Towards an Ontology of Task Dependence in Organizations

Formal Model

Mena Rizk[1] [a], Mark Fox[1] [b] and Daniela Rosu[1] [c]
[1] *Enterprise Integration Laboratory, University of Toronto*
mena.rizk@mail.utoronto.ca, msf@eil.utoronto.ca, drosu@mie.utoronto.ca

1 Agents, Tasks, and Goals

hasTask connects agents with tasks:

$$\forall a, t(hasTask(a, t) \rightarrow (Agent(a) \wedge Task(t))) \tag{1}$$

hasGoal connects agents with goals:

$$\forall a, g(hasGoal(a, g) \rightarrow (Agent(a) \wedge Goal(g))) \tag{2}$$

hasTask and *taskOf* are inverse relations:

$$\forall a, t(hasTask(a, t) \leftrightarrow taskOf(t, a)) \tag{3}$$

hasGoal and *goalOf* are inverse relations:

$$\forall a, g(hasGoal(a, g) \leftrightarrow goalOf(g, a)) \tag{4}$$

Every *Agent* has at least one *Task* and one *Goal*:

$$\begin{aligned} \forall a(Agent(a) \rightarrow (\exists t, g(Task(t) \wedge Goal(g) \\ \wedge hasTask(a, t) \wedge hasGoal(a, g)))) \end{aligned} \tag{5}$$

towards connects *Tasks* with *Goals*:

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

A *Task* (intention of an agent to work towards a goal) is defined in terms of the *Goal* it is oriented *towards*, i.e., every task has a goal it is oriented towards:

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

[a] https://orcid.org/0009-0008-6095-0100
[b] https://orcid.org/0000-0001-7444-6310
[c] https://orcid.org/0000-0002-5877-9681

hasActivity connects a *Task* with an *Activity* associated with it:

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

hasActivity and *activityOf* are inverse relations:

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

A *Goal* is defined in terms of an *Agent* which has the goal and a desired *State*:

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

hasDesiredState connects goals and (desired) states:

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

hasDesiredState and *desiredStateOf* are inverse relations:

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

2 States

A *ComplexState* is a *State* that has at least one (proper) substate:

$$\begin{gathered}\forall s_1(ComplexState(s_1) \rightarrow \exists s_2(State(s_2) \\ \wedge hasSubState(s_1, s_2) \wedge (s_1 \neq s_2)))\end{gathered} \tag{13}$$

A *ComplexState* can be either a *ConjunctiveState* or *DisjunctiveState*, both of which are mutually exclusive:

$$\forall s(ComplexState(s) \rightarrow (ConjunctiveState(s) \vee DisjunctiveState(s))) \tag{14}$$

$$\forall s_1, s_2((ConjunctiveState(s_1) \wedge (DisjunctiveState(s_2)) \rightarrow (s_1 \neq s_2))) \tag{15}$$

An *AtomicState* cannot have a substate:

$$\forall s_1(AtomicState(s_1) \rightarrow \neg(\exists s_2(hasSubState(s_1, s_2)))) \tag{16}$$

A *State* is complex or atomic:

$$\forall s(State(s) \rightarrow (ComplexState(s) \vee AtomicState(s))) \tag{17}$$

*ComplexState*s and *AtomicState*s are disjoint:

$$\forall s_1, s_2(ComplexState(s_1) \wedge AtomicState(s_2) \rightarrow (s_1 \neq s_2)) \tag{18}$$

hasSubState connects a state to another state:

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

A state cannot be a substate of itself:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow (s_1 \neq s_2)) \tag{20}$$

hasSubState and *subStateOf* are inverse relations:

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

A state cannot have itself as a substate:

$$\forall s_1, s_2(subStateOf(s_1, s_2) \rightarrow (s_1 \neq s_2)) \tag{22}$$

A state cannot be a substate of its substate:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow \neg subStateOf(s_1, s_2)) \tag{23}$$

An *AtomicState* is defined in terms of a *StateCharacteristic*, *Operator*, and *Value*:

$$\begin{aligned} \forall s(AtomicState(s) \equiv \exists c, o, v(StateCharacteristic(c) \wedge Operator(o) \\ \wedge Value(v) \wedge hasStateCharacteristic(s, c) \\ \wedge hasOperator(s, o) \wedge hasValue(s, v))) \end{aligned} \tag{24}$$

hasStateCharacteristic and *stateCharacteristicOf* are inverse relationships:

$$\forall s, c(hasStateCharacteristic(s, c) \rightarrow stateCharacteristicOf(c, s)) \tag{25}$$

hasOperator and *operatorOf* are inverse relationships:

$$\forall s, o(hasOperator(s, o) \rightarrow operatorOf(o, s)) \tag{26}$$

hasValue and *valueOf* are inverse relationships:

$$\forall s, v(hasValue(s, v) \rightarrow valueOf(v, s)) \tag{27}$$

3 Activities and Resources

An *Activity* is a function, or well-defined pattern of operation, and *causes* a *State*:

$$\forall a(Activity(a) \rightarrow \exists s(causes(a, s))) \tag{28}$$

causes connects activities and states:

$$\forall a, s(causes(a, s) \rightarrow (Activity(a) \wedge State(s))) \tag{29}$$

performedBy connects activities and agents, i.e., *Activity* is performed by an *Agent*:

$$\forall a, g(performedBy(a, g) \rightarrow (Activity(a) \wedge Agent(g))) \tag{30}$$

activityOf connects activities and tasks.

$$\forall a, t(activityOf(a, t) \rightarrow (Activity(a) \wedge Task(t))) \tag{31}$$

causes and *causedBy* are inverse relationships:

$$\forall a, s(causes(a, s) \leftrightarrow causedBy(s, a)) \tag{32}$$

performedBy and *performs* are inverse relationships:

$$\forall a, g(performedBy(a, g) \leftrightarrow performs(g, a)) \tag{33}$$

activityOf and *hasActivity* are inverse relationships:

$$\forall a, t(activityOf(a, t) \leftrightarrow hasActivity(t, a)) \tag{34}$$

An *Outcome* is a *State* that is *causedBy* an *Activity*:

$$\forall s, act(causedBy(s, act) \rightarrow Outcome(s)) \tag{35}$$

enables connects states with the activities they enable.

$$\forall a, s(enables(s, a) \rightarrow (State(s) \wedge Activity(a))) \tag{36}$$

enables and *enabledBy* are inverse relations:

$$\forall a, s(enables(s, a) \leftrightarrow enabledBy(a, s)) \tag{37}$$

3.1 Resources

A *Resource* is an entity that an *Activity* either *requires* or *produces*:

$$\forall r(Resource(r) \leftrightarrow \exists a(requires(a, r) \vee produces(a, r))) \tag{38}$$

requires connects an *Activity* with a *Resource* it needs. *produces* connects an *Activity* with a *Resource* it produces:

$$\forall a, r(requires(a, r) \rightarrow Activity(a) \wedge Resource(r)) \tag{39}$$

$$\forall a, r(produces(a, r) \rightarrow Activity(a) \wedge Resource(r)) \tag{40}$$

requires and *requiredBy* are inverse relationships. *produces* and *producedBy* are inverse relationships:

$$\forall r, a(requires(a, r) \leftrightarrow requiredBy(r, a))) \tag{41}$$

$$\forall r, a(produces(a, r) \leftrightarrow producedBy(r, a))) \tag{42}$$

A resource can either be shareable or nonshareable:

$$\begin{gathered}\forall r(Resource(r) \rightarrow ShareableResource(r) \\ \vee NonShareableResource(r)))\end{gathered} \tag{43}$$

NonShareableResource and *ShareableResource* are disjoint classes:

$$\forall r_1, r_2(ShareableResource(r_1) \wedge NonShareableResource(r_2) \\ \rightarrow (r_1 \neq r_2)) \tag{44}$$

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

$$\forall r(Resource(r) \rightarrow ConsumableResource(r) \\ \vee NonConsumableResource(r))) \tag{45}$$

ConsumableResource and *NonConsumableResource* are disjoint classes:

$$\forall r_1, r_2(ConsumableResource(r_1) \wedge NonConsumableResource(r_2) \\ \rightarrow (r_1 \neq r_2)) \tag{46}$$

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

$$\forall a, r(uses(a, r) \rightarrow NonConsumableResource(r) \wedge requires(a, r)) \tag{47}$$

$$\forall a, r(consumes(a, r) \rightarrow ConsumableResource(r) \wedge requires(a, r)) \tag{48}$$

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):

$$\forall d(ActivityDecomposition(d) \rightarrow \exists a_1, a_2, a_3(Activity(a_1) \wedge Activity(a_2) \\ \wedge Activity(a_3) \wedge decompositionOf(d, a_1) \wedge compositionOf(d, a_2) \\ \wedge CompositionOf(d, a_3) \wedge (a_1 \neq a_2) \wedge (a_2 \neq a_3) \wedge (a_1 \neq a_3))) \tag{49}$$

decompositionOf and *compositionOf* connect activity decompositions with activities:

$$\forall d, a(decompositionOf(d, a) \rightarrow ActivityDecomposition(d) \\ \wedge Activity(a)) \tag{50}$$

$$\forall d, a(compositionOf(d, a) \rightarrow ActivityDecomposition(d) \\ \wedge Activity(a)) \tag{51}$$

decompositionOf and *hasDecomposition* are inverse relations. *compositionOf* and *hasComposition* are inverse relations:

$$\forall d, a(decompositionOf(d, a) \leftrightarrow hasDecomposition(a, d)) \tag{52}$$

$$\forall d, a(compositionOf(d, a) \leftrightarrow hasComposition(a, d)) \tag{53}$$

hasSubActivity connects an activity with another activity:

$$\forall a_1, a_2(hasSubActivity(a_1, a_2) \rightarrow Activity(a_1) \wedge Activity(a_2) \\ \wedge (a_1 \neq a_2)) \tag{54}$$

If an activity has a decomposition which is a composition of another activity, then the former activity has the latter as a subactivity:

$$\begin{aligned}\forall d, a_1, a_2(\mathit{Activity}(a_1) \wedge \mathit{Activity}(a_2) \wedge \mathit{hasDecomposition}(a_1, d) \\ \wedge \mathit{compositionOf}(d, a_2) \rightarrow \mathit{hasSubActivity}(a_1, a_2))\end{aligned} \tag{55}$$

hasSubActivity and *subActivityOf* are inverse relations:

$$\forall a_1, a_2(\mathit{hasSubActivity}(a_1, a_2) \leftrightarrow \mathit{subActivityOf}(a_2, a_1)) \tag{56}$$

hasFirstActivity and *hasLastActivity* connects an activity decomposition with an activity:

$$\begin{aligned}\forall d, a(\mathit{hasFirstActivity}(d, a) \rightarrow \mathit{ActivityDecomposition}(d) \\ \wedge \mathit{Activity}(a))\end{aligned} \tag{57}$$

$$\forall d, a(\mathit{hasLastActivity}(d, a) \rightarrow \mathit{ActivityDecomposition}(d) \wedge \mathit{Activity}(a)) \tag{58}$$

hasFirstActivity and *firstActivityOf* are inverse relations. *hasLastActivity* and *lastActivityOf* are inverse relations:

$$\forall d, a(\mathit{hasFirstActivity}(d, a) \leftrightarrow \mathit{firstActivityOf}(a, d)) \tag{59}$$

$$\forall d, a(\mathit{hasLastActivity}(d, a) \leftrightarrow \mathit{lastActivityOf}(a, d)) \tag{60}$$

hasSuccessor and *hasPredecessor* connect an activity with another activity:

$$\begin{aligned}\forall a_1, a_2(\mathit{hasSuccessor}(a_1, a_2) \rightarrow \mathit{Activity}(a_1) \\ \wedge \mathit{Activity}(a_2) \wedge (a_1 \neq a_2))\end{aligned} \tag{61}$$

$$\begin{aligned}\forall a_1, a_2(\mathit{hasPredecessor}(a_1, a_2) \rightarrow \mathit{Activity}(a_1) \\ \wedge \mathit{Activity}(a_2) \wedge (a_1 \neq a_2))\end{aligned} \tag{62}$$

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

$$\forall d, a_1(\mathit{hasFirstActivity}(d, a_1) \rightarrow \neg(\exists a_2(\mathit{hasPredecessor}(a_1, a_2))) \tag{63}$$

$$\forall d, a_1(\mathit{hasLastActivity}(d, a_1) \rightarrow \neg(\exists a_2(\mathit{hasSuccessor}(a_1, a_2))) \tag{64}$$

An activity cannot be both a successor and predecessor of another activity:

$$\forall a_1, a_2(\mathit{hasSuccessor}(a_1, a_2) \rightarrow \neg \mathit{hasPredecessor}(a_1, a_2)) \tag{65}$$

$$\forall a_1, a_2(\mathit{hasPredecessor}(a_1, a_2) \rightarrow \neg \mathit{hasSuccessor}(a_1, a_2)) \tag{66}$$

hasSuccessor and *hasPredecessor* are inverse relations:

$$\forall a_1, a_2(\mathit{hasSuccessor}(a_1, a_2) \leftrightarrow \mathit{hasPredecessor}(a_2, a_1)) \tag{67}$$

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

$$\forall a_1, a_2, r(subActivityOf(a_2, a_1) \land requires(a_2, r) \to requires(a_1, r) \tag{68}$$

$$\forall a_1, a_2, r(subActivityOf(a_2, a_1) \land produces(a_2, r) \to produces(a_1, r) \tag{69}$$

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

$$\begin{gathered}\forall a_1, d, r(hasDecomposition(a_1, d) \land requires(a_1, r) \\ \to \exists a_2(hasSubActivity(a_1, a_2) \land requires(a_2, d)))\end{gathered} \tag{70}$$

$$\begin{gathered}\forall a_1, d, r(hasDecomposition(a_1, d) \land produces(a_1, r) \\ \to \exists a_2(hasSubActivity(a_1, a_2) \land produces(a_2, d)))\end{gathered} \tag{71}$$

3.3 Activity Characteristics

hasActivityCharacteristic connects an activity with an activity characteristic associated with it:

$$\begin{gathered}\forall a, c(hasActivityCharacteristic(a, c) \to Activity(a) \\ \land ActivityCharacteristic(c))\end{gathered} \tag{72}$$

hasActivityCharacteristic and *activityCharacteristicOf* are inverse relation:

$$\begin{gathered}\forall a, c(hasActivityCharacteristic(a, c) \\ \leftrightarrow activityCharacteristicOf(c, a))\end{gathered} \tag{73}$$

Each *ActivityCharacteristic* must be tied to a specific *Activity*:

$$\forall c(ActivityCharacteristic(c) \to \exists a(activityCharacteristicOf(c, a))) \tag{74}$$

Each *ActivityCharacteristic* belongs to exactly one *Activity*:

$$\begin{gathered}\forall a_1, a_2, c(hasActivityCharacteristic(a_1, c) \\ \land hasActivityCharacteristic(a_2, c) \land Activity(a_1) \\ \land Activity(a_2) \to (a_1 = a_2))\end{gathered} \tag{75}$$

SpatialCharacteristic, *SpatialCharacteristic*, *InputCharacteristic*, *ProcessCharacteristic*, and *OutputCharacteristic* are types of activity characteristics:

$$\forall c(TemporalCharacteristic(c) \to ActivityCharacteristic(c)) \tag{76}$$

$$\forall c(SpatialCharacteristic(c) \to ActivityCharacteristic(c)) \tag{77}$$

$$\forall c(InputCharacteristic(c) \to ActivityCharacteristic(c)) \tag{78}$$

$$\forall c(ProcessCharacteristic(c) \to ActivityCharacteristic(c)) \tag{79}$$

$$\forall c(OutputCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{80}$$

Start, *End*, and *Duration* are types of temporal characteristics:

$$\forall c(Start(c) \rightarrow TemporalCharacteristic(c)) \tag{81}$$

$$\forall c(End(c) \rightarrow TemporalCharacteristic(c)) \tag{82}$$

$$\forall c(Duration(c) \rightarrow TemporalCharacteristic(c)) \tag{83}$$

Location is a type of spatial characteristics:

$$\forall c(Location(c) \rightarrow SpatialCharacteristic(c)) \tag{84}$$

Material, *ResourceConsumption*, and *Cost* are types of input characteristics:

$$\forall c(Material(c) \rightarrow InputCharacteristic(c)) \tag{85}$$

$$\forall c(ResourceConsumption(c) \rightarrow InputCharacteristic(c)) \tag{86}$$

$$\forall c(Cost(c) \rightarrow InputCharacteristic(c)) \tag{87}$$

Implementation and *Method* are types of process characteristics:

$$\forall p(Implementation(c) \rightarrow ProcessCharacteristic(c)) \tag{88}$$

$$\forall p(Method(c) \rightarrow ProcessCharacteristic(c)) \tag{89}$$

Quantity, *Quality*, and *Design* are types of output characteristics:

$$\forall p(Quantity(c) \rightarrow OutputCharacteristic(c)) \tag{90}$$

$$\forall p(Quality(c) \rightarrow OutputCharacteristic(c)) \tag{91}$$

$$\forall p(Design(c) \rightarrow OutputCharacteristic(c)) \tag{92}$$

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

$$\begin{gathered}\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1) \\ \wedge hasActivityCharacteristic(a, c_2) \\ \wedge((Start(c_1) \wedge Start(c_2)) \vee (End(c_1) \wedge End(c_2)) \\ \vee(Duration(c_1) \wedge Duration(c_2))) \rightarrow (c_1 = c_2))\end{gathered} \tag{93}$$

$$\begin{gathered}\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1) \\ \wedge hasActivityCharacteristic(a, c_2) \\ \wedge((Design(c_1) \wedge Design(c_2)) \vee (Quality(c_1) \wedge Quality(c_2)) \\ \vee(Quantity(c_1) \wedge Quantity(c_2))) \rightarrow (c_1 = c_2))\end{gathered} \tag{94}$$

$$
\begin{gathered}
\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1) \\
\wedge hasActivityCharacteristic(a, c_2) \\
\wedge((Material(c_1) \wedge Material(c_2)) \\
\vee(ResourceConsumption(c_1) \wedge ResourceConsumption(c_2)) \\
\vee(Cost(c_1) \wedge Cost(c_2))) \rightarrow (c_1 = c_2))
\end{gathered} \tag{95}
$$

$$
\begin{gathered}
\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1) \\
\wedge hasActivityCharacteristic(a, c_2) \\
\wedge(Location(c_1) \wedge Location(c_2)) \rightarrow (c_1 = c_2))
\end{gathered} \tag{96}
$$

$$
\begin{gathered}
\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1) \\
\wedge hasActivityCharacteristic(a, c_2) \\
\wedge((Method(c_1) \wedge Method(c_2)) \\
\vee(Implementation(c_1) \wedge Implementation(c_2))) \\
\rightarrow (c_1 = c_2))
\end{gathered} \tag{97}
$$

TemporalCharacteristic, *SpatialCharacteristic*, *InputCharacteristic*, *ProcessCharacteristic*, and *OutputCharacteristic* are disjoint classes:

$$
\begin{gathered}
\forall c(\neg((TemporalCharacteristic(c) \wedge SpatialCharacteristic(c)) \\
\vee(TemporalCharacteristic(c) \wedge InputCharacteristic(c)) \\
\vee(TemporalCharacteristic(c) \wedge ProcessCharacteristic(c)) \\
\vee(TemporalCharacteristic(c) \wedge OutputCharacteristic(c)) \\
\vee(SpatialCharacteristic(c) \wedge InputCharacteristic(c)) \\
\vee(SpatialCharacteristic(c) \wedge ProcessCharacteristic(c)) \\
\vee(SpatialCharacteristic(c) \wedge OutputCharacteristic(c)) \\
\vee(InputCharacteristic(c) \wedge ProcessCharacteristic(c)) \\
\vee(InputCharacteristic(c) \wedge OutputCharacteristic(c)) \\
\vee(ProcessCharacteristic(c) \wedge OutputCharacteristic(c))))
\end{gathered} \tag{98}
$$

Start, *End*, and *Duration* are disjoint classes:

$$
\begin{gathered}
\forall c(\neg((Start(c) \wedge Duration(c)) \vee (End(c) \wedge Duration(c)) \\
\vee(Start(c) \wedge End(c)))
\end{gathered} \tag{99}
$$

Design, *Quality*, and *Quantity* are disjoint classes:

$$
\begin{gathered}
\forall c(\neg((Design(c) \wedge Quality(c)) \vee (Design(c) \wedge Quantity(c)) \\
\vee(Quality(c) \wedge Quantity(c)))
\end{gathered} \tag{100}
$$

ResourceConsumption, *Material*, and *Cost* are disjoint classes:

$$
\begin{gathered}
\forall c(\neg((ResourceConsumption(c) \wedge Cost(c)) \vee (Cost(c) \wedge Material(c)) \\
\vee(Material(c) \wedge ResourceConsumption(c)))
\end{gathered} \tag{101}
$$

Method and *Implementation* are disjoint classes:

$$\forall c(\neg((Method(c) \wedge Implementation(c))) \tag{102}$$

4 Task Dependence

A *BasisOfDependence* is defined in terms of the dependum through which it exists:

$$\begin{gathered}\forall b(BasisOfDependence(b) \equiv \exists d(hasDependum(b,d) \\ \wedge(Activity(d) \vee State(d) \\ \vee ActivityCharacteristic(d) \vee StateCharacteristic(d) \\ \vee Resource(d))))\end{gathered} \tag{103}$$

hasDependum and *dependumOf* are inverse relations:

$$\forall b, d(hasDependum(b,d) \leftrightarrow dependumOf(d,b)) \tag{104}$$

A basis of dependence has exactly one dependum:

$$\forall b, d_1, d_2(hasDependum(b,d_1) \wedge hasDependum(b,d_2) \rightarrow (d_1 = d_2)) \tag{105}$$

constrains and *alteredBy* connects a basis of dependence with either a task, activity, or activity characteristic:

$$\begin{gathered}\forall b, x(constrains(b,x) \rightarrow BasisOfDependence(b) \\ \wedge(Task(x) \vee Activity(x) \vee ActivityCharacteristic(x)))\end{gathered} \tag{106}$$

$$\begin{gathered}\forall b, x(alteredBy(b,x) \rightarrow BasisOfDependence(b) \\ \wedge(Task(x) \vee Activity(x) \vee ActivityCharacteristic(x)))\end{gathered} \tag{107}$$

constrains and *constrainedBy* are inverse relations. *alteredBy* and *alters* are inverse relations

$$\forall b, x(constrains(b,x) \leftrightarrow constrainedBy(x,b)) \tag{108}$$

$$\forall b, x(alteredBy(b,x) \leftrightarrow alters(x,b)) \tag{109}$$

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

$$\begin{gathered}\forall b(Availability(b) \rightarrow BasisOfDependence(b) \\ \wedge\exists r(Resource(r) \wedge hasDependum(b,r)))\end{gathered} \tag{110}$$

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

$$\begin{gathered}\forall b(Functionality(b) \rightarrow BasisOfDependence(b) \\ \wedge\exists a(Activity(a) \wedge hasDependum(b,a)))\end{gathered} \tag{111}$$

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

$$\begin{aligned}\forall b(Compatibility(b) \rightarrow BasisOfDependence(b)\\ \wedge \exists x((ActivityCharacteristic(x)\\ \vee StateCharacteristic(x)) \wedge hasDependum(b, s)))\end{aligned} \tag{112}$$

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

$$\begin{aligned}\forall b(Complementarity(b) \rightarrow BasisOfDependence(b)\\ \wedge \exists x((ActivityCharacteristic(x)\\ \vee StateCharacteristic(x)) \wedge hasDependum(b, s)))\end{aligned} \tag{113}$$

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

$$\begin{aligned}\forall b(Uncertainty(b) \rightarrow BasisOfDependence(b)\\ \wedge \exists c((ActivityCharacteristic(c)\\ \vee StateCharacteristic(c)) \wedge hasDependum(b, c)))\end{aligned} \tag{114}$$

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

$$\forall b(Complexity(b) \rightarrow BasisOfDependence(b)) \tag{115}$$

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

$$\begin{aligned}\forall x, y(dependsOn(x, y) \rightarrow ((Task(x) \wedge Task(y))\\ \vee(Activity(x) \wedge Activity(y))\\ \vee(ActivityCharacteristic(x) \wedge ActivityCharacteristic(x))))\end{aligned} \tag{116}$$

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

$$\begin{aligned}\forall b, a_1, a_2, c_1, c_2(hasActivityCharacteristic(a_1, c_1)\\ \wedge hasActivityCharacteristic(a_2, c_2)\\ \wedge constrains(b, c_2) \wedge alteredBy(b, c_1)\\ \wedge(c_1 \neq c_2) \rightarrow dependsOn(c_2, c_1))\end{aligned} \tag{117}$$

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

$$\begin{aligned}\forall a_1, a_2, c_1, c_2(Activity(a_1) \wedge Activity(a_2) \wedge (a_1 \neq a_2)\\ \wedge hasActivityCharacteristic(a_1, c_1)\\ \wedge hasActivityCharacteristic(a_2, c_2)\\ \wedge dependsOn(c_2, c_1) \rightarrow dependsOn(a_2, a_1))\end{aligned} \tag{118}$$

If an activity *dependsOn* 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:

$$\begin{aligned}\forall t_1, t_2, a_1, a_2 (Task(t_1) \wedge Task(t_2) \wedge hasActivity(t_1, a_1) \\ \wedge hasActivity(t_2, a_2) \wedge (t_1 \neq t_2) \\ \wedge dependsOn(a_2, a_1) \rightarrow dependsOn(t_2, t_1))\end{aligned} \tag{119}$$

An activity *dependsOn* 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:

$$\begin{aligned}\forall a_1, a_2, b, c (Activity(a_1) \wedge Activity(a_2) \wedge constrains(b, a_1) \\ \wedge alteredBy(b, c) \wedge hasActivityCharacteristic(a_2, c) \\ \wedge (a_1 \neq a_2) \rightarrow dependsOn(a_1, a_2))\end{aligned} \tag{120}$$

An activity *dependsOn* 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:

$$\begin{aligned}\forall a_1, a_2, b, c (Activity(a_1) \wedge Activity(a_2) \wedge constrains(b, c) \\ \wedge alteredBy(b, a_2) \wedge hasActivityCharacteristic(a_1, c) \\ \wedge (a_1 \neq a_2) \rightarrow dependsOn(a_1, a_2))\end{aligned} \tag{121}$$

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

$$\begin{aligned}\forall a_1, a_2, b (Activity(a_1) \wedge Activity(a_2) \wedge constrains(b, a_1) \\ \wedge alteredBy(b, a_2) \wedge (a_1 \neq a_2) \rightarrow dependsOn(a_1, a_2))\end{aligned} \tag{122}$$

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

$$\begin{aligned}\forall t_1, t_2, b (Task(t_1) \wedge Task(t_2) \wedge constrains(b, t_1) \wedge alteredBy(b, t_2) \\ \wedge (t_1 \neq t_2) \rightarrow dependsOn(t_1, t_2))\end{aligned} \tag{123}$$