An Activity Ontology for Enterprise Modelling

Michael Gruninger and Mark S. Fox [1]

Department of Industrial Engineering, University of Toronto,
4 Taddle Creek Road, Toronto, Ontario M5S 1A4
tel:1-416-978-6823 fax:1-416-971-1373 internet:{gruninger, msf }@ie.utoronto.ca

Abstract

We present a logical framework for representing activities, states, and time in an enterprise integration architecture. We define an ontology for these concepts in first-order logic and consider the problems of temporal projection and reasoning about the occurrence of actions. This framework provides the basis for research in integrated supply chain management and enterprise engineering, as well as a new area for the application of theories of action and time.

Content Area: Knowledge Representation

1.0 Introduction

Enterprise modelling is an essential component in defining the tasks and functionality of the various components of an enterprise.The goal is to create generic, reusable representations of Enterprise Knowledge that can be applied across a variety of enterprises. Towards this end, the TOVE (Toronto Virtual Enterprise) ontology [Fox et al 93] has been developed and applied to enterprise engineering [Fox et al 94], enterprise integration, and integrated supply chain management. An ontology is a formal description of entities and their properties; it forms a shared terminology for the objects of interest in the domain, along with definitions for the meaning of each of the terms. The TOVE ontology currently spans knowledge of activity, time, and causality, resources, and more enterprise oriented knowledge such as cost, quality and organization structure. The TOVE Testbed provides an environment for analyzing enterprise ontologies; it provides a model of an enterprise and tools for browsing, visualization, simulation, and deductive queries.

The development of the TOVE ontology is driven by the specification of tasks that arise from two projects. The first of these is integrated supply chain management. The supply chain is a set of activities which span enterprise functions from the ordering and receipt of raw materials through the manufacturing of products and the distribution and delivery to the customer. The supply chain is

1. This research is supported, in part, by the Natural Science and Engineering Research Council, Digital Equipment Corp., Micro Electronics and Computer Research Corp., and Spar Aerospace.

viewed as being managed by a set of intelligent agents, each responsible for one or more activities in the supply chain, and each interacting with other agents in the planing and execution of their responsibilities. This includes agents for scheduling, dispatching, resource management, logistics, and transportation.

The second project, enterprise engineering, is concerned with the design and execution of enterprises. The goal of the enterprise engineering project is to formalize the knowledge required for business process reengineering ([Davenport 93], [Hammer & Champy 93]) and create an environment that facilitates the application of this knowledge to a particular company. First, the knowledge found in enterprise engineering perspectives such as time-based competition, activity-based costing, quality, agility, and resource management must be formally represented. This knowledge must then be integrated into a software tool that will support the enterprise engineering function by exploring alternative organization models spanning organization structure and behaviour, analyzing each alternative, and providing guidance to the designer.

In this paper we present a logical framework for the TOVE ontology. We present a set of tasks that arise in integrated supply chain management and enterprise engineering and the requirements on any ontology that is used to represent the tasks and their solution. We then apply Reiter's solution of the frame problem [Reiter 91] and Pinto's formalization of occurrence and the incorporation of time within the situation calculus [Pinto & Reiter 93] to the TOVE ontology of activity. This work provides a new testbed for theories of action and time, extending the formalisms to problems outside of the traditional robotics applications.

2.0 Ontologies and Microtheories

The basic entities in the TOVE model are represented as objects with specific properties and relations. Objects are structured into taxonomies and the definitions of objects, attributes and relations are specified in first-order logic. An ontology is defined in the following way. We first identify the objects in our domain of discourse; these will be represented by constants and variables in our language. We then identify the properties of these objects and the relations that exist over these objects; these will be represented by predicates in our language.

We next define a set of axioms in first-order logic to represent the constraints over the objects and predicates in the ontology. This set of axioms constitutes a microtheory ([Lenat & Guha 90]) and provides a declarative specification for the various tasks we wish to model. Further, we need to prove results about the properties of our microtheories in order to provide a characterization and justification for our approach; this enables us to understand the scope and limitations of the approach. We use a set of problems, which we call competency questions, that serve to characterize the various ontologies and microtheories in our enterprise model. The microtheories must contain a necessary and sufficient set of axioms to represent and solve these questions, thus providing a declarative semantics for the system. It is in this sense that we can claim to have an adequate microtheory appropriate for a given task, and it is this rigour that is lacking in previous approaches to enterprise engineering and integrated supply chain management.

The competency questions are generated by requiring that the ontologies and microtheories be necessary and sufficient to represent the tasks and their solutions for the various components of the system. Within integrated supply chain management and enterprise engineering, these include:

- Planning and scheduling -- what sequence of activities must be completed to achieve some goal? At what times must these activities be initiated and terminated?

- Temporal projection -- Given a set of actions that occur at different points in the future, what are the properties of resources and activities at other points in time?

- Execution monitoring and external events -- What are the effects on the schedule of the occurrence of external and unexpected events (such as machine breakdown or the unavailability of resources)?

- Hypothetical reasoning -- what will happen if we move one task ahead of schedule and another task behind schedule? What are the effects on orders if we buy another machine?

- Time-based competition -- we want to design an enterprise that minimizes the cycle time for a product [Blackburn 91]. This is essentially the task of finding a minimum duration plan that minimizes action occurrence and maximizes concurrency of activities.

The primary task which we address in this paper is that of temporal projection in an enterprise. This induces the following set of requirements on the ontologies:

- Temporal projection requires the evaluation of the truth value of a proposition at some point in time in the future. We therefore need to define axioms that express how the truth of a proposition changes over time. In particular, we need to address the frame problem and express the properties and relations that change or do not change as the result of an activity.

- We must define the notion of a state of the world, that is, define what is true of the world before and after performing different activities. This is necessary to express the causal relationship between the preconditions and effects of an activity.

- The time interval over which the state has a certain status is bounded by the times at which the appropriate actions that change status occur. This interval defines the duration of a state if the status is enabled. This is essential for the construction of schedules.

- We want a uniform hierarchical representation for activities (aggregation). Plans and processes are constructed by combining activities. We must precisely define how activities are combined to form new ones. The representation of these combined activities should be the same as the representation of the subactivities. Thus aggregate activities (sets of activities or processes) should themselves be represented as activities.

- The causal and temporal structure of states and subactivities of an activity should be explicit in the representation of the activity.

3.0 Time and Action

In this section we define the ontology of time and action that is used throughout the this paper. We represent time as a continuous line; on this line we define time points and time periods (intervals) as the domain of discourse. We define a relation < over time points with the intended interpretation that $t < t'$ iff t is earlier than t'.

One important property that must be represented is the intuition that some action *a* occurs and then some action *b* occurs, and that there is no intervening event between *a* and *b*. Furthermore, we want to define what holds in the world after performing some action in order to capture the notion of causality. How do we express these notions if we have a continuous time line? The extended situation calculus of [Pinto & Reiter 93] allows us to incorporate the notions of situations and a time line by assigning durations to situations. The primary distinction between our approach and the situation calculus is that we are using a single time line, whereas situation calculus uses a branching time structure. Since situations have duration, they can be defined as a set of distinguished intervals on the time line; they will be denoted by the letters σ. Further, we impose a structure over these intervals that is isomorphic to the natural numbers by introducing the notion of successor situation [Reiter 91]. The function $do(a,\sigma)$ is the name of situation that results from performing action *a* in situation σ. We also define an initial situation denoted by the constant σ_0. The following axioms establish the properties of the relation < over situations:

$$(\forall\, a, \sigma_1,\sigma_2)\ \sigma_1 < do(a,\sigma_2) \equiv \sigma_1 \leq \sigma_2 \qquad \text{(EQ 1)}$$

$$(\forall\, a_1,a_2, \sigma_1)\ do(a_1,\sigma_1) = do(a_2,\sigma_1) \supset a_1 = a_2 \qquad \text{(EQ 2)}$$

$$(\forall\, \sigma_1,\sigma_2)\ \sigma_1 < \sigma_2 \supset \neg\, \sigma_2 < \sigma_1 \qquad \text{(EQ 3)}$$

$$(\forall\, \varphi)[\varphi(\sigma_0) \wedge (\forall\, \sigma,a)\ (\varphi(\sigma) \supset \varphi(do(a, \sigma))] \supset (\forall\, \sigma)\ \varphi(\sigma) \qquad \text{(EQ 4)}$$

This enables us to define the intuition of no intervening events, that is, there is no situation between a situation and its successor, which is a consequence of the axioms:

$$(\forall\, \alpha,\sigma,\sigma')\ \neg\, (\sigma < \sigma' < do(a,\sigma)) \qquad \text{(EQ 5)}$$

Situations are assigned different durations by defining the predicate *start(s,t)* [Pinto & Reiter 93]. Each situation has a unique start time; these times begin at 0 in σ_0 and increase monotonically away from the initial situation.

$$(\forall \sigma)(\exists t)\ start(\sigma,t) \tag{EQ 6}$$

$$start(\sigma_0,0) \tag{EQ 7}$$

$$(\forall \sigma, t,t')\ start(\sigma,t) \wedge start(\sigma,t') \supset t = t' \tag{EQ 8}$$

$$(\forall a. \sigma, t,t')\ start(\sigma,t) \wedge start(do(a,\sigma),t') \supset t< t' \tag{EQ 9}$$

To define the evaluation of the truth value of a sentence at some point in time, we will use the predicate *holds(f,σ)* to represent the fact that some ground literal *f* is true in situation σ. Using the assignment of time to situations, we define the predicate $holds_T(f, t)$ to represent the fact that some ground literal *f* is true at time *t*. A fluent is a predicate or function whose value may change with time.

Another important notion is that actions occur at points in time. To represent this we introduce two predicates, *occurs(a,σ)* and $occurs_T(a,t)$, defined as follows:

$$occurs(a,\sigma) \equiv \sigma_0 < do(a,\sigma) \tag{EQ 10}$$

$$occurs_T(a,t) \equiv occurs(a,\sigma) \wedge start(do(a, \sigma), t) \tag{EQ 11}$$

We will now apply this formalism to the representation of activities in an enterprise.

4.0 Activities and States

At the heart of the TOVE Enterprise Model lies the representation of an *activity* and its corresponding enabling and caused *states* ([Sathi et al. 85], [Fox et al 93]). In this section we examine the notion of states and define how properties of activities are defined in terms of these states. An activity is the basic transformational action primitive with which processes and operations can be represented; it specifies how the world is changed. An enabling state defines what has to be true of the world in order for the activity to be performed. A caused state defines what is true of the world once the activity has been completed.

An activity, along with its enabling and caused states, is called an *activity cluster*. The state tree linked by an *enables* relation to an activity specifies what has to be true in order for the activity to be performed. The state tree linked to an activity by a *causes* relation defines what is true of the world once the activity has been completed. Intermediate states of an activity can be defined by elaborating the aggregate activity into an activity network (see Figure 1).

There are two types of states: *terminal* and *non-terminal*. In Figure 1, *es_fabricate_plug_on_wire* is the nonterminal enabling state for the activity *fabricate_plug_on_wire* and *pro_fabricate_plug_on_wire* is the caused state for the activity. The terminal conjunct substates of *es_fabricate_plug_on_wire* are *consume_wire, consume_plug*, and *use_inject_mold* since all three resources must be present for the activity to occur; the terminal states of *pro_fabricate_plug_on_-wire* are *produce_plug_on_wire* and *release_inject_mold*. The activity *assemble2 wire_switch* is enabled by the consumption of plug_on_wire (*consume plug_on_wire*) and the use of an assembly area (*use asmbly_area*); this is represented by the nonterminal state *es2_assemble_wire_switch*. This activity causes the production of wire_switch (*produce wire_switch*) and the release of the used resource (*release asmbly_area*); this is represented by the nonterminal state *pro2_assemble_wire_switch*.

In TOVE there are four terminal states represented by the following predicates:*use(s,a)*, *consume(s,a)*, *release(s,a), produce(s,a)*. These predicates relate the state with the resource required by the activity. Intuitively, a resource is used and released by an activity if none of the properties of a resource are changed when the activity is successfully terminated and the resource is released. A resource is consumed or produced if some property of the resource is changed after termination of the activity; this includes the existence and quantity of the resource, or some arbitrary property such as color. Thus *consume(s,a)* signifies that a resource is to be used up by the activity and will not exist once the activity is completed, and *produce(s,a)* signifies that a resource, that did not exist prior to the performance of the activity, has been created by the activity. We define use and consume states to be enabling states since the preconditions for activities refer to the properties of these states, while we define release and produce states to be caused states, since their properties are the result of the activity.

Terminal states are also used to represent the amount of a resource that is required for a state to be enabled. For this purpose, the predicate *quantity(s,r,q)* is introduced, where *s* is a state, *r* is the associated resource, and *q* is the amount of resource r that is required. Thus if *s* is a consume state, then *q* is the amount of resource consumed by the activity, if *s* is a use state, then *q* is the amount of resource used by the activity, and if *s* is a produce state, then *q* is the amount of resource produced.

In this section, we formalize the relationship between states and activities. First we examine the notion that an activity specifies a transformation on the world; this requires that we introduce fluents for states and activities, and the actions that change these fluents. The axioms presented adequate for solving the temporal projection problem for these properties of states and activities.

To formalize the notions of nonterminal states and aggregate activities, we introduce occurrence axioms for a set of actions.

4.1 Successor Axioms for Status of Terminal States

The primary fluents we will consider are the values assigned to states to capture the notion of the status of a state. We define a new sort for the domain of the status with the following set of constants:{ *possible, committed, enabled, completed, disenabled, reenabled*}. The status of a state is changed by one of the following actions:*commit(s,a), enable(s,a), complete(s,a), disenable(s,a), reenable(s,a)*. Note that these actions are parametrized by the state and the associated activity.

The next step is to define the successor axioms that specify how the above actions change the status of a state. These axioms provide a complete characterization of the value of a fluent after performing any action, so that we can use the solution to the frame problem in [Reiter 91]. Thus if we are given a set of action occurrences, we can solve the temporal projection problem (determining the value of a fluent at any point in time) by first finding the situation containing that time point, and then using the successor axioms to evaluate the status of the state in that situation.

The status of a state is committed in a situation iff either a commit action occurred in the preceding situation, or the state was already committed and an enable action did not occur.

$$(\forall\, s,a,e,\sigma)\ holds(status(s,a, committed), do(e,\sigma)) \equiv (e = commit(s,a) \wedge holds(status(s,a,possible),\sigma)) \vee \neg(e=enable(s,a)) \wedge holds(status(s,a, committed),\sigma) \quad \text{(EQ 12)}$$

The status of a state is enabled in a situation iff either an enable action occurred in the preceding situation, or the state was already committed and a complete action or disenable action did not occur.

$$(\forall\, s,a,e,\sigma)\ holds(status(s,a, enabled), do(e,\sigma)) \equiv (e = enable(s,a) \wedge holds(status(s,a,committed),\sigma)) \vee \neg[(e=complete(s,a) \vee e=disenable(s,a)) \wedge holds(status(s,a, enabled),\sigma)] \quad \text{(EQ 13)}$$

The status of a state is completed in a situation iff either a complete action occurred in the preceding situation, or the state was already completed.

$$(\forall\, s,a,e,\sigma)\ holds(status(s,a, completed), do(e,\sigma)) \equiv [e = complete(s,a) \wedge (holds(status(s,a,enabled),\sigma) \vee holds(status\ (s,a,reenabled),\sigma))] \vee holds(status(s,a, completed),\sigma) \quad \text{(EQ 14)}$$

The status of a state is disenabled in a situation iff either a disenable action occurred in the preceding situation, or the state was already disenabled and a reenable action did not occur.

$$(\forall\, s,a,e,\sigma)\ holds(status(s,a, disenabled), do(e,\sigma)) \equiv [e = disenable(s,a) \wedge (holds(status(s,a,enabled),\sigma) \vee holds(status\ (s,a,reenabled),\sigma))] \vee \neg\ (e=reenable(s,a)) \wedge holds(status(s,a, disenabled),\sigma) \quad \text{(EQ 15)}$$

The status of a state is reenabled in a situation iff either a reenable action occurred in the preceding situation, or the state was already reenabled and a complete action or disenable action did not occur.

$$(\forall\, s,a,e,\sigma)\ holds(status(s,a, reeenabled), do(e,\sigma)) \equiv (e = reeenable(s,a) \wedge holds(status(s,a,disenabled),\sigma)) \vee \neg(e=complete(s,a) \vee e=disenable(s,a)) \wedge holds(status(s,a, reenabled),\sigma) \quad \text{(EQ 16)}$$

Note that in each of these axioms we also specify the precondition for the action. These preconditions can equivalently be expressed as the following occurrence axioms:

$$(\forall\ s,a,\sigma)\ occurs(commit(s,a),\sigma) \supset holds(status(s,a,possible),\sigma) \quad \text{(EQ 17)}$$

$$(\forall\ s,a,\sigma)\ occurs(enable(s,a),\sigma) \supset holds(status(s,a,committed),\sigma) \quad \text{(EQ 18)}$$

$$(\forall\ s,a,\sigma)\ occurs(complete(s,a),\sigma) \supset (holds(status(s,a,enabled),\sigma) \vee holds(status(s,a,reenabled),\sigma)) \quad \text{(EQ 19)}$$

$$(\forall\ s,a,\sigma)\ occurs(disenable(s,a),\sigma) \supset (holds(status(s,a,enabled),\sigma) \vee holds(status(s,a,reenabled),\sigma)) \quad \text{(EQ 20)}$$

$$(\forall\ s,a,\sigma)\ occurs(reenable(s,a),\sigma) \supset holds(status(s,a,disenabled),\sigma) \quad \text{(EQ 21)}$$

How are these incorporated into the activity-state clusters, which only represent the causal relationships among states and activities? The occurrence of a commit action is not explicitly given in the specification of an activity. However, since the status fluents can only be changed by the above set of actions, the following sentences can be derived from the axioms:

$$(\forall\ s,a,\sigma)\ occurs(enable(s,a),\sigma) \supset (\exists\sigma')\ occurs(commit(s,a),\sigma') \quad \text{(EQ 22)}$$

$$(\forall\ s,a,\sigma)\ occurs(complete(s,a),\sigma) \supset (\exists\sigma')\ (occurs(enable(s,a),\sigma') \vee occurs(reenable(s,a),\sigma')) \quad \text{(EQ 23)}$$

$$(\forall\ s,a,\sigma)\ occurs(disenable(s,a),\sigma) \supset (\exists\sigma')\ (occurs(enable(s,a),\sigma') \vee occurs(reenable(s,a),\sigma')) \quad \text{(EQ 24)}$$

$$(\forall\ s,a,\sigma)\ occurs(reenable(s,a),\sigma) \supset (\exists\sigma')\ occurs(disenable(s,a),\sigma') \quad \text{(EQ 25)}$$

Similarly, the precondition for the *commit* action is that the state be *possible*. In [Fadel 94] it is shown how the *possible* status is defined in terms of the availability of a resource for the activity. This includes the configuration or setup of a resource as well as capacity constraints for the concurrent execution of activities with a shared resource. Axioms similar to those above would be used to express the occurrence of the appropriate setup activities for some activity. This is necessary for formalizing time-based competition, where the occurrence of setup activities is minimized.

4.2 Status of Non-Terminal States

In TOVE, non-terminal states enable the boolean combination of states. We will consider four non-terminal states:*conjunctive, disjunctive, exclusive, not*. What precisely does it mean for a non-terminal state to be a boolean combination of states? For example, how do we define the status of a non-terminal state given the status of each substate? To define this notion, we must refer to the occurrence of the actions that change the status of the states.

Disjunctive states are used to formalize the intuition of a resource pool. We may have a set of resources, such as machines or operators, that can possibly be used by an activity. The activity only requires one of these resources, so the activity only needs to nondeterministically choose one of the alternative resources in the pool. Thus, the status of the disjunctive state changes if one of the resources has been selected and its status has been changed. For example, we have

$$(\forall\, s,s_1,\ldots,s_n,a,\sigma)\ disjunctive(s,a) \wedge substate(s_1,s) \wedge \ldots \wedge substate(s_n,s) \supset occurs(enable(s,a),\sigma) \equiv occurs(enable(s_1,a),\sigma) \vee \ldots \vee occurs(enable(s_n,a),\sigma) \tag{EQ 26}$$

The successor axioms for the other values of status are defined in the same way. In other words, the occurrence of an action for a disjunctive state is equivalent to a disjunctive sentence of occurrence literals for each disjunct substate.

Similarly, we have the following constraints on conjunctive states:

$$(\forall\, s,s_1,\ldots,s_n,a,\sigma)\ conjunctive(s,a) \wedge substate(s_1,s) \wedge \ldots \wedge substate(s_n,s) \supset occurs(enable(s,a),\sigma) \equiv occurs(enable(s_1,a),\sigma) \wedge \ldots \wedge occurs(enable(s_n,a),\sigma) \tag{EQ 27}$$

The occurrence of an action for a conjunctive state is equivalent to a conjunctive sentence of occurrence literals for each conjunct substate. Note that we make the assumption that all conjunct substates change their status at the same time.

For exclusive states we have constraints of the form

$$(\forall\, s,s_1,\ldots,s_n,a,\sigma)\ exclusive(s,a) \wedge substate(s_1,s) \wedge \ldots \wedge substate(s_n,s) \supset occurs(enable(s,a),\sigma) \equiv occurs(enable(s_1,a),\sigma) \vee \ldots \vee occurs(enable(s_n,a),\sigma) \tag{EQ 28}$$

$$(\forall\, s,s_i,s_j,a,\sigma)\ exclusive(s,a) \wedge substate(s_i,s) \wedge substate(s_j,s) \wedge occurs(enable(s,a),\sigma) \supset (occurs(enable(s_i,a),\sigma) \equiv \neg occurs(enable(s_j,a),\sigma) \tag{EQ 29}$$

so that the occurrence of an action for an exclusive state is equivalent to the occurrence of the action for exactly one of the substates.

For not states we have the constraint that the action for the substate does not occur when the action for the nonterminal state occurs:

$$(\forall\, s,s_1,a,\sigma)\ not(s,a) \wedge substate(s_1,s) \supset occurs(enable(s,a),\sigma) \equiv \neg\, occurs(enable(s_1,a),\sigma) \tag{EQ 30}$$

In this way we can define arbitrary nonterminal states as occurrence axioms.

4.3 Status of Activities

Just as status was defined for states, we can define the status of an activity. We define a new sort for the domain of the status of an activity with the following set of constants:{ *dormant, executing, suspended, reExecuting, terminated* }. The status of an activity is determined by the status of its enabling and caused states.

An activity is dormant iff its enabling state is committed. In this case, the resources associated with the state are committed but not yet enabled.

$$(\forall\, a,s,\sigma)\ enabling(s,a) \supset holds(status(a, dormant), \sigma) \equiv holds(status(s,a,committed), \sigma) \tag{EQ 31}$$

An activity is executing iff either its enabling or caused state is enabled.

$$(\forall\, a,\sigma)\ holds(status(a, executing), \sigma) \equiv (\exists\, s)\ (enabling(s,a) \vee caused(s,a)) \wedge holds(status(s,a, enabled), \sigma) \tag{EQ 32}$$

An activity is suspended iff its enabling and caused states are disenabled.

$$(\forall\, a,s,\sigma)\ (enabling(s,a) \vee caused(s,a)) \supset holds(status(a, suspended), \sigma) \equiv holds(status(s,a, disenabled), \sigma) \tag{EQ 33}$$

An activity is reexecuting iff its enabling and caused states are reenabled.

$$(\forall\, a,s,\sigma)\ (enabling(s,a) \vee caused(s,a)) \supset holds(status(a, reExecuting), \sigma) \equiv holds(status(s,a, reenabled), \sigma) \tag{EQ 34}$$

An activity is terminated iff its enabling and caused states are completed.

$$(\forall\, a,s,\sigma)\ (enabling(s,a) \vee caused(s,a)) \supset holds(status(a, terminated), \sigma) \equiv holds(status(s,a,completed), \sigma) \tag{EQ 35}$$

4.4 Duration

By combining the ontology of time with the ontology of states of activities, we arrive at the notion of duration, which is essential for scheduling and the analysis of activities in time-based competition. The duration of a state is defined as the time period beginning at the time that the state is enabled and ending at the time that the state is completed. Similarly, the duration of an activity is defined as the time period beginning at the time that activity begins the status of executing and ending at the time that the activity begins the status of terminated. The duration of a state is represented by the predicate *state_duration(s,d)*, while the duration of an activity is represented by the predicate *activity_duration(a,d)*. In the representation of an activity, the duration of a state satisfies the occurrence axiom

$$(\forall\, a,s,t,t',d)\ state\_duration(s,d) \equiv occurs_T(enable(s,a), t) \wedge occurs_T(complete(s,a), t') \wedge d = t - t' \tag{EQ 36}$$

We can also define intervals for the remaining status values, such as committed:

$$(\forall\, a,s,t,t',d)\ committed\_duration(s,d) \equiv occurs_T(commit(s,a), t) \wedge occurs_T(complete(s,a), t') \wedge d = t - t' \tag{EQ 37}$$

In [Fadel 94], the committed duration is necessary to schedule the availability of a resource for a set of activities over some time interval. The resource must have sufficient capacity to support each activity at every time point in the interval and the resource may not be available to one activity if it is committed to other activities .

5.0 Aggregation of Activities

An important requirement for an ontology for activities is the ability to aggregate a set of activities to form a new activity. Activity clusters may be also aggregated to form multiple levels of abstraction. An activity is elaborated to an aggregate activity (an activity network), which then has activities [Sathi, et al 85]. These activities are subactivities of the aggregate activity. We introduce the predicate *subactivity(a,a′)* to denote that activity *a′* is a subactivity of activity *a*. For example, consider the activity clusters in Figure 1; the activities *fabricate plug_on_wire* and *assemble2 wire_switch* are sub-activities of **a***ssemble_ws aggregation*. The states *es_fabricate plug_on_wire* and *es2_assemble wire_switch* are substates of *enable_ws aggregation*, and the states *pro_fabricate plug_on_wire* and *pro_assemble wire_switch* are substates of *pro_ws_aggregation*.

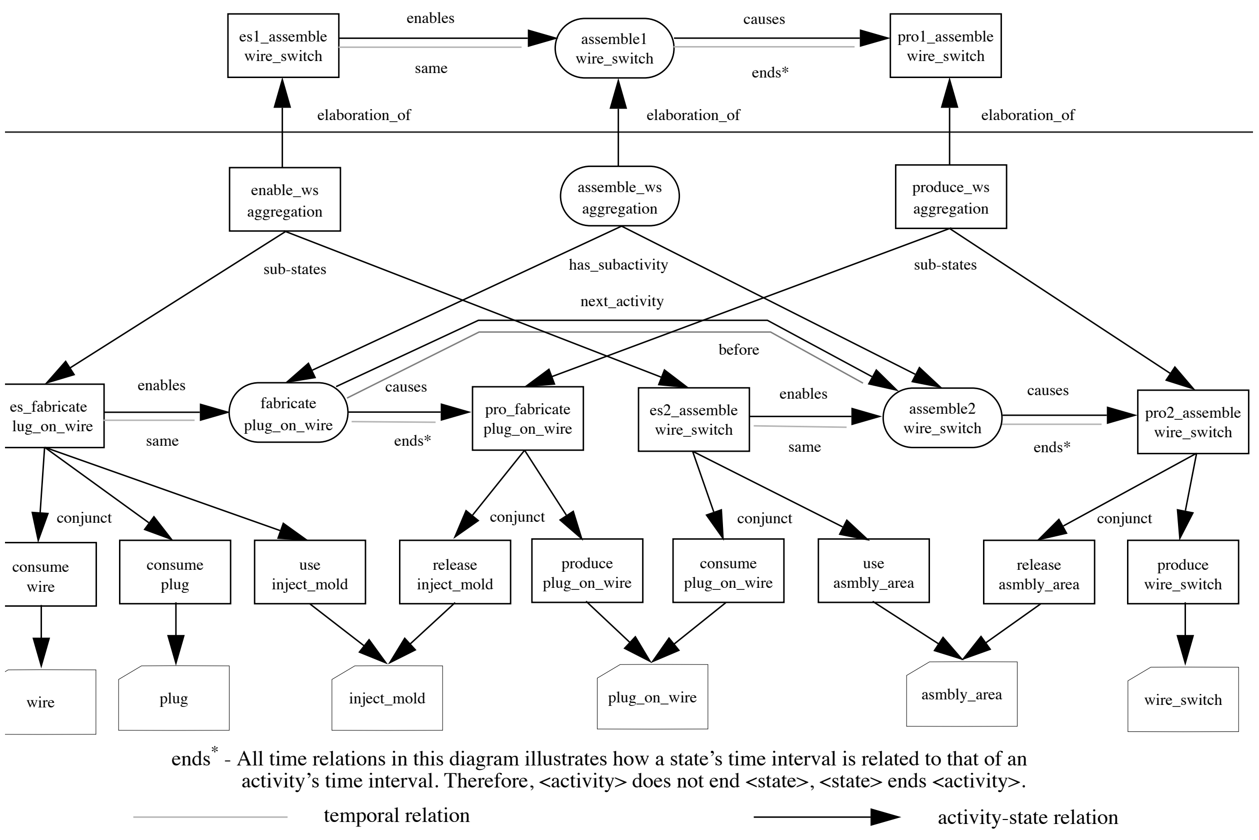
To completely specify an aggregate activity, we must define the temporal relations over its subactivities and the states of the subactivities. Indeed, the definition of status for activities allows to represent the temporal structure of an aggregate activity in terms of the enabling and caused states of each subactivity; we do this using occurrence axioms.

Essentially, the representation of an activity consists of a sequence of actions that commit, enable, and complete states. These actions may be partially ordered; once the actions in an activity have been totally ordered, we then assign times to the situations in which the actions occur. Thus activities will be represented by an occurrence axiom of the form:

$$\begin{gathered}(\forall\, a, s, s_1,\ldots,s_n,\sigma)\ enabling(s,a) \wedge substate(s_1,s) \wedge \ldots \wedge substate(s_n,s) \supset [occurs(enable(s,a),\sigma) \supset (\exists\, \sigma_1,\ldots,\sigma_n,t_1,\ldots t_n) \\ occurs(enable(s_1,a), \sigma_1) \wedge \ldots \wedge occurs(enable(s_k,a), \sigma_i) \wedge occurs(complete(s_1,a), \sigma_{i+1}) \wedge \ldots \wedge occurs(complete(s_k,a), \\ \sigma_n) \wedge occurs_T(enable(s_1,a), t_1) \wedge \ldots \wedge occurs_T(enable(s_k,a), t_i) \wedge occurs_T(complete(s_1,a), t_{i+1}) \wedge \ldots \wedge \\ occurs_T(complete(s_n,a), t_n)] \qquad \text{(EQ 38)}\end{gathered}$$

Note that this specification of the activity does not place any constraints on the ordering over the situations and times in which the actions occur. The complete specification of the aggregation of a set of arbitrary activities would impose a total ordering over the occurrences. In general, this is a scheduling problem which remains for future work.

FIGURE 1 plug_on_wire activity cluster



6.0 Summary

In this paper, we presented a logical formalization of the TOVE ontology of activity and time that has been designed to specify the tasks that arise in integrated supply chain management and enterprise engineering. To this end, we have defined the TOVE ontologies for activities, states, and time within first-order logic. This formalization allows deduction of properties of activities and states at different points in time by formalizing how these properties do or do not change as the result of an activity (temporal projection). The representation of aggregate activities, and the role of temporal structure in this aggregation, is accomplished through axioms that allow us to reason about the occurrence of actions.

The ontologies for activities, states, and time defined in this paper have been implemented on top of C++ using the ROCK knowledge representation tool from Carnegie Group. The successor state axioms and occurrence axioms have been implemented using Quintus Prolog. The formalization of activities is being extended to handle concurrent activities, reasoning about the availability and capacity of resources, and activity-based costing.

This work serves as a testbed for research into enterprise engineering, enterprise integration and integrated supply chain management, as well as opening new areas of application for theoretical work in reasoning about time, actions, and plans.

7.0 References

[Blackburn 91] Blackburn J. *Time-based Competition*. Business One Irwin, 1991.

[Fadel 94] Fadel, F. *Resource Ontology for Enterprise Modelling*. M.A.Sc. thesis, Department of Industrial Engineering, University of Toronto.

[Fox et al. 93] Fox, M.S., Chionglo, J., Fadel, F. A Common-Sense Model of the Enterprise, *Proceedings of the Industrial Engineering Research Conference 1993*.

[Fox et al 94]Fox, M. S., Gruninger, M., Zhan, Y.. Enterprise engineering: An information systems perspective (to appear in *Proceedings of the Industrial Engineering Research Conference 1994*).

[Davenport 93] Davenport, T.H. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Press, 1993.

[Hammer & Champy 93] Hammer, M. and Champy J. *Reengineering the Corporation*. Harper Business, 1993.

[Lenat & Guha 90] Lenat, D. and Guha, R.V. *Building Large Knowledge-based Systems: Representation and Inference in the CYC Project*. Addison Wesley, 1990.

[Pinto & Reiter 93] Pinto, J. and Reiter, R. Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of the Tenth International Conference on Logic Programming* (Budapest, June 1993).

[Reiter 91] Reiter, R. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press, San Diego, 1991.

[Sathi et al 85] Sathi, A., Fox, M.S., and Greenberg, M. Representation of activity knowledge for project management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. PAMI-7:531-552, September, 1985.