A Situation Theoretic Approach to the Representation of Processes

Christopher Menzel and Richard J. Mayer
Texas A&M University and Knowledge Based Systems, Inc.
1500 University Avenue E., College Station, TX 77843 (409) 260-5274 (phone), (409) 260-1965 (fax), cmenzel@tamu.edu, rmayer@kbsi.com

Abstract

Typical methods for representing business, engineering, and manufacturing processes represent process information by means of rather restricted, often graphical languages. These languages are often fine as far as they go, but for many purposes—information sharing, in particular—much more precise, detailed representations of enterprise processes are required. In this paper we develop an approach to the rigorous representation of process information based on *situation theory*. We begin with an informal account of the semantic categories of the approach including situations, infons, types, activities, and processes, as well as the central relations that can hold between them. A framework known as ST that builds upon the Knowledge Interchange Format (KIF) is introduced for expressing information in these terms. The use of ST is then illustrated in detail by means of a series of examples.

Keywords

Process modeling, situation theory, enterprise integration, knowledge sharing, Knowledge Interchange Format, ontology

1 INTRODUCTION

A complex enterprise can be characterized essentially by its business processes, i.e., the events situations, states of affairs, and other highly-structured, dynamic objects according to which the enterprise functions. An integrated enterprise must capture and maintain detailed information about these dynamic objects in order to be able to coordinate them and, when necessary, reengineer them. That is, such an enterprise must be able to model its business processes effectively. This capacity, in turn, requires a well-defined theoretical framework, i.e., a well-defined language for representing process information, and a clear semantics that determines the meanings of the constructs of the language.

© Springer Science+Business Media Dordrecht 1996

In this paper we base such an account on situation theory, a relatively recent theory of information that we believe provides an excellent conceptual and theoretical foundation for representing process information (see, e.g., Barwise and Perry (1983), Barwise (1987), or Devlin (1991)). We will begin by introducing the basic concepts of the theory. We will then develop a language for expressing process information that builds upon the Knowledge Interchange Format, or KIF (Genesereth and Fikes (1992)). Finally, using that language, we will illustrate the power of the theory by means of a few of examples that require the specification of detailed process information. For detailed formal development of the theory, the reader is referred to Menzel and Mayer (forthcoming).

2 BASIC SITUATION THEORY

The notion of a situation is hardly unfamiliar in the literature of knowledge representation. Perhaps its earliest manifestation was the situation calculus of McCarthy (1968), which has been developed and applied more recently, and to great effect, to problems in enterprise integration by the TOVE project, especially Grüninger and Fox (1994, 1995). The notion in situation theory, though related, is significantly different.

2.1 Situations and infons

In situation theory, situations are (typically) concrete, spatially and temporally extended pieces of the real world, such as a baseball game, a math class, a manufacturing system (though situations within nonconcrete systems are admitted as well, e.g., the field of real numbers). Can any more be said about them other than this, however? What is it, for example, that makes a situation what it is, that distinguishes it from any other situation? The situation theoretic answer to this question is that a situation is what it is in virtue of the pieces of information it *supports*, or that *hold* in it. That, of course, just pushes the question back a step: what is a piece of information? Situation theory proper provides an elaborate and quite unique answer to this question. In situation theory, individual pieces of information are known as *infons*. The infons within a given domain are themselves constituted by objects, properties, and relations that exist within the domain. (Objects here are construed broadly to include not only physical objects, but also abstract ones like numbers and intervals of time.) More specifically, the *basic* infons in a given situation s are the fundamental units of information, good and bad, "generated" combinatorially from the relations and appropriate arguments for those relations within s; that is, the basic infons of s consist of all possible legitimate units of information of the form

objects $a_1, ..., a_n$ stand in relation r,

and

objects $a_1, ..., a_n$ do not stand in relation r, where the a_i are all constituents of s. (Objects here may be concrete, like persons and machines, or abstract, like numbers and intervals of time. Relations that hold among objects are known as **first-order** relations.*) These infons will be represented in the language that will be developed here as '(r a_1 ..., a_n +)' and '(r a_1 ..., a_n –)', respectively. A situation s *supports* a basic "positive" infon (r a_1 ..., a_n +) just in case its component objects a_1, ..., a_n stand in the relation r in s, and s supports a basic "negative" infon (r a_1 ... a_n –) just in case a_1, ..., a_n are present in s but do not stand in that relation in s. Thus, for example, the infons (mother-of Hillary Chelsea +) and (mother-of Chelsea Hillary –) are supported by, or hold in, typical White House situations s in 1993. In the language here, these facts would be expressed as '(supports s (mother-of Hillary Chelsea +))', '(supports s (mother-of Chelsea Hillary –))', '(denies s (mother-of Hillary Chelsea –))' and '(denies s (mother-of Chelsea Hillary +)), respectively, where is s is an appropriate White House situation.

Note that, because situations are (in general) **limited** pieces of the world, an object b that exists in one situation s may not exist in another s′. Hence, s′ will be "silent" on b; more exactly, it will support no information about b. Situations, that is to say, are **partial** with respect to information; they do not answer every question about every individual or every state of affairs. A typical White House situation, for example, carries no information about, say, the number of birds nesting on Heron Island in the Great Barrier Reef. It is as important for a theory of information to be able to represent partiality as it is to represent misinformation. In our adaptation of situation theory, partiality is captured by allowing the *supports* relation to be "gappy," or nonbivalent; that is, it is not the case in general that, for every situation s and infon p, either (supports s p +) or (supports s p –). However, the logic of situations is still classical, in the sense that the law of excluded middle still holds, i.e., we still have, for any s and p, either (supports s p +) or (not (supports s p +)).

To say that s supports a given basic infon (r a_1 ... a_n +) is to say that the individuals a_1, ..., a_n stand in the relation r throughout s. However, things can change within a situation—e.g., one changes from sleeping to waking in typical morning situations. This can be captured in situation theory by counting temporal intervals as individuals and including a temporal parameter explicitly among the arguments of first-order relations whenever appropriate. Thus, for instance, the property *asleep* will be conceived to be a 2-place relation that holds between individuals and temporal intervals. Thus, if s is a typical morning situation between 6:00 a.m. and 8:00 a.m. at an individual b's house, it is likely the case both that (supports s (asleep b 0600 +)) and that (supports s (asleep b 0800 –)). (If the relevant temporal parameter is understood; then, of course, it can be suppressed as a matter of convenience.) It is presupposed in the semantics of the version of situation theory presented here that all subintervals of the interval over which a situation occurs are present in the situation. Hence, a situation occurring from 6:00 a.m. to 8:00

*In the account developed here, infons will be restricted to individuals and first-order relations only. Full blown situation theory does not have this restriction, but this engenders significant technical complications that do not seem at all necessary for the purposes of enterprise process modeling.

a.m. supports all relevant temporal information within that period, e.g., that the interval from 6:00 to 6:15 precedes the interval from 6:30 to 6:45.

2.2 Types, activities, and processes

In most physical systems one observes multiple occurrences of situations that are similar in some respect. In such cases, the similar situations are said to be of the same type. For instance, a situation in which Bill Clinton is running on Tuesday and another in which he is running on Wednesday, though perhaps different in many respects, are similar insofar as Clinton is running in them, and hence are instances of the same type of situation. Situation types are thus general, repeatable patterns that can be exhibited by many different specific situations. A situation type, or activity, is specified in situation theory by means of an operator that abstracts over similar situations and an appropriate abstraction variable;* here we will use the operator 'type-of'. Thus, the activity just noted is represented as '(type-of ?sit (supports ?sit (running Clinton)))'. Similarly, distinct objects can be the same in certain respects, and hence can be thought of as instances of the same object type. Thus, Bill Clinton and Jimmy Carter are alike insofar as they are male politicians, i.e., they are both of the type *male politician.*

The importance of types in the context of process modeling—and, indeed, of modeling generally—is that the semantic content of most all process models concerns types. More exactly, a typical process is best thought of as a structured collection of situation types related to one another in a manner that reflects the process flow in a given activation of the process, i.e., the temporal relations between the instances of those types in an activation. For instance, consider the painting process depicted in Figure 1. (We use the graphical notation of the IDEF3 process description capture method, but nothing essential hinges upon this choice.) This diagram depicts a general process that must begin with an instance of *Paint Part* (represented by the *Paint Part* box with no predecessor), followed by an instance of *Test Coverage*. At that point, depending, presumably, on the outcome of the test, an instance of the process can either loop back to another instance of *Paint Part* or continue on to have the part dried. Thus, there are, in principle, infinitely many possible ways this single process can be instantiated by particular courses of events, depending on how many times such a course of events loops back to repeat the *Paint Part* activity.

*In situation theory proper, variables correspond semantically to actual 'variable objects' in the world, known sometimes as 'parameters' or 'indeterminates'. For purposes here, these entities can be avoided, though there are certain representational needs that require them, and hence they would be present in a complete account.

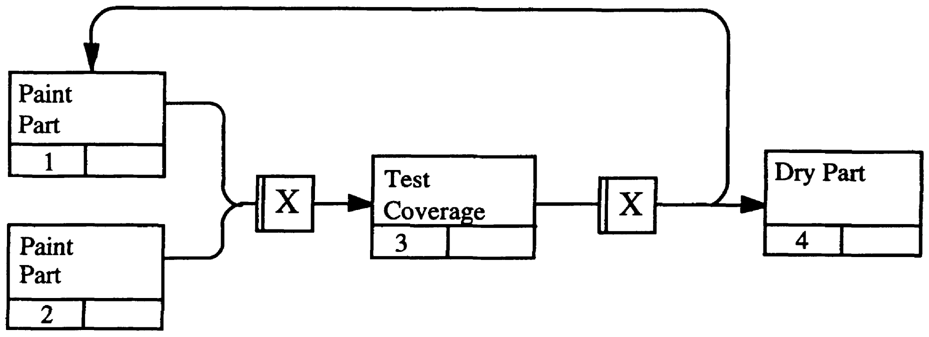

Figure 1 Paint/Test/Dry Scenario.

More generally, then, an activity, or situation type, is specified in terms of the variable-binding operator 'type-of', a variable '?sit' ranging over arbitrary situations, and a formula φ specifying the conditions common to situations of that type: '(type-of ?sit φ)', read 'the type of situation such that φ.' Thus, recalling the example above, '(type-of ?sit (supports ?sit (running Clinton)))' is read 'the type of situation such that it supports Clinton running,' or a little more naturally in this case, 'the type of situation in which Clinton is running.' A situation s is of **of type** T = (type-of ?sit φ) just in case φ is true when '?sit' refers to s. ('?sit', of course, will typically occur as a free variable in φ.) If φ is of the form '(supports ?sit ι)', where ι is an infon term, the activity is said to be specified **internally**; otherwise it is specified **externally**. The difference is that an internal specification describes the activity in terms of the infons that its occurrences support, whereas an external specification may refer instead to properties of the activity beyond the infons that its occurrences support, such as, e.g., the causes of its occurrences or the costs involved in maintaining them.

Analogously, an object type T is specified in terms of a variable ?x ranging over arbitrary objects and a sentence φ specifying the conditions common to objects of that type; specifically, '(type-of ?x φ)'. Analogous to situation types, an object *a* is of type T = (type-of ?x φ) just in case the sentence φ is true when '?x' refers to *a*.

2.3 A budget of relations

Situation theory is highly typed in the sense that the world it describes is partitioned into a number of different semantic categories, most notably, objects, first-order properties and relations, infons, situations, courses-of-events, object types, situation types, processes, and temporal intervals. To capture these distinctions, the theory of situations developed in this paper defines terms that denote each of these categories. In addition, a variety of terms are defined that signify a class of special relations, along with axioms that express precisely what categories of objects can stand in these relations. With these terms at his or her disposal, a user is able clearly to express any additional information or constraints not expressible in terms of a typical graphical language.

Specifically, then, the *supports* relation between situations and infons was discussed at length above. The *occurrence-of* relation holds between a situation s and an activity A just in case s is an instance of A. The *activation-of* relation holds between a course-of-events c and the process P just in case c is an activation of P. The *occurs-in* relation holds between a situation s and courses-of-events c just in case s occurs in c. The *activity-in* relation mirrors this relation at the type level: it holds between an activity A and a process P just in case A is among the situation types that constitute P. The *of-type* relation holds between a situation s and an activity A just in case s is an instance of A. The *object-in* relation holds between an object b and either a situation s or an activity A just in case b occurs in s or in instances of A.

A variety of temporal relations are needed to describe the temporal structure of complex processes. The only primitive relation required is *meets*, where, intuitively, one interval *i* meets another *j* just in case the endpoint of *i* is the starting point of *j*. Further relations—e.g., *precedes*, *starts*, *finishes*, *overlaps*, *during*—can be defined in terms of this relation, as illustrated in Allen and Hayes (1987). It should be noted that because intervals are treated as first-order objects, i.e., individuals, the temporal relations are all first-order relations. Temporal relations are used to define a variety of corresponding temporal relations among situations, as illustrated below.

3 USEFUL DEFINITIONS AND AXIOMS

The KIF axiomatizations for logical connectives, quantifiers, and other logical operators, sets, numbers, and lists, and the like are presupposed and are not repeated here. For details, see Genereth and Fikes (1992). Similarly, a standard axiomatization of the temporal predicates expressing the intuitive semantics noted above, such as in Allen and Hayes (1987) or van Benthem (1983) is assumed.

The following definitions and axioms are especially useful or important. Most of these are expressed as axiom schemas that pick out an infinite number of axioms that differ uniformly only in their choice of terms or sentences. To achieve this, the metavariables φ and ψ will serve as placeholders for any sentences, and V, V1, V2, etc. are any variables. An **instance** of a schema is the result of replacing each occurrence of a metavariable uniformly with an appropriate linguistic item (making sure always to replace all occurrences of the same metavariable with occurrences of the same linguistic item). φ[V/V1] is the result of replacing every free occurrence of V in φ with V1. An occurrence of a variable V in a sentence or term ε occurs **free** in ε just in case it does not occur within an expression of the form (OP V ...) or (OP (V_1 ... V ... V_n) ...) or (OP (V_1 ... V ... V_n such-that ...) ...) in ε, where OP is (in the first form) the term operator 'the' or 'setofall' or one of the numerical quantifiers 'exists-1', 'exists!-1', 'exists-2', exists!-2', etc., or (in any of the three forms) either 'forall' or 'exists'. It is assumed below that V1 is **free for** V in φ in the sense that V1 occurs free in φ[V/V1] wherever V is free in φ.

We will call the theory developed here "ST" for short.

Restricting variables with 'such-that'

Frequently when one uses universal quantification, one qualifies the range of things one is talking about. The way that this phenomenon manifests itself in a formal language is through the use of a conditional. For example, if the range of things one is talking about includes all people, then to talk about all the members of one specific class of person—professors, say—the claim is qualified in the following fashion: "For all x, if x is a professor, then x is overpaid." In terms of ST,

(forall ?x (=> (professor ?x) (overpaid ?x))).

In ordinary language, however, it is rare that a conditional "if ... then" would be used to express such a proposition. Rather, one would say simply "All professors are overpaid." Something of this naturalness can be captured using the 'such that' construction, which enables one to restrict the class of things one is talking about without a conditional; one essentially builds the qualification into the variable directly, as follows:

(forall (?x such-that (professor ?x)) (overpaid ?x)),

i.e., in logician's English, "For all x such that x is a professor, x is overpaid." Granted, there is generally no savings in the number of characters, but the form itself comports better with the grammar of ordinary language, hence making the proposition being expressed more understandable.

Note that the expression 'such-that' has no actual semantic content; it is there only to enhance readability. Hence, it could be eliminated from the formal syntax entirely. Though this would reduce clutter, it is significantly less readable, and readability, after all, was the point of introducing the construct in the first place. A reasonable compromise (following a fairly common practice in logic) is to permit the optional use of a colon in place of 'such-that'. Thus, the example above can also be written as follows.

(forall (?x : (professor ?x)) (overpaid ?x)).

This option will be used in the remainder of this paper. The 'such-that' construct is axiomatized generally in the obvious way as follows.

(<=> (forall (V_1 ... V_m : $\varphi_1 \ldots \varphi_n$) ψ) (forall ($V_1 \ldots V_m$) (=> (and $\varphi_1 \ldots \varphi_n$) ψ)))

(<=> (exists (V_1 ... V_m : $\varphi_1 \ldots \varphi_n$) ψ) (exists (V_1 ... V_m) (and $\varphi_1 \ldots \varphi_n$ ψ)))

(<=> (exists-ν (V : $\varphi_1 \ldots \varphi_n$) ψ) (exists-ν V (and $\varphi_1 \ldots \varphi_n$ ψ))), ν is any numeral other than '0'.

(<=> (exists!-ν (V : $\varphi_1 \ldots \varphi_n$) ψ) (exists!-ν V (and $\varphi_1 \ldots \varphi_n$ ψ))), ν is any numeral other than '0'.

Numerical quantifiers

Next, we introduce axioms for numerical quantifiers. Numerical quantifiers enable one to say that some specific number of things have a certain property without having to introduce that number of variables to indicate the objects. The required axioms illustrate the difference in syntactic complexity quite markedly. The weak numerical quantifiers are introduced first. The meaning of these quantifiers is that **at least** a certain specific number *n* of objects (of the type over which the variable V ranges) satisfy the sentence φ. These quantifiers are axiomatized as follows.

```
(<=> (exists-1 V φ)
     (exists V φ))

(<=> (exists-2 V φ)
     (exists (V1 V2 : (/= V1 V2))
             (and φ[V/V1] φ[V/V2])))

(<=> (exists-3 V φ)
     (exists (V1 V2 V3 : (/= V1 V2) (/= V2 V3) (/= V1 V3))
             (and φ[V/V1] φ[V/V2] φ[V/V3])))
```

So it continues, for all natural numbers *n*. The strong numerical quantifiers simply add to each of these axioms the qualification that no other objects besides the specific ones in question have the specified property, i.e., that **exactly** *n* things (of the type over which the variable V ranges) satisfy the sentence φ. The strong numerical quantifiers are thus axiomatized by adding this constraint to the axioms for the weak numerical quantifiers as follows.

```
(<=>  (exists!-1 V φ)
      (exists (V : φ) (forall (V1 : φ[V1/V]) (= V V1))))

(<=>  (exists!-2 V φ)
      (exists(V1 V2 : (/= V1 V2))
             (and φ[V/V1] φ[V/V2]
                  (forall (V3 : φ[V/V3]) (or (= V3 V1) (= V3 V2))))))

(<=>  (exists!-3 V φ)
      (exists (V1 V2 V3 : (/= V1 V2) (/= V2 V3) (/= V1 V3))
             (and φ[V/V1] φ[V/V2] φ[V/V3]
                  (forall (V4 : φ[V/V3]) (or (= V4 V1) (= V4 V2) (= V4 V3))))))
```

Similarly, once again, for all numbers *n*.

Number-of

Given the strong numerical quantifiers, it is important to axiomatize their connection with the *number-of* function. Specifically, the number of things that satisfy the sentence φ is *n* if and only if there exist exactly *n* things that satisfy φ, i.e.:

(<=> (= (number-of V φ) ?n) (exists!-?n V φ)).

The interval over which a situation occurs

In describing processes and their activations, it is very important to be able to talk about the interval of time over which a given situation occurs. For this reason, a function is defined (using the "define-function" operator) that, when applied to a given situation, yields exactly that interval. (As noted, an interval logic such as that defined in Allen and Hayes (1987) is assumed.) First, an axiom is provided to the effect that every situation occurs over exactly one temporal interval:

(forall ?sit (exists!-1 ?t (occurs-over ?sit ?t))).

This fact is now used to justify the desired definition:

(define-function interval-of (?sit) := (the ?t (occurs-over ?sit ?t))).

That is, the *interval-of* function applied to a situation returns the interval of time over which it occurs.

The starting and ending points of an interval

Given the ability to pick out the interval over which a situation occurs, it becomes equally important to be able to talk about its starting and ending points. This is accomplished by defining two more functions that yield the required points. Clearly, however, this requires that the property of being a temporal point be defined. This is accomplished with the following definition.

(define-relation point (?t) := (not (exists ?t1 (and (/= ?t ?t1) (during ?t1 ?t))))).

That is, a point is an interval during which no other interval occurs. Given facts that are left implicit here about the structure, or topology, of the space of temporal intervals, an interval satisfying this property is as small as possible, i.e., it is a point. The desired functions are then defined in the obvious way:

```
(define-function start-of (?t) :=
    (the ?t1 (and (point ?t1) (starts ?t1 ?t))))

(define-function end-of (?t) :=
    (the ?t1 (and (point ?t1) (ends ?t1 ?t)))).
```

That is, (start-of ?t) is the (unique) point that starts ?t; and (end-of ?t) is the point that ends it.

The starting and ending points of a situation

Given these functions, it is useful to define the starting (ending) point of a situation to be the starting (ending) point of its interval.

```
(define-function start-of (?sit) :=
    (start-of (interval-of ?sit)))

(define-function end-of (?sit) :=
    (end-of (interval-of ?sit))).
```

Note that the highly typed character of ST permits 'start-of' to be "polymorphic" and to select the appropriate function in virtue of the type of its argument.

Precedence for situations

In the same fashion, it is useful to define temporal precedence for situations in terms of their intervals as well:

```
(define-relation precedes (?sit1 ?sit2) :=
    (precedes (interval-of ?sit1) (interval-of ?sit2))).
```

Similar definitions can be given for all of the other temporal relations as well.

The ***next*** *relation*

The *interval-of* function also permit the definition of another useful notion, namely, the relation *next* that holds between occurrences s and s′ of two activities A and A′ in an activation e of a process when s′ is the first occurrence of A′ in e to follow s.

```
(define-relation next (?coe ?P ?A1 ?A2 ?sit1 ?sit2) :=
    (and(activation-of ?coe ?P)
        (activity-in ?A1 ?P)
        (activity-in ?A2 ?P)
        (occurs-in ?s1 ?coe)
        (occurs-in ?s2 ?coe)
        (occurrence-of ?s1 ?A1)
        (occurrence-of ?s2 ?A2)
        (precedes ?s1 ?s2)
        (forall(?s3 : (/= ?s3 ?s2) (occurs-in ?s3 ?coe)
                    (occurrence-of ?s3 ?A2) (precedes ?s1 ?s3))
            (precedes ?s2 ?s3)))).
```

That is, the *next* relation holds between s and s′ within an activation e just in case s precedes s′ and s′ precedes any other occurrence in e of the same activity.

The immediate successor relation

Finally, a somewhat more general notion than *next* is that of the *immediate successor* of a situation in an activation, or more generally, in a course-of-events. This relation is defined to hold between two situations s and s′ and a course of events e just in case s precedes s′ in e and there is no other situation s″ in e that s precedes but which starts before s′ does. Specifically:

```
(define-relation imm-succcessor (?sit1 ?sit2 ?coe) :=
    (and(occurs-in ?sit1 ?coe)
        (occurs-in ?sit2 ?coe)
        (precedes ?sit1 ?sit2)
        (not (exists ?sit3 (and (occurs-in ?sit3 ?coe)
                                (precedes ?sit1 ?sit3)
                                (precedes (start-of ?sit3) (start-of ?sit2)))))).
```

4 ILLUSTRATIVE EXAMPLES

To illustrate the use of ST, consider a more complex Paint/Test/Dry process than the one depicted in Figure 1, namely, the Paint/Review/Queue/Dry process depicted in Figure 2; call this process 'PRQD'.

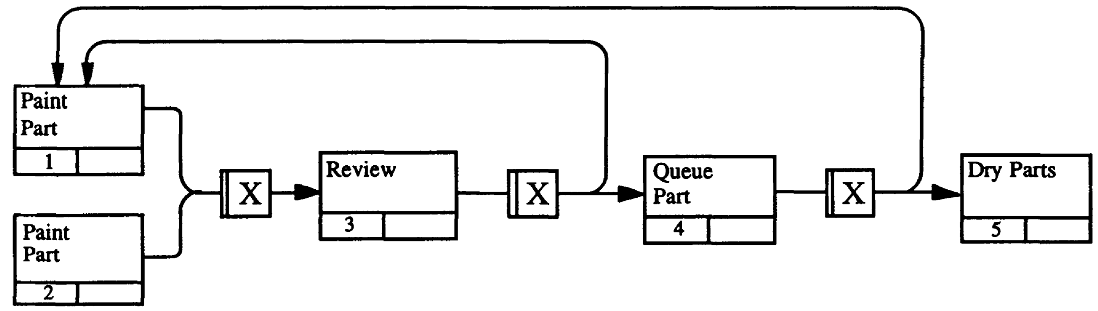

Figure 2 Paint/Review/Queue/Dry Scenario with an Additional Loopback.

The schematic itself provides a good general picture of the structure of PRQD. However, certain features of the process cannot be represented in the graphical language explicitly. To capture these, it will be best to define several auxillary notions. First, the new objects, properties, and relations introduced by this scenario need to be introduced into ST explicitly. There is, for instance, the first-order properties *being painted* and *being dried*, the object types *Part* and *Queue*, and so on. These are introduced into the language with explicit declarations:

```
(define-FO-relation painted)

(define-FO-relation dried)

(define-type Part)

(define-type Queue (exists!-1 ?x (of-type ?x Queue))).
```

Note that in the case of 'Queue', an 'axiom' is given that further characterizes the type. (The form is known as a 'partial definition' in KIF jargon.) This is one important way that basic, unchanging facts about the domain being described are expressed. In this case, the axiom expresses that the queue in PRQD is understood as a single object that remains constant across all activations—despite its changing length from moment to moment. (This fact is simply being assumed as part of the example; it is obviously not a fact about queues *per se*.) The existence of a single queue in the domain, in turn, justifies introducing a name 'Q' for the queue by means of a complete definition:

```
(define-object Q := (the ?x (of-type ?x Queue))),
```

that is, Q is the unique object of type Queue.

To express information involving the queue, it will be useful to introduce a relation *in-queue* that holds between an object, a queue, and a temporal interval (or point) just in case the first is in the second during the third. Since this is a primitive predicate, it is possible only to circumscribe its meaning by indicating that it only holds between an object and a queue. Accordingly, the new predicate is given an appropriate partial definition that specifies this condition explicitly, namely,

```
(define-relation in-queue (?x1 ?x2 ?t) :=>
   (and (of-type ?x2 Queue) (interval ?t))).
```

The presence of the ':=>' operator indicates that the sentence that follows expresses a necessary condition that must be met if the relation is to hold. That is, the following statement follows logically from the above definition:

```
(forall (?x1 ?x2) (=> (in-queue ?x1 ?x2 ?t) (and (of-type ?x2 Queue) (interval ?t)))).
```

It is now possible to express some of the features of PRQD that are merely implicit in the above schematic. For instance, clearly, the following constraint is intended:

In an activation of PRQD, the part that is queued in an occurrence of ***Queue Part*** *is different from the part that is painted in the next occurrence of* ***Paint Part*** *(if there is such an occurrence).*

Though not expressible in the graphical language, this proposition can be expressed easily in ST. As is often the case, however, some groundwork needs to be laid. Because there are typically numerous parts in the queue during an occurrence of *Queue Part*, the part that is queued during that occurrence needs to be distinguished from the others already in the queue. This could be accomplished in a number of ways, but here it will be assumed that the (unique) part that is added to the queue in an occurrence of *Queue Part* is added in such a fashion that it is not in the queue at the beginning of that occurrence (it is being moved, for instance), but is in the queue at the end. In terms of ST,

```
(forall (?coe : (activation-of ?coe PRQD))
        (forall (?sit : (occurs-in ?sit ?coe) (occurrence-of ?sit Queue-Part))
                (exists!-1 ?x (and  (of-type ?x Part)
                                    (supports ?sit (in-queue ?x Q (start-of ?sit) –))
                                    (supports ?sit (in-queue ?x Q (end-of ?sit) +)))))).
```

(Note that this constraint could have been expressed as '(forall (?coe ?sit : . . .)', but it is often more natural to group sequences of universally quantified variables according to type.) The above constraint can now be expressed as follows.

```
(forall (?coe : (activation-of ?coe PRQD))
        (forall (?sit1 ?sit2 : (occurs-in ?sit1 ?coe) (occurrence-of ?sit1 Queue-part)
                               (occurs-in ?sit2 ?coe) (occurrence-of ?sit2 Paint-part)
                               (next ?coe PQD Queue-part Paint-part ?sit1 ?sit2))
                (/= (the ?x (and (object-in ?x ?sit1) (of-type ?x Part)))
                    (the ?x (and (object-in ?x ?sit2) (of-type ?x Part))))))
```

Contrast this with the first loopback in the scenario depicted in Figure 2, in which the following is also clearly intended:

In an activation of PRQD, the part under review in an occurrence of ***Review*** *is identical with the part in the next occurrence of* ***Paint Part***.

This is so, of course, because an activation "loops back" to *Paint Part* after an occurrence of *Review* only if the part in that occurrence itself needs to be repainted after failing the test for adequate coverage. This general constraint on PRQD can be expressed much as the previous constraint.

```
(forall (?coe : (activation-of ?coe PRQD))
        (forall (?sit1 ?sit2 : (and  (occurs-in ?sit1 ?coe) (occurrence-of ?sit1 Review)
                                     (occurs-in ?sit2 ?coe) (occurrence-of ?sit2 Paint-part)
                                     (next ?coe PRQD Review Paint-part ?sit1 ?sit2)))
                (= (the ?x (and (object-in ?x ?sit1) (of-type ?x Part)))
                   (the ?x (and (object-in ?x ?sit2) (of-type ?x Part))))))
```

It is important to note that these two types of cycle in PRQD—one in which a part returns to *Paint Part* and another in which a fresh part is painted—are depicted in the graphical language in precisely the same fashion, and hence the difference between the two goes unrepresented unless expressed explicitly in ST.

Consider now a simplified version of PRQD in which the *Review* activity has been removed, as depicted in Figure 3. Call this process 'PQD'. In addition to obviously intended constraints such as those just indicated, there could be a wide variety of additional constraints on the process that cannot be expressed in the graphical language.

In an activation of PQD, exactly one part is painted in any given occurrence of ***Paint Part****.*

This constraint is expressed as follows.

```
(forall (?coe : (activation-of ?coe PQD))
        (forall (?sit : (occurs-in ?sit ?coe) (occurrence-of ?sit Paint-part))
                (exists!-1 ?x (supports ?sit (painted ?x +)))))
```

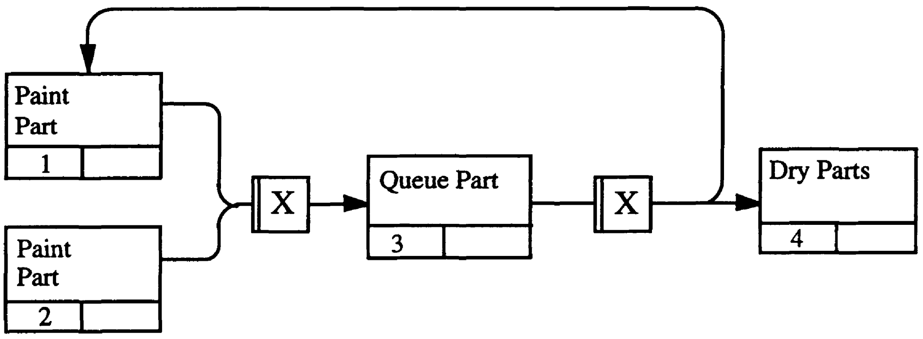

Figure 3 Paint/Queue/Dry Process.

This example illustrates the usefulness of the strong numerical quantifier 'exists!-1' and the use of the 'such-that' operator for qualifying the range of a variable. Note that this constraint, as expressed, attaches to the entire PQD scenario depicted in the diagram. However, it is more natural to add the constraint directly to the characterization of *Paint Part,* where it is intended to apply to each occurrence of *Paint Part* in a given activation of PQD. The general universally quantified conditions at the beginning of the constraint can thus be dropped, as they are implicit, and thus the constraint can be expressed much more simply and directly as follows:

(exists!-1 ?x (supports ?sit (painted ?x +))).

Note that, as a constraint on *Paint Part*, the situation variable '?sit' is not to be thought of as implicitly universally quantified, but rather as a parameter playing the role of a given occurrence of *Paint Part* in a given activation; the same is true for the object variable '?x'.

In an activation of PQD, no instance of ***Paint Part*** *begins at any time if there are five objects in the queue at that time.*

```
(forall (?coe : (activation-of ?coe PQD))
        (forall (?sit : (occurs-in ?sit ?coe) (occurrence-of ?sit Paint-part))
                (not (exists-5 (?x : (instance-of ?x Part))
                                     (supports ?sit (in-queue ?x Q (start-of (interval-of ?sit)) +))))))
```

That is, for any activation of PQD there is in that activation no occurrence *s* of *Paint Part* such that there are five objects in the queue at the start of *s*. Again, this constraint is expressed generally about PQD, but if it is added directly to the characterization of *Paint Part* where it is intended to apply to the occurrences of *Paint Part* within a given activation, it can be expressed directly as follows.

```
(not (exists-5 ?x (and (instance-of ?x Part)
                       (supports ?sit (in-queue ?x Q (start-of (interval-of ?sit)) +)))))
```

A final constraint on PQD illustrates the decision logic attached to the second of the two XOR junctions, namely:

If the number of parts in the queue at the end of an occurrence of ***Queue Part*** *is less than 5, the next activity to occur is* ***Paint Part****; otherwise, if the number of parts in the queue is equal to 5, then the next activity to occur is* ***Dry Parts***.

Attached to the junction, this constraint is easily expressible in terms of the *immediate successor* relation (defined above) as follows:

```
(forall (?sit1 ?coe : (occurs-in ?sit1 ?coe) (occurrence-of ?sit1 Queue-part))
        (and (=> (< (number-of ?x (supports ?sit (in-queue ?x Q (end-of ?sit) +))) 5)
                 (exists ?sit2 (and (occurrence-of ?sit2 Paint-part)
                                    (imm-succcessor ?sit1 ?sit2 ?coe))))
             (=> (= (number-of ?x (supports ?sit (in-queue ?x Q (end-of ?sit) +))) 5)
                 (exists ?sit2 (and (occurrence-of ?sit2 Dry-parts)
                                    (imm-succcessor ?sit1 ?sit2 ?coe)))))).
```

Since the constraint is attached to the junction, that ?coe is an activation of PQD is determined by context and hence can be left implicit.

5 REFERENCES

Allen, J., and Hayes, P. (1987) Moments and points in an interval-based temporal logic. Technical Report TR180, Departments of Computer Science and Philosophy, University of Rochester.

Barwise, J., and Perry, J. (1983) *Situations and Attitudes*. MIT Press/Bradford Books, Cambridge.

Barwise, J. (1987) *The Situation in Logic*. CSLI Lecture Notes, CSLI Publications, Center for the Study of Language and Information, Stanford University.

Devlin, K. (1991) *Logic and Information*. Cambridge University Press, Cambridge.

Enderton, H. (1972) *A Mathematical Introduction to Logic*. Academic Press, New York.

Genesereth, M. R., and Fikes, R. E. (1992) Knowledge Interchange Format version 3.0 — Reference Manual. Technical report Logic-92-1, Logic Group, Department of Computer Science, Stanford University, CA.

Grüninger M., and Fox, M. (1994) The design and evaluation of ontologies for enterprise engineering. Ms., Department of Industrial Engineering, University of Toronto.

Grüninger, M., and Fox, M. (1995) Methodology for the design and evaluation of ontologies. *IJCAI-95 Workshop on Basic Issues in Ontology*, Montreal, August 1995.

Grüninger M., and Pinto, J. (1995) A theory of complex actions for enterprise modelling, *AAAI Spring Symposium: Extending Theories of Action*, Stanford University, March 1995.

Hayes, P. (1977) In defense of logic. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 559-565.

Lee, J., Yost G., and the PIF Working Group (1994) The PIF process interchange format and framework. Working Paper 180, MIT Center for Coordination Science. Available in postscript format by anonymous ftp from pound.mit.edu as CCSWP180.ps in the directory /PUB/CCSWorking_Papers/.

McCarthy J., and Hayes, P. (1969) Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie (eds.), *Machine Intelligence* **4**, Edinburgh University Press, Edinburgh, pp. 463-502. Also in B. L. Weber and N. Nilsson, *Readings in Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1981.

McCarthy, J. (1968) Programs with Common Sense. In M. Minsky (ed.), *Semantic Information Processing*, MIT Press, Cambridge, MA, 403-418.

Menzel, C., and Mayer, R. (forthcoming) A Situation Theoretic Framework for Process Modeling. Forthcoming in the *International Journal for Concurrent Engineering Research Applications*.

van Bentham, J. (1983) *The Logic of Time*. D. Reidel Publishing Co., Dordrecht, Holland.

6 BIOGRAPHIES

Christopher Menzel is an associate professor in the Department of Philosophy at Texas A&M University. He also works as a senior consultant with Knowledge Based Systems, Inc., in College Station, Texas. Dr. Menzel has published widely in the area of philosophical logic. His more recent research focuses on both formal and philosophical issues in quantified modal logic. More generally, he is interested in the nature of formal representation, an interest that has led to a separate line of research in the application of formal methods to the domain of information modeling. He is particularly interested in logic-based approaches to enterprise model integration.

Richard J. Mayer is an associate professor in the Department of Industrial Engineering and director of the Knowledge Based Systems Laboratory at Texas A&M. He is also president and co-founder of Knowledge Based Systems, Inc. From 1977 through 1984, Dr. Mayer was project manager of the Integrated Computer Aided Manufacturing (ICAM) effort for the Manufacturing Technology Division at Wright-Patterson Air Force Base. His areas of expertise include: large scale information integration of logistical engineering and manufacturing information; AI applications to manufacturing, design, and engineering; knowledge engineering tool and method development; and information engineering methods development.