only allow for a single duration associated with any transition or place
Date(s) and Time(s)	**Cannot**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.

Requirements	Petri Nets	Descriptions
Iterative Loops	**Completely**	by having a event that is both the input and output to a transition with a corresponding criteria
Manual vs. Automated Tasks	**Partially**	although there are no constructs that convey this, anything can be associated with a task - including an attribute which states if it is manual or automatable
Manufacturing Product Quantity	**Partially**	colored petri nets. the number of a certain type of token which represents the number of products to be manufacturing can represent this
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	combination of places and transitions. two transitions don't have to be related to each other to be in the same system
Parameters and Variables	**Completely**	predicated petri nets. the tokens can themselves be parameters which are attributed
Pre- and Post-processing Constraints	**Completely**	places (circles). places represent pre- and post- conditions
Queues, Stacks, Lists	**Partially**	FIFO petri nets. First-In-First-Out (FIFO) petri nets keep track of when a token is put in a place and lets out the first token that came in first
Resource Categorization and Grouping	**Partially**	A specific color of token in a colored petri net. All resources that can perform a specific operation may be represented by a red token. All red tokens would make up a resource group.
Resource Location	**Partially**	if a certain type of token represents a resource, you can associate any type of attributes you like with that token - including location
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	place1-transition1-place2-transition2 when transition1 fires, it moves a token from place1 to place2. Transition2 cannot fire until place2 has a token.
State Existence Constraints	**Completely**	a transition can not fire until all of the tokens in the input places are present. If you want a state existence constraint, include it as an input place.
State Representations	**Completely**	circles represent states. the tokens in the circles at any given time represent the state of the system at that time
Temporal Constraints	**Cannot**	No construct/feature specified.
Uncertainty / Variability / Tolerance	**Partially**	stochastic petri nets. uncertainty can be associated with time
Ability to Insert or Attach a Highlight(milestones)	**Partially**	one can consider any state (place) a milestone. Therefore, any state can have a milestone associated with it.
Complex Precedence	**Not sure**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Partially**	colored petri nets. one can use colored tokens which represent resources. Those resources which are of a certain color are eligible.
Exception Handling and Recovery	**Not sure**	No construct/feature specified.
Information Exchange Between Tasks	**Partially**	tokens. information can be partially exchanged by using the tokens as the exchange mechanism
Mathematical and Logical Operations	**Completely**	interpreted petri nets. can be used for conditions

Requirements	Petri Nets	Descriptions
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Partially**	one can have nested petri nets to represent multiple levels of abstraction
Synchronization of Multiple, Parallel Task Sequences	**Completely**	places, token, and transitions. have the two or more transitions rely on the same token (output from another transition) to begin

Process Flow Representation (PFR)

Requirements	PFR	Descriptions
ad hoc Notes	**Completely**	:advice. Used for both stylized and completely free form notes in a property list style. No direct support for non-text (graphics, audio); have to name a file.
Cost Data	**Partially**	quality control, e.g. applications have been written that associate cost of variation with a process step
Level of Effort	**Partially**	no specific construct
Product Characteristics	**Completely**	:change-wafer-state
Resource	**Partially**	:machine specifies equipment to be used by its name. PFR does not have sophisticated descriptions of resources (handled in CAFE by a separate representation language)
Resource Requirements for a Task	**Partially**	see resource
Simple Groupings	**Completely**	(flow (:body step1 step2 ...))
Simple Resource Capability / Characteristics	**Cannot**	not in PFR language itself (PFR does not have any sophisticated descriptions of resources. This is handled in CAFE be a separate representation language.)
Simple Sequences	**Completely**	flow is hierarchically recursive concept (see groups of tasks)
Simple Task Representation and Characteristics	**Completely**	all process steps (flow) attributes can be specified at any hierarchical level
Task Duration	**Completely**	:time-required
Task Executor	**Partially**	:instructions. specify operator instructions
Extensibility	**Completely**	:advice
Resource Allocation/deallocation for one or many tasks	**Partially**	:machine. specifies equipment to be used by a step
Simple Precedence	**Completely**	No construct/feature specified.
Composition / Decomposition	**Completely**	No construct/feature specified.
Incompleteness / Vagueness	**Completely**	No construct/feature specified.
Alternative Task	**Not sure**	No construct/feature specified.
Associated Illustrations and Drawings	**Partially**	must point to a file or database object
Complex Groups of Tasks	**Not sure**	No construct/feature specified.

Requirements	PFR	Descriptions
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Partially**	:wafers. specified which wafers (sublot) are to be processed. Thus serial/parallel can be inferred
Complex Task Representation and Parameters	**Partially**	(:delay :minimal) -- flow is uninterruptible see also :advice
Concurrent Tasks	**Cannot**	PFR does not have a way to say "these two operations MUST be performed concurrently". (It would not be hard to add; we have just never found a specific need for it here.)
Conditional Tasks	**Partially**	(if) extension language :advice
Confidence Levels	**Completely**	(:mean :range) (:gaussian :mean :variance). numerical values can have uncertainty
Constraints	**Partially**	see :advice, confidence level
Multiple Duration(s)	**Completely**	see confidence level
Date(s) and Time(s)	**Partially**	these are stored in the CAFE database
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Partially**	(in practice we find operator intervention is always required anyway
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Partially**	(see :wafer :advice)
Material Constraints	**Partially**	named materials
Parallel Tasks	**Completely**	
Parameters and Variables	**Completely**	CAFE database
Pre- and Post-processing Constraints	**Partially**	(see :advice)
Queues, Stacks, Lists	**Partially**	:sequence
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	No construct/feature specified.
State Existence Constraints	**Not sure**	No construct/feature specified.
State Representations	**Not sure**	No construct/feature specified.
Temporal Constraints	**Completely**	No construct/feature specified.
Uncertainty/Variability/ Tolerance	**Completely**	see confidence level
Ability to Insert or Attach a Highlight(milestones)	**Completely**	:advice, "dummy" tasks
Complex Precedence	**Partially**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Partially**	maybe, not quite sure what you mean by specialization here.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.

Requirements	PFR	Descriptions
Eligible Resources	**Partially**	:machines -- can specify list of possible resources
Exception Handling and Recovery	**Partially**	runtime mechanism, usually involves operator intervention
Information Exchange Between Tasks	**Completely**	use database
Mathematical and Logical Operations	**Completely**	all standard math plus function (lambda) abstraction
Support for Task/Process Templates	**Completely**	functional abstraction to create parameterized flows library/database search/load mechanism for source/object inclusion
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Partially**	(extension language)

Process Interchange Format (PIF) Core Version 1.1

Requirements	PIF Core V. 1.1	Descriptions
ad hoc Notes	**Partially**	This requirement is partially filled through the use of the "documentation" slot and the "user-attributes" that can be attached to PIF entities.
Cost Data	**Cannot**	PIF does not address resource or task cost.
Level of Effort	**Cannot**	PIF can say that an activity uses some object, but does not have a mechanism to quantify the usage.
Product Characteristics	**Partially**	While PIF can represent objects that are created, modified, or used during an activity, there is no provision for attaching characteristics to that object. Represent [a] dependency via Precondition and Postcondition of an activity. A resource X requires another resource Y if the Use activity that uses X has as a precondition the availability of the resource Y
Resource	**Completely**	Activities can specify which objects (resources) were created, modified or used.
Resource Requirements for a Task	**Partially**	PIF cannot represent quantity of an object (resource).
Simple Groupings	**Completely**	PIF can express grouping of activities through decomposition relationships. satisfied if the grouping is a deterministic activity, but is not satisfied in general for nondeterministic activities.
Simple Resource Capability / Characteristics	**Partially**	PIF Core does not have an explicit mechanism to describe the "capability" of an object (when used in the role of resource) but it does allow for attributes of such objects to be stated. Capability limited to agents.

Requirements	PIF Core V. 1.1	Descriptions
Simple Sequences	**Completely**	The use of timepoints and temporal relationships will provide simple sequences. This requirement is satisfied insofar as we can write the definition of an activity for simple and complex sequences. However, we cannot express the definition of simple and complex sequences using PIF-Core.
Simple Task Representation and Characteristics	**Completely**	PIF can represent a task with its effects, conditions, etc. Also, textual high-level descriptions can be attached via the documentation attribute.
Task Duration	**Completely**	An activity can contain begin and end points, but PIF Core itself does not support quantities for duration. This can be captured, since the axiomatization of time points in the situation calculus means that time points are isomorphic to the real numbers.
Task Executor	**Completely**	PIF can describe a "performs" relationship between activities and agents.
Extensibility	**Completely**	PIF has a "user-attributes" slot defined at the highest level of the hierarchy that can store user-defined information.
Resource Allocation / deallocation for one or many tasks	**Partially**	Tate's input that pointed out that specifying individual resource units are not part of the requirement. PIF can represent objects that are created, modified, or used during an activity.
Simple Precedence	**Completely**	PIF can provide a detailed description of activities' relationships to other activities. Temporal, causal, and decompositional relationships can be used to impose constraints on the precedence.
Composition/Decomposition	**Completely**	PIF supports decompositional relationships between activities. Therefore activities can be arranged in an abstract, incomplete, or ambiguous fashion.
Incompleteness/Vagueness	**Completely**	PIF supports decompositional relationships between activities. Therefore activities can be arranged in an abstract, incomplete, or ambiguous fashion.
Alternative Task	**Completely**	PIF's use of decisions allows for a selection of alternative tasks. Although decisions can be used to select an activity based on state, this cannot be used to define arbitrary nondeterministic choices e.g. do A or do B.
Associated Illustrations and Drawings	**Partially**	[a PIF user] can use the documentation or user attribute slots for this.
Complex Groups of Tasks	**Partially**	This requirement asks for information that goes beyond specifying which sub-activities occur in a group and asks whether there is explicit representation about the overall group (total cost, total resources used in decomposition, etc.) satisfied if the grouping is a deterministic activity, but is not satisfied in general for nondeterministic activities.
Complex Resource Characteristics	**Cannot**	See annotation at "simple resource capability/characteristics".
Complex Sequences	**Completely**	PIF can handle concepts such as: alternative, parallel, serial, concurrent activities. Timepoints and temporal relationships provide these requirements. This requirement is satisfied insofar as we can write the definition of an activity for simple and complex sequences. However, we cannot express the definition of simple and complex sequences using PIF-Core.
Complex Task Representation and Parameters	**Partially**	PIF's highly expressive pif-sentences can be used to give a detailed representation of what an activity needs and does (hierarchical activities are considered "grouped"). More specialized charac. (e.g. uniterruptability) cannot be expressed.
Concurrent Tasks	**Completely**	See Complex Sequences Annotation.

Requirements	PIF Core V. 1.1	Descriptions
Conditional Tasks	**Completely**	PIF uses the entity, decision, to represent a conditional activity.
Confidence Levels	**Cannot**	PIF sentences are boolean.
Constraints	**Completely**	Activities and objects (resources) inherit constraint slots for such purposes.
Multiple Duration(s)	**Partially**	Activities can express durations through relative begin and end timepoints.
Date(s) and Time(s)	**Partially**	relative begin and end timepoints can be specified.
Implicit/Explicit Resource Association	**Completely**	PIF representation of object (resource) component can be interpreted to some extent as a dependency. (e.g. Object A has components Object B and C. Therefore using Object A implies using Object B and C as well.) Represent [a] dependency via Precondition and Postcondition of an activity. A resource X requires another resource Y if the Use activity that uses X has as a precondition the availability of the resource Y.
Iterative Loops	**Completely**	This can be satisfied using decisions or preconditions and postconditions of activities.
Manual vs. Automated Tasks	**Cannot**	No information is explicitly captured for this in the PIF-CORE.
Manufacturing Product Quantity	**Cannot**	Not in PIF-CORE, sounds like a PSV.
Material Constraints	**Cannot**	PIF contains no support for material constraints.
Parallel Tasks	**Completely**	See Complex Sequences Annotation. This requirement is satisfied insofar as we can write the definition of an activity for parallel or serial tasks. However, we cannot express the definition of parallel or serial tasks using PIF-Core.
Parameters and Variables	**Completely**	PIF activities utilize variables in pif-sentences.
Pre- and Post-processing Constraints	**Completely**	PIF declares what must be true before an activity is performed and also asserts what must be true after the activity completes.
Queues, Stacks, Lists	**Partially**	PIF provides a list structure.
Resource Categorization and Grouping	**Partially**	To some extent, PIF can address this by explicitly listing which objects (resources) each activity uses/creates/modifies. PIF can also describe which objects are components of other objects, but logical grouping (outside of activity) seems to be absent.
Resource Location	**Cannot**	There is no facility in the PIF-CORE to address this relationship.
Resource/Task Combined Characteristics	**Cannot**	There is no facility in the PIF-CORE to address this relationship.
Serial Tasks	**Completely**	PIF can order activities in serial. This requirement is satisfied insofar as we can write the definition of an activity for parallel or serial tasks. However, we cannot express the definition of parallel or serial tasks using PIF-Core.
State Existence Constraints	**Completely**	PIF can constrain when activities can be executed using activity preconditions.
State Representations	**Partially**	PIF can describe state changes for activities but not for objects (resources).
Temporal Constraints	**Partially**	PIF can relate activities through shared begin/end points, but a PSV is required to appropriately address temporal relationships (other than "before"). X to #, BEFORE is available and ... some temporal constraints can be expressed using the begin and end timepoints.

Requirements	PIF Core V. 1.1	Descriptions
Uncertainty/Variability/Tolerance	**Cannot**	PIF does not have a facility for managing uncertainty.
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	Not in PIF-CORE.
Complex Precedence	**Completely**	The use of preconditions and decisions allows for complex, conditional activity orderings. This requirement is satisfied insofar as we can write the definition of an activity for simple and complex sequences. However, we cannot express the definition of simple and complex sequences using PIF-Core.
Convey the Ancestry or Class of a Task	**Completely**	PIF's decompositional relationships define a hierarchy of specialization.
Deadline Management	**Partially**	PIF's activities utilize an activity-status relation which is linked to timepoints. Therefore PIF can set timepoints for when an activity must be at a certain status.
Dispatching	**Cannot**	This level of object detail is not explicitly represented in PIF.
Eligible Resources	**Partially**	In terms of agents (as resources) PIF can explicitly describe their capability, thus making them eligible. PIF does not provide the "eligibility" of other non-agent objects.
Exception Handling and Recovery	**Completely**	PIF's conditional activities can respond to exception handling and recovery. Depends on interpretation. PIF cannot (a) while the (b) is simply a decision activity. (see below) There are two interpretations of this construct: a) global occurrence constraint which must be satisfied regardless of what activities are occurring at any point in time.b) a conditional activity which occurs at specific points during a complex activity.
Information Exchange Between Tasks	**Partially**	The flow can be partially mapped out by illustrating which activities create/modify/use objects.
Mathematical and Logical Operations	**Completely**	PIF has a mechanism to derive boolean results for conditionals, etc.
Support for Task/Process Templates	**Partially**	PIF does not provide this explicit form of meta element, but PIF design elements can be reused.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Completely**	PIF decomposition allows a designer to attach/modify activity relationships at any level of abstraction.
Synchronization of Multiple, Parallel Task Sequences	**Partially**	PIF does not contain any real-time event signaling and notification that could manage multiple, parallel, activities. X to #: The definition of synchronized activities does not depend on real-time event signaling and notification; this only becomes an issue when coordinating agents within an organization.

Quirk Model

Requirements	Quirk Model	Descriptions
ad hoc Notes	**Cannot**	No construct/feature specified.
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Cannot**	No construct/feature specified.
Resource	**Cannot**	No construct/feature specified.
Resource Requirements for a Task	**Cannot**	No construct/feature specified.
Simple Groupings	**Partially**	A Q-model consists of a group of processes, which can be thought of as tasks. Thus, the model itself is a group of tasks. The tasks may only be grouped provided that they make up a plan, or algorithm, or a system. Hence, we can't use Q-model to group tasks based on their functionality, execution durations, etc.
Simple Resource Capability / Characteristics	**Cannot**	No construct/feature specified.
Simple Sequences	**Partially**	By lining up the processes connected by synchronous channels, we can represent a time-sequential groups of tasks.
Simple Task Representation and Characteristics	**Cannot**	No construct/feature specified.
Task Duration	**Partially**	This may be specified as an "interval estimate for process execution time" of the process which represents the particular task in question. All time intervals in a Q-model are supposed to be estimates. Thus, there's no explicit construct for actual duration, etc.
Task Executor	**Partially**	A task executor can be represented as a "common process" which simply sends activation signals to the process that is the task it executes.
Extensibility	**Partially**	One can certainly change the Q-model as long as the application that implements it allows. However, changes are not guaranteed to be local, i.e., one may need to alter other processes and channels when a new process is introduced into the model.
Resource Allocation/deallocation for one or many tasks	**Partially**	A "selector process" can decide which input channel's input it wants. Thus, we can connect the input channels to "common processes" representing resources, and the input channels that are selected would correspond to allocated resources.
Simple Precedence	**Cannot**	The constraints that apply to the processes in a Q-model are assumed to be implicit to the processes, and thus, constraints of any sort cannot be represented by a Q-model.
Composition / Decomposition	**Cannot**	No construct/feature specified.
Incompleteness / Vagueness	**Cannot**	No construct/feature specified.
Alternative Task	**Partially**	Using "common processes" with channels connected to an "input selector process," we can represent a list of alternative tasks as those "common processes."
Associated Illustrations and Drawings	**Cannot**	No construct/feature specified.

Requirements	Quirk Model	Descriptions
Complex Groups of Tasks	**Partially**	A Q-model consists of a group of processes, which can be thought of as tasks. Thus, the model itself is a group of tasks. The tasks may only be grouped provided that they make up a plan, or algorithm, or a system. Hence, we can't use Q-model to group tasks based on their functionality, execution durations, etc.
Complex Resource Characteristics	**Cannot**	No construct/feature specified.
Complex Sequences	**Partially**	Serial tasks can be represented as "common processes" which are connected, one after another, via synchronous channels. Alternative tasks, as mentioned above, can be represented with the help of an "input selector process." Parallel tasks can sort of be represented because two Q-models are basically two groups of tasks that can occur at any time independent of each other. Concurrent tasks cannot be represented.
Complex Task Representation and Parameters	**Cannot**	No construct/feature specified.
Concurrent Tasks	**Cannot**	No construct/feature specified.
Conditional Tasks	**Partially**	A "common process" can be thought of as conditional if it receives input from an "output selector process" which, based on certain algorithm, determines whether output will be sent to this process.
Confidence Levels	**Cannot**	
Constraints	**Partially**	Only temporal constraints can be represented. This will be explained below under "temporal constraints."
Multiple Duration(s)	**Partially**	A task's or a resource's duration can be represented as the "estimate interval of execution time" of a process which denotes the task or resource. Multiple Q-model "processes" are needed to represent multiple durations of one task/resource.
Date(s) and Time(s)	**Cannot**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Partially**	Iterative loops can be represented by connecting a channel from the last process back to the first one. The problem is that it is assumed that the processes have some way of generating output that will cause the termination of the loop. This terminal condition is not explicitly represented in a Q-model.
Manual vs. Automated Tasks	**Cannot**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Partially**	No explicit construct, but two tasks represented as two different Q-models can presumably occur at any time independent of each other. Thus, they are parallel.
Parameters and Variables	**Cannot**	No construct/feature specified.
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Cannot**	No construct/feature specified.

Requirements	Quirk Model	Descriptions
Resource Categorization and Grouping	**Cannot**	No construct/feature specified.
Resource Location	**Cannot**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	can be represented by connecting that processes serially with channels.
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Partially**	The estimate interval of execution time of a process and dely of input/output of a process can be used to represent some sort of temporal constraints.
Uncertainty / Variability / Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight (milestones)	**Cannot**	No construct/feature specified.
Complex Precedence	**Cannot**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Cannot**	No construct/feature specified.
Exception Handling and Recovery	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Partially**	Channels connecting processes allow flow of information among processes. The problem here is that it can only specify that some information is passed around, but exactly what it is not explicit in the model.
Mathematical and Logical Operations	**Cannot**	No construct/feature specified.
Support for Task/Process Templates	**Cannot**	No construct/feature specified.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Cannot**	No construct/feature specified.
Synchronization of Multiple, Parallel Task Sequences	**Partially**	Multiple processes may receive inputs from synchronous channels connected to the same process. These processes may, thus, begin at around the same time as they receive the same input.

Visual Process Modeling Language (VPML)[1]

Requirements	VPML	Descriptions
ad hoc Notes	**Completely**	This is handled by the 'annotations' construct which is described as text used for explanation
Cost Data	**Cannot**	No construct/feature specified.
Level of Effort	**Cannot**	No construct/feature specified.
Product Characteristics	**Completely**	These are items that are used, created, modified, and transferred among activities in a process. This includes documents, messages, and artifacts (a default category). All of these are modeled using some "subtype" of the 'product' construct.
Resource	**Completely**	through the 'resource' construct. These are used to model real-world resources that are required to perform an activity. This could be either human or non-human resources.
Resource Requirements for a Task	**Partially**	a resource type is associated with an activity early on and then a resource instance is associated later. Resource amount is not explicitly represented but can be added if necessary.
Simple Groupings	**Completely**	grouping of tasks can be done with the 'composite activity' construct grouping of resources can be done with the 'group' construct which is part of the 'resource' construct
Simple Resource Capability / Characteristics	**Completely**	through the 'resources' construct
Simple Sequences	**Completely**	through the 'finsh_start' connection
Simple Task Representation and Characteristics	**Completely**	through the 'activity' construct. Included in this construct is only the name of the activity.
Task Duration	**Cannot**	No construct/feature specified.
Task Executor	**Partially**	each resource can have a 'role' that specifies how it is related to an activity. A role for a resource could be 'task executor'.
Extensibility	**Cannot**	this is a language requirement anayway so it is insignificant
Resource Allocation/deallocation for one or many tasks	**Cannot**	No construct/feature specified.
Simple Precedence	**Partially**	you can use the 'data flow' construct to show the direction that data or products are moving through the system. Since an activity can't start until it receives this data or product, precedence is inferred.
Composition / Decomposition	**Partially**	composition/decomposition is handles by the 'composite activity' construct. This allows hierarchial decomposition of activities, which allows process definers to represent the same process in varying levels of detail.
Incompleteness / Vagueness	**Cannot**	Using the 'composite activity' construct. process definers can represent the same process in varying levels of detail. Thus, a process with incomplete information can be represented at a very high level, and so on.
Alternative Task	**Completely**	through the 'output_or' construct

[1] An earlier version of VPML was used in the analysis that is described in this paper. Just prior to publication, ISSI, the developers of VPML and related tools, contributed a second analysis of VPML based on a more recent version. This additional analysis is included here for completeness.

Requirements	VPML	Descriptions
Associated Illustrations and Drawings	**Completely**	this is captured by the 'document' construct which is part of the 'product' construct. A document is something that is created, modified, or used as an activity is carried out.
Complex Groups of Tasks	**Cannot**	No construct/feature specified.
Complex Resource Characteristics	**Completely**	through resource attribute specifications
Complex Sequences	**Partially**	and/or split can be accomplished through the 'input_and', 'input_or', 'output_and', and 'output_or' constructs. There are no constructs for conditional tasks. Serial tasks can be represented with the 'finish_start' construct. Concurrent tasks can be represented with the 'timer' which models periodic events of events that happen at a specific time. If two events happen at the same specific time, they are concurrent.
Complex Task Representation and Parameters	**Completely**	allows for attribute specifications
Concurrent Tasks	**Completely**	supports temporal synchronization using the 'timer' construct
Conditional Tasks	**Completely**	supports conditional paths using the 'output_or' branching data flow
Confidence Levels	**Cannot**	
Constraints	**Partially**	in the simple case yes - when you get to more detailed constraints, no material constraints - no temporal constraints - yes pre and post condition - kind_of, in the simple cases state existence constraints - no
Multiple Duration(s)	**Cannot**	durations are not represented
Date(s) and Time(s)	**Completely**	dates and times are handled by the 'timer'
Implicit/Explicit Resource Association	**Cannot**	No construct/feature specified.
Iterative Loops	**Completely**	iteration is described as a feature of VPML in the ProSLCSE homepage
Manual vs. Automated Tasks	**Completely**	an activity can either be a leaf activity (manual) or an automatic activity (automated)
Manufacturing Product Quantity	**Cannot**	No construct/feature specified.
Material Constraints	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	tasks can be in any order with respect to each other
Parameters and Variables	**Not sure**	No construct/feature specified.
Pre- and Post-processing Constraints	**Cannot**	No construct/feature specified.
Queues, Stacks, Lists	**Partially**	These can be handled by the folder (composite product) construct which is part of the product construct. A folder is a dynamic collection of products. When a folder is connected to an automatic activity, the activity's script specifies the procedure by which products are extracted from or inserted into the folder.
Resource Categorization and Grouping	**Partially**	resource grouping can be done at both the type and instance level using the 'group' construct in the 'resource' construct resource categorization can not be represented explicitly but the 'group' construct can be used to do this in an ad hoc way
Resource Location	**Completely**	the 'location' construct in the 'resource' construct is used to model a physical place that is need to perform an activity. This could be a location type (a meeting room) or a location instance (meeting room 200).
Resource/Task Combined Characteristics	**Cannot**	No construct/feature specified.

Requirements	VPML	Descriptions
Serial Tasks	**Completely**	through the 'finish_start' data flow construct
State Existence Constraints	**Cannot**	No construct/feature specified.
State Representations	**Cannot**	No construct/feature specified.
Temporal Constraints	**Completely**	the following temporal relationship can be represented: finish_to_start start_to_start finish_to_finish start_after_start finish_after_finish
Uncertainty/Variability /Tolerance	**Cannot**	No construct/feature specified.
Ability to Insert or Attach a Highlight(milestones)	**Completely**	through the 'milestone' construct which is described as 'modeling significant events in a process'
Complex Precedence	**Cannot**	simple precedence can be accomplished through data dependencies with the 'data flow' constructs but this is the extent of what it can do
Convey the Ancestry or Class of a Task	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	No construct/feature specified.
Eligible Resources	**Partially**	you can create a resource group which satisfy a given requirements but this can only be done manually
Exception Handling and Recovery	**Cannot**	only through the use of alternative tasks but this really doesn't cover it
Information Exchange Between Tasks	**Completely**	This is handled with the 'message' construct that is a type of the 'product' construct. Messages are defined as information that is output from or input to an activity.
Mathematical and Logical Operations	**Not sure**	insignificant - this is a language characteristic
Support for Task/Process Templates	**Cannot**	only machine, tool .location, and group types (or templates) are supported
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Partially**	For the association of resources with activities, a resource type is associated with an activity early on and it is instantiated closer to execution.
Synchronization of Multiple, Parallel Task Sequences	**Completely**	This is accomplished through the 'timer' construct. Temporal synchronization is a feature advertised in the ProSLCSE homepage.

VPML (Latest release and supporting tools)[1]

Requirements	VPML Old	VPML New	Comments
ad hoc Notes	**Completely**	**Completely**	No construct/feature specified.
Cost Data	**Cannot**	**Completely**	The cost data for each activity and total cost can be found after process simulation.
Level of Effort	**Cannot**	**Completely**	The data of effort can be found after process simulation
Product Characteristics	**Completely**	**Completely**	No construct/feature specified.
Resource	**Completely**	**Completely**	No construct/feature specified.
Resource Requirements for a Task	**partially**	**Completely**	Resource type and amount can be set in process definition, and be used in process simulation.
Simple Groupings	**Completely**	**Completely**	No construct/feature specified.
Simple Resource Capability/Characteristics	**Completely**	**Completely**	No construct/feature specified.
Simple Sequences	**Completely**	**Completely**	No construct/feature specified.
Simple Task Representation and Characteristics	**Completely**	**Completely**	No construct/feature specified.
Task Duration	**Cannot**	**Completely**	Activity duration is an attribute of activities, which can be set in activity specification in process definition, and be used in process simulation
Task Executor	**Partially**	**Completely**	One or more different roles can be assigned to an activity. The number of persons in the role should be specified to indicate how many people would execute the activity.
Extensibility	**Cannot**	**Cannot**	No construct/feature specified.
Resource Allocation / deallocation for one or many tasks	**Cannot**	**Completely**	Explicitly support resource allocation for one or more activities.
Simple Precedence	**Partially**	**Completely**	directed graph approach
Composition / Decomposition	**Partially**	**Partially**	No construct/feature specified.
Incompleteness / Vagueness	**Cannot**	**Completely**	VPML provides completeness check to ensure a process is completed or not.
Alternative Task	**Completely**	**Completely**	No construct/feature specified.
Associated Illustrations and	**Completely**	**Completely**	No construct/feature specified.

[1] An earlier version of VPML was used in the analysis that is described in this paper. Just prior to publication, ISSI, the developers of VPML and related tools, contributed a second analysis of VPML based on a more recent version. This additional analysis is included here for completeness. This analysis shows the changes made and comments where there are differences. No approval or endorsement of any commercial product in this paper by the National Institute of Standards and Technology is intended or implied.

Requirements	**VPML Old**	**VPML New**	**Comments**
Drawings			
Complex Groups of Tasks	**Cannot**	**Completely**	Via composite activities and logical connectors.
Complex Resource Characteristics	**Completely**	**Completely**	No construct/feature specified.
Complex Sequences	**Partially**	**Partially**	No construct/feature specified.
Complex Task Representation and Parameters	**Completely**	**Completely**	No construct/feature specified.
Concurrent Tasks	**Completely**	**Completely**	No construct/feature specified.
Conditional Tasks	**Completely**	**Completely**	No construct/feature specified.
Confidence Levels	**Cannot**	**Cannot**	No construct/feature specified.
Constraints	**Partially**	**Partially**	No construct/feature specified.
Multiple Duration(s)	**Cannot**	**Completely**	Duration is an attribute of an activity, there are six kinds of distributions are available for users to describe task duration.
Date(s) and Time(s)	**Completely**	**Completely**	No construct/feature specified.
Implicit/Explicit Resource Association	**Cannot**	**Not Sure**	One or more resources can be assigned to one or more manual activities. The amount for each resource can also be specified.
Iterative Loops	**Completely**	**Completely**	No construct/feature specified.
Manual vs. Automated Tasks	**Completely**	**Completely**	No construct/feature specified.
Manufacturing Product Quantity	**Cannot**	**Completely**	The number of products that will be produced can be calculated in process simulation.
Material Constraints	**Cannot**	**Cannot**	No construct/feature specified.
Parallel Tasks	**Completely**	**Completely**	No construct/feature specified.
Parameters and Variables	**Not sure**	**Partially**	There are many parameters can be filled in for activities, products, resources, etc. in process definition. The process will be simulated based on these parameters
Pre and Post-Processing Constraints	**Cannot**	**Not Sure**	The basic pre-condition of an activity is required resource and input product available; The basic past condition of an activity is output product produced and takes the amount of duration time.
Queues, Stacks, Lists	**Partially**	**Completely**	Each data flow between an activity and leaf product has a queue to indicate how many instances in the product at simulation time.
Resource Categorization and Grouping	**Partially**	**Partially**	No construct/feature specified.
Resource Location	**Completely**	**Completely**	No construct/feature specified.
Resource/Task Combined Characteristics	**Cannot**	**Cannot**	No construct/feature specified.
Serial Tasks	**Completely**	**Partially**	Through timer and timer connection.
State Existence Constraints	**Cannot**	**Partially**	If an activity does not get enough input product, it in pending state; If it gets get enough input and wait for resource, it is in ready state; An activity can be active, it must have enough resource and input product.
State Representations	**Cannot**	**Partially**	An activity has following states at least: Pending,

Requirements	VPML Old	VPML New	Comments
			Ready and Active.
Temporal Constraints	**Completely**	**Partially**	Using timer and timer connection.
Uncertainty / Variability / Tolerance	**Cannot**	**Partially**	Output-or connector is introduced in current VPML, which can be connected with activity and product. The probability can be set for each output branch of the connector. Which product will be produced depends on the execution of process simulation.
Ability to Insert or Attach a Highlight (milestones)	**Completely**	**Completely**	No construct/feature specified.
Complex Precedence	**Cannot**	**Not Sure**	No construct/feature specified.
Convey the Ancestry or Class of a Task	**Cannot**	**Cannot**	No construct/feature specified.
Deadline Management	**Cannot**	**Cannot**	No construct/feature specified.
Dispatching	**Cannot**	**Cannot**	No construct/feature specified.
Eligible Resources	**Partially**	**Not Sure**	No construct/feature specified.
Exception Handing and Recovery	**Cannot**	**Cannot**	No construct/feature specified.
Information Exchange Between Tasks	**Completely**	**Completely**	No construct/feature specified.
Mathematical and Logical Operations	**Not sure**	**Partially**	VPML has input-or, input-and, output-or, output-and to represent the logical connection.
Support for Task/Process Templates	**Cannot**	**Partially**	User can build template in one process, and copy/paste to another process.
Support for Simultaneously Maintained Associations of Multiple Levels of Abstraction	**Partially**	**Not Sure**	No construct/feature specified.
Synchronization of Multiple Parallel Task Sequences	**Completely**	**Completely**	No construct/feature specified.
Business Practices, Rules, Constraints		**Partially**	Timers can be connected to activities. While batch activities include by-time and/or by-amount controls
Configuration Management Information and Processes		**Partially**	Resource types can be defined and assigned.
Customer-driven Process Specification and Constraints		**Completely**	Stream-like source products can be defined.
Priority Attributes		**Completely**	Each activity has a priority attribute.
Representation of the Origin of Task(s)		**Partially**	Can be described in activity's specs.
Analysis Characteristics		**Completely**	ProSimulator provides simulation and analysis capability.
Critical Task		**Completely**	At modeling time, the priority attribute can be used to describe activity's significance; while at simulation

Requirements	VPML Old	VPML New	Comments
			time, the statistic report shows the data such as costs, waiting time, running times and so on.
Predictive and Time-dependent Resource Availability		**Partially**	Timer, activity and resource types can be used.
Prescriptive Task Behavior		**Partially**	Common activities have an attribute describing the estimated procedure.
Task/Process Performance Measurement		**Completely**	ProSimulator can be used to simulate a process model.
Dynamic Model Modification		**Cannot**	No construct/feature specified.
Optimization		**Partially**	Optimization can be done manually according to the simulation analysis.
Resource/System/Process Monitoring and Feedback		**Partially**	ProSimulator provides animation capability.
Support for Validation of the Entire Process Plan		**Completely**	via simulation
Tracking of Changes in the System		**Cannot**	No construct/feature specified.
Resource Amount and Availability		**Completely**	Resource planning in ProBuilder
Resource Interruptions		**Completely**	Resource can be deprived when needed by a higher priority activity.
Event Signaling and Notification		**Partially**	Timer event can be triggered to control the input of an activity. Activities can be connected by dataflow and/or reference controls via products
Resource Behavior During Processing		**Cannot**	No construct/feature specified.
Track In-progress Goods		**Cannot**	No construct/feature specified.
Task/Process Purpose		**Partially**	Each activity has a text attribute for its description of purpose.
Characteristics of Groups of Resources		**Completely**	Resource types can be defined from categories such as Role, Machine Type, Location Type and Tool Type.
Implicit Task Association		**Partially**	via connections like dataflows and references.
Parallelism		**Completely**	Activity cloning happens whenever enough resources are available.
Stochastic Properties		**Partially**	Different distributions such as constant, normal, etc. can be defined in stream-like source products. Output connector can be used to conduct activity's output to different out-going paths.
Uncertainty of Sequences		**Completely**	A given seed can be used to create a stream for each random variable.
Simulation - queue entry and exit rates		**Completely**	Available from simulation reports.
Workflow - manual vs. automatable tasks		**Completely**	Manual activities can be accomplished by human roles, while automatic activities can be performed by

Requirements	**VPML Old**	**VPML New**	**Comments**
			machines and tools.
Workflow - invoked tool capability		**Completely**	Tool types can be connected to any product.
Workflow - support specifications of task structure (control flow)		**Completely**	Composition and decomposition of activities.
Project Management - work breakdown ids		**Completely**	Each activity can be numbered by a work breakdown.