Organizational Dependence Ontology

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

1 Conceptual Model

In this section we introduce our conceptual model (partially depicted in Figure 1), with each subsection covering a different layer. It is worth emphasizing that this model is not intended to exhaustively represent organizational structures, for which many frameworks exist (Vernadat, 2020). Rather, it is intended to serve as a higher-level model of the different types of dependencies and their relationship to coordination needs and cooperation risks. This conceptual model, and its formalization, can be extended with domain specific theories for a finer grained reasoning, based on specialized field knowledge.

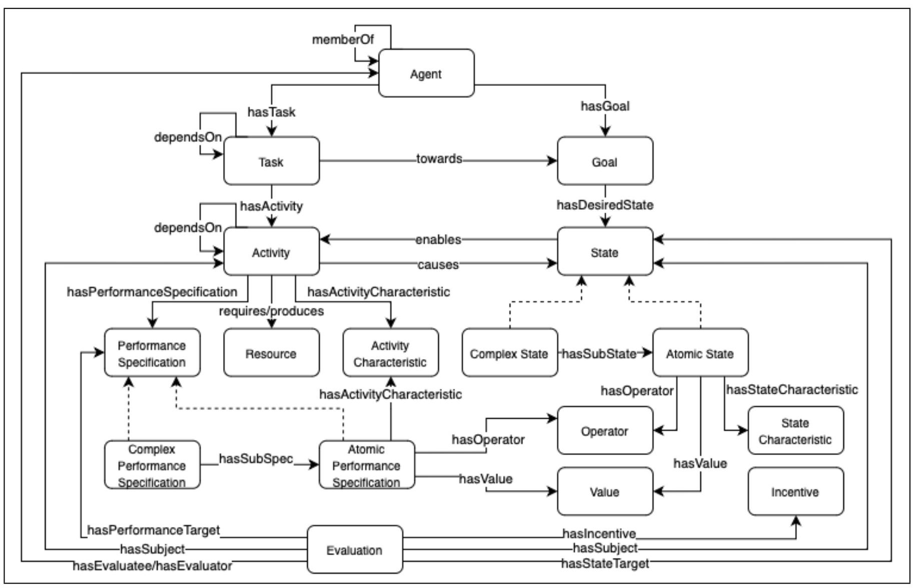

Figure 1: Simplified conceptual model. subClassOf relations shown as dashed arrows.

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

1.1 Agents, Task Structure, and Task Dependence

Central to our understanding of coordination and cooperation risks is the inherent task structure, which necessitates such coordination and cooperation. Within this framework, an Agent is an entity capable of possessing goals, intentions to strive towards them, and acting on those intention. An agent can either be an Individual or a Collective, which in itself might be composed of individuals or other collectives. A Task within our model denotes an agent's intention to contribute effort towards a goal. Conversely, a Goal represents a desired "state-of-affairs", defined in relation to the particular State that is desired. Tasks may also be specified in terms of the activities through which goals can be achieved. The difference between tasks and activities is subtle but significant. While a task encapsulates the intention to act towards a goal, an activity specifies the precise way in which an agent can act. This distinction enables us to comprehend coordination and cooperation risks across a spectrum of task environments (Raveendran et al., 2020) and addresses the problematic dual usage of the term "task" as the achievement of goals and performance of activities in the literature (Crowston, 1994; Chandrasekaran et al., 1998).

An Activity is 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 ActivityCharacteristic represents a particular dimension of the performance of an activity that is subject to variation (such as start time, quality or design of output, and location of execution). A PerformanceSpecification defines a particular specification of values for an activity's characteristics. It can be either a ComplexPerformanceSpecification or an AtomicPerformanceSpecification, where the former is described in terms of the conjunction or disjunction of other complex or atomic specifications, and the latter is expressed in terms of a single ActivityCharacteristic, Operator (e.g., $\leq$, $\geq$, $=$, $\neq$)., and Value combination. The value represents the specific value of the characteristic defined in an atomic specification. It can be based on a unit of measure, an ordinal or nominal scale, or any data type. An agent is a contributorTo an activity if they perform the activity and they are a soleContributorTo the activity if no other agent is a contributor to it. A State describes a particular aspect of an object or situation. Similarly to performance specification, It can be either a ComplexState (which can be a conjunction or disjunction of other states) or an AtomicState. An atomic state is defined in terms of a state characteristic, operator, and value combination. An agent is a contributorTo a state if they perform an activity that causes that state and they are a soleContributorTo the state if no other agent contributes to it.

A task dependsOn another task if the way in which the goal of the former task is achieved is constrained by the way in which the goal of the latter is achieved. An activity dependsOn another activity if the way in which the former is performed is constrained by latter. Note that these relations hold irrespective of the agents that have the tasks or perform the activities. The efforts of agents may in some cases be strategic complements or supplements. Two tasks or activities are strategicComplements through a state if their combined effects on a state can be more or less than the sum of their individual contributions. Two tasks or activities are strategicSubstitutes with respect to a state if they mutually offset the other (i.e. the value of one with respect to its effect on the state reduces the value of the other).

dependsOn, strategicComplements, and strategicSubstitutes relationships can be directly asserted by experts, or inferred automatically based on sufficient conditions implemented by extensions with domain-specific knowledge.

1.2 Evaluations and Incentives

Underpinning the notions of outcome, reward, and epistemic dependence are the concepts of evaluation and incentives. An Evaluation represents an instance of measurement or appraisal of the work of agents. It is defined in terms of an evaluatee(s) (the agent whose work is being evaluated), evaluator(s) (the agent who is evaluating), the target of evaluation, and the subject of evaluation. The target of evaluation is the specific standard, benchmark, or expectation that the evaluatees are being measured against. A target can be a state or performance specification, which is akin to outcome and behaviour controls, respectively (Ouchi, 1979), allowing for a flexible representation of a wide variety of evaluation schemes in organizations. Each evaluation has exactly one target associated with it. The subject of evaluation is the particular efforts of an evaluatee which is being evaluated with respect to the target. Subjects can be either an activity performed by the evaluatee(s) or a state that is caused by the performance of their activity(s). Said differently, an evaluatee is answerable to an evaluator for the extent to which a target is satisfied, and the evaluation may consist of examining particular efforts of the evaluatee to assess their contribution to the target. An evaluation may also have an Incentive associated with it. An incentive can be either a Reward or a Sanction. It is worth noting at this point that we only aim to capture "extrinsic" performance-based rewards and sanctions, such as performance bonuses and performance improvement plans. In this model, incentives are necessarily tied to evaluations since they are means for encouraging the contribution of effort towards performance or state targets, therefore these incentives are administered based on an appraisal of an evaluatee's efforts.

1.3 Dependence, Coordination, and Cooperation

We may now proceed with operationalizing the notions of outcome, epistemic, and reward dependence:

- **Outcome dependence**: Two agents are outcome interdependent if they both share in an evaluation. Two agents share in an evaluation when they are both individual evaluatees of the same Evaluation, or if they are both members of a collective which is itself the evaluatee of an evaluation.

- **Epistemic dependence**: Agent A is epistemically dependent on Agent B if A is the recipient of an incentive for an evaluation where their work is the subject of evaluation and the work depends on the work of Agent B.

Since Agent A's incentive is contingent upon their performance on their work, and their work is constrained by the way in which B performs their work, to perform optimally (i.e. performing in a way that increases their position to recieve their reward), A must know how B will act on their work.

- **Reward dependence**: In the symmetric case, two agents are reward interdependent when they are outcome interdependent through an evaluation that has an incentive that they are both recipients of. In the asymmetric case, Agent A is reward dependent on Agent B if A is epistemically dependent on B, since B's performance on their work constrains A's performance on their work, influencing whether A receives their incentive or not.

Based on the arrangement of the three types of dependencies between agents, we may now represent the need to coordinate and the risks for cooperation failures between them. A coordinationNeed exists between two agents when at least one is epistemically dependent on the other, since there is a requirement for at least one agent to have knowledge about the actions of the other to perform their work optimally. A cooperationRisk exists between two agents when there is a risk for free-riding, shirking, or sub-goal optimization between them. A free-riding risk exists when an agent is an evaluatee of an evaluation that includes other evaluatees yet there is no subject of evaluation that the agent solely contributes to. This can lead to a cooperation risk if at least one agent in an evaluation has a free-ride risk with that evaluation. A shirking risk exists when an agent has a task or activity for which there is no evaluation. This poses a cooperation risk when one agent is epistemically dependent on another, yet there is a shirk risk between the latter agent and the work that the former agent's work depends on. Finally, sub-goal optimization risks occur when two agents have tasks or activities that are strategic complements, there are evaluations for their individual work, yet there does not exist an evaluation for the complementary state. This may cause a cooperation risk since neither of the agents may be motivated to exert efforts towards the complementary state since they are only being evaluated on their individual contributions.

2 Formal Model

2.1 Agents, Tasks, and Goals

An *Agent* can be either a *Collective* or an *Individual*:

$$\forall a(Agent(a) \rightarrow Collective(a) \lor Individual(a)) \tag{1}$$

An *Agent* can only be a member of a *Collective*:

$$\forall a_1, a_2(memberOf(a_1, a_2) \rightarrow (Agent(a_1) \land Collective(a_2))) \tag{2}$$

An *Agent* cannot be a member of itself:

$$\forall a_1, a_2(memberOf(a_1, a_2) \rightarrow (a_1 \neq a_2)) \tag{3}$$

memberOf and *hasMember* are inverse relations:

$$\forall a_1, a_2(memberOf(a_1, a_2) \leftrightarrow hasMember(a_2, a_1)) \tag{4}$$

An *Agent* cannot have itself as a member:

$$\forall a_1, a_2(hasMember(a_1, a_2) \rightarrow (a_1 \neq a_2)) \tag{5}$$

hasTask connects agents with tasks:

$$\forall a, t(hasTask(a, t) \rightarrow (Agent(a) \wedge Task(t))) \tag{6}$$

hasGoal connects agents with goals:

$$\forall a, g(hasGoal(a, g) \rightarrow (Agent(a) \wedge Goal(g))) \tag{7}$$

hasTask and *taskOf* are inverse relations:

$$\forall a, t(hasTask(a, t) \leftrightarrow taskOf(t, a)) \tag{8}$$

hasGoal and *goalOf* are inverse relations:

$$\forall a, g(hasGoal(a, g) \leftrightarrow goalOf(g, a)) \tag{9}$$

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

$$\begin{aligned} &\forall a(Agent(a) \rightarrow (\exists t, g(Task(t) \wedge Goal(g) \\ &\wedge hasTask(a, t) \wedge hasGoal(a, g)))) \end{aligned} \tag{10}$$

towards connects *Tasks* with *Goals*:

$$\forall t, g(towards(t, g) \rightarrow (Task(t) \wedge Goal(g))) \tag{11}$$

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:

$$\forall t(Task(t) \rightarrow (\exists g(towards(t, g)))) \tag{12}$$

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

$$\forall t, a(hasActivity(t, a) \rightarrow (Task(t) \wedge Activity(a))) \tag{13}$$

hasActivity and *activityOf* are inverse relations:

$$\forall t, a(hasActivity(t, a) \leftrightarrow activityOf(a, t)) \tag{14}$$

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

$$\begin{aligned} &\forall g(Goal(g) \equiv (\exists a, s(Agent(a) \wedge State(s) \\ &\wedge goalOf(g, a) \wedge hasDesiredState(g, s)))) \end{aligned} \tag{15}$$

hasDesiredState connects goals and (desired) states:

$$\forall g, s(hasDesiredState(g, s) \rightarrow (Goal(g) \wedge State(s))) \tag{16}$$

hasDesiredState and *desiredStateOf* are inverse relations:

$$\forall g, s(hasDesiredState(g, s) \leftrightarrow desiredStateOf(s, g)) \tag{17}$$

2.2 States

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

$$\begin{aligned}\forall s_1(ComplexState(s_1) \rightarrow \exists s_2(State(s_2)\\ \wedge hasSubState(s_1, s_2) \wedge (s_1 \neq s_2)))\end{aligned} \tag{18}$$

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

$$\forall s(ComplexState(s) \rightarrow (ConjunctiveState(s) \vee DisjunctiveState(s))) \tag{19}$$

$$\forall s_1, s_2((ConjunctiveState(s_1) \wedge (DisjunctiveState(s_2)) \rightarrow (s_1 \neq s_2))) \tag{20}$$

An *AtomicState* cannot have a substate:

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

A *State* is complex or atomic:

$$\forall s(State(s) \rightarrow (ComplexState(s) \vee AtomicState(s))) \tag{22}$$

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

$$\forall s_1, s_2(ComplexState(s_1) \wedge AtomicState(s_2) \rightarrow (s_1 \neq s_2)) \tag{23}$$

hasSubState connects a state to another state:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow (State(s_1) \wedge State(s_2))) \tag{24}$$

A state cannot be a substate of itself:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow (s_1 \neq s_2)) \tag{25}$$

hasSubState and *subStateOf* are inverse relations:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \leftrightarrow subStateOf(s_2, s_1)) \tag{26}$$

A state cannot have itself as a substate:

$$\forall s_1, s_2(subStateOf(s_1, s_2) \rightarrow (s_1 \neq s_2)) \tag{27}$$

A state cannot be a substate of its substate:

$$\forall s_1, s_2(hasSubState(s_1, s_2) \rightarrow \neg subStateOf(s_1, s_2)) \tag{28}$$

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

$$\begin{aligned}\forall s(AtomicState(s) \equiv \exists c, o, v(StateCharacteristic(c) \wedge Operator(o)\\ \wedge Value(v) \wedge hasStateCharacteristic(s, c)\\ \wedge hasOperator(s, o) \wedge hasValue(s, v)))\end{aligned} \tag{29}$$

hasStateCharacteristic and *stateCharacteristicOf* are inverse relationships:

$$\forall s, c(hasStateCharacteristic(s, c) \rightarrow stateCharacteristicOf(c, s)) \tag{30}$$

hasOperator and *operatorOf* are inverse relationships:

$$\forall s, o(hasOperator(s, o) \rightarrow operatorOf(o, s)) \tag{31}$$

hasValue and *valueOf* are inverse relationships:

$$\forall s, v(hasValue(s, v) \rightarrow valueOf(v, s)) \tag{32}$$

2.3 Activities and Resources

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

$$\forall a(Activity(a) \rightarrow \exists s(causes(a, s))) \tag{33}$$

causes connects activities and states:

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

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

$$\forall a, g(performedBy(a, g) \rightarrow (Activity(a) \wedge Agent(g))) \tag{35}$$

activityOf connects activities and tasks.

$$\forall a, t(activityOf(a, t) \rightarrow (Activity(a) \wedge Task(t))) \tag{36}$$

causes and *causedBy* are inverse relationships:

$$\forall a, s(causes(a, s) \leftrightarrow causedBy(s, a)) \tag{37}$$

performedBy and *performs* are inverse relationships:

$$\forall a, g(performedBy(a, g) \leftrightarrow performs(g, a)) \tag{38}$$

activityOf and *hasActivity* are inverse relationships:

$$\forall a, t(activityOf(a, t) \leftrightarrow hasActivity(t, a)) \tag{39}$$

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

$$\forall s, act(causedBy(s, act) \rightarrow Outcome(s)) \tag{40}$$

enables connects states with the activities they enable.

$$\forall a, s(enables(s, a) \rightarrow (State(s) \wedge Activity(a))) \tag{41}$$

enables and *enabledBy* are inverse relations:

$$\forall a, s(enables(s, a) \leftrightarrow enabledBy(a, s)) \tag{42}$$

2.3.1 Resources

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

$$\forall r(Resource(r) \leftrightarrow \exists a(requires(a, r) \vee produces(a, r))) \tag{43}$$

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

$$\forall a, r(requires(a, r) \rightarrow Activity(a) \wedge Resource(r)) \tag{44}$$

$$\forall a, r(produces(a, r) \rightarrow Activity(a) \wedge Resource(r)) \tag{45}$$

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

$$\forall r, a(requires(a, r) \leftrightarrow requiredBy(r, a))) \tag{46}$$

$$\forall r, a(produces(a, r) \leftrightarrow producedBy(r, a))) \tag{47}$$

A resource can either be shareable or nonshareable:

$$\begin{gathered}\forall r(Resource(r) \rightarrow ShareableResource(r) \\ \vee NonShareableResource(r)))\end{gathered} \tag{48}$$

NonShareableResource and *ShareableResource* are disjoint classes:

$$\begin{gathered}\forall r_1, r_2(ShareableResource(r_1) \wedge NonShareableResource(r_2) \\ \rightarrow (r_1 \neq r_2))\end{gathered} \tag{49}$$

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

$$\begin{gathered}\forall r(Resource(r) \rightarrow ConsumableResource(r) \\ \vee NonConsumableResource(r)))\end{gathered} \tag{50}$$

ConsumableResource and *NonConsumableResource* are disjoint classes:

$$\begin{gathered}\forall r_1, r_2(ConsumableResource(r_1) \wedge NonConsumableResource(r_2) \\ \rightarrow (r_1 \neq r_2))\end{gathered} \tag{51}$$

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

$$\forall a, r(uses(a, r) \rightarrow NonConsumableResource(r) \wedge requires(a, r)) \tag{52}$$

$$\forall a, r(consumes(a, r) \rightarrow ConsumableResource(r) \wedge requires(a, r)) \tag{53}$$

2.3.2 Activity Decomposition

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

$$\begin{gathered}\forall d(ActivityDecomposition(d) \rightarrow \exists a_1, a_2, a_3(Activity(a_1) \wedge Activity(a_2) \\ \wedge Activity(a_3) \wedge decompositionOf(d, a_1) \wedge compositionOf(d, a_2) \\ \wedge CompositionOf(d, a_3) \wedge (a_1 \neq a_2) \wedge (a_2 \neq a_3) \wedge (a_1 \neq a_3)))\end{gathered} \tag{54}$$

decompositionOf and *compositionOf* connect activity decompositions with activities:

$$\begin{gathered}\forall d, a(decompositionOf(d, a) \rightarrow ActivityDecomposition(d) \\ \wedge Activity(a))\end{gathered} \tag{55}$$

$$\forall d, a(compositionOf(d, a) \rightarrow ActivityDecomposition(d) \wedge Activity(a)) \tag{56}$$

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

$$\forall d, a(decompositionOf(d, a) \leftrightarrow hasDecomposition(a, d)) \tag{57}$$

$$\forall d, a(compositionOf(d, a) \leftrightarrow hasComposition(a, d)) \tag{58}$$

hasSubActivity connects an activity with another activity:

$$\begin{gathered}\forall a_1, a_2(hasSubActivity(a_1, a_2) \rightarrow Activity(a_1) \\ \wedge Activity(a_2) \wedge (a_1 \neq a_2))\end{gathered} \tag{59}$$

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

$$\begin{gathered}\forall d, a_1, a_2(Activity(a_1) \wedge Activity(a_2) \wedge hasDecomposition(a_1, d) \\ \wedge compositionOf(d, a_2) \rightarrow hasSubActivity(a_1, a_2))\end{gathered} \tag{60}$$

hasSubActivity and *subActivityOf* are inverse relations:

$$\forall a_1, a_2(hasSubActivity(a_1, a_2) \leftrightarrow subActivityOf(a_2, a_1)) \tag{61}$$

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

$$\begin{gathered}\forall d, a(hasFirstActivity(d, a) \rightarrow ActivityDecomposition(d) \\ \wedge Activity(a))\end{gathered} \tag{62}$$

$$\forall d, a(hasLastActivity(d, a) \rightarrow ActivityDecomposition(d) \wedge Activity(a)) \tag{63}$$

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

$$\forall d, a(hasFirstActivity(d, a) \leftrightarrow firstActivityOf(a, d)) \tag{64}$$

$$\forall d, a(hasLastActivity(d, a) \leftrightarrow lastActivityOf(a, d)) \tag{65}$$

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

$$\forall a_1, a_2(hasSuccessor(a_1, a_2) \rightarrow Activity(a_1) \land Activity(a_2) \land (a_1 \neq a_2)) \tag{66}$$

$$\begin{gathered}\forall a_1, a_2(hasPredecessor(a_1, a_2) \rightarrow Activity(a_1) \\ \land Activity(a_2) \land (a_1 \neq a_2))\end{gathered} \tag{67}$$

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

$$\forall d, a_1(hasFirstActivity(d, a_1) \rightarrow \neg(\exists a_2(hasPredecessor(a_1, a_2))) \tag{68}$$

$$\forall d, a_1(hasLastActivity(d, a_1) \rightarrow \neg(\exists a_2(hasSuccessor(a_1, a_2))) \tag{69}$$

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

$$\forall a_1, a_2(hasSuccessor(a_1, a_2) \rightarrow \neg hasPredecessor(a_1, a_2)) \tag{70}$$

$$\forall a_1, a_2(hasPredecessor(a_1, a_2) \rightarrow \neg hasSuccessor(a_1, a_2)) \tag{71}$$

hasSuccessor and *hasPredecessor* are inverse relations:

$$\forall a_1, a_2(hasSuccessor(a_1, a_2) \leftrightarrow hasPredecessor(a_2, a_1)) \tag{72}$$

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

$$\forall a_1, a_2, r(subActivityOf(a_2, a_1) \land requires(a_2, r) \rightarrow requires(a_1, r) \tag{73}$$

$$\forall a_1, a_2, r(subActivityOf(a_2, a_1) \land produces(a_2, r) \rightarrow produces(a_1, r) \tag{74}$$

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

$$\begin{gathered}\forall a_1, d, r(hasDecomposition(a_1, d) \land requires(a_1, r) \\ \rightarrow \exists a_2(hasSubActivity(a_1, a_2) \land requires(a_2, d)))\end{gathered} \tag{75}$$

$$\begin{gathered}\forall a_1, d, r(hasDecomposition(a_1, d) \land produces(a_1, r) \\ \rightarrow \exists a_2(hasSubActivity(a_1, a_2) \land produces(a_2, d)))\end{gathered} \tag{76}$$

2.3.3 Activity Characteristics and Performance Specification

hasActivityCharacteristic and *activityCharacteristicOf* are inverse relation:

$$\begin{gathered}\forall a, c(hasActivityCharacteristic(a, c) \\ \leftrightarrow activityCharacteristicOf(c, a))\end{gathered} \tag{77}$$

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

$$\forall c(ActivityCharacteristic(c) \rightarrow \exists a(activityCharacteristicOf(c, a))) \tag{78}$$

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

$$\forall a_1, a_2, c(hasActivityCharacteristic(a_1, c) \wedge hasActivityCharacteristic(a_2, c) \\ \wedge Activity(a_1) \wedge Activity(a_2) \rightarrow (a_1 = a_2)) \tag{79}$$

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

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

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

$$\forall c(InputCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{82}$$

$$\forall c(ProcessCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{83}$$

$$\forall c(OutputCharacteristic(c) \rightarrow ActivityCharacteristic(c)) \tag{84}$$

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

$$\forall c(Start(c) \rightarrow TemporalCharacteristic(c)) \tag{85}$$

$$\forall c(End(c) \rightarrow TemporalCharacteristic(c)) \tag{86}$$

$$\forall c(Duration(c) \rightarrow TemporalCharacteristic(c)) \tag{87}$$

Location is a type of spatial characteristics:

$$\forall c(Location(c) \rightarrow SpatialCharacteristic(c)) \tag{88}$$

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

$$\forall c(Material(c) \rightarrow InputCharacteristic(c)) \tag{89}$$

$$\forall c(ResourceConsumption(c) \rightarrow InputCharacteristic(c)) \tag{90}$$

$$\forall c(Cost(c) \rightarrow InputCharacteristic(c)) \tag{91}$$

Implementation and *Method* are types of process characteristics:

$$\forall p(Implementation(c) \rightarrow ProcessCharacteristic(c)) \tag{92}$$

$$\forall p(Method(c) \rightarrow ProcessCharacteristic(c)) \tag{93}$$

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

$$\forall p(Quantity(c) \rightarrow OutputCharacteristic(c)) \tag{94}$$

$$\forall p(Quality(c) \rightarrow OutputCharacteristic(c)) \tag{95}$$

$$\forall p(Design(c) \rightarrow OutputCharacteristic(c)) \tag{96}$$

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

$$\begin{aligned}&\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1)\\&\wedge hasActivityCharacteristic(a, c_2)\\&\wedge((Start(c_1) \wedge Start(c_2)) \vee (End(c_1) \wedge End(c_2))\\&\vee(Duration(c_1) \wedge Duration(c_2))) \rightarrow (c_1 = c_2))\end{aligned} \tag{97}$$

$$\begin{aligned}&\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1)\\&\wedge hasActivityCharacteristic(a, c_2)\\&\wedge((Design(c_1) \wedge Design(c_2)) \vee (Quality(c_1) \wedge Quality(c_2))\\&\vee(Quantity(c_1) \wedge Quantity(c_2))) \rightarrow (c_1 = c_2))\end{aligned} \tag{98}$$

$$\begin{aligned}&\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1)\\&\wedge hasActivityCharacteristic(a, c_2)\\&\wedge((Material(c_1) \wedge Material(c_2))\\&\vee(ResourceConsumption(c_1) \wedge ResourceConsumption(c_2))\\&\vee(Cost(c_1) \wedge Cost(c_2))) \rightarrow (c_1 = c_2))\end{aligned} \tag{99}$$

$$\begin{aligned}&\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1)\\&\wedge hasActivityCharacteristic(a, c_2)\\&\wedge(Location(c_1) \wedge Location(c_2)) \rightarrow (c_1 = c_2))\end{aligned} \tag{100}$$

$$\begin{aligned}&\forall a, c_1, c_2(hasActivityCharacteristic(a, c_1)\\&\wedge hasActivityCharacteristic(a, c_2) \wedge ((Method(c_1) \wedge Method(c_2))\\&\vee(Implementation(c_1) \wedge Implementation(c_2))) \rightarrow (c_1 = c_2))\end{aligned} \tag{101}$$

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

$$\begin{aligned}&\forall c(\neg((TemporalCharacteristic(c) \wedge SpatialCharacteristic(c))\\&\vee(TemporalCharacteristic(c) \wedge InputCharacteristic(c))\\&\vee(TemporalCharacteristic(c) \wedge ProcessCharacteristic(c))\\&\vee(TemporalCharacteristic(c) \wedge OutputCharacteristic(c))\\&\vee(SpatialCharacteristic(c) \wedge InputCharacteristic(c))\\&\vee(SpatialCharacteristic(c) \wedge ProcessCharacteristic(c))\\&\vee(SpatialCharacteristic(c) \wedge OutputCharacteristic(c))\\&\vee(InputCharacteristic(c) \wedge ProcessCharacteristic(c))\\&\vee(InputCharacteristic(c) \wedge OutputCharacteristic(c))\\&\vee(ProcessCharacteristic(c) \wedge OutputCharacteristic(c))))\end{aligned} \tag{102}$$

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

$$\begin{aligned}&\forall c(\neg((Start(c) \wedge Duration(c)) \vee (End(c) \wedge Duration(c))\\&\vee(Start(c) \wedge End(c)))\end{aligned} \tag{103}$$

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

$$\begin{aligned}\forall c(\neg((Design(c) \wedge Quality(c)) \vee (Design(c) \wedge Quantity(c))\\ \vee(Quality(c) \wedge Quantity(c)))\end{aligned} \tag{104}$$

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

$$\begin{aligned}\forall c(\neg((ResourceConsumption(c) \wedge Cost(c)) \vee (Cost(c) \wedge Material(c))\\ \vee(Material(c) \wedge ResourceConsumption(c)))\end{aligned} \tag{105}$$

Method and *Implementation* are disjoint classes:

$$\forall c(\neg((Method(c) \wedge Implementation(c))) \tag{106}$$

hasPerformanceSpecification connects an activity with a performance specification:

$$\begin{aligned}\forall a, p(hasPerformanceSpecification(a, p)\\ \to Activity(a) \wedge PerformanceSpecification(p))\end{aligned} \tag{107}$$

hasPerformanceSpecification and *performanceSpecificationOf* are inverse relations:

$$\begin{aligned}\forall a, p(hasPerformanceSpecification(a, p)\\ \leftrightarrow performanceSpecificationOf(p, a))\end{aligned} \tag{108}$$

A *PerformanceSpecification* can be either be complex (*ComplexPerfSpec*) or atomic (*AtomicPerfSpec*), both of which are disjoint subclasses:

$$\begin{aligned}\forall p(PerformanceSpecification(p)\\ \to ComplexPerfSpec(p) \vee AtomicPerfSpec(p))\end{aligned} \tag{109}$$

$$\forall p_1, p_2(ComplexPerfSpec(p_1) \wedge AtomicSpec(p_2) \to (p_1 \neq p_2)) \tag{110}$$

A *ComplexPerfSpec* has at least two subspecifications:

$$\begin{aligned}\forall p_1(ComplexPerfSpec(p_1) \to \exists p_2, p_3(hasSubSpec(p_1, p_2)\\ \wedge hasSubSpec(p_1, p_3) \wedge (p_2 \neq p_3)))\end{aligned} \tag{111}$$

hasSubSpec connects a performance specification with another performance specification that it contains.

$$\begin{aligned}\forall p_1, p_2(hasSubSpec(p_1, p_2) \to ComplexPerfSpec(p_1)\\ \wedge PerformanceSpecification(p_2)))\end{aligned} \tag{112}$$

A performance specification cannot contain itself as a subspecification:

$$\forall p_1, p_2(hasSubSpec(p_1, p_2) \to (p_1 \neq p_2)) \tag{113}$$

hasSubSpec and *subSpecOf* are inverse relations:

$$\forall p_1, p_2(hasSubSpec(p_1, p_2) \leftrightarrow subSpecOf(p_2, p_1))) \tag{114}$$

A performance specification cannot be a subspecification of itself:

$$\forall p_1, p_2(subSpecOf(p_1, p_2) \rightarrow (p_1 \neq p_2)) \tag{115}$$

An *AtomicSpec* is defined in terms of an *ActivityCharacteristic*, *Operator*, and *Value*:

$$\begin{aligned}\forall p(AtomicSpec(p) \leftrightarrow \exists c, o, v(hasActivityCharacteristic(p, c)\\ \wedge hasOperator(p, o) \wedge hasValue(p, v))))\end{aligned} \tag{116}$$

2.3.4 Contribution, Strategic Substitutes, and Strategic Complements

If an *Agent performs* an *Activity*, then they are a *contributorTo* it.

$$\forall a_1, act_1(performs(a_1, act_1) \rightarrow contributorTo((a_1, act_1))) \tag{117}$$

If an *Agent performs* an *Activity* that *causes* a *State*, then they are a *contributorTo* the *State*:

$$\begin{aligned}\forall a_1, act_1, s(performs(a_1, act_1) \wedge causes(act_1, s)\\ \rightarrow contributorTo((a_1, s)))\end{aligned} \tag{118}$$

An *Agent* is a *soleContributorTo* to something if no other agent contributes to it:

$$\begin{aligned}\forall a_1, x(contributorTo(a_1, x) \wedge \neg(\exists a_2(contributorTo(a_2, x) \wedge (a_1 \neq a_2)))\\ \rightarrow soleContributorTo(a_1, x))\end{aligned} \tag{119}$$

contributorTo connects agents with the activity or state they contribute to.

$$\forall a, x(contributorTo(a, x) \rightarrow Agent(a) \wedge (Activity(x) \vee State(x))) \tag{120}$$

soleContributorTo connects an agent with an activity, or state, they are the only contributors to.

$$\forall a, x(soleContributorTo(a, x) \rightarrow Agent(a) \wedge (Activity(x) \vee State(x))) \tag{121}$$

strategicComplements is a ternary relation connecting two tasks, or two activities, and a state:

$$\begin{aligned}\forall x, y, s(strategicCompliments(x, y, s) \rightarrow ((Activity(x) \wedge Activity(y))\\ \vee(Task(x) \wedge Task(y))) \wedge State(s) \wedge (x \neq y))\end{aligned} \tag{122}$$

strategicSubstitutes is a ternary relation between two tasks or two activities, and a state:

$$\begin{aligned}\forall x, y, s(strategicSubstitutes(x, y, s) \rightarrow ((Activity(x) \wedge Activity(y))\\ \vee(Task(x) \wedge Task(y))) \wedge State(s) \wedge (x \neq y))\end{aligned} \tag{123}$$

2.4 Task Dependence

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

$$\begin{gathered}\forall b(BasisOfDependence(b) \equiv \exists d(hasDependum(b,d) \wedge (Activity(d) \\ \vee State(d) \vee ActivityCharacteristic(d) \\ \vee StateCharacteristic(d) \vee Resource(d))))\end{gathered} \tag{124}$$

hasDependum and *dependumOf* are inverse relations:

$$\forall b, d(hasDependum(b,d) \leftrightarrow dependumOf(d,b)) \tag{125}$$

A basis of dependence has exactly one dependum:

$$\forall b, d_1, d_2(hasDependum(b,d_1) \wedge hasDependum(b,d_2) \rightarrow (d_1 = d_2)) \tag{126}$$

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

$$\begin{gathered}\forall b, x(constrains(b,x) \rightarrow BasisOfDependence(b) \\ \wedge(Task(x) \vee Activity(x) \vee ActivityCharacteristic(x)))\end{gathered} \tag{127}$$

$$\begin{gathered}\forall b, x(alteredBy(b,x) \rightarrow BasisOfDependence(b) \\ \wedge(Task(x) \vee Activity(x) \vee ActivityCharacteristic(x)))\end{gathered} \tag{128}$$

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

$$\forall b, x(constrains(b,x) \leftrightarrow constrainedBy(x,b)) \tag{129}$$

$$\forall b, x(alteredBy(b,x) \leftrightarrow alters(x,b)) \tag{130}$$

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

$$\begin{gathered}\forall b(Availability(b) \rightarrow BasisOfDependence(b) \\ \wedge \exists r(Resource(r) \wedge hasDependum(b,r)))\end{gathered} \tag{131}$$

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

$$\begin{gathered}\forall b(Functionality(b) \rightarrow BasisOfDependence(b) \\ \wedge \exists a(Activity(a) \wedge hasDependum(b,a)))\end{gathered} \tag{132}$$

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

$$\begin{gathered}\forall b(Compatibility(b) \rightarrow BasisOfDependence(b) \\ \wedge \exists x((ActivityCharacteristic(x) \\ \vee StateCharacteristic(x)) \wedge hasDependum(b,s)))\end{gathered} \tag{133}$$

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

$$\begin{gathered}\forall b(Complementarity(b) \rightarrow BasisOfDependence(b) \\ \wedge \exists x((ActivityCharacteristic(x) \vee StateCharacteristic(x)) \\ \wedge hasDependum(b, s)))\end{gathered} \tag{134}$$

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

$$\begin{gathered}\forall b(Uncertainty(b) \rightarrow BasisOfDependence(b) \\ \wedge \exists c(ActivityCharacteristic(c) \vee StateCharacteristic(c) \\ \wedge hasDependum(b, c)))\end{gathered} \tag{135}$$

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

$$\forall b(Complexity(b) \rightarrow BasisOfDependence(b)) \tag{136}$$

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

$$\begin{gathered}\forall x, y(dependsOn(x, y) \rightarrow ((Task(x) \wedge Task(y)) \\ \vee(Activity(x) \wedge Activity(y)) \\ \vee(ActivityCharacteristic(x) \wedge ActivityCharacteristic(x))))\end{gathered} \tag{137}$$

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:

$$\begin{gathered}\forall b, a_1, a_2, c_1, c_2(hasActivityCharacteristic(a_1, c_1) \\ \wedge hasActivityCharacteristic(a_2, c_2) \wedge constrains(b, c_2) \\ \wedge alteredBy(b, c_1) \wedge (c_1 \neq c_2) \rightarrow dependsOn(c_2, c_1))\end{gathered} \tag{138}$$

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:

$$\begin{gathered}\forall a_1, a_2, c_1, c_2(Activity(a_1) \wedge Activity(a_2) \wedge (a_1 \neq a_2) \\ \wedge hasActivityCharacteristic(a_1, c_1) \\ \wedge hasActivityCharacteristic(a_2, c_2) \\ \wedge dependsOn(c_2, c_1) \rightarrow dependsOn(a_2, a_1))\end{gathered} \tag{139}$$

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:

$$\begin{gathered}\forall t_1, t_2, a_1, a_2(Task(t_1) \wedge Task(t_2) \wedge hasActivity(t_1, a_1) \\ \wedge hasActivity(t_2, a_2) \wedge (t_1 \neq t_2) \wedge dependsOn(a_2, a_1) \\ \rightarrow dependsOn(t_2, t_1))\end{gathered} \tag{140}$$

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:

$$\begin{aligned}\forall a_1, a_2, b, c(\mathit{Activity}(a_1) \wedge \mathit{Activity}(a_2) \wedge \mathit{constrains}(b, a_1) \\ \wedge \mathit{alteredBy}(b, c) \wedge \mathit{hasActivityCharacteristic}(a_2, c) \\ \wedge (a_1 \neq a_2) \rightarrow \mathit{dependsOn}(a_1, a_2))\end{aligned} \tag{141}$$

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:

$$\begin{aligned}\forall a_1, a_2, b, c(\mathit{Activity}(a_1) \wedge \mathit{Activity}(a_2) \wedge \mathit{constrains}(b, c) \\ \wedge \mathit{alteredBy}(b, a_2) \wedge \mathit{hasActivityCharacteristic}(a_1, c) \\ \wedge (a_1 \neq a_2) \rightarrow \mathit{dependsOn}(a_1, a_2))\end{aligned} \tag{142}$$

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:

$$\begin{aligned}\forall a_1, a_2, b(\mathit{Activity}(a_1) \wedge \mathit{Activity}(a_2) \wedge \mathit{constrains}(b, a_1) \\ \wedge \mathit{alteredBy}(b, a_2) \wedge (a_1 \neq a_2) \rightarrow \mathit{dependsOn}(a_1, a_2))\end{aligned} \tag{143}$$

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

$$\begin{aligned}\forall t_1, t_2, b(Task(t_1) \wedge Task(t_2) \wedge \mathit{constrains}(b, t_1) \wedge \mathit{alteredBy}(b, t_2) \\ \wedge (t_1 \neq t_2) \rightarrow \mathit{dependsOn}(t_1, t_2))\end{aligned} \tag{144}$$

2.5 Evaluations and Incentives

An *Evaluation* is defined in terms of its evaluatee(s), evaluator(s), and target of evaluation:

$$\begin{aligned}\forall e(Evaluation(e) \leftrightarrow \exists t, a_1, a_2(hasEvaluatee(e, a_1) \\ \wedge hasEvaluator(e, a_2) \wedge hasTarget(e, t)))\end{aligned} \tag{145}$$

hasEvaluatee connects an *Evaluation* with an *Agent* it has as an evaluatee:

$$\forall e, a(hasEvaluatee(e, a) \rightarrow Evaluation(e) \wedge Agent(a)) \tag{146}$$

hasEvaluator connects an *Evaluation* with an *Agent* it has as an evaluator:

$$\forall e, a(hasEvaluator(e, a) \rightarrow Evaluation(e) \wedge Agent(a)) \tag{147}$$

hasTarget connects an evaluation with a target, that can be either a *State* or *PerformanceSpecification*:

$$\begin{aligned}\forall e, t(hasTarget(e, t) \rightarrow Evaluation(e) \wedge (State(t) \\ \vee PerformanceSpecification(t)))\end{aligned} \tag{148}$$

hasStateTarget is a restriction of relation *hasTarget* where the target must be a *State*:

$$\forall e, t(hasStateTarget(e, t) \rightarrow hasTarget(e, t) \wedge State(t)) \tag{149}$$

hasPerformanceTarget is a restriction of relation *hasTarget* where the target must be a *PerformanceSpecification*:

$$\begin{gathered}\forall e, t(hasPerformanceTarget(e, t) \rightarrow hasTarget(e, t) \\ \wedge PerformanceSpecification(t))\end{gathered} \tag{150}$$

An *Evaluation* has exactly one target:

$$\forall e, t_1, t_2(hasTarget(e, t_1) \wedge hasTarget(e, t_2) \rightarrow (t_1 = t_2)) \tag{151}$$

hasSubject connects an *Evaluation* with an *Activity*, or *State*, that is the subject of the evaluation:

$$\forall e, s(hasSubject(e, s) \rightarrow Evaluation(e) \wedge (State(s) \vee Activity(s))) \tag{152}$$

An *Incentive* is defined in terms of the *Evaluation* that forms the basis of its provision:

$$\forall i(Incentive(i) \leftrightarrow \exists e(hasIncentive(e, i))) \tag{153}$$

An *Incentive* can either be a *Reward* or *Sanction*:

$$\forall i(Incentive(i) \rightarrow Reward(i) \vee Sanction(i)) \tag{154}$$

Rewards and *Sanctions* are disjoint classes:

$$\forall i_1, i_2(Reward(i_1) \wedge Sanction(i_2) \rightarrow (i_1 \neq i_2)) \tag{155}$$

hasIncentive connects an evaluation with an incentive:

$$\forall e, i(hasIncentive(e, i) \rightarrow Evaluation(e) \wedge Incentive(i)) \tag{156}$$

hasRecipient connects an incentive with an agent that receives it:

$$\forall i, a(hasRecipient(i, a) \rightarrow Incentive(i) \wedge Agent(a)) \tag{157}$$

hasAdministrator connects an incentive with an agent that administers it:

$$\forall i, a(hasAdministrator(i, a) \rightarrow (Incentive(i) \wedge Agent(a))) \tag{158}$$

An *Evaluation* can have at most a single *Incentive*:

$$\forall e, i_1, i_2(hasIncentive(e, i_1) \wedge hasIncentive(e, i_2) \rightarrow (i_1 = i_2)) \tag{159}$$

2.6 Outcome, Reward, and Epistemic Dependence

outcomeDependentOn connects two agents with an evaluation:

$$\begin{aligned}\forall e, a_1, a_2(&outcomeDependentOn(a_1, a_2, e)\\ &\rightarrow (Agent(a_1) \wedge Agent(a_2) \wedge Evaluation(e)))\end{aligned} \tag{160}$$

Two *Agents* are outcome dependent through an *Evaluation* if they are both evaluatees of the evaluation:

$$\begin{aligned}\forall e, a_1, a_2((&hasEvaluatee(e, a_1) \wedge hasEvaluatee(e, a_2) \wedge (a_1 \neq a_2))\\ &\rightarrow outcomeDependentOn(a_1, a_2, e))\end{aligned} \tag{161}$$

Two *Agents* are outcome dependent through an *Evaluation* if they are both members of a *Collective* that is itself an evaluaatee of the evaluation:

$$\begin{aligned}&\forall e, a_1, a_2, a_3((hasEvaluatee(e, a_1) \wedge memberOf(a_2, a_1)\\ &\wedge memberOf(a_3, a_1) \wedge (a_1 \neq a_2)) \rightarrow outcomeDependentOn(a_2, a_3, e))\end{aligned} \tag{162}$$

outcomeDependentOn is symmetric in its first two arguments:

$$\begin{aligned}\forall a_1, a_2, e(&outcomeDependentOn(a_1, a_2, e)\\ &\leftrightarrow outcomeDependentOn(a_2, a_1, e))\end{aligned} \tag{163}$$

An *Agent* cannot be *outcomeDependentOn* on itself:

$$\forall a_1, a_2, e(outcomeDependentOn(a_1, a_2, e) \rightarrow (a_1 \neq a_2)) \tag{164}$$

predictiveNeed connects two agents and two actions:

$$\begin{aligned}\forall a_1, a_2, act_1, act_2(&predictiveNeed(a_1, a_2, act_1, act_2)\\ &\rightarrow (Agent(a_1) \wedge Agent(a_2) \wedge Activity(act_1) \wedge Activity(act_2)))\end{aligned} \tag{165}$$

An *Agent* has a *predictiveNeed* of another *Agent* if the former *performs* an *Activity* that depends on an *Activity* that the latter *performs*:

$$\begin{aligned}\forall a_1, a_2, act_1, act_2((&performs(a_1, act_1) \wedge performs(a_2, act_2)\\ &\wedge dependsOn(act_1, act_2)) \rightarrow predictiveNeed(a_1, a_2, act_1, act_2))\end{aligned} \tag{166}$$

Two agents in a *predictiveNeed* need relation must be unique:

$$\forall a_1, a_2, act_1, act_2(predictiveNeed(a_1, a_2, act_1, act_2) \rightarrow (a_1 \neq a_2)) \tag{167}$$

epistemicallyDependentOn connects two agents and an evaluation:

$$\begin{aligned}\forall a_1, a_2, e(&epistemicallyDependentOn(a_1, a_2, e)\\ &\rightarrow (Agent(a_1) \wedge Agent(a_2) \wedge Evaluation(e)))\end{aligned} \tag{168}$$

An agent is *epistemicallyDependentOn* another agent through an evaluation if the former agent is an evaluatee of the evaluation, is the recipient of an incentive associated with the evaluation, and performs an activity that they have a predictive need of the latter agent for, and either the activity or its outcome is the subject of evaluation:

$$\begin{gathered}\forall a_1, a_2, act_1, act_2, e, i(predictiveNeed(a_1, a_2, act_1, act_2) \\ \wedge hasEvaluatee(e, a_1) \wedge hasIncentive(e, i) \\ \wedge hasRecipient(i, a_1) \wedge (hasSubject(e, act_1) \\ \vee \exists s(hasSubject(e, s) \wedge causes(act_1, s))) \\ \rightarrow epistemicallyDependentOn(a_1, a_2, e))\end{gathered} \tag{169}$$

An *Agent* cannot be *epistemicallyDependentOn* itself:

$$\forall a_1, a_2, e(epistemicallyDependentOn(a_1, a_2, e) \rightarrow (a_1 \neq a_2)) \tag{170}$$

rewardDependentOn connects two agents and an evaluation:

$$\begin{gathered}\forall e, a_1, a_2(rewardDependentOn(a_1, a_2, e) \rightarrow Agent(a_1) \\ \wedge Agent(a_2) \wedge Evaluation(e))\end{gathered} \tag{171}$$

Two agents are *rewardDependentOn* each other through an evaluation if they are outcome dependent on each other through that evaluation, and the evaluation has an incentive associated with it that they are both recipients of:

$$\begin{gathered}\forall e, i, a_1, a_2((outcomeDependentOn(a_1, a_2, e) \wedge hasIncentive(e, i) \\ \wedge recipientOf(a_1, i) \wedge recipientOf(a_2, i)) \\ \rightarrow (rewardDependentOn(a_1, a_2, e) \wedge rewardDependentOn(a_2, a_1, e)))\end{gathered} \tag{172}$$

If an agent is *epistemicallyDependentOn* another agent through an evaluation, then they are also *rewardDependentOn* them through the same evaluation:

$$\begin{gathered}\forall a_1, a_2, e(epistemicallyDependentOn(a_1, a_2, e) \\ \rightarrow rewardDependentOn(a_1, a_2, e))\end{gathered} \tag{173}$$

An *Agent* cannot be *rewardDependentOn* itself:

$$\forall a_1, a_2, e(rewardDependentOn(a_1, a_2, e) \rightarrow (a_1 \neq a_2)) \tag{174}$$

2.7 Coordination Needs and Cooperation Risks

coordinationRequirement connects agents, and only agents:

$$\forall a_1, a_2(coordinationRequirement(a_1, a_2) \rightarrow (Agent(a_1) \wedge Agent(a_2))) \tag{175}$$

A *coordinationRequirement* between two agents exists when at least one of them is epistemically dependent on the other:

$$\begin{gathered}\forall a_1, a_2(coordinationRequirement(a_1, a_2) \leftrightarrow \\ \exists e(epistemicallyDependentOn(a_1, a_2, e) \\ \vee epistemicallyDependentOn(a_2, a_1, e)))\end{gathered} \tag{176}$$

coordinationRequirement is a symmetric relation:

$$\begin{aligned}\forall a_1, a_2(coordinationRequirement(a_1, a_2)\\ \rightarrow coordinationRequirement(a_2, a_1))\end{aligned} \tag{177}$$

coordinationRequirement is a non-reflexive relation:

$$\forall a_1, a_2(coordinationRequirement(a_1, a_2) \rightarrow (a_1 \neq a_2)) \tag{178}$$

freeRideRisk can exist only between an agent and an evaluation:

$$\forall a(freeRideRisk(a, e) \rightarrow Agent(a) \wedge Evaluation(e)) \tag{179}$$

A *freeRideRisk* exists between an agent and an evaluation when the agent is an evaluatee of an evaluation that includes other evaluatees yet there is no subject of evaluation that the former agent solely contributes to:

$$\begin{gathered}\forall a_1, a_2, e(outcomeDependentOn(a_1, a_2, e)\\ \wedge\neg(\exists x(hasSubject(e, x) \wedge soleContributorTo(a_1, x)))\\ \rightarrow freeRideRisk(a_1, e))\end{gathered} \tag{180}$$

A *freeRideRisk* exists between an agent and an evaluation when the agent is an evaluatee of an evaluation that includes another evaluatee where they both have activities that are subjects of evaluation and the activities are strategic substitutes:

$$\begin{gathered}\forall a_1, a_2, act_1, act_2, s, e(performs(a_1, act_1) \wedge performs(a_2, act_2)\\ \wedge hasEvaluatee(e, a_1) \wedge hasEvaluatee(e, a_2)\\ \wedge hasSubject(e, act_1) \wedge hasSubject(e, act_2)\\ \wedge strategicSubstitutes(act_1, act_2, s) \rightarrow freeRideRisk(a_1, e))\end{gathered} \tag{181}$$

shirkRisk connects an agent and either a task or activity:

$$\forall a, x(shirkRisk(a, x) \rightarrow Agent(a) \wedge (Task(x) \vee Activity(x))) \tag{182}$$

A *shirkRisk* exists between an agent and an activity when an agent performs a task and there does not exist an evaluation where the agent is an evaluatee and the target of evaluation is a performance target that is a performance specification of the activity:

$$\begin{gathered}\forall a, act(performs(a, act) \wedge \neg(\exists e, pt(hasEvaluatee(e, a)\\ \wedge hasPerformanceTarget(e, pt)\\ \wedge performanceSpecificationOf(p, act))) \rightarrow shirkRisk(a, act))\end{gathered} \tag{183}$$

A *shirkRisk* exists between an agent and a task when the agent has a task towards a goal, and there does not exist an evaluation where the agent is an evaluatee and the target of evaluation is the desired state of the goal:

$$\begin{gathered}\forall a, t, g, s(hasTask(a, t) \wedge towards(t, g) \wedge hasDesiredState(g, s)\\ \wedge\neg(\exists e(hasEvaluatee(e, a) \wedge hasStateTarget(e, s))) \rightarrow shirkRisk(a, t))\end{gathered} \tag{184}$$

subGoalOptRisk connects two agents and a state:

$$\forall a_1, a_2, s(subGoalOptRisk(a_1, a_2, s) \to Agent(a_1) \\ \wedge Agent(a_2) \wedge State(s)) \tag{185}$$

A *subGoalOptRisk* risk exists between two agents and a state when each agent performs an activity that causes a state, the activities are strategic compliments with respect to another state, each agent is an evaluatee of an evaluation that has a state target for the respective outcomes of each activity, yet there does not exist an evaluation for the complementary state:

$$\forall a_1, a_2, act_1, act_2, g, e_1, e_2, s_1, s_2, s_3(hasDesiredState(g, s_3) \\ \wedge performs(a_1, act_1) \wedge performs(a_2, act_2) \wedge causes(act_1, s_1) \\ \wedge causes(act_2, s_2) \wedge strategicComplements(act_1, act_2, s_3) \\ \wedge hasEvaluatee(e_1, a_1) \wedge hasEvaluatee(e_2, a_2) \\ \wedge hasStateTarget(e_1, s_1) \wedge hasStateTarget(e_2, s_2) \\ \wedge \neg(\exists e_3(hasStateTarget(e_3, s_3))) \to subGoalOptRisk(a_1, a_2, s_3)) \tag{186}$$

A *subGoalOptRisk* cannot exist between and agent and themselves:

$$\forall a_1, a_2, s(subGoalOptRisk(a_1, a_2, s) \to (a_1 \neq a_2)) \tag{187}$$

A *cooperationRisk* can exist only between two agents:

$$\forall a_1, a_2(cooperationRisk(a_1, a_2) \to Agent(a_1) \wedge Agent(a_2)) \tag{188}$$

A *cooperationRisk* can occur between the evaluatees of an evaluation when at least one of the evaluatees has a free-riding risk with that evaluation:

$$\forall a_1, a_2, e(((hasEvaluatee(e, a_1) \wedge hasEvaluatee(e, a_2)) \\ \vee(\exists a_3(hasEvaluatee(e, a_3) \wedge memberOf(a_1, a_3) \\ \wedge memberOf(a_2, a_3)))) \wedge (freeRideRisk(a_1, e) \\ \vee freeRideRisk(a_2, e)) \to cooperationRisk(a_1, a_2)) \tag{189}$$

A *cooperationRisk* can exist between two agents when one agent epistemically depends on another through an evaluation, and the evaluation has a subject that the latter agent can shirk on:

$$\forall a_1, a_2, e(epistemicallyDependentOn(a_1, a_2, e) \wedge hasSubject(e, x) \\ \wedge shirkRisk(a_2, x) \to cooperationRisk(a_1, a_2)) \tag{190}$$

A *cooperationRisk* can exist between two agents when there is a risk of sub-goal optimization between them:

$$\forall a_1, a_2, s(subGoalOptRisk(a_1, a_2, s) \to cooperationRisk(a_1, a_2)) \tag{191}$$

cooperationRisk is a symmetric relation:

$$\forall a_1, a_2(cooperationRisk(a_1, a_2) \leftrightarrow cooperationRisk(a_2, a_1)) \tag{192}$$

cooperationRisk is a non-reflexive relation:

$$\forall a_1, a_2(cooperationRisk(a_1, a_2) \to (a_1 \neq a_2)) \tag{193}$$

References

Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1998). The ontology of tasks and methods.

Crowston, K. (1994). A taxonomy of organizational dependencies and coordination mechanisms. Technical Report 174, Massachusetts Institute of Technology.

Ouchi, W. G. (1979). A Conceptual Framework for the Design of Organizational Control Mechanisms. *Management Science*, 25(9):833–848.

Raveendran, M., Silvestri, L., and Gulati, R. (2020). The Role of Interdependence in the Micro-Foundations of Organization Design: Task, Goal, and Knowledge Interdependence. *Academy of Management Annals*, 14(2):828–868.

Vernadat, F. (2020). Enterprise modelling: Research review and outlook. *Computers in Industry*, 122:103265.