International Journal of Production Research

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

# Combining RFID with ontologies to create smart objects

Michael Grüninger [a], Steven Shapiro [b], Mark S. Fox [a] & Harald Weppner [c]

[a] Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada

[b] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada

[c] SAP Research, SAP Labs LLC, Palo Alto, CA, USA

Available online: 18 Mar 2010

To cite this article: Michael Grüninger, Steven Shapiro, Mark S. Fox & Harald Weppner (2010): Combining RFID with ontologies to create smart objects, International Journal of Production Research, 48:9, 2633-2654

To link to this article: http://dx.doi.org/10.1080/00207540903564975

*International Journal of Production Research*
Vol. 48, No. 9, 1 May 2010, 2633–2654

Taylor & Francis
Taylor & Francis Group



# RESEARCH ARTICLE

## Combining RFID with ontologies to create smart objects

Michael Grüninger[a]*, Steven Shapiro[b], Mark S. Fox[a] and Harald Weppner[c]

[a]*Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada;* [b]*Department of Computer Science, University of Toronto, Toronto, Ontario, Canada;* [c]*SAP Research, SAP Labs LLC, Palo Alto, CA, USA*

(*Revision received November 2009*)

Radio frequency identification (RFID) technology has long been known for its ability to uniquely identify objects. Recent years have witnessed a significant increase in storage capacity on the tag, which is giving rise to a new set of application scenarios. As the tag itself can carry relevant context information, processes can be managed locally rather than relying on a centralised system infrastructure. This in turn results in a massive interoperability challenge. We propose to solve this problem by combining RFID technology with ontologies to create *smart objects* in the context of manufacturing process control. The idea is to store information originating from an SAP ERP system using the PSL Ontology (ISO 18629) for representing processes and time directly on the RFID tags. As an item flows through a manufacturing process, information about the item can be stored on its tag. This information, along with the PSL axioms, can be used to make inferences about the manufacturing process and the item in particular. In this paper, we discuss our formalisation of an ontology for the SAP data model and show an example of translating data from an SAP ERP system into PSL axioms, and answering queries about a manufacturing process.

**Keywords:** RFID; smart objects; manufacturing; ontologies; process specification language; automated reasoning

### 1. Introduction

In today's global economy, manufacturing enterprises must employ increasingly effective and efficient information systems. Such systems should result in the seamless integration of manufacturing applications and exchange of manufacturing process information between applications within and across enterprise boundaries. A new approach to achieve this integration has been the notion of proactive computing, in which information systems act in anticipation of future problems, needs, or changes of the user.

To be proactive, a computer system must understand the user's context and how it changes over time. Within manufacturing enterprises, this can be facilitated by the recording of process constraints associated with a particular resource as it passes through the set of activities performed within the supply chain.

In these enterprises, RFID (radio frequency identification) technology is used to capture, retain, and transmit data about an object. An RFID tag is a device that can serve as a means of identifying objects using radio transmissions; data can be written and retrieved using readers that are in close proximity to the tag. Some companies have used active or

*Corresponding author. Email: gruninger@mie.utoronto.ca

ISSN 0020–7543 print/ISSN 1366–588X online

DOI: 10.1080/00207540903564975
http://www.informaworld.com

passive RFID tags to record a detailed history of the steps and conditions as each object or batch of objects goes through the manufacturing process. While RFID tags originally held only 32–128 bits of information, the current generation of passive tags already hold 32 kilobytes while active tags hold up to 4 megabytes of data or more. The memory size is expected to increase in the future. Some companies write manufacturing instructions to the tag, which can then be recalled at each step of the manufacturing process. In all of these cases, the idea is to create smart objects by deploying RFID technologies throughout the supply network. By analysing data and events in real time, objects become self-directing, processes become self-managing, and the supply chain becomes self-correcting.

Nevertheless, current deployments of RFID have not fully achieved this vision. One problem has been that the information being encoded on the tags has often been represented in an informal and *ad hoc* fashion. As a result, it has been difficult to provide any automated dynamic decision support. In particular, truly proactive systems need to be able to predict possible future behaviours. For example, if we can identify the subsequent activities that may possibly occur so that the complete set of constraints will be satisfied, then the set of predictions can be used to dynamically determine the routing of the resource and prevent the occurrences of any activities that are inconsistent with the set of process constraints.

Even in cases where the appropriate information is encoded on the RFID tags, an approach known as 'data-on-tag', enterprises must provide access to the information by people, software applications, and business processes, anytime and anywhere. Unfortunately, different applications and databases ascribe disparate meanings to the same terms or use distinct terms to convey the same meaning. This clash over the meaning of the terms prevents the seamless exchange of information among the applications.

The objective of this paper is to specify a set of theories in first-order logic which can support the design and implementation of smart objects that are able to predict their possible future behaviours by deduction from their history and background manufacturing process knowledge. The emphasis is on the following issues:

- specifying the manufacturing process knowledge that is to be encoded on RFID tags;
- performing automated reasoning with the knowledge encoded on RFID tags; and
- integrating this knowledge with the enterprise's manufacturing software systems (e.g. process modelling, process planning, scheduling, production planners, and workflow management systems).

The representation of the process knowledge must be generic and reusable, so that it can be used in multiple manufacturing scenarios. It needs to be expressive enough to capture process plans and their properties, as well as the potential queries related to future behaviours. Finally, it must be powerful enough to deduce the solutions to queries from the axioms of the theory alone, rather than use extralogical mechanisms, since the process knowledge is specified declaratively on the RFID tags independently of domain-specific algorithms.

## 2. Related work

Diekmann *et al*. (2007) compare two approaches on how object-related data and processes are managed. In the 'data-on-network' approach, only identifying information is stored on

the RFID tag while related data and processes are managed centrally. In the 'data-on-tag' approach, information is held on the tag, at least initially. Advantages and disadvantages of each approach are discussed and it is concluded that they should be seen as complementary. Melski *et al.* (2007) further explore the specific benefits of 'data-on-tag' in comparison to 'data-on-network' applications. They identify a lack of standards, which are required to allow reading and writing data-rich tags across company boundaries. Through case studies they identify four general functions that 'data-on-tag' applications specifically provide: informational, documentational, temporary storage and a control function. The control function results in increased flexibility and adaptability, a clear advantage of decentralised over centralised process controlling. Guenther and Tribowski (2009) identify the future interoperability and integration problems for 'data-on-tag' applications whenever syntax and semantic are not agreed upon by collaborating enterprises. They highlight the lack of a standard for storing object-related data on RFID tags and raise this as a significant issue for a wider spread adoption. ISO standard 13584 is explored to address the informational function, i.e. deliver additional information on the object. Ruta *et al.* (2007) have shown the possibilities of organising the tag memory and enhancing the current standard data exchange protocol in a compatible fashion.

## 3. Manufacturing process scenario

### 3.1 *Dynamic process routing*

The storyboard in Figure 1 motivates queries related to dynamic self-routing of objects through the various process plans within the set of manufacturing processes. Each product is associated with a set of process plans, which are partially ordered sequences of manufacturing processes. In more general scenarios, such process plans may also be

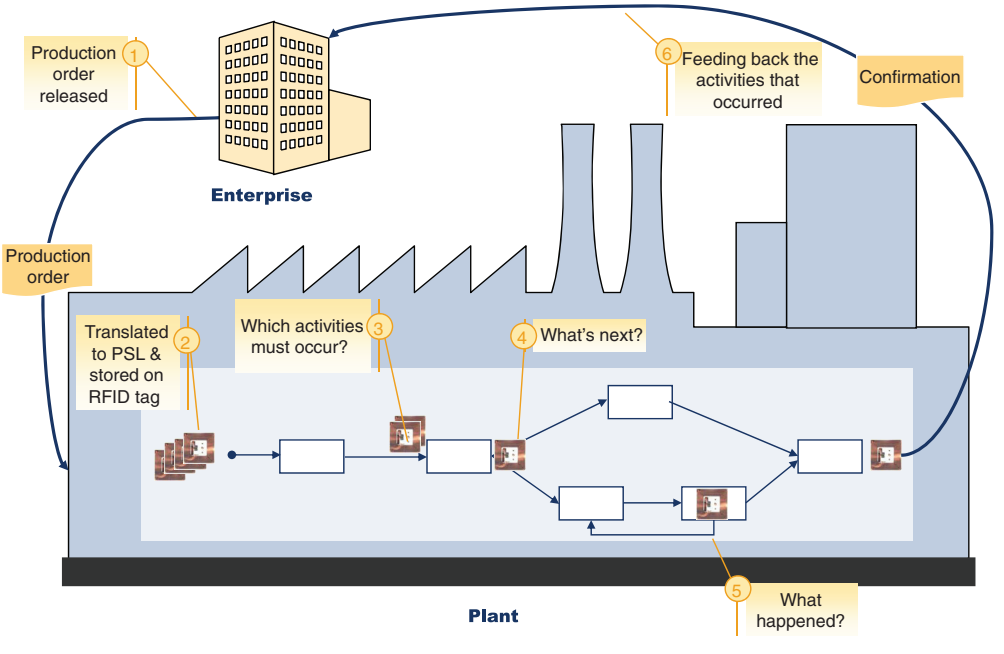



Figure 1. Manufacturing process scenario.

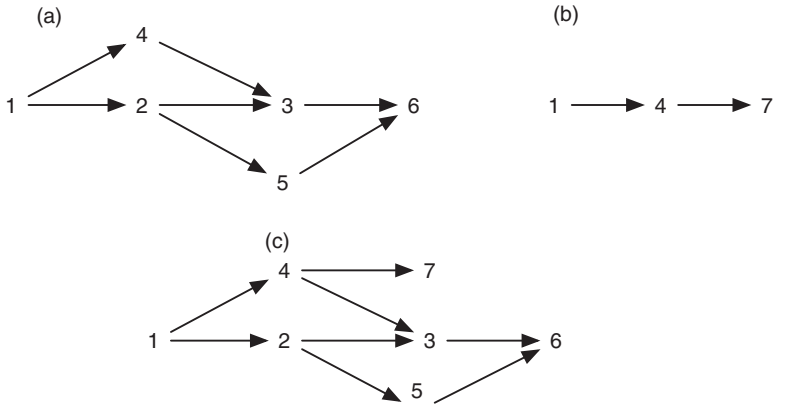



Figure 2. Process plans.

non-deterministic (that is, involve different choices of sequences of manufacturing processes). Objects 'flow' through the sequence of processes. At any point in a process plan, there are multiple activities that can possibly occur next.

Furthermore, different process plans may have manufacturing processes in common, so that an object may participate in an activity that is part of multiple process plans. In Figure 2(a) and (b), we see two different process plans, and their combination in Figure 2(c).

Each manufacturing process imposes constraints on the objects that participate in the process. State constraints require that the objects satisfy specific properties before the process can occur. If the object does not satisfy the state constraints, then additional activities may occur so that the necessary state properties for the object can be achieved.

The state constraints on manufacturing processes also influence possible routings of objects – the next activity that occurs may depend on the properties of the object. Temporal constraints require that different processes occur before specified deadlines. In such cases, an object may be routed to different manufacturing processes.

The history of an object is the set of activity occurrences in which an object participates as well as the order in which the activities occurred, and the times at which the activities occurred. All of the dynamic process routing queries are answered relative to the set of process plans that could possibly be occurring, which is the set of process plans whose initial sequences of subactivity occurrences are consistent with the history of the object.

### 3.2 *Queries*

All of the dynamic process routing queries focus on the choices that the object has at different points in a process. Since no decisions need to be made where there are no choices, we need to first identify the choicepoints in a process plan, that is, the set of subactivities in the process plan after which we have a choice over what activity occurs next. The object then needs to determine the nature of the choices, and the impact of the choice on future decisions. The following informal queries capture these intuitions.

**Query 3.1:** Is the object currently at a choicepoint?

**Query 3.2:** Is this a choice of which activities occur, or a choice of the ordering in which to perform activities?

**Query 3.3:** Given that the object is at a choicepoint, determine which activities can possibly occur next.

**Query 3.4:** Given that the object is at a choicepoint, can the choice to perform some activity be postponed until later in the process? If not, then it must occur next (even if other activities can possibly occur next).

**Query 3.5:** Which activities must not occur next, that is, which activities must occur later?

**Query 3.6:** Which activities must occur in the same ordering, for any occurrence of the process?

**Query 3.7:** Is there a point in the process after which the same activities occur? If such a point exists, then all of the remaining choices will simply be over the order in which the activities occur.

**Query 3.8:** Is there a point in the process after which the order of the activities is the same for all occurrences of the process? After such a point, we cannot make any choices about the order in which the activities occur.

The dynamic process routing queries are also related to process verification. Process plans specify the set of activities that are intended to occur, while the object history records the set of activities that actually occurred. If any activity occurred which should not have occurred or an activity occurrence violated an ordering constraint, then the object history will not correspond to the occurrence of any process plan.

### 3.3 *Inferences*

The dynamic process routing queries are answered using the following sets of sentences, which are written on the RFID tag for the object:

- object history
- all possible process plans
- manufacturing process ontology

In the following sections, we will consider each of these aspects.

## 4. Ontology for manufacturing processes

The key component in the representation of the knowledge written on the RFID tag is the manufacturing process ontology, which specifies the semantics of concepts such as process plans, activity occurrences, and ordering constraints. In this section, we give an overview of the ontology. We begin by considering why ontologies are needed in the first place.

### 4.1 *The need for ontologies*

Ontologies address two major challenges with respect to the sharability and reusability of knowledge. The first challenge is a lack of interoperability among the various applications that the enterprises use. Interoperability is hindered because different applications may use different terminology and representations of the domain. Even when applications use the

same terminology, they often associate different semantics with the terms. This clash over the meaning of the terms prevents the seamless exchange of information among the applications. Typically, point-to-point translation programs are written to enable communication from one specific application to another. However, as the number of applications has increased and the information has become more complex, it has been more difficult for software developers to provide translators between every pair of applications that must cooperate. What is needed is some way of explicitly specifying the terminology of the applications in an unambiguous fashion.

The second problem faced by enterprises today is a lack of reusability. The knowledge bases that capture the domain knowledge of engineering applications are often tailored to specific tasks and projects. When the application is deployed in a different domain, it does not perform as expected, often because assumptions are implicitly made about the concepts in the application, and these assumptions are not generic across domains. For example, machine models are often designed specifically about a particular set of properties about specific machines rather than characterising generic properties of machines, such as reusability, setup activities, and operating conditions.

To address these challenges, various groups within industry, academia, and government have been developing sharable and reusable models known as ontologies. All ontologies consist of a vocabulary along with some specification of the meaning or semantics of the terminology within the vocabulary. In doing so, ontologies support interoperability by providing a common vocabulary with a shared semantics. Rather than develop point-to-point translators for every pair of applications, one simply needs to write one translator between the application's terminology and the common ontology. Similarly, ontologies support reusability by providing a shared understanding of generic concepts that span across multiple projects, tasks and environments.

The various ontologies that have been developed can be distinguished by their degree of formality in the specification of meaning. With informal ontologies, the definitions are expressed loosely in natural language. Semi-formal ontologies provide weak constraints, such as taxonomies, of the terminology. Formal ontologies use languages based on mathematical logic. Informal and semi-formal ontologies can serve as a framework for shared understanding among people, but they are often insufficient to support interoperability, since any ambiguity can lead to inconsistent interpretations and hence hinder integration. Thus, simply sharing terminology is insufficient to support interoperability – the applications must share their semantics as well.

### 4.2 *Process specification language*

The Process Specification Language (PSL) (Schlenoff *et al*. 1999, Grüninger and Menzel 2003, Grüninger 2004, Bock and Grüninger 2005) has been designed to facilitate correct and complete exchange of process information.[1] The primary purpose of PSL is to enable the semantic interoperability of manufacturing process descriptions between manufacturing engineering and business software applications such as process planning, scheduling, workflow, and project management. Additional applications of PSL include business process design and analysis, the implementation of business processes as web services, and enterprise modelling.

The PSL Ontology is a modular set of theories in the language of first-order logic. All core theories within the ontology are consistent extensions of a theory referred to as

PSL-Core, which introduces the basic ontological commitment to a domain of activities, activity occurrences, timepoints, and objects that participate in activities. Additional core theories capture the basic intuitions for the composition of activities, and the relationship between the occurrence of a complex activity and occurrences of its subactivities.

In order to formally specify a broad variety of properties and constraints on complex activities, we need to explicitly describe and quantify over complex activities and their occurrences. Within the PSL Ontology, complex activities and occurrences of activities are elements of the domain and the *occurrence_of* relation is used to capture the relationship between different occurrences of the same activity.

A second requirement for formalising queries is to specify the composition of activities and occurrences. The PSL Ontology uses the *subactivity* relation to capture the basic intuitions for the composition of activities. Complex activities are composed of sets of *atomic* activities, which in turn are either *primitive* (i.e. they have no proper subactivities) or they are concurrent combinations of primitive activities.

Corresponding to the composition relation over activities, *subactivity_occurrence* is the composition relation over activity occurrences. Given an occurrence of a complex activity, subactivity occurrences are occurrences of subactivities of the complex activity.

Finally, we need some way to specify ordering constraints over the subactivity occurrences of a complex activity. The PSL Ontology uses the *min_precedes*($s_1$, $s_2$, $a$) relation to denote that subactivity occurrence $s_1$ precedes the subactivity occurrence $s_2$ in occurrences of the complex activity $a$. Note that there could be other subactivity occurrences between $s_1$ and $s_2$. We use *next_subocc*($s_1$, $s_2$, $a$) to denote that $s_2$ is the next subactivity occurrence after $s_1$ in occurrences of the complex activity $a$.

A fundamental structure within the models of the axioms of the PSL Ontology is the *occurrence tree*, whose branches are equivalent to all discrete sequences of occurrences of atomic activities in the domain. Elements of the occurrence tree are referred to as *arboreal* occurrences.

Although occurrence trees characterise all sequences of activity occurrences, not all of these sequences will intuitively be physically possible within a given domain. We therefore consider the subtree of the occurrence tree that consists only of possible sequences of activity occurrences, which we refer to as the *legal occurrence tree*. The *legal*($o$) relation specifies that the atomic activity occurrence $o$ is an element of the legal occurrence tree.

The basic structure that characterises occurrences of complex activities within models of the ontology is the *activity tree*, which is a subtree of the legal occurrence tree that consists of all possible sequences of atomic subactivity occurrences of an activity; the relation *root*($s$, $a$) denotes that the subactivity occurrence $s$ is the root of an activity tree for $a$. Elements of the tree are ordered by the *min_precedes* relation; each branch of an activity tree is a linearly ordered set of occurrences of subactivities of the complex activity. In addition, there is a one-to-one correspondence between occurrences of complex activities and branches of the associated activity trees.

In a sense, an activity tree is a microcosm of the occurrence tree, in which we consider all of the ways in which the world unfolds in the context of an occurrence of the complex activity. Different subactivities may occur on different branches of the activity tree – different occurrences of an activity may have different subactivity occurrences or different orderings on the same subactivity occurrences (see the examples in Figure 3). This distinction plays a key role in the specification of the reasoning problems in this paper.

### 4.3 *Ontology of process plans*

There have been many proposals for the representation of process plans (Cho and Wysk 1995, Sormaz and Khoshnevis 2003), particularly with respect to structures such as AND/OR graphs. Nevertheless, such approaches are not directly suitable for the task at hand, since the intended semantics of the process plans is implicit in the algorithms used to interpret the AND/OR graphs. As a result, automated reasoning through theorem provers with declarative specifications of processes is inhibited.

The ontology of process plans discussed in this paper is a theory within the PSL Ontology containing axioms for classes of activities whose activity trees are defined with respect to partial orderings over their subactivity occurrences. In particular, the set of axioms in the PSL Ontology captures all of the intuitions of the graph-theoretic representation of process plans.

In this paper we also introduce two new consistent extensions of the PSL Ontology which are used to capture the notion of process plan composition and intuitions specific to the issue of choicepoints within a process plan.



#### 4.3.1 *Process plans*

The fundamental intuition in the ontology of process plans is the notion of a partial ordering that is embedded within the activity trees of a complex activity. Different classes of process plans are defined by characterising the relationship between the partial ordering and the set of possible occurrences of a process plan.

Three new relations are introduced to specify the relationship between the partial ordering (referred to as the *subactivity occurrence ordering*) and the activity tree. The relation $soo(s, a)$ denotes that the activity occurrence $s$ is an element of the subactivity occurrence ordering for the activity $a$. The relation $soo\_precedes(s_1, s_2, a)$ captures the ordering over the elements. For example, the partial ordering in Figure 3(a) can be defined by[2]

$$\begin{aligned} soo(s_1^1, a) \wedge soo(s_2^2, a), soo(s_3^3, a) \wedge soo(s_4^4, a) \wedge soo(s_5^5, a) \\ \wedge soo\_precedes(s_1^1, s_2^2, a) \wedge soo\_precedes(s_1^1, s_3^3, a) \wedge soo\_precedes(s_2^2, s_4^4, a) \\ \wedge soo\_precedes(s_2^2, s_5^5, a) \wedge soo\_precedes(s_3^3, s_4^4, a) \wedge soo\_precedes(s_3^3, s_5^5, a). \end{aligned}$$

The activity trees in Figure 3(b), (c), (d) and (e) can all be mapped to the partial ordering in Figure 3(a).

The relation $mono(s_1, s_2, a)$ indicates that $s_1$ and $s_2$ are occurrences of the same subactivity on different branches of the activity tree for $a$. In the activity tree in Figure 3(c) we have

$$\begin{aligned} mono(s_2^2, s_7^2, a) \wedge mono(s_3^3, s_6^3, a) \wedge mono(s_4^4, s_9^4, a) \wedge mono(s_4^4, s_{10}^4, a) \wedge mono(s_4^4, s_{13}^4, a) \\ \wedge mono(s_5^5, s_8^5, a) \wedge mono(s_5^5, s_{11}^5, a) \wedge mono(s_5^5, s_{12}^5, a). \end{aligned}$$

There are three basic classes of process plans, each of which imposes different occurrence constraints on the elements of the partial ordering:

- strong poset activities
- choice poset activities
- complex poset activities

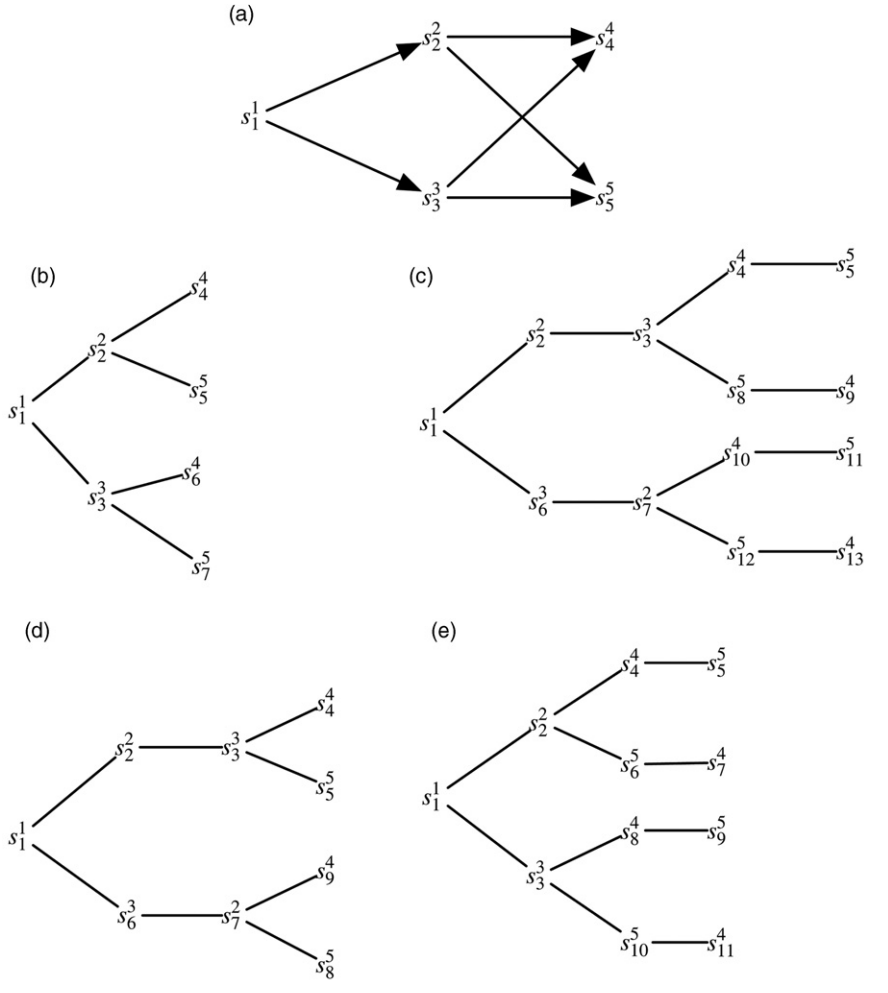



Figure 3. Classes of poset activities.

For strong poset activities, there is a one-to-one correspondence between branches of the activity tree and the linear extensions of the partial ordering. The partial ordering in Figure 3(a) has four linear extensions:

$$s_1^1 s_2^2 s_3^3 s_4^4 s_5^5, \quad s_1^1 s_2^2 s_3^3 s_5^5 s_4^4, \quad s_1^1 s_3^3 s_2^2 s_4^4 s_5^5, \quad s_1^1 s_3^3 s_2^2 s_5^5 s_4^4,$$

which correspond to the branches of the strong poset activity tree in Figure 3(c).

For choice poset activities, there is a one-to-one correspondence between branches of the activity tree and the maximal chains in the partial ordering. The partial ordering in Figure 3(a) has four maximal chains:

$$s_1^1 s_2^2 s_4^4, \quad s_1^1 s_2^2 s_5^5, \quad s_1^1 s_3^3 s_4^4, \quad s_1^1 s_3^3 s_5^5,$$

which correspond to the branches of the choice poset activity tree in Figure 3(b).

A complex poset activity is the union of the strong poset activities corresponding to a set of linear extensions for *suborderings* of the partial ordering.

For example, given the partial ordering in Figure 3(a), the activity tree in Figure 3(b) is a choice poset activity, the activity tree in Figure 3(c) is a strong poset activity, and the activity trees in Figure 3(d) and (e) are complex poset activities.

For a strong or choice poset activity, there exists a unique activity tree corresponding to the partially ordered set. For complex poset activities, the relationship between the activity tree and the partial ordering is a little more complicated, since there can be multiple possible activity trees corresponding to the same partial ordering. The key is to consider the role of the incomparable elements in the partial ordering. In strong poset activities, incomparable elements in the partial ordering correspond to subactivities that occur in any order. In choice poset activities, incomparable elements in the partial ordering correspond to subactivities that never occur on the same branch of the activity tree.

Two relations are introduced in the PSL Ontology that allow one to specify whether or not two incomparable elements in the partial ordering correspond to subactivities that occur in any order or whether they are subactivities that never occur on the same branch of the activity tree. The *same_bag* relation is used to specify suborderings whose linear extensions are contained in branches of the activity tree. The *alternate* relation is used to specify the sets of subactivity occurrences that can never be elements of the same branch of the activity tree; this is equivalent to specifying the suborderings whose chains are contained in branches of the activity tree.

For example, consider the partial ordering in Figure 3(a), in which $s_2^2$ and $s_3^3$ are incomparable, as are $s_4^4$ and $s_5^5$. The choice poset activity tree in Figure 3(b) can be described by

$$alternate(s_2^2, s_3^3, a) \wedge alternate(s_4^4, s_5^5, a).$$

Note that, for choice posets, all incomparable elements in the ordering are *alternate* with respect to each other. On the other hand, for strong posets, all incomparable elements in the ordering are in the *same_bag*, so that the strong poset activity tree in Figure 3(c) is described by

$$same\_bag(s_2^2, s_3^3, a) \wedge same\_bag(s_4^4, s_5^5, a).$$

For complex posets, we use both relations; for example, the complex poset activity tree in Figure 3(d) is specified by

$$same\_bag(s_2^2, s_3^3, a) \wedge alternate(s_4^4, s_5^5, a),$$

because every linear extension of $s_2^2$ and $s_3^3$ is contained in the branches and every chain in the subordering defined by $s_4^4$ and $s_5^5$ is contained in the branches of the activity tree.

#### 4.3.2 *Composition of process plans*

In order to represent the problem of identifying the set of process plans that could possibly be occurring, given the object history, we will consider the complex activity that consists of all possible process plans as subactivities. This approach allows us to explicitly represent the set of all possible activity occurrences; choices are made with respect to this set.

Since an activity tree represents all possible sequences of subactivity occurrences corresponding to occurrences of the activity, the composition of a set of process plans is

A maximum activity is one that contains all process plans as subactivities

$$(\forall a)\ maximum(a) \equiv ((\forall a_1)\ activity(a_1) \supset subactivity(a_1, a)). \tag{10}$$

The activity $a_1$ is a maximal subactivity of the activity $a_2$ if there is no other activity that is a subactivity of $a_2$ that is not already a subactivity of $a_1$

$$(\forall a_1, a_2)\ maximal\_sub(a_1, a_2) \equiv subactivity(a_1, a_2)$$
$$\wedge ((\forall a_3)\ subactivity(a_1, a_3) \wedge subactivity(a_3, a_2) \supset ((a_3 = a_1) \vee (a_3 = a_2))). \tag{11}$$

There exists a maximum activity

$$(\exists a)\ maximum(a). \tag{12}$$

All maximal subactivities of the maximum activity are process plans

$$(\forall a_1, a_2)\ maximum(a_2) \wedge maximal\_sub(a_1, a_2)$$
$$\supset ((strong\_poset(a_1) \vee choice\_poset(a_1) \vee complex\_poset(a_1))). \tag{13}$$

The maximum activity is the composition of the process plans, that is, the activity trees for the maximum activity is the union of the activity trees for the process plans that are subactivities

$$(\forall a)\ maximum(a) \supset$$
$$((\forall s_1, s_2)\ min\_precedes(s_1, s_2, a) \equiv (\exists a_1)\ maximal\_sub(a_1, a) \wedge min\_precedes(s_1, s_2, a_1). \tag{14}$$

Figure 4. $T_{\max}$: axioms for maximum activities.

equivalent to the union of the activity trees for those process plans. The activity that contains all possible process plans as subactivities is the *maximum activity*, and it can be proven that such an activity is unique for a given set of process plans. Finally, we want to restrict the composition to the set of process plans, rather than consider the composition of all possible primitive activities; on the other hand, the transitivity of the *subactivity* relation means that the primitive activities are also subactivities of the maximum activity. For this reason, we introduce the notion of a *maximal subactivity* to allow us to distinguish between subactivities of the maximum activity that are process plans and subactivities that are primitive activities.

Figure 4 shows the axioms (called $T_{\max}$) that extend the PSL Ontology to specify the composition of process plans.

#### 4.3.3 *Properties of activity trees*

Finally, we need two additional relations that characterise structural properties of activity trees that are relevant to the intuition of making a choice at some point of a process plan. For example, consider the activity tree in Figure 3(c). After the element $s_1^1$, either activity $a_2$ or $a_3$ may occur next, although both subactivities occur on every branch of the activity tree; the only choice in this case is selecting the order in which to perform $a_2$ and $a_3$. On the other hand, after the element $s_3^3$, a choice must be made to either perform $a_4$ or $a_5$ next.

The subactivity occurrence $s$ is a choicepoint in an activity tree for the activity $a$ if it has multiple distinct successors in the activity tree

$$(\forall s, a)\ choicepoint(s, a) \equiv$$
$$((\forall s_1)\ mono(s, s_1, a) \supset (\exists s_2, s_3)\ next\_subocc(s_1, s_2, a)$$
$$\wedge\ next\_subocc(s_1, s_3, a) \wedge (s_2 \neq s_3)). \quad (15)$$

The subactivity occurrence $s$ is a weak choicepoint in an activity tree for the activity $a$ if all of its successors in the activity tree occur on each branch

$$(\forall s, a)\ weak\_choicepoint(s, a) \equiv$$
$$(\forall s_1, s_2)\ next\_subocc(s, s_1, a) \wedge next\_subocc(s, s_2, a)$$
$$\supset (\exists s_3)\ next\_subocc(s_1, s_3, a) \wedge mono(s_2, s_3, a). \quad (16)$$

Figure 5. $T_{\text{choicepoint}}$: Axioms for properties of activity trees.

The *choicepoint* relation captures the idea that there are multiple possible subactivities that may occur next. In Figure 3(c), $s_3^3$ and $s_7^2$ are both choicepoints. The *weak_choicepoint* relation captures the intuition that the choice involved is with respect to determining the ordering in which to perform the following subactivities. In Figure 3(c), $s_1^1$ is a weak choicepoint. The axioms defining these relations (called $T_{\text{choicepoint}}$) are given in Figure 5; these axioms are a consistent extension of the PSL Ontology.

### 4.4 *Formalisation of process constraints*

In addition to the ontology of process plans, we also need to specify classes of sentences that capture the constraints of the particular domain; intuitively, this corresponds to the knowledge about instances of classes of objects and processes.

#### 4.4.1 *Object history*

The history of an object is the set of activity occurrences in which an object participates as well as the order in which the activities occurred, and the times at which the activities occurred. This is formalised as the following class of sentences, which is parameterised by an object, $K$.

**Definition 4.1:** $\Sigma_{\text{history}}(K)$ is a sentence of the form

$$(\exists o_1, \ldots, o_n, o, a) occurrence\_of(o_1, A_1) \wedge \ldots \wedge occurrence\_of(o_n, A_m)$$
$$\wedge\ actual(o_1) \wedge \ldots \wedge actual(o_n)$$
$$\wedge\ participates\_in(K, o_1, T_1) \wedge \ldots \wedge participates\_in(K, o_n, T_n)$$
$$\wedge\ before(T_1, T_2) \wedge \ldots \wedge before(T_{n-1}, T_n) \wedge root(o_1, a)$$
$$\wedge\ min\_precedes(o_1, o_2, a) \wedge \ldots \wedge min\_precedes(o_{n-1}, o_n, a)$$
$$\wedge\ maximum(a) \wedge occurrence\_of(o, a).$$

The object history is rewritten on the object's RFID tag after each activity occurrence. After each activity occurs, this sentence is augmented with the corresponding information

about which activity occurred and the time at which it occurred.[3] Ordering information is captured by the *before* and *min_precedes* literals.

The object history sentence also asserts the existence of an occurrence of a maximum activity that contains all activity occurrences in the object history for the object $K$ as subactivity occurrences. This additional condition allows us to characterise the dynamic process routing queries with respect to properties of the activity trees of the maximum activity.

#### 4.4.2 *Process descriptions*

A process description is an axiom schema that can be used to specify the occurrence and ordering constraints on an instance of a process plan in some class. For example, a strong poset activity $a$ has a process description of the form

$$\begin{gathered}\forall o.occurrence\_of(o,a) \supset \\ \exists s_1,\ldots,s_n.occurrence\_of(s_1,a_1) \wedge \ldots \wedge occurrence\_of(s_n,a_n) \\ \wedge subactivity\_occurrence(s_1,o) \wedge \ldots \wedge subactivity\_occurrence(s_n,o) \\ \wedge min\_precedes(s_{i_1},s_{j_1},a) \wedge \ldots \wedge min\_precedes(s_{i_k},s_{j_k},a),\end{gathered} \tag{1}$$

where $i_1,\ldots,i_k, j_1,\ldots,j_k \in \{1,\ldots,n\}$.

This schema says that for any occurrence $o$ of activity $a$ there is a set of occurrences $s_1,\ldots,s_n$ that are occurrences of activities $a_1\ldots a_n$, respectively, and they are subactivity occurrences of $o$. The *min_precedes* literals (partially) constrain the order in which the subactivity occurrences occur. This is an axiom schema because $a, a_1,\ldots,a_n$ are unbound. Here $a$ is the activity corresponding to the strong poset activity, and $a_1,\ldots,a_n$ are atomic subactivities of $a$. In an instantiation of the schema, these unbound variables are replaced with appropriate activity terms to describe a particular strong poset activity. The axioms for the process plans will be denoted by $\Sigma_{\text{pd}}$.

#### 4.4.3 *Antecedents for the reasoning problems*

In summary, every reasoning problem has the following sets of sentences in the antecedent:

$$T_{\text{psl}} \cup T_{\max} \cup T_{\text{choicepoint}} \cup \Sigma_{\text{history}}(K) \cup \Sigma_{\text{pd}}.$$

If this set of sentences is inconsistent, then either an unexpected activity has occurred at some point in the object history, or an activity occurrence in the object history has violated an ordering constraint.

### 4.5 *Formalisation of queries*

The dynamic process routing queries characterise properties of the activity trees of the maximum activity. Using the ontology of manufacturing processes, we can formalise the informal queries from Section 2.2 as the following sentences.

- Is the current subactivity occurrence a choice point?

$$(\exists s,a)maximum(a) \wedge choicepoint(s,a) \wedge occurrence\_of(s,a_1). \tag{2}$$

- Is the current subactivity occurrence a weak choice point?

$$(\exists s, a)maximum(a) \wedge weak\_choicepoint(s, a) \wedge occurrence\_of(s, a_1). \tag{3}$$

- Which subactivities can possibly occur next after an occurrence of $a_1$?

$$\begin{gathered}(\exists a)maximum(a) \\ \wedge ((\forall s_1)occurrence\_of(s_1, a_1) \supset (\exists a_2, s_2)subactivity(a_2, a) \\ \wedge occurrence\_of(s_2, a_2) \wedge next\_subocc(s_1, s_2, a))).\end{gathered} \tag{4}$$

- Does there exist a subactivity that must occur next after an occurrence of $a_1$?

$$\begin{gathered}(\exists a)maximum(a) \\ \wedge ((\forall o, s_1)occurrence\_of(s_1, a_1) \wedge occurrence\_of(o, a) \\ occurrence\_of(s_2, a_2) \wedge subactivity\_occurrence(s_2, o) \\ \wedge next\_subocc(s_1, s_2, a)).\end{gathered} \tag{5}$$

- Which subactivities must occur later after an occurrence of $a_1$?

$$\begin{gathered}(\exists a)maximum(a) \wedge ((\forall o, s_1)occurrence\_of(s_1, a_1) \wedge occurrence\_of(o, a) \\ \wedge subactivity\_occurrence(s_1, o) \supset (\exists a_2, s_2)occurrence\_of(s_2, a_2) \\ \wedge min\_precedes(s_1, s_2, a) \wedge subactivity\_occurrence(s_2, o)).\end{gathered} \tag{6}$$

- Which activities must occur in the same order?

$$\begin{gathered}(\exists a_1, a_2, a)maximum(a) \wedge subactivity(a_1, a) \wedge subactivity(a_2, a) \\ \wedge ((\forall s, s_1, s_2, s_3, s_4)mono(s_1, s_3, a) \wedge mono(s_2, s_4, a) \wedge occurrence\_of(s_1, a_1) \\ \wedge occurrence\_of(s_2, a_2) \wedge occurrence\_of(s_3, a_1) \wedge occurrence\_of(s_4, a_2) \\ \wedge min\_precedes(s, s_3, a) \wedge root(s, a) \wedge min\_precedes(s_1, s_2, a) \\ \wedge min\_precedes(s, s_4, a) \supset \neg min\_precedes(s_4, s_3, a)).\end{gathered} \tag{7}$$

- Does there exist a point in an activity tree $a$ after which the same subactivities occur? (After this point, there will be no choice about which activities occur.)

$$\begin{gathered}(\exists a, a_1, s_1)maximum(a) \wedge subactivity(a_1, a) \wedge occurrence\_of(s_1, a_1) \\ \wedge ((\forall o_1, o_2)occurrence\_of(o_1, a) \wedge occurrence\_of(o_2, a) \\ \wedge subactivity\_occurrence(s_1, o_1) \wedge subactivity\_occurrence(s_1, o_2) \\ \wedge min\_precedes(s_1, s_2, a) \\ \supset (\exists s_3)subactivity\_occurrence(s_3, o_2) \wedge min\_precedes(s_1, s_3, a) \wedge mono(s_2, s_3, a).\end{gathered} \tag{8}$$

- Does there exist a point in each activity tree for $a$ after which the ordering over the subactivities $a_1$ and $a_2$ is the same? (After this point, there will be no choice about

the ordering in which the activities occur.)

$$\begin{aligned}
&(\exists a, a_1)\mathit{maximum}(a) \wedge \mathit{subactivity}(a_1, a) \wedge \mathit{occurrence_of}(s, a_1) \\
&\wedge ((\forall s', s_1, s_2, s_3, s_4)\mathit{root}(s', a) \wedge \mathit{min_precedes}(s', s, a) \wedge \mathit{min_precedes}(s, s_1, a) \\
&\wedge \mathit{min_precedes}(s, s_2, a) \wedge \mathit{mono}(s_1, s_3, a) \wedge \mathit{mono}(s_2, s_4, a) \\
&\wedge \mathit{min_precedes}(s, s_3, a) \wedge \mathit{min_precedes}(s, s_4, a) \wedge \mathit{occurrence_of}(s_1, a_1) \\
&\wedge \mathit{occurrence_of}(s_2, a_2) \wedge \mathit{occurrence_of}(s_3, a_1) \wedge \mathit{occurrence_of}(s_4, a_2) \\
&\supset ((\mathit{min_precedes}(s_1, s_2, a) \equiv \mathit{min_precedes}(s_3, s_4, a))).
\end{aligned} \tag{9}$$



## 5. Semantic integration with SAP ERP

As an example of the application of these ideas, we formalised a fragment of the SAP ERP data model for discrete manufacturing processes in PSL, which we call the *SAP Ontology*. We then wrote a program which translates a production order exported from SAP ERP into the SAP Ontology. Once the production order was represented in this form, we could use an automated theorem prover to answer queries about the production order. With the addition of axioms about the history of the item being produced (which, in this example, is a pump) as it flows through the manufacturing process, we can also use the theorem prover to answer queries about the item as it is being manufactured.

### 5.1 *Formalising the SAP data model*

The central construct in the SAP Ontology is the SAP production order. A production order can contain multiple 'sequences' of operations. The sequences can be of three types: *simple sequence*, *alternate sequence* or *parallel sequence*. A simple sequence is a finite, linearly ordered sequence of operations. Each production order has a main simple sequence called the *standard sequence*. Branching off from the standard sequence can be parallel or alternate sequences. Each parallel or alternate sequence is also a simple sequence, but it also has start and end branch points to indicate the subsequence of the standard sequence that it is parallel to or alternates with. Parallel sequences occur in parallel with the corresponding standard subsequence, but alternate sequences, if they are chosen for execution, will occur instead of the corresponding subsequence of the main sequence. The definitions of the classes of process plans in the SAP Ontology can be found in Figure 6.

There are tags and values associated with a production order, sequence, and operation. For example, each production order has a material number and quantity associated with it. The production order will produce that quantity of the material. There are also tags to specify a name for the material, the sequences, and the operations, as well as various start and end times and dates, e.g. earliest and latest scheduled start and end times, processing times, wait times, etc. An operation can also have components associated with it; these are the components that are consumed as part of the operation.

The full SAP Ontology includes definitions for concepts such as production orders, as well as temporal relations, such as the earliest and latest start and finish times of an operation. Other objects, such as the items being manufactured and their components, are modelled as PSL objects.

An activity is a simple sequence if each of its activity trees has a unique branch

$$(\forall a)\ simple\_sequence(a) \equiv$$
$$((\forall s_1, s_2, s_3)\ min\_precedes(s_1, s_2, a) \wedge min\_precedes(s_1, s_3, a) \supset$$
$$(min\_precedes(s_2, s_3, a) \vee min\_precedes(s_3, s_2, a) \vee (s_2 = s_3)). \quad (17)$$

An activity $a_1$ is a standard sequence for an activity $a_2$ if it is a simple sequence that contains subactivity occurrences that correspond to the choicepoints in activity trees for $a_2$

$$(\forall a_1, a_2)\ standard\_sequence(a_1, a_2) \equiv$$
$$((simple\_sequence(a_1) \wedge subactivity(a_1, a_2)$$
$$\wedge ((\forall s_1)\ choicepoint(s_1, a_2) \supset (\exists s_2)\ soo(s_2, a_1) \wedge mono(s_1, s_2, a_2)). \quad (18)$$

An activity is an SAP process plan if it is a poset activity tree that contains a standard sequence

$$(\forall a)\ sap\_process\_plan(a) \equiv$$
$$((\exists a')\ standard\_sequence(a', a)$$
$$\wedge (choice\_poset(a) \vee strong\_poset(a) \vee complex\_poset(a))). \quad (19)$$

An activity is a parallel sequence if it is a simple sequence such that all of its subactivity occurrences are in the same bag with respect to the standard sequence of the process plan

$$(\forall a)\ parallel\_sequence(a) \equiv simple\_sequence(a)$$
$$\wedge ((\forall a_1, a_2, s_1, s_2)\ sap\_process\_plan(a_1) \wedge subactivity(a, a_1) \wedge standard\_sequence(a_2, a_1)$$
$$\wedge soo(s_1, a) \wedge soo(s_2, a_2) \supset same\_bag(s_1, s_2, a_1)). \quad (20)$$

An activity is an alternate sequence if it is a simple sequence such that all of its subactivity occurrences are alternate with respect to the standard sequence of the process plan

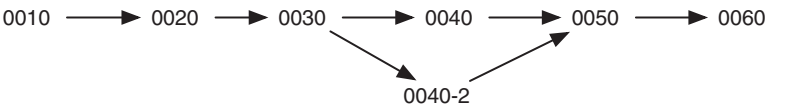
$$(\forall a)\ alternate\_sequence(a) \equiv simple\_sequence(a)$$
$$\wedge ((\forall a_1, a_2, s_1, s_2)\ sap\_process\_plan(a_1) \wedge subactivity(a, a_1) \wedge standard\_sequence(a_2, a_1)$$
$$\wedge soo(s_1, a) \wedge soo(s_2, a_2) \supset alternate(s_1, s_2, a_1)). \quad (21)$$

Figure 6. Axioms for SAP process plans.

### 5.2 *Formalisation of the manufacturing process control scenario*

In this section, we discuss the formalisation and results of a manufacturing process control scenario. Our example is of a pump manufacturing process. The process plan for the production order (60004907) is shown in Figure 7.

The operation numbers (as assigned by the SAP ERP system) are shown in parentheses. Note that the final assembly is accomplished in two operations (0040 and 0040-2) which are performed in parallel. Here, steps 1–3 are done sequentially, following which, steps 4 and 5 are done in parallel, and then steps 6 and 7 are done sequentially. The standard sequence for this production order is the sequence of operations: 0010, 0020, 0030, 0040, 0050, 0060. There is also a parallel sequence that consists only of the operation 0040-2 and is in parallel with the standard subsequence consisting of the single operation 0040.




[1.] Retrieval of items on the picking list (0010)
[2.] Assembly according to the component drawing (0020)
[3.] Coat and paint (0030)
[4.] Final assembly (0040)
[5.] Final assembly (0040-2)
[6.] Inspect (0050)
[7.] Deliver to storage (0060)

Figure 7. Motivating scenario: operations for production order 60004907.

```
( E1AFKOL ( AUFNR "60004907" )( APRIO  )( APROZ "0.00" ) ( AUART "PP01" )
( AUFLD "20070831" )( AUTYP "10" ) ( BAUMNG "0.000" )( BMEINS "PCE" )
( BMENGE "10.000" ) ( CY_SEQNR "00000000000000" )( DISPO "101" )
( FEVOR "101" ) ( FHORI "001" )( FLG_MLTPS  )( FREIZ "005" )
( FTRMI "20070815" ) (FTRMS"20070903" )( GAMNG "10.000" )
( GASMG "0.000" ) ( GETRI "00000000" )( GEUZI "000000" )
( GLTRI "00000000" ) ( GLTRP "20070914" )( GLTRS "20070912" )
( GLUZP "000000" ) ( GLUZS "150000" )( GMEIN "PCE" )
( GSTRI "00000000" ) ( GSTRP "20070906" )( GSTRS "20070910" )...
```

Figure 8. Excerpt of an exported IDOC file.

This production order was created using the SAP ERP system and exported into a custom format with IDOC[4] tags and values that are embedded in a LISP list structure. An excerpt of an example of an exported IDOC file is shown in Figure 8.

This output is then read into a LISP program which translates the IDOC tags into a set of PSL axioms that capture the production order using the PSL and SAP ontologies. This representation of the production order, along with the PSL axioms and axioms for the SAP ontology, are then input into Otter (McCune 2003), a first-order logic theorem prover. Otter can then be used to answer queries about the manufacturing process.

We now present the output of our translation program after we ran it on the IDOC file excerpted in Figure 8. The first axiom declares 60004907 to be an SAP production order

$$sap\_production\_order(60004907).$$

The production order number was generated by the SAP ERP system and is taken directly from the IDOC file. The axioms for the SAP ontology state that SAP production orders are activities.

Next, we specify the standard sequence for the production order. To do this a new symbol G1263 is generated and is declared to be a simple sequence. This is the standard sequence for the production order

$$simple\_sequence(G1263).$$

The name G1263 (and all other symbols beginning with 'G') was generated by the translation program.

G1263 is declared to be a subactivity of 60004907

$$\mathit{subactivity}(\text{G1263}, 60004907).$$

Then 0010 is declared to be an SAP operation in the production order 60004907 and a subactivity of G1263

$$\mathit{sap\_operation}(0010, 60004907)$$
$$\mathit{subactivity}(0010, \text{G1263}).$$

This is repeated for all the other operations in the standard sequence, but we omit the axioms here. Recall that parallel sequences consist of a simple subsequence of the standard sequence in parallel with one or more other simple sequences. In this case we have only one parallel sequence where operation 0040-2 is in parallel with 0040 of the standard sequence. We declare G1266 and G1264 to be simple sequences with 0040 and 0040-2 (respectively) as subactivities

$$\mathit{simple\_sequence}(\text{G1266})$$
$$\mathit{subactivity}(0040, \text{G1266})$$
$$\mathit{simple\_sequence}(\text{G1264})$$
$$\mathit{subactivity}(\text{0040-2}, \text{G1264}).$$

Then, we say that G1265 is a parallel sequence, it is a subactivity of 60004907, and it has two subactivities: G1266 and G1264

$$\mathit{parallel\_sequence}(\text{G1265})$$
$$\mathit{subactivity}(\text{G1265}, 60004907)$$
$$\mathit{subactivity}(\text{G1266}, \text{G1265})$$
$$\mathit{subactivity}(\text{G1264}, \text{G1265}).$$

Finally, we have an axiom that specifies the order among the subactivity occurrences of the production order. It says that for any occurrence of the activity 60004907, there must be occurrences of the operations in Figure 7 and they must occur in the order shown in the figure

$$\begin{aligned}
&\forall \mathit{occpo}.\mathit{occurrence_of}(\mathit{occpo}, 60004907)\\
&\quad \supset \exists \mathit{occ10}, \mathit{occ20}, \mathit{occ30}, \mathit{occ40}, \mathit{occ50}, \mathit{occ60}, \mathit{occ40\text{-}2}.\\
&\quad \mathit{occurrence_of}(\mathit{occ10}, 0010) \wedge \mathit{occurrence_of}(\mathit{occ20}, 0020)\\
&\quad \wedge \mathit{occurrence_of}(\mathit{occ30}, 0030) \wedge \mathit{occurrence_of}(\mathit{occ40}, 0040)\\
&\quad \wedge \mathit{occurrence_of}(\mathit{occ50}, 0050) \wedge \mathit{occurrence_of}(\mathit{occ60}, 0060)\\
&\quad \wedge \mathit{occurrence_of}(\mathit{occ40\text{-}2}, \text{0040-2}) \wedge \mathit{subactivity_occurrence}(\mathit{occ10}, \mathit{occpo})\\
&\quad \wedge \mathit{subactivity_occurrence}(\mathit{occ20}, \mathit{occpo}) \wedge \mathit{subactivity_occurrence}(\mathit{occ30}, \mathit{occpo})\\
&\quad \wedge \mathit{subactivity_occurrence}(\mathit{occ40}, \mathit{occpo}) \wedge \mathit{subactivity_occurrence}(\mathit{occ50}, \mathit{occpo})\\
&\quad \wedge \mathit{subactivity_occurrence}(\mathit{occ60}, \mathit{occpo}) \wedge \mathit{subactivity_occurrence}(\mathit{occ40\text{-}2}, \mathit{occpo})\\
&\quad \wedge \mathit{min_precedes}(\mathit{occ10}, \mathit{occ20}, 60004907) \wedge \mathit{min_precedes}(\mathit{occ20}, \mathit{occ30}, 60004907)\\
&\quad \wedge \mathit{min_precedes}(\mathit{occ30}, \mathit{occ40}, 60004907) \wedge \mathit{min_precedes}(\mathit{occ40}, \mathit{occ50}, 60004907)
\end{aligned}$$

$$\wedge\, min\_precedes(occ50, occ60, 60004907) \wedge min\_precedes(occ30, occ40\text{-}2, 60004907)$$
$$\wedge\, min\_precedes(occ40\text{-}2, occ50, 60004907).$$

We also have unique names axioms for the activities, but we omit them here.

The RFID tags attached to items in the manufacturing process can record information about what happens to them at different stages during manufacturing. This information could be stored directly in the form of PSL axioms on a tag and then used as further input to the theorem prover. To simulate this process, we added some axioms that describe some simple information about an object, PUMP1, that is being produced in this example production order. The axioms say that PUMP1 is an object and it participates in the operations 0010 and 0040-2. The idea is that once each of these operations is completed, the corresponding axioms could be written directly on the RFID tag for PUMP1

$$object(\mathbf{PUMP1})$$
$$\exists o.occurrence\_of(o, 0010) \wedge actual(o) \wedge participates\_in(\mathbf{PUMP1}, o, beginof(o))$$
$$\exists o.occurrence\_of(o, 0040\text{-}2) \wedge actual(o) \wedge participates\_in(\mathbf{PUMP1}, o, beginof(o)),$$

where $beginof(o)$ denotes the start time of the activity occurrence $o$.

The above axioms, along with the axioms for PSL and the SAP ontology, can be input into a theorem prover, which can then answer questions about the process of manufacturing the pump. We now show the queries that were correctly answered by the Otter theorem prover.

What are the simple sequences that are subactivities of the production order?

$$\exists a_1.simple\_sequence(a_1) \wedge sap\_production\_order(60004907)$$
$$\wedge\, subactivity(a_1, 60004907).$$

What are the subactivities of any parallel sequences that are subactivities of the production order?

$$\exists a_1, a_2.parallel\_sequence(a_1) \wedge sap\_production\_order(60004907)$$
$$\wedge\, subactivity(a_1, 60004907) \wedge subactivity(a_2, a_1).$$

Which activities must occur in the production order?

$$\exists a_1.subactivity(a_1, 60004907) \wedge a_1 \neq 60004907$$
$$\wedge\, \forall occ.occurrence\_of(occ, 60004907)$$
$$\supset \exists s.occurrence\_of(s, a_1) \wedge subactivity\_occurrence(s, occ).$$

Does some activity $a_1$ always occur before some activity $a_2$ in the production order?

$$\forall o.occurrence\_of(o, 60004907)$$
$$\supset \exists s_1, s_2, a_1, a_2.$$
$$occurrence\_of(s_1, a_1) \wedge occurrence\_of(s_2, a_2)$$
$$\wedge\, subactivity\_occurrence(s_1, o) \wedge subactivity\_occurrence(s_2, o)$$
$$\wedge\, min\_precedes(s_1, s_2, 60004907).$$

Which subactivities occur before activity 0030 in the production order?

$$\exists a_1.subactivity(a_1, 60004907)$$
$$\wedge \forall o.occurrence\_of(o, 60004907)$$
$$\supset \exists s_1, s_2.occurrence\_of(s_1, a_1) \wedge occurrence\_of(s_2, 0030)$$
$$\wedge subactivity\_occurrence(s_1, o) \wedge subactivity\_occurrence(s_2, o)$$
$$\wedge min\_precedes(s_1, s_2, 60004907).$$

What process steps did the pump object go through?

$$\exists o, t, a.participates\_in(\textsc{pump}1, o, t) \wedge occurrence\_of(o, a)$$
$$\wedge actual(o) \wedge subactivity(a, 60004907).$$

Did the pump object participate in a parallel sequence in a process plan?

$$\exists o, a_1, t, a_2.participates\_in(\textsc{pump}1, o, t) \wedge occurrence\_of(o, a_1) \wedge$$
$$actual(o) \wedge parallel\_sequence(a_2) \wedge subactivity(a_1, a_2).$$

These queries were successfully proven by Otter, and Otter can also output the bindings for the existentially quantified variables in its proofs, if desired.

We now summarise the steps in the process just described. A production order representing a manufacturing process is created and released in the SAP ERP system. This production order is then exported into the SAP IDOC format. The IDOC file is input into our translation program, which outputs an axiomatisation of the manufacturing process in PSL. These axioms, along with the axioms for PSL, the SAP ontology and the axioms on the RFID tags are input into the Otter theorem prover to answer queries about the manufacturing process. Ideally, at this point, the answers to the queries could be translated back into the appropriate SAP IDOC format and input to the SAP ERP system, e.g. as a production order confirmation. However, we reserve this step for future work.

## 6. Summary

While analysing the reasons for a lack of integration between applications within and across enterprise boundaries we identified the clash over the meaning of terms by different applications as a leading cause. We outlined a manufacturing process control scenario and identified a number of questions an ideal system should provide answers to. This scenario illustrated and motivated the need for ontologies, which led to the introduction of PSL and its primary purpose to enable the semantic interoperability of manufacturing process descriptions between manufacturing, engineering and business software applications. We further covered extensions to PSL and transformed the informal questions into their formal representations. We then explored this approach by integrating a particular business software application, SAP ERP, and formalised a relevant subset of the SAP data model by building an extension to the ontology that made the implicit semantics of the application's concepts explicit. We covered the translation process for the key concept in this subset, the SAP production order, and illustrated how a theorem prover can now provide answers to the questions posed.

Given the increasing capability to store information on an RFID tag the outcome is that the physical flow of goods can become fully aligned with the corresponding information flow. Accurate, individual product information becomes available when and where needed and other software applications can interoperate with relative ease because the semantics of concepts are clearly defined. The deductive capabilities of this system were also highlighted as answers to queries were found that were well beyond the scope it was initially designed for.

Further research is needed into solving a broader range of queries for dynamic process control. The queries in this paper have focussed on the ordering constraint over subactivities; in general, we will also need to incorporate state and temporal constraints, as well as reason about the preconditions and effects of activities within a process plan. This will also enable the solution of queries related to quality management and activity-based costing.

A second area of future work is to demonstrate provably correct semantic integration. Although ontologies can support the semantic integration of software applications, we are faced with the additional challenge that few applications have an explicitly axiomatised ontology. In this paper, we have used an ontology that formalises the data model of one such application; nevertheless, we need to prove that the ontology is the correct ontology *for that application*. Furthermore, a general methodology for attribution and evaluation of ontologies for software systems is required.

One limitation of this ontology-based approach arises from the intractability of automated reasoning with first-order logic theories. There are cases where automated theorem provers such as Otter are unable to find a proof for a particular query, even though the query sentence is in fact entailed by the axioms of the ontology and process description. Even in cases where solutions are found, some queries take minutes to solve, whereas the dynamic environment of a factory floor requires decisions to be made within seconds. We are actively pursuing several strategies to address this issue. The first is to identify tractable subsets of the ontology which are required for the range of possible queries. Another idea is to specify additional sentences which are consequences of the ontology and which can be used to more efficiently find proofs. The third approach is to identify subclasses of process plans for which the solutions of queries are tractable.

Before concluding we would like to highlight some of the key potential business benefits that can be expected from adopting the outlined approach.

- Improved local decisions that observe constraints of or even improve central schedules.
- Better business insight, e.g. determining an actual cost at item level.
- Greater responsiveness to changes and disruptions such as late configuration changes or machine breakdowns.
- Ability to dynamically reroute work in process while managing production process constraints.
- Immediate identification and notification of quality problems.
- Increased coordination among business partners across the the supply network.

Overall, we believe this that utilisation of RFID can lead to lower manufacturing costs, higher product quality and a greatly increased agility within the manufacturing enterprise and across its supply network.

## Notes

1. PSL has been published as an International Standard (ISO 18629) within the International Organisation of Standardisation. The full set of axioms (which we call $T_{psl}$) in the Common Logic Interchange Format is available at http://www.mel.nist.gov/psl/ontology.html
2. Each activity occurrence is an occurrence of a unique activity but activities can have multiple occurrences, so we label activity occurrences using the following convention: for activity occurrence $s_i^j$, $i$ is a unique label for the occurrence and $j$ denotes the activity of which it is an occurrence. For example, $s_1^1$ and $s_2^1$ are two occurrences of the activity $a_1$.
3. The *actual* relation is used to distinguish activity occurrences that have actually occurred, as distinct from activity occurrences that are possible within the activity tree
4. IDOC is a data interchange format for SAP programs.

## References

Bock, C. and Grüninger, M., 2005. PSL: A semantic domain for flow models. *Software and Systems Modeling Journal*, 2 (2), 209–231.

Cho, H. and Wysk, R., 1995. Intelligent workstation controller for computer-integrated manufacturing: Problems and models. *Journal of Manufacturing Systems*, 14 (4), 252–263.

Diekmann, T., Melski, A., and Schumann, M., 2007. Data-on-network vs. data-on-tag: Managing data in complex RFID environments. Waikoloa, USA, January.

Grüninger, M., 2004. Ontology of the Process Specification Language. *In*: S. Staab and R. Studer, eds. *Handbook of ontologies in information systems*. Berlin: Springer.

Grüninger, M. and Menzel, C., 2003. Process Specification Language: Principles and applications. *AI Magazine*, 24 (3), 63–74.

Guenther, O. and Tribowski, C., 2009. Storing data on RFID tags: A standards-based approach. Verona, Italy, June. To be published.

McCune, W., 2003. OTTER 3.3 Reference manual. Technical memorandum No. 263, Argonne National Laboratory ANL/MCS-TM-263.

Melski, A., *et al*., 2007. Beyond EPC – Insights from multiple RFID case studies on the storage of additional data on tag. Chicago, August, 281–286.

Ruta, M., *et al*., 2007. Semantic-enhanced EPCglobal Radio-Frequency Identification. *In*: G. Semeraro, *et al*., eds. *Proceedings of SWAP 2007, the 4th Italian Semantic Web Workshop*, 18–20 December 2007, Bari, Italy. Available from: http://ceur-ws.org/Vol-314/56.pdf

Schlenoff, C., Grüninger, M., and Ciociou, M., 1999. The essence of the Process Specification Language. *Transactions of the Society for Computer Simulation*, 16 (4), 204–216.

Sormaz, D. and Khoshnevis, B., 2003. Generation of alternative process plans in integrated manufacturing systems. *Journal of Intelligent Manufacturing*, 14 (6), 509–526.