Task Dependence Ontology Axiomatization

July 2023

1 Introduction

This document presents the task dependence ontology. Section 2 introduces the model conceptually. Section 3 contains the formal model's axiomatization in first-order logic. In Section 4, we discuss the validation of our formal model.

2 Conceptual Model

In this section, we introduce our conceptual model, with each subsection covering a different layer. Figure 1 displays the ontology design pattern, highlighting the main concepts and relations discussed.

2.1 Agent, Intentions, and Desires

The foundational concepts in our model are agents, their tasks, and their goals. We define an Agent as an entity capable of possessing goals, having intentions to strive for those goals, and executing actions to attain them. Agents can be individuals, teams, departments, or entire organizations and even temporarily formed entities like committees or task forces. A Task signifies an agent's intention to exert effort towards a goal. It is defined based on the agent's intent and the goal, which is the target of this intent. A Goal represents a desired "state of affairs". It is defined in terms of the agent who has the goal and the desired State (detailed later) of the agent. At this stage, we don't consider the reasons behind an agent's goal or task. Also worth noting, our model doesn't incorporate organizational roles since our focus is on capturing dependencies between work, not dependent on specific organizational settings or formal structures.

Tasks may also be defined via Activities through which goals can be achieved. The difference between tasks and activities is important yet subtle. Tasks convey the intent to work towards a goal, while activities are concrete operations that might lead to a state, regardless of whether that state is desirable. Tasks represent the intention to act towards a goal, while activities specify the exact manner in which one can act.

This distinction is useful in contrasting well-defined and ill-defined task structures. For well-defined task structures, where necessary actions to achieve

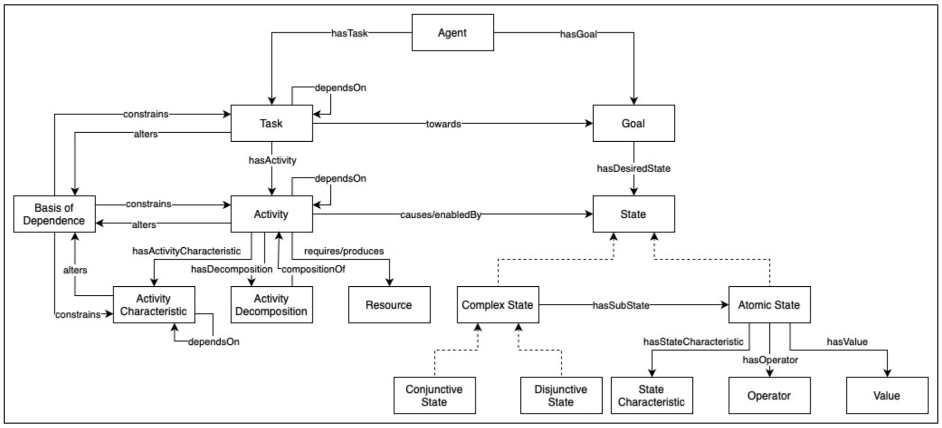

Figure 1: Task dependence ontology design pattern showing main classes and properties (dashed arrows for subclassOf).

a goal are known, we can understand the specific contribution of each agent. Conversely, in ill-defined task structures, where agents' specific actions are unpredictable or can't be predefined, we can still capture their collective intention towards the shared goal. Furthermore, this distinction allows us to express task dependence in terms of achieving goals rather than merely performing activities. In scenarios where the accomplishment of one goal influences or restricts another, but the exact method of influence can't be predetermined (as in ill-defined task structures), we model tasks as the intention to work towards a goal. This way, we can articulate the influence of one goal on another through the tasks involved in achieving both goals. Thus, the achievement of a state may be constrained by how another state is achieved, even if the precise "how" (i.e., an activity) is not known or cannot be represented. This gives us a higher level of representation, at the level of "intention to act towards" rather than the specific "act".

2.2 Activities and Resources

An Activity refers to a well-defined operation that an agent can perform as part of a task, causing an outcome state. It may also be enabled by a state and may require or produce a Resource. An activity can be seen as a production technology (Puranam, 2018) or an IDEF∅ function. An activity may have an Activity Decomposition, which breaks down complex activities into constituent sub-activities. This construct captures relationships between activities at different abstraction levels, reminiscent of hierarchical modelling in IDEF∅ or the *AggregateActivity* construct in TOVE (Fox et al., 1993). Explicit modelling of the decomposition as a construct allows representation, querying, and visualization of task structures at various abstraction levels. Capturing the specific ways in which an activity's performance can vary is crucial for activity dependence (the performance of one activity being influenced by another). For this, we use the Activity Characteristic construct, defining a feature of an activity subject to variation. Not every activity characteristic may be part of a dependency. We classify these characteristics into five main categories (not intended to be collectively exhaustive):

- *Input* characteristics: Variations in activity inputs such as *material selection*, *resource consumption*, *cost*, and *information source*.
- *Process* characteristics: Variations in the *method* or *implementation* of an activity.
- *Output* characteristics: Activity performance dimensions that define the output, like *quality*, *quantity*, and *design*.
- *Spatial* characteristics: Activity performance dimensions related to the physical or virtual *location* where an activity occurs.
- *Temporal* characteristics: Temporal features of a task, including *start* time, *end* time, and *duration*.

2.3 States

A State describes a particular aspect of an object or situation, capturing the idea that the modelled object or situation can have varying conditions over time. States can be complex or atomic. A Complex State can be either a Conjunctive State or a Disjunctive State, representing the conjunction or disjunction of other (sub)states, respectively.

An Atomic State is defined in terms of a state characteristic, operator, and value combination. The State Characteristic is an identifiable property of an object or situation. The Operator defines the relationship between a characteristic and value (e.g. $\leq$, $\geq$, $=$, $\neq$). The Value represents the specific value of the state characteristic defined in an atomic state. It can be based on a unit of measure, an ordinal or nominal scale, or any data type. This flexibility allows for representation of specific states and reasoning about the achievement of goals that can be satisfied as well as satisficed (Simon, 1947), a feature also seen in i* (Yu and Mylopoulos, 1995). This is useful where the exact condition for state satisfaction cannot be predetermined (e.g., minimize construction costs) or when a particular state characteristic is hard to quantify (e.g., improve design aesthetic).

2.4 Dependence

A Basis of Dependence captures the underlying reason for one task, activity, or activity characteristic depending on another, establishing the nature of their relationship. This construct is key to our model, as it underpins how task and activity dependencies are inferred. A basis of dependence may constrain or be altered by how a task is carried out, an activity is performed, or an activity characteristic varies. If a task or activity (or one of its characteristics) is constrained by a basis of dependence and that basis is altered by another task or activity, then the former entity depends on the latter. The semantics of the depends on relation vary based on the class of entities being related.

Also crucial to defining a basis of dependence is the Dependum concept – the entity through which a basis of dependence exists. For example, if an activity produces a resource used by another, the resource is the dependum causing a dependency. While this is not an exhaustive classification of all possible bases of dependence, we've identified six that collectively allow the representation of a variety of dependencies between organizational tasks and activities:

- *Availability*: The availability of a common resource (either as a common input or intermediary object) may be the basis of dependence between two tasks or activities. Here, the common resource is the dependum.

- *Functional*: Arises when the ability to perform an activity (both dependum and subject of dependence) varies with if or how another activity is performed or task accomplished.

- *Complementarity*: This basis of dependence comes into play when the overall effects of two activities or tasks are either greater or less than the sum of their parts. The non-additive state characteristic is the dependum here. For example, to minimize traffic disruptions, a municipality may seek to bundle several planned maintenance activities (e.g.; road repavement, sewer improvements, utility relocation) to occur at the same time.

- *Compatibility*: This basis of dependence arises when two tasks or activities need to coexist, but may not functionally influence each other.

- *Uncertainty*: This basis of dependence comes into play when two tasks or activities depend on each other due to the uncertainty of a state or activity characteristic.

- *Complexity*: This basis of dependence applies when the exact nature of the dependency relationship between two tasks or activities is unknown or cannot be explicitly stated.

We don't impose any set relations between these bases of dependence; depending on the domain the ontology is applied to, some bases may be sub-bases of others or disjoint. The complexity basis may serve as a catch-all for bases of dependence that can't be captured by the ones we've defined.

To illustrate the model, consider two activities requiring the same non-shareable, non-consumable, unary-capacity resource. An availability basis of dependence exists between them through the shared resource. Since the start time of either activity affects the resource's availability, both activity characteristics "alter" the basis of dependence. As the resource availability constrains when each activity can start, the basis of dependence "constrains" both activities. Hence, they both depend on each other.

3 Formal Model

3.1 Agents, Tasks, and Goals

An agent is an entity that has tasks and goals:

$$\begin{gathered}\forall a(Agent(a) \rightarrow \exists t, g(Task(t) \wedge Goal(g) \wedge \\ \wedge\, hasTask(a,t) \wedge hasGoal(a,g)))\end{gathered} \tag{1}$$

Domain and range constraint on $hasTask$ and $hasGoal$. The domains of each are agent, and the ranges must be tasks and goals, respectively:

$$\forall a, t(hasTask(a,t) \rightarrow Agent(a) \wedge Task(t)) \tag{2}$$

$$\forall a, g(hasGoal(a,g) \rightarrow Agent(a) \wedge Goal(g)) \tag{3}$$

Inverse relation for $hasTask$ and $hasGoal$:

$$\forall a, t(hasTask(a,t) \leftrightarrow taskOf(t,a)) \tag{4}$$

$$\forall a, g(hasGoal(a,g) \leftrightarrow goalOf(g,a)) \tag{5}$$

Domain and range constraint for $towards$:

$$\forall t, g(towards(t,g) \rightarrow Task(t) \wedge Goal(g)) \tag{6}$$

Task definition; a task must be defined in terms of the goal it is oriented $towards$:

$$\forall t(Task(t) \rightarrow \exists g(towards(t,g)) \tag{7}$$

Domain and range constraint for $hasActivity$. A task may have activities associated with it:

$$\forall t, a(hasActivity(t,a) \rightarrow Task(t) \wedge Activity(a)) \tag{8}$$

Inverse relation for $hasActivity$:

$$\forall t, a(hasActivity(t,a) \rightarrow activityOf(a,t)) \tag{9}$$

A goal is defined in terms of an agent which has the goal and a desired state:

$$\begin{aligned}\forall g(Goal(g) \equiv \exists a, s(&Agent(a) \wedge State(s) \\ &\wedge goalOf(g,a) \wedge hasDesiredState(g,s)))\end{aligned} \tag{10}$$

Domain and range constraint for $hasDesiredState$:

$$\forall g, s(hasDesiredState(g,s) \rightarrow Goal(g) \wedge State(s)) \tag{11}$$

Inverse relation for $hasDesiredState$:

$$\forall g, s(hasDesiredState(g,s) \leftrightarrow desiredStateOf(s,g)) \tag{12}$$

3.2 States

All states are either complex or atomic states:

$$\forall s(State(s) \rightarrow ComplexState(s) \lor AtomicState(s)) \tag{13}$$

Complex states and atomic states are disjoint subclasses of state:

$$\forall s_1, s_2(ComplexState(s_1) \land AtomicState(s_2) \rightarrow (s_1 \neq s_2)) \tag{14}$$

A complex state is a state that has atleast one substate:

$$\begin{aligned} &\forall s_1(ComplexState(s_1) \rightarrow \\ &\qquad \exists s_2(State(s_2) \land hasSubState(s_1, s_2) \land (s_1 \neq s_2))) \end{aligned} \tag{15}$$

A complex state can be either conjunctive or disjunctive, which are mutually exclusive:

$$\forall s(ComplexState(s) \rightarrow ConjunctiveState(s) \lor DisjunctiveState(s)) \tag{16}$$

$$\forall s_1, s_2(ConjunctiveState(s_1) \land DisjunctiveState(s_2) \rightarrow (s_1 \neq s_2)) \tag{17}$$

An atomic state cannot have a substate:

$$\forall s_1(AtomicState(s_1) \rightarrow \nexists s_2(hasSubState(s_1, s_2))) \tag{18}$$

Domain and range constraint on *hasSubState*:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow State(s_1) \land State(s_2)) \tag{19}$$

Inverse relation for *hasSubState*:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \leftrightarrow subStateOf(s_2, s_1)) \tag{20}$$

If a state has a substate, then it cannot be a substate of its substate:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow \neg subStateOf(s_1, s_2)) \tag{21}$$

An atomic state is defined in terms of a characteristic, operator, and value:

$$\begin{aligned} \forall s(AtomicState(s) \equiv \exists c, o, v(&StateCharacteristic(c) \land Operator(o) \\ &\land Value(v) \land hasStateCharacteristic(s, c) \\ &\land hasOperator(s, o) \land hasValue(s, v))) \end{aligned} \tag{22}$$

Inverse relationships between an atomic state and its characteristic, operator, and value:

$$\forall s, c(hasStateCharacteristic(s, c) \rightarrow stateCharacteristicOf(c, s)) \tag{23}$$

$$\forall s, o(hasOperator(s, o) \rightarrow operatorOf(o, s)) \tag{24}$$

$$\forall s, v(hasValue(s, v) \rightarrow valueOf(v, s)) \tag{25}$$

3.3 Activities and Resources

An activity is a function, or well-defined pattern of operation that causes a state:

$$\forall a(Activity(a) \equiv \exists s(causes(a, s))) \tag{26}$$

Domain and range constraint of *causes*:

$$\forall a, s(causes(a, s) \rightarrow Activity(a) \wedge State(s)) \tag{27}$$

Domain and range constraint of $performedBy$. An *Activity* may be performed by an *Agent*:

$$\forall a, g(performedBy(a, g) \rightarrow Activity(a) \wedge Agent(g)) \tag{28}$$

Domain and range constraint of $activityOf$. An *Activity* may be defined as part of a $Task$:

$$\forall a, t(activityOf(a, t) \rightarrow Activity(a) \wedge Task(t)) \tag{29}$$

Inverse relationships for *causes*, $performedBy$, and $activityOf$:

$$\forall a, s(causes(a, s) \rightarrow causedBy(s, a)) \tag{30}$$

$$\forall a, g(performedBy(a, g) \rightarrow performs(g, a)) \tag{31}$$

$$\forall a, t(activityOf(a, t) \rightarrow hasActivity(t, a)) \tag{32}$$

An outcome is a state that is caused by an activity:

$$\forall x(Outcome(x) \rightarrow State(x) \wedge \exists a(causedBy(x, a))) \tag{33}$$

Domain and range constraint of *enables*. An activity may also be enabled by a state (a precondition):

$$\forall a, s(enables(s, a) \rightarrow State(s) \wedge Activity(a)) \tag{34}$$

Inverse relationship for *enables*:

$$\forall a, s(enables(s, a) \leftrightarrow enabledBy(a, s)) \tag{35}$$

3.3.1 Resources

A resource is an entity that an activity either requires or produces:

$$\forall r(Resource(r) \leftrightarrow \exists a(requires(a, r) \vee produces(a, r))) \tag{36}$$

Domain and range constraints on *requires* and *produces* relationships, which can only be between a resource and activity:

$$\forall a, r(requires(a, r) \rightarrow Activity(a) \wedge Resource(r)) \tag{37}$$

$$\forall a, r(produces(a, r) \rightarrow Activity(a) \wedge Resource(r)) \tag{38}$$

Inverse relationships for *requires* and *produces*:

$$\forall r, a(requires(a, r) \leftrightarrow requiredBy(r, a))) \tag{39}$$

$$\forall r, a(produces(a, r) \leftrightarrow producedBy(r, a))) \tag{40}$$

A resource can either be shareable or nonshareable, both of which are disjoint:

$$\begin{aligned}\forall r(Resource(r) \rightarrow &ShareableResource(r)\\ &\vee NonShareableResource(r)))\end{aligned} \tag{41}$$

$$\begin{aligned}\forall r_1, r_2(&ShareableResource(r_1)\\ &\wedge NonShareableResource(r_2) \rightarrow (r_1 \neq r_2))\end{aligned} \tag{42}$$

A resource can either be consumable or nonconsumable, both of which are disjoint:

$$\begin{aligned}\forall r(Resource(r) \rightarrow &ConsumableResource(r)\\ &\vee NonConsumableResource(r)))\end{aligned} \tag{43}$$

$$\begin{aligned}\forall r_1, r_2(&ConsumableResource(r_1)\\ &\wedge NonConsumableResource(r_2) \rightarrow (r_1 \neq r_2))\end{aligned} \tag{44}$$

The *uses* and *consumes* are subrelations of *requires* when the resource is nonconsumable or consumable, respectively:

$$\forall a, r(uses(a, r) \rightarrow NonConsumableResource(r) \wedge requires(a, r)) \tag{45}$$

$$\forall a, r(consumes(a, r) \rightarrow ConsumableResource(r) \wedge requires(a, r)) \tag{46}$$

3.3.2 Activity Decomposition

An activity decomposition is a decomposition of an activity into at least two other activities (which it is a composition of):

$$\begin{aligned}\forall d(&ActivityDecomposition(d) \rightarrow \exists a_1, a_2, a_3(Activity(a_1) \wedge Activity(a_2)\\ &\wedge Activity(a_3) \wedge decompositionOf(d, a_1) \wedge compositionOf(d, a_2)\\ &\wedge CompositionOf(d, a_3) \wedge (a_1 \neq a_2) \wedge (a_2 \neq a_3) \wedge (a_1 \neq a_3)))\end{aligned} \tag{47}$$

Domain and range of $decompositionOf$ and $compositionOf$:

$$\begin{aligned}\forall d, a(&decompositionOf(d, a) \rightarrow\\ &ActivityDecomposition(d) \wedge Activity(a))\end{aligned} \tag{48}$$

$$\begin{aligned}\forall d, a(&compositionOf(d, a) \rightarrow\\ &ActivityDecomposition(d) \wedge Activity(a))\end{aligned} \tag{49}$$

Inverse relationships for $decompositionOf$ and $compositionOf$:

$$\forall d, a(decompositionOf(d, a) \leftrightarrow hasDecomposition(a, d)) \tag{50}$$

$$\forall d, a(compositionOf(d, a) \leftrightarrow hasComposition(a, d)) \tag{51}$$

Domain and range of *hasSubActivity*. If an activity can be decomposed into another activity, then the latter activity is a subactivity of the former:

$$\forall a_1, a_2(hasSubActivity(a_1, a_2) \rightarrow Activity(a_1) \wedge Activity(a_2)) \tag{52}$$

If an activity is decomposed into another activity, then the latter activity is a subactivity of the decomposed activity:

$$\begin{aligned} \forall d, a_1, a_2(Activity(a_1) \wedge Activity(a_2) \wedge hasDecomposition(a_1, d) \\ \wedge compositionOf(d, a_2) \rightarrow hasSubActivity(a_1, a_2)) \end{aligned} \tag{53}$$

Inverse relations for *hasSubActivity* :

$$\forall a_1, a_2(hasSubActivity(a_1, a_2) \leftrightarrow subActivityOf(a_2, a_1)) \tag{54}$$

An activity's decomposition may have an ordering of its subactivities that specifies the first and last activity. *hasFirstActivity* and *hasLastActivity* specify the first and last activities in an activity decomposition, respectively:

$$\begin{aligned} &\forall d, a(hasFirstActivity(d, a) \rightarrow \\ &\qquad ActivityDecomposition(d) \wedge Activity(a)) \end{aligned} \tag{55}$$

$$\begin{aligned} &\forall d, a(hasLastActivity(d, a) \rightarrow \\ &\qquad ActivityDecomposition(d) \wedge Activity(a)) \end{aligned} \tag{56}$$

Inverse relationships for *hasFirstActivity* and *hasLastActivity*:

$$\forall d, a(hasFirstActivity(d, a) \leftrightarrow firstActivityOf(a, d)) \tag{57}$$

$$\forall d, a(hasLastActivity(d, a) \leftrightarrow lastActivityOf(a, d)) \tag{58}$$

An activity may be either be succeeded or preceded by another activity:

$$\forall a_1, a_2(hasSuccessor(a_1, a_2) \rightarrow Activity(a_1) \wedge Activity(a_2)) \tag{59}$$

$$\forall a_1, a_2(hasPredecessor(a_1, a_2) \rightarrow Activity(a_1) \wedge Activity(a_2)) \tag{60}$$

First and last activities in a decomposition are not preceded or succeeded by any activity, respectively:

$$\forall d, a_1(hasFirstActivity(d, a_1) \rightarrow \neg(\exists a_2(hasPredecessor(a_1, a_2))) \tag{61}$$

$$\forall d, a_1(hasLastActivity(d, a_1) \rightarrow \neg(\exists a_2(hasSuccessor(a_1, a_2))) \tag{62}$$

An activity cannot both precede and succeed the same activity:

$$\forall a_1, a_2(hasSuccessor(a_1, a_2) \rightarrow \neg hasPredecessor(a_1, a_2)) \tag{63}$$

Inverse relationships for *hasSuccessor* and *hasPredecessor*:

$$\forall a_1, a_2(hasSuccessor(a_1, a_2) \leftrightarrow hasPredecessor(a_2, a_1)) \tag{64}$$

If an activity's subactivity requires or produces a resource, then so does the activity:

$$\forall a_1, a_2, r(subActivityOf(a_2, a_1) \wedge requires(a_2, r) \rightarrow requires(a_1, r) \tag{65}$$

$$\forall a_1, a_2, r(subActivityOf(a_2, a_1) \wedge produces(a_2, r) \rightarrow requires(a_1, r) \tag{66}$$

If an activity requires or produces a resource, then it must have a subactivity that requires or produces the resource:

$$\begin{aligned}\forall a_1, d, r(&hasDecomposition(a_1, d) \wedge requires(a_1, r) \rightarrow \\ &\exists a_2(hasSubActivity(a_1, a_2) \wedge requires(a_2, d)))\end{aligned} \tag{67}$$

$$\begin{aligned}\forall a_1, d, r(&hasDecomposition(a_1, d) \wedge produces(a_1, r) \rightarrow \\ &\exists a_2(hasSubActivity(a_1, a_2) \wedge produces(a_2, d)))\end{aligned} \tag{68}$$

3.3.3 Activity Characteristics

An activity may have an activity characteristic. Domain and range constraint for *hasActivityCharacteristic*:

$$\begin{aligned}\forall a, c(&hasActivityCharacteristic(a, c) \rightarrow \\ &Activity(a) \wedge ActivityCharacteristic(c))\end{aligned} \tag{69}$$

Inverse relation for *hasActivityCharacteristic*:

$$\begin{aligned}\forall a, c(&hasActivityCharacteristic(a, c) \leftrightarrow \\ &activityCharacteristicOf(c, a)\end{aligned} \tag{70}$$

Each activity characteristic must be defined as a characteristic of a specific activity:

$$\begin{aligned}\forall c(&ActivityCharacteristic(c) \rightarrow \\ &\exists a(activityCharacteristicOf(c, a)))\end{aligned} \tag{71}$$

Each activity characteristic belongs to a single activity:

$$\begin{aligned}\forall a_1, a_2, c(&hasActivityCharacteristic(a_1, c) \\ &\wedge hasActivityCharacteristic(a_2, c) \rightarrow (a_1 = a_2))\end{aligned} \tag{72}$$

The five main types of activity characteristics:

$$\forall c(TemporalCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{73}$$

$$\forall c(SpatialCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{74}$$

$$\forall c(InputCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{75}$$

$$\forall c(ProcessCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{76}$$

$$\forall c(OutputCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{77}$$

Types of temporal characteristics:

$$\forall c(Start(c) \rightarrow TemporalCharacteristic(c)) \tag{78}$$
$$\forall c(End(c) \rightarrow TemporalCharacteristic(c)) \tag{79}$$
$$\forall c(Duration(c) \rightarrow TemporalCharacteristic(c)) \tag{80}$$

Types of spatial characteristics:

$$\forall c(Location(c) \rightarrow SpatialCharacteristic(c)) \tag{81}$$

Types of input characteristics:

$$\forall c(Material(c) \rightarrow InputCharacteristic(c)) \tag{82}$$
$$\forall c(ResourceConsumption(c) \rightarrow InputCharacteristic(c)) \tag{83}$$
$$\forall c(Cost(c) \rightarrow InputCharacteristic(c)) \tag{84}$$

Types of process characteristics:

$$\forall p(Implementation(c) \rightarrow ProcessCharacteristic(c)) \tag{85}$$
$$\forall p(Method(c) \rightarrow ProcessCharacteristic(c)) \tag{86}$$

Types of output characteristics:

$$\forall p(Quantity(c) \rightarrow OutputCharacteristic(c)) \tag{87}$$
$$\forall p(Quality(c) \rightarrow OutputCharacteristic(c)) \tag{88}$$
$$\forall p(Design(c) \rightarrow OutputCharacteristic(c)) \tag{89}$$

There can be at most a single instance of a characteristic of an activity for the following types of characteristics:

$$
\begin{aligned}
\forall a, c_1, c_2 (&hasActivityCharacteristic(a, c_1) \\
&\wedge hasActivityCharacteristic(a, c_2) \\
&\wedge ((Start(c_1) \wedge Start(c_2)) \\
&\vee (End(c_1) \wedge End(c_2)) \\
&\vee (Duration(c_1) \wedge Duration(c_2))) \\
&\rightarrow (c_1 = c_2))
\end{aligned}
\tag{90}
$$

$$
\begin{aligned}
\forall a, c_1, c_2 (&hasActivityCharacteristic(a, c_1) \\
&\wedge hasActivityCharacteristic(a, c_2) \\
&\wedge ((Design(c_1) \wedge Design(c_2)) \\
&\vee (Quality(c_1) \wedge Quality(c_2)) \\
&\vee (Quantity(c_1) \wedge Quantity(c_2))) \\
&\rightarrow (c_1 = c_2))
\end{aligned}
\tag{91}
$$

$$
\begin{aligned}
\forall a, c_1, c_2 (&hasActivityCharacteristic(a, c_1) \\
&\wedge hasActivityCharacteristic(a, c_2) \\
&\wedge ((Material(c_1) \wedge Material(c_2)) \\
&\vee (ResourceConsumption(c_1) \wedge ResourceConsumption(c_2)) \\
&\vee (Cost(c_1) \wedge Cost(c_2))) \\
&\rightarrow (c_1 = c_2))
\end{aligned}
\tag{92}
$$

$$
\begin{aligned}
\forall a, c_1, c_2 (&hasActivityCharacteristic(a, c_1) \\
&\wedge hasActivityCharacteristic(a, c_2) \\
&\wedge (Location(c_1) \wedge Location(c_2)) \\
&\rightarrow (c_1 = c_2))
\end{aligned}
\tag{93}
$$

$$
\begin{aligned}
\forall a, c_1, c_2 (&hasActivityCharacteristic(a, c_1) \\
&\wedge hasActivityCharacteristic(a, c_2) \\
&\wedge ((Method(c_1) \wedge Method(c_2)) \\
&\vee (Implementation(c_1) \wedge Implementation(c_2))) \\
&\rightarrow (c_1 = c_2))
\end{aligned}
\tag{94}
$$

Disjointness of activity characteristic classes:

$$\begin{aligned}\forall c(\neg((&TemporalCharacteristic(c) \wedge SpatialCharacteristic(c))\\ &\vee (TemporalCharacteristic(c) \wedge InputCharacteristic(c))\\ &\vee (TemporalCharacteristic(c) \wedge ProcessCharacteristic(c))\\ &\vee (TemporalCharacteristic(c) \wedge OutputCharacteristic(c))\\ &\vee (SpatialCharacteristic(c) \wedge InputCharacteristic(c))\\ &\vee (SpatialCharacteristic(c) \wedge ProcessCharacteristic(c))\\ &\vee (SpatialCharacteristic(c) \wedge OutputCharacteristic(c))\\ &\vee (InputCharacteristic(c) \wedge ProcessCharacteristic(c))\\ &\vee (InputCharacteristic(c) \wedge OutputCharacteristic(c))\\ &\vee (ProcessCharacteristic(c) \wedge OutputCharacteristic(c))))\end{aligned} \tag{95}$$

$$\begin{aligned}\forall c(\neg((&Start(c) \wedge Duration(c)) \vee (End(c) \wedge Duration(c))\\ &\vee (Start(c) \wedge End(c)))\end{aligned} \tag{96}$$

$$\begin{aligned}\forall c(\neg((&Design(c) \wedge Quality(c)) \vee (Design(c) \wedge Quantity(c))\\ &\vee (Quality(c) \wedge Quantity(c)))\end{aligned} \tag{97}$$

$$\begin{aligned}\forall c(\neg((&ResourceConsumption(c) \wedge Cost(c)) \vee (Cost(c) \wedge Material(c))\\ &\vee (Material(c) \wedge ResourceConsumption(c)))\end{aligned} \tag{98}$$

$$\forall c(\neg((Method(c) \wedge Implementation(c))) \tag{99}$$

3.4 Bases of Dependence

A basis of dependence is defined in terms of the dependum through which it exists:

$$\begin{aligned}&\forall b(BasisOfDependenc(b) \equiv\\ &\qquad\qquad \exists d(Dependum(d) \wedge hasDependum(b, d)))\end{aligned} \tag{100}$$

Inverse relation for *hasDependum*:

$$\forall b, d(hasDependum(b, d) \leftrightarrow dependumOf(d, b)) \tag{101}$$

A basis of dependence has exactly one dependum:

$$\forall b, d_1, d_2(hasDependum(b, d_1) \wedge hasDependum(b, d_2) \rightarrow (d_1 = d_2)) \tag{102}$$

Domain and range constraints for *constrains* and *alteredBy*. A basis of dependence may constrain or be altered by a task, activity, or particular characteristic of an activity. Therefore the domain of both relations are bases of dependence, and the range can be either a task, activity, or activity resource:

$$\begin{aligned}&\forall b, x(constrains(b, x) \rightarrow BasisOfDependence(b) \wedge (Task(x)\\ &\qquad\qquad \vee Activity(x) \vee ActivityCharacteristic(x)))\end{aligned} \tag{103}$$

$$\begin{aligned}&\forall b, x(alteredBy(b, x) \rightarrow BasisOfDependence(b) \wedge (Task(x)\\ &\qquad\qquad \vee Activity(x) \vee ActivityCharacteristic(x)))\end{aligned} \tag{104}$$

Inverse relations for *constrains* and *alteredBy*:

$$\forall b, x(constrains(b, x) \leftrightarrow constrainedBy(x, b)) \tag{105}$$

$$\forall b, x(alteredBy(b, x) \leftrightarrow alters(x, b)) \tag{106}$$

Availability is a type of *BasisOfDependence* where the dependum must be a resource:

$$\begin{split}\forall b(Availability(b) \rightarrow BasisOfDependence(b)\\ \wedge \exists r(Resource(r) \wedge hasDependum(b, r)))\end{split} \tag{107}$$

Functionality is a type of *BasisOfDependence* where the dependum must be an activity:

$$\begin{split}\forall b(Functionality(b) \rightarrow BasisOfDependence(b)\\ \wedge \exists a(Activity(a) \wedge hasDependum(b, a)))\end{split} \tag{108}$$

Compatibility is a type of *BasisOfDependence* where the dependum must be a characteristic (either state or activity):

$$\begin{split}&\forall b(Compatibility(b) \rightarrow BasisOfDependence(b)\\ &\quad \wedge \exists x((ActivityCharacteristic(x)\\ &\quad \vee StateCharacteristic(x)) \wedge hasDependum(b, s)))\end{split} \tag{109}$$

Complementarity is a type of *BasisOfDependence* where the dependum must be a characteristic (either state or activity):

$$\begin{split}&\forall b(Complementarity(b) \rightarrow BasisOfDependence(b)\\ &\quad \wedge \exists x((ActivityCharacteristic(x)\\ &\quad \vee StateCharacteristic(x)) \wedge hasDependum(b, s)))\end{split} \tag{110}$$

Uncertainty is a type of *BasisOfDependence* where the dependum must be a characteristic (either state or activity):

$$\begin{split}&\forall b(Uncertainty(b) \rightarrow BasisOfDependence(b)\\ &\quad \wedge \exists c(ActivityCharacteristic(c)\\ &\quad \vee StateCharacteristic(c) \wedge hasDependum(b, c)))\end{split} \tag{111}$$

Complexity is a type of *BasisOfDependence* that does not necessarily specify the type of dependum:

$$\forall b(Complexity(b) \rightarrow BasisOfDependence(b)) \tag{112}$$

3.5 Task Dependence

A *dependsOn* relation can only be between two tasks, activities, or activity characteristic:

$$\begin{split}&\forall x, y(dependsOn(x, y) \rightarrow ((Task(x) \wedge Task(y))\\ &\quad \vee (Activity(x) \wedge Activity(y)) \vee (ActivityCharacteristic(x)\\ &\quad \wedge ActivityCharacteristic(x))))\end{split} \tag{113}$$

An activity characteristic depends on another activity characteristic if there exists a basis of dependence that constrains the former activity characteristic and is also altered by the latter characteristic:

$$\begin{aligned}\forall b, a_1, a_2, c_1, c_2(&\mathit{hasActivityCharacteristic}(a_1, c_1)\\ &\wedge \mathit{hasActivityCharacteristic}(a_2, c_2) \wedge \mathit{constrains}(b, c_2)\\ &\wedge \mathit{alteredBy}(b, c_1) \rightarrow \mathit{dependsOn}(c_2, c_1)\end{aligned} \tag{114}$$

If an activity characteristic depends on another activity characteristic, then the activity of the former activity characteristic is dependent on the activity of the second activity characteristic:

$$\begin{aligned}\forall a_1, a_2, c_1, c_2(&\mathit{hasActivityCharacteristic}(a_1, c_1)\\ &\wedge \mathit{hasActivityCharacteristic}(a_2, c_2)\\ &\wedge \mathit{dependsOn}(c_2, c_1) \rightarrow \mathit{dependsOn}(a_2, a_1)\end{aligned} \tag{115}$$

If an activity depends on another activity, then the task that the former activity is a part of is dependent on the task that the second activity is a part of:

$$\begin{aligned}&\forall t_1, t_2, a_1, a_2(\mathit{hasActivity}(t_1, a_1) \wedge \mathit{hasActivity}(t_2, a_2)\\ &\quad \wedge (t_1 \neq t_2) \wedge \mathit{dependsOn}(a_2, a_1) \rightarrow \mathit{dependsOn}(t_2, t_1)\end{aligned} \tag{116}$$

An activity depends on another activity if there exists a basis of dependence that constrains the former activity and is also altered by a characteristic of the latter activity:

$$\begin{aligned}&\forall a_1, a_2, b, c(\mathit{Activity}(a_1) \wedge \mathit{constrains}(b, a_1)\\ &\quad \wedge \mathit{alteredBy}(b, c) \wedge \mathit{hasActivityCharacteristic}(a_2, c)\\ &\quad \wedge (a_1 \neq a_2) \rightarrow \mathit{dependsOn}(a_1, a_2)\end{aligned} \tag{117}$$

An activity depends on another activity if there exists a basis of dependence that constrains a characteristic of the former activity and is also altered by the latter activity:

$$\begin{aligned}&\forall a_1, a_2, b, c(\mathit{Activity}(a_2) \wedge \mathit{constrains}(b, c)\\ &\quad \wedge \mathit{alteredBy}(b, a_2) \wedge \mathit{hasActivityCharacteristic}(a_1, c)\\ &\quad \wedge (a_1 \neq a_2) \rightarrow \mathit{dependsOn}(a_1, a_2)\end{aligned} \tag{118}$$

An activity depends on another activity if there exists a basis of dependence that constrains the former activity and is also altered by the latter activity:

$$\begin{aligned}\forall a_1, a_2, b(&\mathit{Activity}(a_1) \wedge \mathit{Activity}(a_2)\\ &\wedge \mathit{constrains}(b, a_1) \wedge \mathit{alteredBy}(b, a_2)\\ &\wedge (a_1 \neq a_2) \rightarrow \mathit{dependsOn}(a_1, a_2))\end{aligned} \tag{119}$$

A task depends on another task if there exists a basis of dependence that constrains the former task and is also altered by the task:

$$\begin{aligned}\forall t_1, t_2, b(&Task(t_1) \wedge Task(t_2)\\ &\wedge constrains(b, t_1) \wedge alteredBy(b, t_2)\\ &\wedge (t_1 \neq t_2) \rightarrow dependsOn(t_1, t_2))\end{aligned} \tag{120}$$

4 Validation

We demonstrate the correctness of the axiomatization using Prover9 and Mace4. After encoding our axiomatization, provided in `tdo_axioms.in`, Prover9 was unable to prove that our model was inconsistent.

To further validate our axiomatization, we tested whether Mace4 is able to generate a model based on the axioms and the scenarios specified in the Application section of the paper. Specifically, in `instance.in`, we append assertions of all the instances shown in Figure 2 of the paper to `tdo_axioms.in`, and run Mace4 to generate a model.

References

Fox, M. S., Chionglo, J. F., and Fadel, F. G. (1993). A common-sense model of the enterprise. In *Proceedings of Industrial Engineering Research Conference*, pages 178–194.

Puranam, P. (2018). *The Microstructure of Organizations.* Oxford University Press, Oxford, United Kingdom.

Simon, H. A. (1947). *Administrative Behavior: A Study of Decision-Making Processes in Administrative Organizations.* Macmillan, New York, NY.

Yu, E. S. and Mylopoulos, J. (1995). From E-R to "A-R"—modelling strategic actor relationships for business process reengineering. *International Journal of Cooperative Information Systems*, 4(2):125–144.