DELIVERABLE

Task Taxonomies for Knowledge Content
D07

Task Number - Name	**T31, T21, T22, T23, T24**
Issuing Date	**ongoing, 2005**
Distribution	**Public**
Document Web Link	
File name	**D07_v21.pdf**
Author/Editor	**Aldo Gangemi, Stefano Borgo, Carola Catenacci, Jos Lehmann**
QA Mentor	
Lead/Contact Person	**Aldo Gangemi**

Change History

Version	Status	Comments / Changes
2.1	post-final-3	Update May 30th, 2005: - Debugging of reification semantics

2.0	post-final-2	Update May 4th, 2005: - Update of reification semantics - Corrections on axioms related to agentive notions - Some more clarification on inherited DOLCE notions - Updated OWL-DL version of DDPO
1.8	post-final	Update February 15th, 2005: - Addition of a reification semantics for D&S - Updates on FOL formalization - Migration of most control tasks to individuals - Updates to sample domain models - Updated OWL-DL version of DDPO
1.2	final	October 1st, 2004: Signed off for delivery to project officer
1.0	wip	July 6th, 2004: Intermediate version

Table of Content

Executive Summary

1.1 Definition

This deliverable concerns the logical and ontological foundation of so-called *task taxonomies for knowledge content*. Firstly, we clarify a bit this concern.
Task taxonomies can be initially defined from a mathematical viewpoint as graphs that create an ordering over sets of action types. Task taxonomies are mainly used in so-called *workflow management systems* [12], [13].
Knowledge content objects can be initially defined as (materialized) information objects that are tagged by means of metadata, and are the target elements of library management and the Semantic Web.
In Metokis, the two concepts match for the scope of the project: building a demonstration platform that allows a formal definition of certain types (mainly digital) of knowledge content objects (news, clinical data, management documents, etc.), in conjunction with a formal definition of the action types (and their ordering) that involve that kind of knowledge content. The platform should enhance the manipulation and the management of content, specially within organizations, communities, intranets, etc. This objective clearly relates to the ongoing efforts for a Semantic Web, and to the semantic foundation of Web Services, but in Metokis we do not take such a widespread perspective.

1.2 Objective

The objective of this deliverable (across its evolution) is to explain how task taxonomies and knowledge content objects can be designed by means of logical languages and reusable conceptual models, called **formal ontologies**. It is also suggested how the models of tasks and contents can be grounded into system components.

1.3 Contents

This version of the deliverable contains a preliminary review of the related literature, the reusable formal ontologies that are used, a first version of the formal ontology of plans and tasks, an analysis of an example from the Klett and ORFG use cases, models that partly formalizes the examples by applying the ontology of plans, a preliminary ontology of information objects, and the machine readable code - the encoding language is OWL-DL - of the ontology library.
Future versions will include a more complete review of the literature, improved versions of the ontologies, and extended examples and best practices derived from the use cases.

1.3.1 Literature overview

The literature reviewed in this version of the deliverable only covers task-related work. The expected coverage includes process and plan ontologies (reviewed in this version), classical AI planning, the BDI (Belief-Desire-Intention) paradigm, MAS (Multi-Agent Systems), KQML (Knowledge Query and Manipulation Language), and workflow management systems.
Future versions will consider also the literature on content metadata and information objects.
Literature review is included in the deliverable for positioning the solutions proposed by Metokis, but it is not an objective *per se*.

1.3.2 Reusable ontologies

Ontologies for Metokis will be designed as an **ontology library**. This choice enables us to reuse existing components. In particular, the current library includes the DOLCE foundational ontology, some of its extensions for time, space, and information objects, and the DnS ontology. An experimental ontology of plans has been substantially reworked within Metokis, and it currently constitutes the major contribution of the deliverable.

1.3.3 Ontology of plans and tasks

Based on the abovementioned components, an experimental ontology of plans has been substantially reworked within Metokis, and it currently constitutes the major contribution of the deliverable.
Tasks are treated as concepts defined within plans, which refer to actions (e.g. "write a report"). Control constructs (e.g. "choose between the following alternatives", or "loop on the following") from traditional planning and workflows are represented as "control tasks", also defined within plans. Ordering of tasks is formalised by using part (*mereological*) relations, control tasks and a *successor* relation.
A rich subclass hierarchy of tasks has been developed by deepening these basic assumptions.
The ontology of plans and tasks aims at reaching an *intersubjective agreement* among the designers as well as the users of task taxonomies. Axioms for the definition of plan types, task types, goals, control constructs, etc. are provided, together with axioms for the automatic matching of plan executions against plan descriptions.
On the other hand, the *operationalization* of plans and tasks is left to appropriate choices among available information system components.

1.3.4 Sample models

This version includes two sample models representing a task (and role) taxonomy used by Klett, and a plan structure for the ORFG Business executives case study used by Templeton College, two of Metokis partners. Future versions will include other and extended examples, and best practices.

1.3.5 Ontology of information objects

Currently, the proposed ontology for information objects is adapted and improved from an extension of DOLCE, and is currently used for the KCO (Knowledge Content Objects) ontology.
Part of the reused ontology has been developed within the WonderWeb EU project [15], [17], [14], [10].
In order to put metadata on content, we need to know what kind of properties those metadata are talking about.
According to the reused ontology, content of any modality is assumed to be equivalent to information objects having the following properties: a *support*, one or more *combinatorial structure(s)*, a *meaning* and a *reference*.

1.3.6 Ontology grounding

Ontologies aim at an explicitation of the intended meaning, but running systems require that meaning to be operationalized into operations of a system. Operationalization is implemented (usually without the support of an explicit ontology) into workflow management systems and planners.
Within Metokis, the design choice is to have explicit detailed ontologies that are explicitly *grounded* into some information system specification language, e,g, WSDL, IDL, etc. Those

languages can be *executed*, then grounding results in having operational systems that implement requirements coming from ontologies.
The advantages of having explicit ontologies include: *i)* logical consistency, *ii)* conceptual transparency, *iii)* fair matching between user requirements and software design.

1.4 Versioning policy

This deliverable will have a versioning policy based on the evolution of the Metokis use cases and the synergies with other projects: each version will improve the previous one on these dimensions:

i) inclusion of examples of best practices from use cases
ii) changes in the ontology library, related to the needs from the use cases
iii) improvement of the literature review and of the state of art, specially from synergies with research groups from other projects (e.g. University of Karlsruhe from aceMedia, Knowledge Media Institute from DIP)

2 Literature Overview

2.1 General Problem Definition

Historically and theoretically speaking, Planning is a defining research problem for Artificial Intelligence (AI) and related disciplines (chiefly Robotics). For an insider view on this history see [1].
The 90s have seen a renewed interest in Planning from a Knowledge Representation perspective, more specifically from an Ontological perspective. The key assumptions behind this recent focus on Planning may be summed up in the following points:

- Planning is an important benchmark for any new approach to AI.
- Classical planning paradigms mainly suffer of lack of specificity, which hinders their useful application in real-life information systems -- scaling-up -- and, therefore, their proper testing.
- Ontology is a new approach to AI.
- Ontology may prove itself useful to AI by providing insights and solutions for the issues stated in the second point.

The application domains of ontological research in Planning have mainly been military and industrial ones, and this may probably be traced back to two reasons: both these domains seek strategic innovation by automation – thus making funding available -- and both these domains rely on highly structured workflows – thus providing to the proposed approaches a testing ground with a reduced complexity, therefore, making validation a little easier. It remains a question whether the focus on these types of applications has somehow restricted the “exportability” of the proposed approaches. This question may tentatively be answered in the negative, as most of the proposals are fairly well grounded in general knowledge models of what is a plan, which in turn guarantees a fair degree of applicability of most of the proposals in many different domains.

In the section 2.2, an overview is provided of the following Ontological approaches to Planning:

a. the BDI paradigm.
b. planners constructed through Problem Solving Methods.
c. the Act Formalism.
d. the Shared Planning and Activity Representation.
e. the Core Plan Representation.
f. the Process Specification Language.
g. the PLAN semantic NET.
h. the Enterprise Ontology.
i. the Ontology with Polymorphic Types.
j. the Core Ontology of Services.
k. an Ontological Formalization of the Planning Task.

Other approaches of (indirect) interest to Planning might be added to this review in the future (e.g., more Web-Services based approaches or Belief-Desire-Intention based approaches).
For each newly introduced approach, the following indications are provided:

1. a source document,
2. the motivation behind the considered approach,
3. the most significant part of the knowledge model formalized in the approach,

4. a tentative evaluation of the level of formalization of the approach. It should be noticed that, in order to perform the evaluation in a founded way, the level of formalization of an ontology should be measured on *at least* three axes indicating: how much of the ontology is implemented in a formal language; what is the expressivity of the implementation language of the ontology (ranging, for instance, from RDF schemas to higher order logics); how tractable/executable is the implemented ontology. At the moment, there are no stable methods for defining and combining these axes, though. It would require some significant research effort to come up with such a definition. But it is unclear whether this piece of research would directly be relevant to Metokis. Therefore, for the moment we only intuitively evaluate the level of formalization of a given approach on a (continuous) scale like: low, medium, high; where high (roughly) means that the considered ontology has been fully implemented, in an expressive and tractable/executable language.

Moreover, the convention is adopted that any notion defined in any given approach is written in CAPITAL letters.

Finally, in order to make it more accessible, all the material is presented in a non-formal fashion. The main purpose of this overview is to acquaint the reader with the terminology used and the concepts defined in the Ontological literature on Planning. Having a general, yet not too generic overview of the problems treated in the literature, may help the reading of Section 3, where the proposed Plan Ontology is introduced. The reader should become aware of the “minimal” set of notions usually targeted by research on Planning: plan, of course, but also activity, task, action, execution, strategy, schedule, service, world-state, state of affairs, objective, goal, desire, purpose, commitment, pre-condition, post-condition, constraint, resource, agent, role, risk, probability, capability, skill, cost, description, situation and a few more… By going through the existing sets of definitions of these, two threads should become apparent:

1. For most languages dedicated to Planning, internal consistency still is the main issue. Given the plethora of notions that may be involved in Planning, and given the usual computational restrictions, it is more desirable for a language to consistently cover a well defined portion of the conceptual space on Planning, rather than to cover it all and try to achieve conceptual completeness.
2. One of the most significant aspects according to which languages for Planning may be classified is their suitability for execution vs. representation. Executable languages (algebras) are conceptually simpler but also more keen to real-time control. On the contrary, rich languages provide conceptual sophistication but usually guarantee less control at execution.

Section 2.3 provides a preliminary presentation and positioning of DDPO (DOLCE+DnS Plan Ontology), the ontology of plans proposed in section 3.

2.2 Existing Approaches

2.2.1 BDI: Belief, Desire, Intention Approach

Source Document: A. S. Rao and M. P. Georgeff (1995) "BDI Agents: From Theory to Practice"

Motivation: The design of agent-based systems for high-level management and control tasks in complex dynamic environments. The approach is based on modal logic and attempts a different solution to the standard Artificial Intelligence and Decision Theory methods in planning since these are not apt for resource- and knowledge-bounded agents.

Knowledge Model: the BDI approach is based on three operators which are characterized by the standard KD45 modal system (but other formalizations have been exploited). The operators are: Bel (for belief), Des (for desire), and Int (for intention). These operators are related to each other by the notions of goal and commitment.

Level of Formalization: High.

2.2.2 PSM: Planners Constructed Through Problem Solving Methods

Source Document: Benjamins, R., Nunes de Barros, L., Valente, A., (1996) "Constructing Planners Through Problem-Solving Methods" available on website http://ksi.cpsc.ucalgary.ca/KAW/KAW96/benjamins/doc.html.

Motivation: To show how a general, knowledge-level framework for conceptually specifying knowledge-based systems, can be of concrete use to support knowledge acquisition for planning systems. The framework encompasses three interrelated components: (1) problem-solving methods, (2) their assumptions and (3) domain knowledge. The presented analysis of planning performed in the framework can be considered as a library with reusable components, based on which planners can be configured.

Knowledge Model: PSM's planning ontology is built around the following notions (informally grouped in Dynamic Roles, Static Roles and Basic PSMs):

-- Dynamic Roles in Planning

1. CURRENT STATE, a description of the world in the initial state.
2. GOAL, a description of the changes to the world that must be accomplished by the plan. The content of GOAL can be a set of conditions or a set of actions to be accomplished. Initially this role points to the original problem goal. During the planning process the content of the role is dynamically modified by establishing new subgoals and deleting achieved goals.
3. PLAN, the dynamic knowledge role PLAN is a composite role whose content is constantly modified during the planning process until a solution is found. It consists of the following:
    a. PLAN-STEPS which are the steps in the plan that correspond to actions in the domain.
    b. ORDERING CONSTRAINTS, over the plan-steps, such as that one action precedes another. The type of order imposed on the plan-steps in the plan (e.g., partial or total) depends on the static ROLE PLAN STRUCTURE (which will be described later) employed by the planner.
    c. VARIABLE BINDINGS CONSTRAINTS, which keep track of how variables of plan-steps are instantiated with domain knowledge such as objects, resources and agents.
    d. AUXILIARY CONSTRAINTS, that represent temporal and truth constraints between plan-steps and conditions. Auxiliary constraints are present in the plan only as support knowledge for the planning process. When a solution plan is found, they are no longer useful, unless the plan is to be reused. An example of an auxiliary constraint is a CAUSAL LINK, which is defined by (i) a condition in the plan that has to be true (e.g., a goal condition), (ii) a plan-step that needs

this condition to be true, and (iii) another plan-step that makes this condition true. Another example of an auxiliary constraint is POINT TRUTH CONSTRAINT, which requires that some condition be true before a certain plan-step can occur.

4. CONFLICT, contains the result of checking the plan for inconsistencies with respect to its conditions. Whenever a condition is unexpectedly false, a conflict is detected. The CONFLICT role can point directly to a plan-step that violates some interval of the truth value of a condition, or just point to a set of inconsistent constraints.

-- Static Roles in Planning

5. A PLAN MODEL defines what a plan is and what it is made of. It consists of two parts: the WORLD DESCRIPTION and the PLAN DESCRIPTION. Below is a brief description of these roles and their sub-roles.
6. WORLD DESCRIPTION, describes the world about which the planning is done and comprises two sub-roles: STATE DESCRIPTION and STATE CHANGES. The STATE DESCRIPTION contains the knowledge necessary to represent or describe the state of the world The STATE CHANGES role comprehends all the information connected to the specification of changes in the state of the world. This is also the specification of the elements a plan is composed of (but not how they are composed, see PLAN COMPOSITION below).
7. PLAN DESCRIPTION, describes the structure and features of the plan being generated and comprises two sub-roles: PLAN STRUCTURE and the (optional) PLAN ASSESSMENT KNOWLEDGE:
   a. PLAN STRUCTURE, this role specifies how the parts of a plan (actions, sub-plans) are assembled together. It also specifies (indirectly) how the plan is to be executed. There are several varieties in the structure of plans that can be identified in the literature. They can be described by two main knowledge roles: the PLAN COMPOSITION role contains the description of the plan with respect to how the state changes are arranged in order to make up a plan. This includes, for instance, whether the plan will be a partial or a total ordering of a set of state changes, or whether it includes iteration or conditional operators. The composition may also be hierarchical: plans are composed of SUB-PLANS, and so on up to ATOMIC plans, which are normally state changes. The STATE CHANGE DATA role contains the plan information besides the structure of state changes. For example, important state change data are interval constraints for binding the variables involved in the state changes. It is also possible to assign different RESOURCES to each state change or sub-plan. Two particularly important resources are agents and time.
   b. PLAN ASSESSMENT KNOWLEDGE determines whether a certain plan (or sub-plan) is valid (hard assessment knowledge), or whether a plan is better than another (soft). Based on this knowledge, a plan can be modified or criticized. An example of hard PLAN ASSESSMENT KNOWLEDGE is the TRUTH-CRITERION, which is used to find out if a condition is true at some point in the plan.

-- Basic Problem-Solving Methods for Planning

8. PROPOSE REFINEMENT This task has the goal of adding new steps or constraints to the plan. The input knowledge roles for this task are: WORLD DESCRIPTION, PLAN STRUCTURE and PLAN ASSESSMENT KNOWLEDGE. To realize this

task, there is a method called PROPOSE I, which can be decomposed into three sub-tasks: SELECT GOAL, PROPOSE EXPANSION AND TEST FOR UNACHIEVED GOALS.

a. SELECT GOAL, this task selects a goal from the set of goals to be accomplished. The goal can be either a goal state to be achieved or a goal action to be accomplished by a number of actions. For select goal, three methods can be used. LINEAR SELECT, RANDOM SELECT and SMART SELECT.
b. PROPOSE EXPANSION, this task takes the selected goal, and proposes a way to accomplish it using STATE CHANGES. This can be a new plan step, or an action decomposition which will be added to the plan at the place of the goal action. The propose expansion task can be realized by three alternative methods. DECOMPOSITION PROPOSE proposes a goal decomposition, which means to propose a more detailed way to accomplish the goal. This method is only applied if the goal is a goal action. GOAL-ACHIEVEMENT PROPOSE, selects an operator whose effect includes the selected goal, constraining the place of the operator in the plan to be necessarily before the selected goal. When a new operator is added to the plan, its preconditions are added to the set of goals. When there is already a step in the plan that achieves the goal, only the ordering constraint is added to the plan.
c. TEST FOR UNACHIEVED GOALS This task checks the current plan for unachieved goals, and records them in the dynamic role GOAL. It also tests whether the preconditions (sub-goals) of an operator are already achieved in the current state (when the plan composition is total-order). Three methods are identified to realize this task: the MTC-BASED GOAL-TEST, the CURRENT-STATE GOAL-TEST and the AGENDA-BASED GOAL TEST. Planners that exploit causal-links use the simple agenda-based method, because they only need to check for the existence of goals not yet processed; goals, once achieved, are preserved through the causal links.

9. CRITIQUE PLAN The critique plan task checks for conflicts and the quality of the plan generated so far, using plan assessment knowledge. The role PLAN ASSESSMENT KNOWLEDGE can point to `hard' constraints (interaction and the satisfiability of the plan constraints) and `soft' constraints (the factors that define when a given plan is better than another. One method is defined to realize this task which is called CRITIQUE I. This method consists of two subtasks: consistency critique and interaction critique.
   a. INTERACTION CRITIQUE When checking for conflicts, this task verifies whether the proposed action for accomplishing the goal would interact with other goals in the plan (e.g., one action might undo the precondition of another action). Note that this task involves explicit reasoning about interactions. For realizing the interaction critique task, two methods are identified: (i) the CAUSAL-LINK-BASED CRITIQUE, which checks if the proposed plan-step threats any existent causal link; (ii) and the MTC-BASED CRITIQUE, which uses the modal truth criterion to check the existence of a step that possibly deletes any achieved goal.
   b. CONSISTENCY CRITIQUE This task checks the consistency of the overall constraints on the plan generated so far. This task differs from the interaction critique in the sense that it can find more general conflicts between the constraints than the deleted-condition conflict. More complex planning

systems can also check the consistency of the assigned resources and agents. The CONSTRAINT PROPAGATION method is defined to realize this task.

10. MODIFY PLAN,The modify plan task is responsible for modifying the plan with respect to the results of the critique plan task (a conflict). By using plan assessment knowledge, a modification can be done by adding ordering, binding or secondary preconditions to the plan until the possible conflict (violation) is solved or avoided. Three methods are defined to realize this task:
    a. the CAUSAL-LINK-BASED
    b. the MTC-BASED method for partial-ordered plans, and
    c. the BACKTRACK MODIFICATION method for total-order plans.

Level of Formalization: Medium.

2.2.3 The Act Formalism

Source Document: Myers, K.L., Wilkins, D.E., (1997) "The Act Formalism Version 2.2" available on website http://www.ai.sri.com/~act/act-spec.pdf.

Motivation: Many domains in which AI planning techniques can be profitably employed are dynamic in nature. For example, military operations planning and controlling a mobile robot both exhibit this characteristic: during either plan generation or plan execution, the state of the world can change dramatically as troops are dispatched to an area or a robot navigates through a hallway. For such domains, it is necessary that plan generation systems be sensitive to run-time concerns and that plan execution systems be capable of invoking the plan generator to address unexpected events at run-time.

Knowledge Model: The basic unit of organization in the Act formalism is an ACT, which is further decomposed in the following main elements:
1. GOAL EXPRESSIONS.
2. ACT METAPREDICATES, such as ACHIEVE, ACHIEVE-BY, ACHIEVE-ALL, WAIT-UNTIL, TEST, CONCLUDE, RETRACT, REQUIRE-UNTIL, USE-RESOURCE.
3. ENVIRONMENT CONDITIONS, such as CUE, PRECONDITION, SETTING, RESOURCE,
4. PROPERTIES.
5. PLOTS.
6. TEMPORAL REASONING.
7. VARIABLES.

Roughly speaking, the Act formalism binds the terms listed above by seeing each ACT as describing a set of actions that can be taken to fulfill some designated purpose under certain conditions. The purpose could be either to satisfy a GOAL or to respond to some event in the world. An ACT can represent, among other things, a procedure, a planning "operator" or a plan at one particular level of detail. The purpose and applicability criteria for an ACT are formulated using a fixed set of ENVIRONMENT CONDITIONS. Action specifications are called the PLOT, and consist of a partially ordered set of actions and subgoals. The ENVIRONMENT CONDITIONS and PLOTS are specified using GOAL EXPRESSIONS, each of which consists of one of a predefined set of METAPREDICATES applied to a logical formula. The METAPREDICATES permit the specification of many different modes of activity, including goals of achievement, maintenance, and testing.

Level of Formalization: Medium/High.

2.2.4 SPAR: Shared Planning and Activity Representation

Source Document: Tate, A. (1998) "Roots of SPAR - Shared Planning and Activity Representation", The Knowledge Engineering Review, Vol. 13(1), pp. 121-128, Special Issue on "Putting Ontologies to Use" (eds. Uschold, M. and Tate, A.), Cambridge University Press.

Motivation: The Shared Planning and Activity Representation (SPAR) is intended to contribute to a range of purposes including domain modelling, plan generation, plan analysis, plan case capture, plan communication, behaviour modelling. By having a shared model of what constitutes a plan, process or activity, *organisational knowledge* can be harnessed and used effectively.

Knowledge Model: SPAR's top level is built around the following notions and statements:

1. A PLAN is a SPECIFICATION of ACTIVITY to meet one or more OBJECTIVES.
2. A SPECIFICATION of ACTIVITY denotes or describes one or more ACTIVITIES.
3. An ACTIVITY may change the STATES-OF-AFFAIRS.
4. STATES-OF-AFFAIRS is something that can be evaluated as holding or not.
5. An AGENT can perform ACTIVITIES and/or hold OBJECTIVES.
6. An OBJECTIVE may have one or more EVALUATION-CRITERIA.
7. An EVALUATION-CRITERION is an ASPECT of STATES-OF-AFFAIRS or an ASPECT of PLANS.
8. An EVALUATION is a predicate (holds/does not hold) or a preference ranking on EVALUATION-CRITERIA.
9. An ACTIVITY takes place over a TIME-INTERVAL identified by its two ends, the BEGIN-TIME-POINT and the END-TIME-POINT. The BEGIN-TIME-POINT is temporally before the END-TIME-POINT.
10. An ACTIVITY-SPECIFICATION may have CONSTRAINTS associated with it.
11. An ACTIVITY-SPECIFICATION may be decomposed into one or more ACTIVITY-DECOMPOSITIONS.
12. An ACTIVITY-DECOMPOSITION is the specification of how an ACTIVITY is decomposed into one or more SUB-ACTIVITIES; this may include the specification of constraints on and between the SUB-ACTIVITIES.
13. A SUB-ACTIVITY is the constituent activity designated in any ACTIVITY-DECOMPOSITION.
14. A PRIMITIVE-ACTIVITY is an ACTIVITY with no (further) ACTIVITY-DECOMPOSITION.
15. CONSTRAINTS can be stated with respect to none, one or more than one time point. They express things which are required to hold. They are evaluable with respect to a specific PLAN as holding or not holding. Such constraints may refer to world statements (conditions and effects), resource requirements and usage, authority requirements or provision, etc.

Level of Formalization: Low.

2.2.5 CPR: Core Plan Representation

Source Documents: Pease, A. (1998) "The Warplan: A Method Independent Plan Schema" available on website home.earthlink.net/~adampease/professional/AIPS98.ps; a more detailed version on http://reliant.teknowledge.com/CPR2/Reports/CPR-RFC4/Design.html#_Toc435005571.

Motivation: The design of CPR is an attempt to unify the major concepts and advancements in plan and process representation into one comprehensive model. There are two significant payoffs to the CPR effort. The first is that creation of a base plan representation will facilitate information interchange among different planning systems. Imagine a typical military planning situation. A crisis develops and a joint task force is formed. The leadership and staff use a planning application to develop guidance for their subordinate commands. This guidance includes background on the situation, objectives which must be met to contain the crisis, constraints on the actions of the task force and high level specification of the name. The second payoff is in the creation of common services based on the CPR. There are two broad areas of services with immediate utility: visualization, scheduling.

Knowledge Model: CPR's top level may be expressed by a number of English sentences that describe it in the same format as the SPAR model.

1. A PLAN relates ACTION(s) to OBJECTIVE(s).
2. The execution of an ACTION may change the WORLD-STATE.
3. An ACTOR is a PLAN-OBJECT that can perform activities and/or hold objectives.
4. An ACTION takes place over a time interval identified by its two ends, the BEGIN time and the END time.
5. An ACTION is an EVENT that has or could have (in the domain model) an ACTOR.
6. An ACTOR is a ROLE that an ENTITY can play when it is the motive force behind an ACTION.
7. A RESOURCE is a PLAN-OBJECT that is used, modified, consumed or destroyed during the execution of an ACTION. RESOURCE is a ROLE that an ENTITY can play.
8. A PRODUCT is a PLAN-OBJECT that is created during the execution of an ACTION. PRODUCT is a ROLE that an ENTITY can play.
9. A WORLD-MODEL provides a model of dynamics that allows WORLD-STATEs to be predicted as the result of some ACTIONs.
10. A WORLD-STATE describes a snapshot of the world which is actual, expected, or hypothetical.
11. Each PROPERTY of each ENTITY may have a VALUE, i.e. PROPERTY(ENTITY)=VALUE.
12. VALUEs may be imprecise.
13. VALUEs may have a PROBABILITY.
14. PROBABILITYs may be partitioned into PROBABILITY-PREDICTION and PROBABILITY-SENSED.
15. A PROBABILITY-PREDICTION is the likelihood of a WORLD-STATE-DESCRIPTION being valid in the future.
16. A PROBABILITY-SENSED is a likelihood that a WORLD-STATE-DESCRIPTION did in fact have the specified value in the past or at the current time.
17. An INFLUENCE-NETWORK is a structure which relates PROBABILITYs and specifies their dependency structure.
18. The EFFECTS-RECORD of an ACTION is the record of changes made to the WORLD-STATE by execution of the ACTION.
19. An OBJECTIVE may have one or more EVALUATION-CRITERIA.
20. An EVALUATION-CRITERION may be applied to a WORLD-STATE to create an EVALUATION.
21. An EVALUATION may be a predicate (holds/does not hold) or a partial order on the results of EVALUATION-CRITERIA .

22. A PLAN-LIBRARY contains PLANs or portions of PLANs that may be reused in creating new PLANs. A PLAN-LIBRARY has one or more INDEXes which can be used to catalog PLANs and aid in searching for them.

Level of Formalization: Low/Medium.

2.2.6 PSL: Process Specification Language (including PIF: Process Interchange Format)

Source Document: Schlenoff, C., Gruninger, M., Ciocoiu, M., Lee, J., (1999) "The Essence of the Process Specification Language". Special Issue on Modeling and Simulation of Manufacturing Systems in the Transactions of the Society for Computer Simulation International, available on website http://www.mel.nist.gov/msidlibrary/doc/essence.pdf.

Motivation: The Process Specification Language (PSL) project at the National Institute of Standards and Technology (NIST) addresses Planning by creating a neutral, standard language for process specification to serve as an interlingua to integrate multiple process related applications throughout the manufacturing life cycle. This interchange language is unique due to the formal semantic definitions (the ontology) that underlie the language. Because of these explicit and unambiguous definitions, information exchange can be achieved without relying on hidden assumptions or subjective mappings. The scope of study is limited to the realm of discrete processes related to manufacturing, including all processes in the design/manufacturing life cycle. Business processes and manufacturing engineering processes are included in this work both to ascertain common aspects for process specification and to acknowledge the current and future integration of business and engineering functions.

Knowledge Model: PSL's top level is built around the following notions:
1. ACTIVITY, a class or type of action. For example, 'paint-part' is an activity. It is the class of actions in which parts are being painted.
2. ACTIVITY-OCCURRENCE, an event or action that takes place at a specific place and time. An instance or occurrence of an activity. E.g., paint-part is an activity, painting in Maryland at 2 PM on May 25, 1998 is an activity-occurrence.
3. TIMEPOINT, A point in time.
4. OBJECT, anything that is not a timepoint or an activity.

--The following definitions and axioms provide the ontological structure underlying to the four basic entities of PSL:

5. Definition 1**.** Timepoint q is between timepoints p and r if and only if p is before q and q is before r.
6. Definition 2. Timepoint p is beforeEq timepoint q if and only if p is before or equal to q.
7. Definition 3. Timepoint q is betweenEq timepoints p and r if and only if p is before or equal to q, and q is before or equal to r.
8. Definition 4**.** An object exists-at a timepoint p if and only if p is betweenEq its begin and end points.
9. Definition 5**.** An activity occurrence is-occurring-at a timepoint p if and only if p is betweenEq the activity occurrence's begin and end points.
10. Axiom 1**.** The before relation only holds between timepoints.
11. Axiom 2**.** The before relation is a total ordering.
12. Axiom 3**.** The before relation is non-reflexive.
13. Axiom 4**.** The before relation is transitive.

14. Axiom 5. Inf- is before every other timepoint.
15. Axiom 6. Every timepoint else than inf+ is before inf+
16. Axiom 7. Given any timepoint t other than inf-, there is a timepoint between inf- and t.
17. Axiom 8. Given any timepoint t other than inf+, there is a timepoint between t and inf+.
18. Axiom 9. Everything is either an activity, an activity-occurrence, an object, or a timepoint.
19. Axiom 10**.** Activities, activity-occurrences, objects, and timepoints are all distinct kinds of things.
20. Axiom 11**.** The occurrence-of relation only holds between activities and activity-occurrences.
21. Axiom 12. An activity-occurrence is the occurrence-of a single activity.
22. Axiom 13**.** The begin and end of an activity-occurrence or object are timepoints.
23. Axiom 14. The timepoint at which an activity-occurrence b egins always precedes the timepoint at which the activity-occurrence ends.
24. Axiom 15. The participates-in relation only holds between objects, activities, and timepoints, respectively.
25. Axiom 16. An object can participate in an activity only at those timepoints at which both the object exists and the activity is occurring.

Level of Formalization: High.

2.2.7 PLANET: a PLAN semantic NET

Source Document: Gil, Y., and Blythe, J., (2000) "PLANET: A Shareable and Reusable Ontology for Representing Plans". In AAAI 2000 workshop on Representational Issues for Real-world Planning Systems, available on website http://www.isi.edu/expect/papers/gil-blythe-aaai00-2.pdf.

Motivation: Enhance knowledge modeling, reuse, integration and sharing. As for other ontologies of planning, PLANET was initially developed for and applied in the defense sector.

Knowledge Model: PLANET's top level is built around the following notions:
1. A PLANNING PROBLEM CONTEXT represents the initial, given assumptions about the planning problem. It describes the background scenario in which plans are designed and must operate on. This context includes the initial state, desired goals, and the external constraints.
2. A WORLD STATE is a model of the environment for which the plan is intended. A certain world state description can be chosen as the INITIAL STATE of a given planning problem, and all plans that are solutions of this planning problem must assume this initial state.
3. The DESIRED GOALS express what is to be accomplished in the process of solving the planning problem. Sometimes the initial planning context may not directly specify the goals to be achieved, instead these are deduced from some initial information about the situation and some abstract guidance provided as constraints on the problem.
4. EXTERNAL CONSTRAINTS may be specified as part of the planning context to express desirable or undesirable properties or effects of potential solutions to the problem, including user advice and preferences. Examples of external constraints are that the plan accomplishes a mission in a period of seven days, that the plan does not use a certain type of resource, or that transportation is preferably done in tracked vehicles. Commitments are discussed later. The initial requirements expressed in the

planning problem context need not all be consistent and achievable (for example, initial external constraints and goals may be incompatible), rather its aim is to represent these requirements as given. A plan may satisfy or not satisfy external constraints.

5. A PLANNING PROBLEM is created by forming specific goals, constraints and assumptions about the initial state. Several plans can be created as alternative solutions for a given planning problem. A planning problem also includes information used to compare alternative candidate plans. Planning problems can have descendant planning problems, which impose (or relax) different constraints on the original problem or may assume variations of the initial state.
6. A planning problem may have a number of CANDIDATE PLANS which are potential solutions. A candidate plan can be *untried* (i.e., it is yet to be explored or tested), *rejected* (i.e., for some reason it has been rejected as the preferred plan) or *feasible* (i.e., tried and not rejected). One or more feasible plans may be marked as *selected.* All of these are sub-relations of candidate plan.
7. A GOAL SPECIFICATION represents anything that gets accomplished by a plan, subplan or task. Both capabilities and effects of actions and tasks are subtypes of goal specification, as well as posted goals and objectives. Goals may be variabilized or instantiated.
8. STATE-BASED GOAL SPECIFICATIONS are goal specifications that typically represent goals that refer to some predicate used to describe the state of the World, for example ‘achieve (at JimLAX)’, ‘deny (atRed-BrigadeSouth-Pass)’ or ‘maintain (temperature Room5 30)’.
9. OBJECTIVE BASED GOAL SPECIFICATIONS are goal specifications that are typically stated as verb- or action-based expressions, such as ‘transport brigade5 to Ryad’.
10. Goal specifications also include a HUMAN READABLE DESCRIPTION used to provide a description of a goal to an end user. This is useful because often times users want to view information in a format that is different from the internal format used to store it. This could be a simple string or a more complex structure.
11. PLAN TASK DESCRIPTIONS are the actions that can be taken in the world state. They include templates and their instantiations, and can be abstract or specific. A plan task description models one or more ACTIONS in the external world.
12. A PLAN TASK is a subclass of PLAN task description and represents an instantiation of a task as it appears in a plan. It can be a partial or full instantiation.
13. A PLAN TASK TEMPLATE is also a subclass of PLAN TASK DESCRIPTION that denotes an action or set of actions that can be performed in the world state. In some AI planners the two classes correspond to operator instances and operator schemas respectively, and in others they are called tasks and task decomposition patterns. Plan task descriptions have a set of preconditions, a set of effects, a capability, and can be decomposed into a set of subtasks. Not all these properties need to be specified for a given task description, and typically planners represent tasks differently depending on their approach to reasoning about action.
14. The CAPABILITY of a task or task template describes a goal for which the task can be used.
15. A PRECONDITION represents a necessary condition for the task. If the task is executed, its EFFECTS take place in the given world state. Tasks can be decomposed into SUBTASKS that are themselves task descriptions. Hierarchical task network planners use task decomposition or operator templates (represented here as plan task templates) and instantiate them to generate a plan. Each template includes a statement of the kind of goal it can achieve (represented as a capability), a decomposition

network into subtasks, each subtask is matched against the task templates down to primitive templates, represented as primitive plan task descriptions. Like goal specifications, plan task descriptions also include a human readable description. Some AI planners specify this information as a set of parameters of the task that are used to determine which subset of arguments will be printed when the plan is displayed.

16. PLANNING LEVELS can be associated to task descriptions as well as to goal specifications. Levels are also used in real-world domains, for example military plans are often described in different levels according to the command structure, echelons, or nature of the tasks.
17. A PLAN represents a set of commitments to actions taken by an agent in order to achieve some specified goals. It can be useful to state that a plan forms a sub-plan of another one. For example, military plans often include subplans that represent the movement of assets to the area of operations (i.e., logistics tasks), and subplans that group the operations themselves (i.e., force application tasks).
18. PLAN COMMITMENTS are commitments on the plan as a whole, and may be in the form of actions at variously detailed levels of specification, orderings among actions and other requirements on a plan such as a cost profile. The tasks that will form part of the plan are represented as a subset of the commitments made by the plan.
19. TASK COMMITMENTS are commitments that affect individual tasks or pairs of tasks. An *ordering commitment* is a relation between tasks such as (before A B). A temporal commitment is a commitment on a task with respect to time, such as (before ?task ?time-stamp). Another kind of commitment is the selection of a plan task description because it *accomplishes* a goal specification. This relation records the intent of the planning agent for the task, and is used in PLANET to represent causal links.

Level of Formalization: Medium

2.2.8 EO: Enterprise Ontology

Source Document: The Enteprise Ontology, available on website http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html

Motivation: The Enterprise Ontology is a collection of terms and definitions relevant to business enterprises. The ontology was developed in the Enterprise Project by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partners: IBM, Lloyd's Register, Logica UK Limited, and Unilever. The project was support by the UK's Department of Trade and Industry under the Intelligent Systems Integration Programme(project no IED4/1/8032).

Knowledge Model: The following is a complete list of the terms defined in the Enterprise Ontology. We do not provide the definitional structure here, a selection of which might later be added if applicable to Metokis.

1. ACTIVITY: ACTIVITY SPECIFICATION, EXECUTE, EXECUTED ACTIVITY SPECIFICATION, T-BEGIN, T-END, PRE-CONDITIONS, EFFECT, DOER, SUB-ACTIVITY, AUTHORITY, ACTIVITY OWNER, EVENT, PLAN, SUB-PLAN, PLANNING, PROCESS SPECIFICATION, CAPABILITY, SKILL, RESOURCE, RESOURCE ALLOCATION, RESOURCE SUBSTITUTE.
2. ORGANISATION: PERSON, MACHINE, CORPORATION, PARTNERSHIP, PARTNER, LEGAL ENTITY, ORGANISATIONAL UNIT, MANAGE, DELEGATE, MANAGEMENT LINK, LEGAL OWNERSHIP, NON-LEGAL

OWNERSHIP, OWNERSHIP, OWNER, ASSET, STAKEHOLDER, EMPLOYMENT CONTRACT, SHARE, SHARE HOLDER.
3. STRATEGY: PURPOSE, HOLD PURPOSE, INTENDED PURPOSE, STRATEGIC PURPOSE, OBJECTIVE, VISION, MISSION, GOAL, HELP ACHIEVE, STRATEGY, STRATEGIC PLANNING, STRATEGIC ACTION, DECISION, ASSUMPTION, CRITICAL ASSUMPTION, NON-CRITICAL ASSUMPTION, INFLUENCE FACTOR, CRITICAL INFLUENCE FACTOR, NON-CRITICAL INFLUENCE FACTOR, CRITICAL SUCCESS FACTOR, RISK.
4. MARKETING: SALE, POTENTIAL SALE, FOR SALE, SALE OFFER, VENDOR, ACTUAL CUSTOMER, POTENTIAL CUSTOMER, CUSTOMER, RESELLER, PRODUCT, ASKING PRICE, SALE PRICE, MARKET, SEGMENTATION VARIABLE, MARKET SEGMENT, MARKET RESEARCH, BRAND IMAGE, FEATURE, NEED, MARKET NEED, PROMOTION, COMPETITOR.
5. TIME: TIME LINE, TIME INTERVAL, TIME POINT.

Level of Formalization: Low/Medium.

2.2.9 OPT: Ontology with Polymorphic Types (including PDDL: Planning Domain Definition Language)

Source Document: McDermott (2003) "OPT Manual Version 1.6 *Draft**" available on website
http://cs-www.cs.yale.edu/homes/dvm/papers/opt-manual.pdf.

Motivation: OPT is an attempt to create a general-purpose notation for creating ontologies, defined as formalized conceptual frameworks for domains about which programs are to reason. Its syntax is based on PDDL, but it has a more elaborate type system, which allows users to make use of higher-order constructs such as explicit lambda-expressions. OPT is intended to be (almost) upwardly compatible with PDDL 2.1, the dialect used in the 2002 International Planning Competition.

Knowledge Model: OPT includes the following essential built-in types:
1. ACTION, skip-action or a Hop-action.
2. BOOLEAN, true or false.
3. (CON *c*1 ...*ck*), the type consisting of just the constants (literals) *c*1 to *ck*.
4. FLOAT, floating-point number.
5. (FLUENT *y*), (Fun *y* <- Situation).
6. (FUN *r* <- *a*), Function from type *a* to type *r*.
7. (HOP *r*) :action-expansions The type of an action that might take anywhere from zero time to a long time interval, producing a value of type *r*.
8. HOP-ACTION :action-expansions, an action of type (Hop *r*) for some *r*.
9. OBJ, the universal type; every object is of this type.
10. PROCESS, an entity of type (Slide *r*) for some *r*.
11. (SKIP *r*), the type of an action that takes exactly one infinitesimally long time interval and returns a value of type *r*.
12. SKIP-ACTION, an action of type (Skip *r*) for some *r*.
13. SITUATION, a world state.
14. STRING, string of characters.
15. SYMBOL, a Lisp-style symbol.
16. VOID, the empty type.

Level of Formalization: High.

2.2.10 COS: Core Ontology of Services

Source Document: Oberle, D., Mika , P., Gangemi, A., Sabou, M. (2004) “Foundations for service ontologies: Aligning OWL-S to DOLCE” in Staab, S., and Patel-Schneider, P., (eds.), Proceedings of the World Wide Web Conference (WWW2004), Semantic Web Track.

Motivation: The descriptions of services show a clear contextual nature. One may only have to consider the number of different views that may exist on a service. The concepts used to formulate any given view are clearly separate from the actual objects they act upon and often independent from the concepts appearing in other views. In order to account for this independence COS is defined by reference to DOLCE and its basic extensions, i.e. D & S, Ontology of Plans.

Knowledge Model: COS considers five frequently occurring descriptions of a service, where each represents a separate viewpoint:

1. (SERVICE) OFFERING.
2. REQUEST.
3. AGREEMENT.
4. ASSESSMENT,
5. NORMS.

More views may be added in the future when needs arise. All service views are specializations of S-DESCRIPTION defined in the Descriptions & Situations ontology. Furthermore the following specializations of the notion COURSE OF EVENT are considered:

6. TASK.
7. SERVICE TASK.
8. COMPUTATIONAL TASK.

This allows to model activities in an information system and in the real world. Axioms ensure that SERVICE TASKS only sequence SERVICE ACTIVITIES and that COMPUTATIONAL TASKS only sequence COMPUTATIONAL ACTIVITIES. The activities are new kinds of PERDURANTS especially introduced here. Further axioms also ensure that only INFORMATION OBJECTS (a newly introduced NON-PHYSICAL ENDURANT) participate in COMPUTATIONAL ACTIVITIES.
The Core Ontology of Services may optionally take advantage of a number of concepts from the Ontology of Plans which is another module for DOLCE+. It allows the division of tasks into elementary and complex and the construction of complex tasks from elementary ones among other features.
The Core Ontology of Services also models frequently occurring FUNCTIONAL ROLES:

9. REQUESTOR PROVIDER of a service are conceived as LEGALLY CONSTRUCTED PERSONS, an agentive legal role in DOLCE.
10. EXECUTOR of a service is considered an agentive functional role without a legal nature.
11. (COMPUTATIONAL) INPUTS and OUTPUTS, formalized as instrumentality roles. The comprehensive axiomatization requires that, e.g., a COMPUTATIONAL INPUT is only played by an INFORMATION OBJECT.
12. VALUE OBJECTS, as a subtype of the generic DOLCE+ commerce role. Such a role distinguishes generic Inputs/Outputs from ones to which a value is attributed. The latter is usually done by the actor whose viewpoint is being modelled.

Level of Formalization: Medium/High.

2.2.11 An Ontological Formalization of the Planning Task

Source Document: Rajpathak, D., Motta, E. (2004) "An Ontological Formalization of the Planning Task" Accepted at FOIS-2004.

Motivation: To provide an ontology that formalizes the nature of the planning task independently of any planning paradigm, specific domains, or applications while being a fine-grained, precise and comprehensive characterization of the space of planning problems. In addition, to produce a formal specification that operationalizes the ontology into a set of executable definitions, which provide a concrete reusable resource for knowledge acquisition and system development in planning applications.

Knowledge Model:

1. INITIAL-WORLD-STATE, S0. It describes the state of the world at the beginning of the planning process.
2. GOAL**,** G. It describes the desired state of the world we would like to achieve through a planning process.
3. PLAN-TASKS, PT = {pt1, …., ptn}. A set of *plan-tasks*, which specify intermediate goals which need to be accomplished to achieve the overall goal of the planning task.
4. ACTIONS. For each plan-task, pti, there is a finite set of actions, Ai = {ai1, …., aik}, which must be executed to accomplish pti.
5. AGENTS, AG = {ag1, …., agm}. A set of agents, which are responsible for achieving plantasks through the execution of actions.
6. PARAMETERS, PA = {pa1, …., pal}. Parameters can be seen as meta-level pointers to the domain entities which are relevant to the planning process.
7. TIME-HORIZON, TH. A time window within which the plan is required to take place.
8. CONSTRAINTS, C = {c1, ..., cj}. A set of constraints, which must not be violated by a plan. Typical constraints observed in planning are *variable binding*, *ordering relation*, and *interval preservation*.
9. PREFERENCES, PR = {pr1, …., pro}. A set of criteria for partially ranking competing plans. These are important to support the acquisition and modeling of local optimization criteria during the knowledge acquisition process and indeed they can in practice be mutually unsatisfiable. Preferences are typically called *soft constraints* in many approaches to design and planning, however they are ontologically very different from constraints and therefore we prefer not to use the term "soft constraint".
10. COST-FUNCTION, Cf. A function, which provides a global mechanism for comparing the costs of alternative plans.
11. SOLUTION CRITERION, SOL. A mapping from a plan **P** to {True False}, which determines whether a candidate plan is a solution. A solution criterion usually requires **P** to be *complete* and *valid* - see the following section for the description of these properties.
12. PLAN-MODEL, P = {p1, …., pq}. A candidate plan is a sequence of pairs, <pti, agj>, where pti is a plan-task and agj is an agent able to execute the relevant actions associated with pti.

Level of Formalization: High.

2.2.12 DDPO: DOLCE+DnS Plan Ontology

DOLCE+DnS Plan Ontology (DDPO) specializes the concepts and relations defined in DOLCE, and some of its extensions, notably the Ontology of Descriptions and Situations (DnS). DDPO, like DnS, has a very liberal domain, which includes physical and non-physical objects (social entities, mental objects and states, conceptualizations, information objects, constraints), events, states, regions, qualities, and even constructivist situations. The main target of DDPO are so-called *tasks*, namely the types of actions, their sequencing, and the controls performed on them. In order to accept tasks in the domain - as clearly distinguished from actions and states – control operators of classical planning and process models are considered types of *planning* or *decision* actions, i.e. rational actions, distinguished from purely executive actions. Other typical procedural notions like *precondition*, *postcondition*, *preference*, etc. have a corresponding treatment in DDPO.
As done in section 2.2 for existing approaches, in this section we provide a preliminary description of the ontology of plans presented in section 3 by indicating: source documents, the motivation behind the considered approach, the most significant part of the knowledge model formalized in the approach, a tentative evaluation of the level of formalization of the approach. The convention here is that a newly introduced notion is written in **bold**.

Source Documents: Besides this deliverable, [10], and [13].

Motivation: The intended use of DDPO is to specify plans at an abstract level and independently from existing resources. Its rich set of primitives would require a complex algebra to be implemented as a calculus, but the aim is not to make a plan calculus. On the contrary, DDPO should be implemented, through appropriate tools, as a framework to define detailed or approximate plans from any perspective. The resulting plans could be grounded in systems that implement a set of functionalities and reason according to the specifications given in DDPO-based plans.

Knowledge Model:

1. A **description** is a non-agentive social object.
2. A **situation** is a **setting** for any number of entities.
3. The (external) **time and space of a situation** are the time and space of the entities in the setting.
4. A **concept** is a non-physical object, which is “defined by” a description.
5. The **classifies** relation relates concepts and entities.
6. There are several kinds of concepts reified, the primary ones (**role**, **course**, and **parameter**) being distinguished by the entity types they classify in DOLCE.
7. **Figures** are concepts that do not classify entities.
8. The **component** relation is a proper part qualified by a description in which the proper parts are involved.
9. **Uses** is a subrelation of Component (there is an inherent cycle between the component and uses relations, but it seems unavoidable: functional part requires a description, which is an object with functional parts, etc.).
10. **Defines** is a subrelation of Uses.
11. Roles or figures and courses are related by relations expressing the **attitudes** that (players of) roles or (representatives of) figures can have **towards** a course.
12. Parameters and roles, figures, or courses are related by a **requisite for** relation, expressing the kind of requisites entities that are classified by roles or courses should have:
13. The **satisfaction** (SAT) relation holds between situations and descriptions, and implies that at least some components in a description must classify at least some entity in the

situation setting. There exist a basic typology for the satisfaction relation between situations and descriptions: P-Satisfies means **proactively satisfies**, R-Satisfies means **retroactively satisfies**, and C-Satisfies means **constructively satisfies**.

14. A **plan** is a description that defines or uses at least one task and one agentive role or figure, and that has at least one goal as a part.
15. A **goal** is a desire (another kind of description) that is a part of a plan.
16. **Desires** are characterized by defining or using at least one agentive role or figure, and at least one course towards which the (player or representative of) role or figure has a desire:
17. A **main goal** is a goal that is part of a plan but not of one of its subplans.
18. **Plan executions** are situations that proactively satisfy a plan.
19. A **goal situation** is a situation that satisfies a goal.
20. A **precondition** for a plan can be defined as a relation between a situation and a plan, implying that, for all plan executions of that plan to occur, a situation should preliminarily satisfy some description as well.
21. A **postcondition** for a plan is a relation between a situation and a plan, implying that, after plan executions of that plan occur, a situation should satisfy some description as well.
22. An **accompanying condition** (sometimes called 'constraint' in the planning literature) for a plan can be defined as a relation between a situation and a task, implying that, for all plan executions of that plan to occur, a situation should satisfy some description as well, at the time of some specified perdurant that is sequenced by a task defined in the plan.
23. A circumstantial or **saturated plan** is a plan that cannot be executed twice, since it defines a temporal parameter restricted to one value, e.g. one of its tasks classifies an event that is valued by a definite temporal value.
24. **Tasks** are courses used to sequence activities, usually within plans. Tasks can be complex, and ordered according to an abstract succession relation. Tasks can relate to concrete actions or decision making; the latter deals with typical flowchart content. A task is different both from a flowchart node, and from an action or action type. Several types of tasks may be defined: **scheduled task**, **complex task**, **sequential task**, **hybrid task**, **bag task**, **elementary task**, **action task**, **control task**, **loop task**, **cyclical task**, **branching task**, **case task**, **alternate task, concurrent task, parallel task**, **any order task**, **beginning task**, **ending task** , and **maximal task**.

Level of Formalization: High.

References

[1] Tate, A., Hendler, J. and Drummond, M., (1990). A Review of AI Planning Techniques, In J. Allen, J. Hendler and A. Tate (eds.) *Readings in Planning* Morgan Kaufmann, pp. 26-49

[2] Benjamins, R., Nunes de Barros, L., Valente, A., (1996) "Constructing Planners Through Problem-Solving Methods" available on website http://ksi.cpsc.ucalgary.ca/KAW/KAW96/benjamins/doc.html.

[3] Myers, K.L., Wilkins, D.E., (1997) "The Act Formalism Version 2.2" available on website http://www.ai.sri.com/~act/act-spec.pdf

[4] Tate, A. (1998) "Roots of SPAR - Shared Planning and Activity Representation", The Knowledge Engineering Review, Vol. 13(1), pp. 121-128, Special Issue on "Putting Ontologies to Use" (eds. Uschold, M. and Tate, A.), Cambridge University Press.

[5] Pease, A. (1998) "The Warplan: A Method Independent Plan Schema" available on website home.earthlink.net/~adampease/professional/AIPS98.ps; a more detailed version on http://reliant.teknowledge.com/CPR2/Reports/CPR-RFC4/Design.html#_Toc435005571.

[6] Schlenoff, C., Gruninger, M., Ciocoiu, M., Lee, J., (1999) "The Essence of the Process Specification Language". Special Issue on Modeling and Simulation of Manufacturing Systems in the Transactions of the Society for Computer Simulation International, available on website http://www.mel.nist.gov/msidlibrary/doc/essence.pdf.

[7] Gil, Y., and Blythe, J., (2000) "PLANET: A Shareable and Reusable Ontology for Representing Plans". In AAAI 2000 workshop on Representational Issues for Real-world Planning Systems, available on website http://www.isi.edu/expect/papers/gil-blythe-aaai00-2.pdf.

[8] The Enteprise Ontology, available on website http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html

[9] McDermott (2003) "OPT Manual Version 1.6 *Draft**" available on website http://cs-www.cs.yale.edu/homes/dvm/papers/opt-manual.pdf.

[10] Oberle, D., Mika , P., Gangemi, A., Sabou, M. (2004) "Foundations for service ontologies: Aligning OWL-S to DOLCE" in Staab, S., and Patel-Schneider, P., (eds.), Proceedings of the World Wide Web Conference (WWW2004), Semantic Web Track.

[11] Rajpathak, D., Motta, E. (2004) "An Ontological Formalization of the Planning Task". To appear at FOIS-2004.

[12] http://www.wfmc.org/standards/model.htm.

[13] van Elst, L., Abecker, A.: Ontologies for information management: balancing formality, stability, and sharing scope. Expert Systems with Applications 23 (2002), 357–366.

[14] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., WonderWeb Deliverable D18: The WonderWeb Library of Foundational Ontologies, http://wonderweb.semanticweb.org (2003).

[15] Gangemi, A., Mika, P.: Understanding the Semantic Web through Descriptions and Situations. In Meersman, R., et al. (eds.), Proceedings of ODBASE03 Conference, Springer, Berlin (2003).

[16] Masolo, M., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, G., Guarino, N.: Social Roles and their Descriptions, in Didier Dubois, Christopher Welty, Mary-Anne Williams (eds.), Procedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004) Whistler, BC, Canada June 2-5, 2004 p.267-277.

[17] http://wonderweb.semanticweb.org.

[18] http://www.loa-cnr.it.
[19] Jennings, N.R., Wooldridge, M.J. (eds.): Agent Technology: Foundations, Applications and Markets, Berlin: Springer Verlag, 1998.
[20] Brentano, F., Psychologie vom empirischen Standpunkt, 2. Aufgabe, hrsg. von O. Kraus. Meiner (Eng. trans. ed. by L. L. McAlister, Psychology from an Empirical Standpoint, London), Leipzig, 1924.
[21] Eco, U., Kant e l’ornitorinco, Milano, Bompiani, 1997.
[22] Jakobson, R.: Linguistics and Poetics: Closing Statement. In: Style in Language. MIT Press, Cambridge, MA,1960.
[23] Galton, A.: Reified Temporal Theories and How To Unreify Them. Proceedings of the Int. Joint Conference on Artificial Intelligence, 1991.
[24] Akinkunmi, B. O.: On the Expressive Limits of Reified Theories. Journal of Logic and Computation 10(2) pp. 197–213, April 2000.
[25] Ghidini C., Giunchiglia F., "Local models semantics, or contextual reasoning = locality + compatibility". *Artificial intelligence*, 2001, v. 127, n. 2, p. 221-259.

3 Basic axiomatization for the plan ontology

3.1 Introduction

The DOLCE+DnS Plan Ontology (DDPO) specializes the concepts and relations defined in DOLCE [14], and some of its extensions, notably the Ontology of Descriptions and Situations (DnS).
DDPO, like DnS, has a very liberal domain, which includes physical and non-physical objects (social entities, mental objects and states, conceptualizations, information objects, constraints), events, states, regions, qualities, and even “constructive” situations.
The main target of DDPO are so-called *tasks*, namely types of actions, their sequencing, and the controls performed on them. In order to accept tasks in the domain - as clearly distinguished from actions and states - control operators of classical planning and process models are considered types of *planning* or *decision* actions, i.e. **rational** actions, distinguished from purely **executable** actions. Other typical procedural notions like *precondition*, *postcondition*, *preference*, etc. have a corresponding treatment in DDPO.
The intended use of DDPO is to specify plans at an abstract level, and independently from existing resources. Its rich set of primitives would require a complex algebra to be implemented as a calculus, but our aim is not to make a plan calculus. On the contrary, we expect that DDPO would be implemented - through appropriate tools - as a framework to define detailed or approximate plans for any use (social, personal, computational). The resulting plans would then be **grounded** in some system that implements a set of functionalities and reasons according to the specifications given in DDPO-based plans.

DDPO is presented here in FOL, with appendixes in KIF and OWL-DL. The case studies of Metokis will be specified as either models of DDPO, or extensions/specialisations of it. A sample model for a publisher plan is presented in section 4.

The axiomatization for DDPO reuses or updates the following sources:

- the Deliverable D18 from the WonderWeb Project [14], including the axiomatization of the basic categories of DOLCE, and in particular the categories (e.g. Non-Physical Endurant and its subclasses) that are specialised in the DnS extension of DOLCE
- the DnS extension of DOLCE as presented in the DOLCE-Lite-Plus OWL-DL version (see annex), as well as the axiomatization of social roles and descriptions as presented in [16], and in particular the categories of Description, Concept, Figure, Situation, and their subclasses (Course, Parameter, etc.)
- some of the extensions provided in the DOLCE-Lite-Plus OWL-DL version (see annex), and in particular: the modules including *time* predicates, *space* predicates, *semiotic* roles and *information* objects, notions related to concrete *datatypes*, etc.
- the preliminary plan ontology in the previous DOLCE-Lite-Plus OWL-DL versions (until 3679), and in particular the categories of Plan, Task, etc.

For all concepts and relations that are not explicitly recalled or presented here, please refer to the textual sources, as well as to the KIF and OWL-DL codes in the annexes.

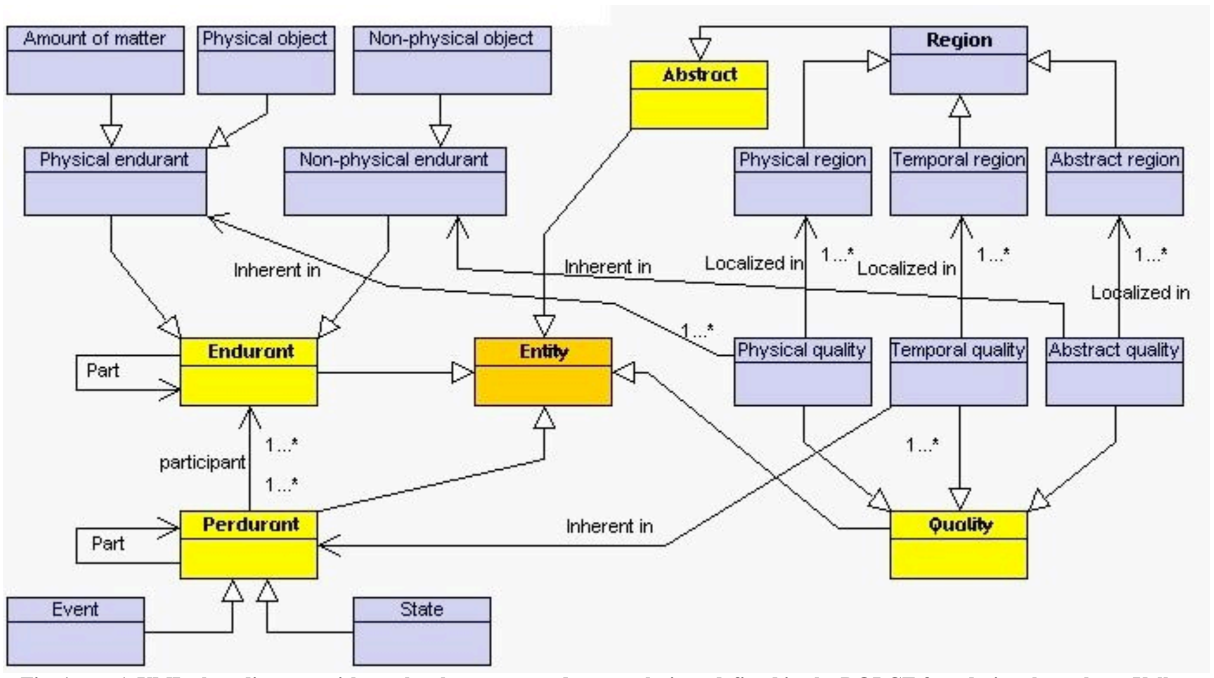

Fig. 1 A UML class diagram with top-level concepts and some relations defined in the DOLCE foundational ontology. Yellow nodes represent the categories. Unlabelled arrows are IS_A (subclass-of) relationships.

3.2 The DOLCE foundational ontology

DOLCE (Descriptive Ontology for Language and Cognitive Engineering, http://dolce.semanticweb.org) has been developed in the context of the WonderWeb project [17] by the Laboratory for Applied Ontology [18] of the ISTC-CNR. It is aimed at supporting the design of *domain ontologies*, and it is currently used in many industrial and academic projects worldwide.
For a detailed presentation of DOLCE, we refer to [16]. We just recall here the main distinctions (Fig.1):

Individuals. **Particular** (*vs* universal) **entities** are the individuals of the DOLCE ontology domain. Particulars can be as varied as possible: in space (e.g. a saxophone) or in time (e.g. a song); physical (e.g. a stone), social (e.g. a company), or mental (e.g. a desire); agentive (e.g. an animal) or non-agentive (e.g. a law); qualities (e.g. the color depending on the pigmentation of a specific eye) or quality spaces (e.g. *sea green* in the Mac palette); substances (e.g. an amount of sand) or systems (e.g. the complex of a car engine, wheels, gears, road, air, driver), etc.
The four top categories of DOLCE particulars are: **Endurant**, **Perdurant**, **Quality**, and **Abstract**.
Endurants are particulars in space, which participate at least in one perdurant (e.g. substances, objects, social entities, concepts). Endurants are distinguished into physical vs. non-physical, and agentive vs. non-agentive.
Perdurants are particulars in time, which have at least one participant (e.g. events, states, processes, phenomena).
Qualities are dependent particulars, “inherent” in either endurants or perdurants (e.g. actual colors, weights, speeds, etc.).
Abstracts are particulars neither is space nor in time (e.g. sets, regions, metric spaces, etc.).
There is a taxonomy that specializes the four categories: endurants are distinguished into physical and non-physical, perdurants into states and events, qualities into physical, temporal, and abstract, etc.

Relations for parthood, connectedness, localization, participation, inherence, dependence, etc. are defined in DOLCE, but not detailed here for brevity. It is possible to refer to [14] for a thorough axiomatization, as well as to the DDPO OWL-DL code in the annex, which is glued together with all the pieces of the ontologies mentioned here (the version of DOLCE-Lite-Plus presented in the annex is 3955).
For the sake of this work, we should mention that *dependency, constitution,* and *inherence* relations are specially important, because they lay down the basis for different strata of reality, e.g.:

- a physical endurant can only be constituted by other physical endurants
- non-physical endurants are always dependent on at least one physical endurant
- the time of an endurant is the time of the perdurants in which it participates
- the space of a perdurant is the space of the participating endurants
- the time and space of a non-physical object are the time and space of the physical endurants on which it depends, etc.

Extensions (Fig.2). The same references hold for the other extensions of DOLCE mentioned above: time, space, semiotic and information, reified situations, and plan-related relations. In Fig.3 a diagram including the basic modules of the current version of DOLCE-Lite-Plus is shown.

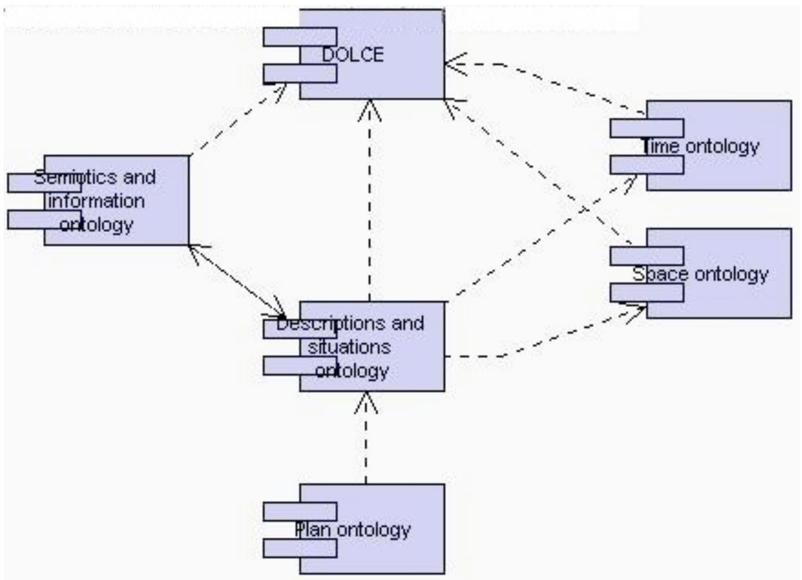
DOLCE
Time ontology
Semiotics and information ontology
Space ontology
Descriptions and situations ontology
Plan ontology

Fig. 2 **A UML component diagram showing the main modules of DOLCE-Lite-Plus. Arrows represent dependencies.**

3.3 Basic classes and relations of the DnS Ontology

Since DDPO heavily relies on DnS, we present it here in some detail (but refer to [16] and to the OWL annex for completeness; updates are available at http://dolce.semanticweb.org).

3.3.1 Introduction

Descriptions and Situations (DnS) is a foundational ontology conceived with the purpose of helping to extend a (possibly, but not exclusively), foundational ontology *O* (the *ground* ontology*)* with another ontology *O'*, by means of:

- *reifying* the *intension* of the relations and classes from *O'* that are not in the vocabulary of *O*, but are used to talk about the same domain of interest as (or a subset of) the one of *O*

- *reifying* the *extension* of the classes and relations from *O'*
- *reifying* the *tuples* from *O'* (the instances from *O'* relations extension)
- *embedding* the resulting individuals into the domain of *O*.

DnS provides a vocabulary and an axiomatization to type the new individuals, to interrelate them, and to link them to the existing predicates from *O*.
DnS *augments* the vocabulary of O, and produces an O^+, and as a consequence it also populates the domain of *O* with entities that are instances of the augmented part of the vocabulary.
For instance, when using DnS to augment DOLCE vocabulary, what you get is DOLCE+. In DOLCE+, DOLCE plays the role of ground ontology.
DOLCE can then be extended to e.g. a *legal* description of social reality, or to a *planning* framework, by *embedding* another ontology. To carry out such an embedding, we can use DOLCE+ and a semantic method that is presented here.
An example shows the way DnS can be used, and how this is different from other approaches to ontology extension.
Suppose that someone wants to express the legal constraints imposed by the norms of a legal system. The ontology that potentially[1] encodes such a system is not a part of DOLCE, even though its domain is intuitively a subset of the domain of DOLCE.
Then you may want to extend DOLCE with a description of social reality under a legal perspective. Typically, this can be done by aligning or merging DOLCE with a legal ontology. On the other hand, with this basic approach there are problems related to arising vocabulary conflicts, alternative accounts for the same domain, etc.
An elegant way of managing these problems is using *formal contexts* [25], and assuming ontologies as a variety of context. But in order to manipulate efficiently all the mappings (*bridging axioms*) required by formal contexts, a powerful meta-level reasoning is required, which deals with motivations to create such mappings.
Here we propose another approach that uses *embedded contexts*, i.e. contexts that are *reified within* an ontology that has been augmented with DnS. The advantages of a reified approach are described in [15]. Notice that this approach is not in competition with formal contexts, and it could be used also to start the research on meta-level reasoning in the formal context framework.
Back to the example, DnS makes it possible to describe the ideal (legal) view on the behaviour of given social entities (as "situations"), according to a given legal system, and without changing the DOLCE+ domain, because the reified predicates and tuples of the legal ontology become instances of the DnS predicates. The legal ontology is then transformed into an *embedded context* within DOLCE+.
In the next section we provide a synthetic overview of a reification semantics for embedded contexts.

3.3.2 A basic semantics for embedded contexts

(a) ontologies

Here we assume ontologies as (strongly) typed first-order axiomatic theories without free variables, having a signature (or vocabulary) Π, a domain of quantification Δ (and its product Δx...xΔ), a set of class extensions Σ, and a set of relation extensions Σ^n. The union of Σ and Σ^n has the same cardinality as Π.

[1] "Potentially", because the ontology extensions we are going to discuss are not necessarily realized: they must only be formalizable *in principle*, i.e. when using DnS, it's possible to work with a *virtual* ontology that is realized only in the reified form we introduce here.

For the sake of this presentation, we assume that Δ is partitioned into the sets from Σ and Σ^n (which can be false in the general case).
Π is partitioned into a set Πc of unary predicates $\varphi_{1...n}$, and a set Πr of ≥2-ary predicates $\rho_{1...n}$.
$\Delta c \subseteq \Delta$ includes the individuals $\iota_{1...n}$, and is partitioned into sets of individuals $\sigma_{1...n} \in \Sigma$ for each predicate φ_i in Πc.
$\Delta r=(\Delta x...x\Delta)$ includes the tuples $\tau_{1...n}$, and is partitioned into the cartesian products $(\sigma_1 x...x\sigma_k)_{1...n} \in \Sigma^n$ $(\sigma_i \subseteq \Delta)$ of the domains and ranges of each predicate ρ_i in Πr.
We will call each φ_i a *class intension*, each ρ_i a *relation intension*, each σ_i a *class extension*, each $(\sigma_1 x...x\sigma_n)_i$ a *relation extension*, each ι_i an *individual*, and each τ_i a *tuple* (or relationship).
Moreover, we consider two special subsets of class and relation intensions: $\Gamma c \subseteq \Pi c$, and $\Gamma r \subseteq \Pi r$, including the *axioms* defined for any φ_i (we are interested in particular to e.g. the following axiom schemata: $[\varphi(x) \wedge \psi(x)]$, $[\varphi(x) \vee \psi(x)]$, $[\neg\varphi(x)]$, $[\exists y.\ \rho(x,y) \wedge \varphi(y)]$, $[\forall y.\ \rho(x,y) \rightarrow \varphi(y)]$, etc.), or for any ρ_i (e.g.: $[\rho(x,y) \wedge \sigma(x)]$, $[\rho(x,y) \vee \sigma(x)]$, $[\neg\rho(x,y)]$, $[(\rho(x,y) \wedge \sigma(y,z)) \rightarrow \rho(x,z)]$, etc.), by using first-order construction operators (quantifiers and boolean operators).
Since no names are usually provided for axioms in the signature Π, we assume *anonymous placeholders* $\alpha_{c1...n} \in \Gamma c$ for class axioms, and $\alpha_{r1...n} \in \Gamma r$ for relation axioms. Correspondingly, for each $\alpha_{ci} \in \Gamma c$ there is a $\sigma^{\Gamma i} \in \Sigma$, and for each $\alpha_{ri} \in \Gamma r$ there is a $(\sigma_1 x...x\sigma_k)_{1...n} \in \Sigma^n$.

(b) type- and token-reification

According to [23], «semantically, to reify a concept is to accord it full ontological status, so that it becomes an entity we can ascribe properties to and, in principle, quantify over». Among the ontology elements we have listed, we have either intensional or extensional represention of concepts, therefore, while we follow the above definition, we also assume a rather large reification vocabulary, consisting of:

- a mapping operator for *type reification* [23][24], $\mathcal{TyR}(\varphi) \rightarrow x$ holding between predicates and individuals so that: $\forall \varphi \in \Pi\ \exists x \in \Delta^{\Pi}.\ \mathcal{TyR}(\varphi,x)$, where Π is any (first-order) ontology O_i signature, Δ is any (first-order) ontology O'_j $(O'_i \neq O_j)$ domain of quantification, and $\Delta^{\Pi} \in \Delta$ is the domain of reified predicates from Π. If φ is a class, $\mathcal{TyR}(\varphi) \rightarrow x$ is called *c-type* reification; if φ is a relation (≥2-ary predicate), $\mathcal{TyR}(\varphi) \rightarrow x$ is called *r-type* reification.
- a mapping operator for *token reification* [23][24], $\mathcal{ToR}(\tau) \rightarrow x$ holding between tuples and individuals so that: $\forall \tau \in \Delta\ \exists x \in \Delta^{\Delta}.\ \mathcal{ToR}(\tau,y)$, where $\tau \in \rho$ –ρ being a ≥2-ary predicate from Π–, Δ is the domain of any (first-order) ontology O_i, and $\Delta^{\Delta} \subseteq \Delta'$ is the domain of reified tuples from Δ –Δ' being the domain of any (first-order) ontology O'_j $(O'_i \neq O_j)$. Notice that the condition $(O'_i \neq O_j)$ is needed in order to exclude the possibility of reifying an ontology into itself.
- a mapping operator for *set reification*, $\mathrm{SR}(\sigma) \rightarrow x$, holding between sets and individuals so that: $\forall \sigma \in \Sigma\ \exists x \in \Delta^{\Sigma}.\ \mathcal{SR}(\sigma,x)$, where Σ is the set of class extensions from any (first-order) ontology O_i, each $\sigma_i \in \Sigma$ being the extension of a $\varphi_i \in \Pi$; Δ is the domain of quantification of any (first-order) ontology O'_j $(O'_i \neq O_j)$, and $\Delta^{\Sigma} \in \Delta$ is the domain of reified sets from Σ.
- a mapping operator for *cartesian set reification*, $\mathcal{CSR}(\sigma_1 x...x\sigma_n) \rightarrow \varphi$, holding between sets of cartesian products and classes so that: $\forall (\sigma_1 x...x\sigma_n) \in \Sigma^n\ \exists \varphi \in \Delta^{\Sigma n}.$ $\mathcal{CSR}[(\sigma_1 x...x\sigma_n),\varphi]$, where Σ^n is the set of relation extensions from any (first-order)

ontology O_i, each $(\sigma_1 x...x\sigma_n)_i \in \Sigma^n$ being the extension of a $\rho_i \in \Pi$; Δ is any (first-order) ontology O'_j ($O'_i \neq O_j$) domain of quantification, and $\Delta^{\Sigma n} \in \Delta$ is the domain of reified sets of cartesian products from Σ^n.

Our reification vocabulary allows to map both the intension and extension of an ontology: second-order entities (classes, relations, tuples, class extensions) are reified as individuals (first-order entities), while third-order entities (relation extensions) are reified as classes (second-order entities).

(c) ontology augmentation

We also assume an *augmentation* operator $\oplus(O_1,O_2) \rightarrow O_3$, where O_i is an ontology, which joins the signatures and domains of O_1 and O_2 into O_3: $\Pi^{O3} \equiv \Pi^{O1} \cup \Pi^{O2}$ and: $\Delta^{O3} \equiv \Delta^{O1} \cup \Delta^{O2}$. Augmentation is the default practice for ontology mapping or extension, but its use here is limited to *qualified* augmentation, as the one we propose here with *DnS*. For this reason, we do not analyze the case of overlapping signatures (which may involve merging procedures), because they are not relevant for the reification issues presented here. Moreover, domain augmentation is trivial in the case of equivalent or partial domains, which is the case addressed for augmentation with *DnS*.

(d) DnS semantics

We consider the following ontologies in order to explain the *DnS* semantics:

(1) an ontology *O*, including a signature Π^O and a domain Δ^O.

(2) the ontology *DnS*, which includes a signature Π^{dns} and a domain Δ^{dns}, and more specifically:
- Π^{dns} is partitioned into Πc^{dns} including unary predicates, and Πr^{dns} including ≥2-ary predicates. Accordingly, Δ^{dns} is the domain of quantification for *DnS*, $\Delta c^{dns} \subseteq \Delta^{dns}$, including individuals, and $\Delta r^{dns} = (\Delta^{dns} x \Delta^{dns})$, including tuples.
- Πc^{dns} includes predicates that are used as types for any reified predicate from ontologies *different* from *DnS*, and is partitioned into the set of Πcon^{dns} *concept types*, the set Πdes^{dns} of *description types*, the set Πsit^{dns} of *situation types*, and the set $\Pi coll^{dns}$ of *collection types*. Accordingly, Δc^{dns} is partitioned into the set of *concepts* Δcon^{dns}, the set of *descriptions* Δdes^{dns}, the set of *situations* Δsit^{dns}, and the set of *collections* $\Delta coll^{dns}$.[2]
- Πr^{dns} contains the binary and ternary predicates used to axiomatize *DnS*, and to link the individuals from Δc^{dns} (see below). Accordingly, Δr^{dns} contains the tuples linking the individuals from Δc^{dns}.
We also introduce here an *embedding* operator $\hookleftarrow(O_1,O_2) \rightarrow O_3$, to be read[3]: *$O_3$ results from embedding O_2 into O_1* – where O_i are ontologies, O_1 is an ontology resulting from $\hookleftarrow(O,\text{DnS}) \rightarrow O_1$, and where O and O_2 share the same domain. The embedding operator implies the following operations:

[2] The theory is still under development, and this partition is not exhaustive: for the sake of clarity, here we do not mention other partitions that have been drawn, and are introduced in the subsequent axiomatization, i.e. *information objects and figures*.

[3] An alternative reading is: *O_3 is the embedding of O_2 into O_1*.

- putting both the domain Δ_1 of $\mathcal{O}_1$, and a domain $\Delta^{\Pi 2}$ – created after the application of the mapping operator $\mathcal{TyR}(\varphi) \rightarrow x$ over all the unary predicates $\varphi_{1...n}$ and all the ≥2-ary predicates $\rho_{1...n}$ of the signature Π_2 from $\mathcal{O}_2$ – into the domain Δ_3 of $\mathcal{O}_3$
- putting the domain $\Delta^{\Delta 2}$ – created after the application of the mapping operator $\mathcal{ToR}(\tau) \rightarrow y$ over all the tuples $\tau_{1...n}$ of the domain Δ^2 from $\mathcal{O}_2$ – into the domain Δ_3 of $\mathcal{O}_3$
- putting the domain $\Delta^{\Sigma 2}$ - created after the application of the mapping operator $\mathcal{SR}(\sigma) \rightarrow z$ over all the sets $\sigma_{1...n}$ from the domain Δ^2 from $\mathcal{O}_2$ – into the domain Δ_3 of $\mathcal{O}_3$
- putting every individual $x_{1...n}$ from the domain $\Delta^{\Pi 2}$ into a set ψ_i created within either Δcon^{dns} or Δdes^{dns}.
- putting every individual $y_{1...n}$ from the domain $\Delta^{\Delta 2}$ into a set χ_i created within Δsit^{dns}.
- putting every individual $z_{1...n}$ from the domain $\Delta^{\Sigma 2}$ into a set ζ_i created within $\Delta coll^{dns}$.

(3) <u>an ontology $\mathcal{O}^+$</u> resulting from $\oplus(\mathcal{O},\mathcal{DnS}) \rightarrow \mathcal{O}^+$. It has a signature $\Pi^{O+} \equiv \Pi^{O} \cup \Pi^{dns}$, and a domain $\Delta^{O+} \equiv \Delta^{O} \cup \Delta^{dns}$.
Moreover, in the case of foundational ontologies like $\mathcal{DOLCE}$, which assume a domain as large as possible[4], the following axiom is applicable: $\forall \varphi \in \Pi^{dns}\ \exists \psi \in \Pi^{O}.\ \varphi \subseteq \psi$, and, consequently, from $\oplus(O,DnS) \rightarrow O^+$, $\Delta^{O} \subseteq \Delta^{dns}$ (e.g. for DOLCE = O and DOLCE$^+$ = O$^+$).

(4) <u>a new ontology $\mathcal{L}$</u> (e.g. a legal one), with a signature Π^L partitioned into Πc^L (for unary predicates) and Πr^L (for ≥2-ary predicates) and with a domain $\Delta^L \subseteq \Delta^O$, with $\Delta c^L \subseteq \Delta^L$, and $\Delta r^L = (\Delta^L x \Delta^L)$.
If $\mathcal{L}$ is considered for reification in $\mathcal{O}^+$, the embedding operator can be applied: $\lrcorner(\mathcal{O}^+,\mathcal{L}) \rightarrow \mathcal{O}^{+L}$.
Embedding causes Π^L to be transformed into a domain $\Delta^{\Pi L} \subseteq \Delta^{dns}$, then the $\mathcal{O}^{+L}$ domain, Δ^{O+L}, subsumes $\Delta^{\Pi L}$ by transitivity of subsumption (since $\Delta^{dns} \subseteq \Delta^{O+L}$). Δr^L, on its turn, is transformed into a domain $\Delta^{\Delta L} \subseteq \Delta^{dns}$.
The signature of $\mathcal{O}^+$ is in principle unchanged: $\Pi^{O+L} = \Pi^{O+}$, modulo possible adjustments of the taxonomy from Π^{dns} as inherited by Π^{O+}. In fact, in $\mathcal{O}^{+L}$, it is a theorem that $\forall x \in \Delta^{\Pi L}\ \forall y \in \Delta^{\Delta L}\ \exists (\varphi,\chi) \in \Pi^{dns}.\ \varphi(x) \wedge \chi(y)$. The theorem is inferrable from the axioms for the reification and embedding operators.
In practice, by admitting classes derived from the application of the cartesian-set reification operator $\mathcal{CSR}$ over n-ary relations extensions, Π^{O+} is changed in Π^{O+L} by the set of all situation classes corresponding to satisfiable descriptions.[5]
In particular:

- each $\varphi_i \in \Pi c^L$ (unary predicates) is type-reified as an individual concept $c_i \in \Delta^{\Pi cL} \subseteq \Delta con^{dns}$
- each $\rho_i \in \Pi r^L$ (≥ 2-ary predicates) is type-reified as an individual description $d_i \in \Delta^{\Pi rL} \subseteq \Delta des^{dns}$
- each $\tau_i \in \Delta r^L$ is token-reified as an individual situation $s_i \in \Delta^{\Delta rL} \subseteq \Delta sit^{dns}$
- each $\sigma_i \in \Sigma^L$ is set-reified as an individual collection $z_i \in \Delta^{\Sigma L} \subseteq \Delta coll^{dns}$, and
- each $(\sigma_1 X...X\sigma_n)_i \in \Sigma^{nL}$ is cartesian-set-reified as a situation class $S_i \in \Pi^{\Sigma nL} \subseteq \Pi^{dns}$.

[4] For example, DOLCE ranges on *possibilia*: all possible entities – independently of their actual existence; in practical terms: the entities that *are assumed to* exist according to someone for some reason.
[5] The reason why we tend to understate this change of signature is the dependence of situation classes on descriptions: no class can be added to Π^{O+} unless there is a corresponding satisfiable description in the domain $\Delta^{\Pi rL}$.

Since all domains $\Delta^{\Pi L}$, $\Delta^{\Delta rL}$, and $\Delta^{\Sigma L}$, generated by reification, are subsets of Δ^{dns}, the domain of $\mathcal{O}^{+}$ is not changed by the reification process in $\mathcal{O}^{+L}$.
Since the vocabulary $\Pi^{\Sigma nL}$, generated by reification, is a subset of the vocabulary Π^{dns}, then the vocabulary of $\mathcal{O}^{+}$ is only increased in the $\mathcal{D}n\mathcal{S}$ branching of $\mathcal{O}^{+L}$, as required by the method.
Summing up, what's happened to $\mathcal{L}$?

- the signature Π^{L} has been type-reified into Δ^{O+L}
- the domain of individuals Δc^{L} is a subset of Δ^{O} because of the shared domain assumption, then it is into Δ^{O+L} by definition
- the domain of tuples Δr^{L} has been token-reified into $\Delta^{\Delta rL} \subseteq \Delta^{O+L}$, and
- the domain of extensional sets Σ^{L} has been set-reified into $\Delta^{\Sigma rL} \subseteq \Delta^{O+L}$.

That's how context embedding works, once an ontology $\mathcal{O}$ has been augmented with $\mathcal{D}n\mathcal{S}$.
For example, given a tuple $r(a,b,t)$ in Δr^{L}, with A(a), B(b) and T(t) from Δc^{L}, belonging to Σ^{L}, and $r(a,b,t) \in$ R(x,y,z) from Πr^{L}, the predicate R is reified as a description *d,* the relationship *r* is reified as an individual situation *s*, the intensions of A, B and T are reified as concepts con^{a}, con^{b}, con^{t}, the extensions of A, B and T are reified as collections $coll^{a}$, $coll^{b}$ and $coll^{t}$, and the individual constants *a*, *b*, and *t* are already included in Δc^{L} by definition, and are related to *s* in the way presented below.
If we substitute *r(a,b,t)* with a meaningful tuple (meaning that a certain John is eating some chocolate at a certain given night), e.g.: eats(john#,chocolate#,atNight#), with Person(john#), Food(chocolate#), and TimeInterval(atNight#), and Eats(x,y,z) $\rightarrow$ Person(x) $\wedge$ Food(y) $\wedge$ TimeInterval(z), the reified entities will be:

Non-reified	**Reified**	**$\mathcal{D}n\mathcal{S}$ Type**
Eats(x,y,z)	eats#	Description(x)
Person(x)	person#	Concept(x)
Food(y)	food#	Concept(x)
TimeInterval(z)	timeInterval#	Concept(x)
eats(john#,chocolate#,atNight#)	johnEatingChocolateAtNight#	Situation(x)
extension(Person(x))	personCollection#	Collection(x)
extension(Food(y))	foodCollection#	Collection(x)
extension(TimeInterval(z))	timeIntervalCollection#	Collection(x)
extension(Person(x)∧Eats(x,y,z))	eatingPersonCollection#	Collection(x)
extension(Food(y)∧Eats(x,y,z))	eatenFoodCollection#	Collection(x)
extension(TimeInterval(z)∧Eats(x,y,z))	eatingTimeIntervalCollection#	Collection(x)
john#	*n/app*	
chocolate#	*n/app*	
atNight#	*n/app*	

(e) DnS relations

Embedding would be a very partial mapping, if the axioms existing in $\mathcal{L}$ wouldn't be mirrored in $\mathcal{O}^{+L}$. For this purpose, the predicates from Πr^{dns} come into our aid.
Here we only provide a list of the basic predicates, with the minimal axioms for them (see the DOLCE+DnS augmentation for a richer set of predicates and axioms).

Token reification is kept together by the relation *settingFor* $\in \Pi r^{dns}$, which is used to link a situation to the individual constants from the reified tuple; e.g. wrt the example above: *settingFor(s,a)*, *settingFor(s,b)*, *settingFor(s,t)*.
Moreover, we need token reification to be explicitly related to r-type reification, since a tuple must satisfy the predicates and axioms of the theory on which the elements of the tuple depend.
In order to do this, we add more predicates from Πr^{dns}, the most important of them being: *satisfies*(*s,d*), *purelySatisfies*(*s,d*), *strictlySatisfies*(*s,d*), *basiclySatisfies*(*s,d*), *completelySatisfies*(*s,d*), *parameter(p), outOfBound(x,p), uses*(*d,c*), and *classifiedBy*(*x,c*), with $s \in \Delta sit^{dns}$, $d \in \Delta des^{dns}$, $c \in \Delta con^{dns}$, $p \in \Delta con^{dns}$, $x \in \Delta c^{O}$, which are axiomatized as follows wrt to token reification:

(i) Concept(c) $\rightarrow \exists d.$ uses(d,c)
(ii) $\forall s,x.$ (Situation(s) $\wedge$ settingFor(s,x)) $\rightarrow \exists d,c.$ Description(d) $\wedge$ Concept(c) $\wedge$ uses(d,c) $\wedge$ classifiedBy(x,c)
(iii) Parameter(p) $\leftrightarrow$ Concept(p) $\wedge \exists x.$ classifiedBy(x,p) $\wedge \forall x \exists \varphi.$ [classifiedBy(x,p) $\rightarrow \varphi(x)$] $\wedge \exists \psi \in \Pi^{O}.\ \varphi(x) \rightarrow \psi(x)$] [*a parameter is a concept that always classifies at least one value from a set* φ*, and such a set is a subset of the extension of an existing specified class* $\psi \in \Pi^{O}$]
(iv) outOfBound(x,p) $\rightarrow \neg$classifiedBy(x,p) $\wedge \exists \psi \in \Pi^{O}.\ \psi(x) \wedge \forall y.$ classifiedBy(y,p) $\rightarrow$ ($\exists \varphi.\ \varphi(x) \rightarrow \psi(x)$) [*a value is out of bound of a parameter if it is in the same given datatype class as the allowed values, but in a disjoint subset*]
(v) basiclySatisfies(s,d) $\leftrightarrow \exists x.$ settingFor(s,x) $\wedge \exists c.$ Concept(c) $\wedge$ uses(d,c) $\wedge$ classifiedBy(x,c) $\wedge \neg \exists y,p.$ settingFor(s,y) $\wedge$ Parameter(p) $\wedge$ outOfBound(y,p)
(vi) completelySatisfies(s,d) $\leftrightarrow$ basiclySatisfies(s,d) $\wedge \forall c.$ uses(d,c) $\rightarrow \exists x.$ settingFor(s,x) $\wedge$ classifiedBy(x,c)
(vii) purelySatisfies(s,d) $\leftrightarrow \exists x.$ settingFor(s,x) $\wedge \forall x.$ settingFor(s,x) $\rightarrow \exists c.$ uses(d,c) $\wedge$ classifiedBy(x,c)
(viii) strictlySatisfies(s,d) $\leftrightarrow$ purelySatisfies(s,d) $\wedge \forall c.$ uses(d,c) $\rightarrow \exists x.$ settingFor(s,x) $\wedge$ classifiedBy(x,c)

The axioms mean that, if a situation is a *settingFor* some entity, that entity must be classified by a concept used by some description.
Moreover, the notions of *parameter* and *outOfBound* are needed to create constraints on different kinds of reified satisfaction relations holding between situations and descriptions.
If a situation *basiclySatisfies* a description, there exists at least one entity in its setting that is classified by a concept used by that description, and no value in the setting is out of bound of a parameter used by that description.
A situation *completelySatisfies* a description if it basicly satisfies that description, and if each concept used by the description classifies some entity(ies) in the situation setting.
A situation *purelySatisfies* a description if all the entities in its setting are classified by concept(s) used by that description.
A situation *strictlySatisfies* a description if it purelySatisfies that description, and if all the concepts used by the description classify some entity from the situation.
Notice that the outOfBound condition required in 'basiclySatisfies' and 'completelySatisfies' is trivial in 'purelySatisfies' and 'strictly satisfies', because no value in a situation that purelySatisfies a description can be out of bound. In fact, 'strictlySatisfies' is the traditional relation between reified relations and their reified tuples.

$$\begin{tikzcd}
 & \text{basiclySatisfies} & \\
\text{completelySatisfies} \arrow[ur] & & \text{purelySatisfies} \arrow[ul] \\
 & \text{strictlySatisfies} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

We also need set reification to be explicitly related to c-type reification, and the most important predicates from Πr^{dns} for that purpose are: *coveredBy*(*x,c*), *membership*(*a,x*), *unifiedBy*(*x,d*), *strictlyUnifiedBy*(*x,d*), with $c \in \Delta con^{dns}$, $x \in \Delta coll^{dns}$, $a \in \Delta c^{O}$, $d \in \Delta des^{dns}$, which are axiomatized as follows wrt to set reification:

(ix) $\text{Collection}(x) \rightarrow \exists c.\ \text{coveredBy}(x,c) \wedge \text{Concept}(c)$

(x) $\forall a,x.\ (\text{Collection}(x) \wedge \text{membership}(a,x)) \rightarrow \exists c.\ \text{coveredBy}(x,c) \wedge \text{classifiedBy}(a,c)]$

(xi) $\forall x,d.\ (\text{Collection}(x) \wedge \text{unifiedBy(x,d)}) \leftrightarrow \exists c.\ \text{coveredBy}(x,c) \wedge \text{uses}(d,c)$

(xii) $\forall x,d.\ (\text{Collection}(x) \wedge \text{strictlyUnifiedBy(x,d)}) \leftrightarrow \exists c.\ \text{coveredBy}(x,c) \wedge \text{uses}(d,c) \wedge$
$\neg \exists y.\ \text{coveredBy}(x,y) \wedge c \neq y$

(xiii) $\text{Collection}(x) \rightarrow \exists d.\ \text{Description}(d) \wedge \text{uses}(d,c) \wedge \text{unifiedBy}(x,d)$

The axioms mean that a collection must be covered by at least one concept, and all its members will be classified by that concept.
A collection is unified by at least one description that uses a concept that covers the collection.
A collection is *stricly UnifiedBy* by exactly one description iff is covered by only one concept.
StrictlyUnifiedBy also allows to capture the case of collections covered by concepts reified out of axioms (see “anonymous placeholders” at *(a)*).
(ix) is a theorem from *(i)* and *(vii)*.
Back to the example sketched in the table, the following statements hold:

Non-reified	**Reified**
Eats(x,y,z) → Person(x)	uses(eats#,person#)
Eats(x,y,z) → Food(y)	uses(eats#,food#)
Eats(x,y,z) → TimeInterval(z)	uses(eats#,timeInterval#)
Person(john#)	classifiedBy(john#,person#)
Food(chocolate#)	classifiedBy(chocolate#,food#)
TimeInterval(atNight#)	classifiedBy(atNight#,timeInterval#)
eats(john#,_#,_#)	settingFor(johnEatingChocolateAtNight#,john#)
eats(_#,chocolate#,_#)	settingFor(johnEatingChocolateAtNight#,chocolate#)
eats(_#,_#,atNight#)	settingFor(johnEatingChocolateAtNight#,atNight#)
extension(Person(x))	coveredBy(personCollection#,person#)
extension(Food(y))	coveredBy(foodCollection#,food#)
extension(TimeInterval(z))	coveredBy(timeIntervalCollection#,timeInterval#)
extension(Person(x)∧Eats(x,y,z))	coveredBy(eatingPersonCollection#,person#), strictlyUnifiedBy(eatingPersonCollection#,eats#)
extension(Food(y)∧Eats(x,y,z))	coveredBy(eatenFoodCollection#,food#), strictlyUnifiedBy (eatenFoodCollection#,eats#)
extension(TimeInterval(z)∧Eats(x,y,z))	coveredBy(eatingTimeIntervalCollection#,timeInterval#), strictlyUnifiedBy (eatingTimeIntervalCollection#,eats#)

and the following statements can be directly inferred by means of $\mathcal{DnS}$ axioms:

john# ∈ Person(x)	membership(personCollection#,john#)
chocolate# ∈ Food(y)	membership(foodCollection#,chocolate#)
atNight# ∈ TimeInterval(z)	membership(timeIntervalCollection#,atNight#)
john# ∈ (Person(x)∧Eats(x,y,z))	membership(eatingPersonCollection#,john#)
chocolate# ∈ (Food(y)∧Eats(x,y,z))	membership(eatenFoodCollection#,chocolate#)
atNight# ∈ (TimeInterval(z)∧Eats(x,y,z))	membership(eatingTimeIntervalCollection#,atNight#)
[eats(john#,chocolate#,atNight#)] ∈ Eats(x,y,z)	satisfies(johnEatingChocolateAtNight#,eats#)

(e) Axioms on reification kinds
From the reification vocabulary definitions, and $\mathcal{DnS}$ relations, we can infer that:
$\forall \sigma,x.\ (\mathcal{SR}(\sigma) \rightarrow x) \rightarrow \exists \varphi,y.\ [(\mathcal{TyR}(\varphi) \rightarrow y) \wedge \text{coveredBy}(x,y)]$
(set reification implies c-type reification)
$\forall \tau,x.\ (\mathcal{ToR}(\tau) \rightarrow x) \rightarrow \exists \rho,y.\ [\tau\in\rho \wedge \rho(a,\ldots,n) \wedge (\mathcal{TyR}(\rho) \rightarrow y)] \wedge \text{satisfies}(x,y)$
(token reification implies r-type reification)
$\forall(\sigma_1 X \ldots X\sigma_n),\varphi.\ [(\mathcal{CSR}(\sigma_1 X\ldots X\sigma_n) \rightarrow \varphi) \wedge \exists x.\ \varphi(x)] \rightarrow \exists \rho,y.\ [\rho(a,\ldots,n) \wedge (\mathcal{TyR}(\rho) \rightarrow y)] \wedge \text{satisfies}(x,y)$
(cartesian set reification implies r-type reification)

(f) Creative use of $\mathcal{DnS}$
The use of DnS has been profiled here as an ontology of reification, i.e. used *in presence* of an actual non-reified, non-embedded ontology. While this approach seems promising (e.g. for ontology design patterns, ontology elements annotation models, mapping, etc.), the very reason for its definition lies in its capability to build ontologies that embed contexts (viewpoints, configurations) in their domain, or that profile incomplete, "common-sense-like" theories with a potential for discussion, comparison, dynamic enhancement, etc.
In other words, DnS can be used *in absence* of an actual ontology to be reified and embedded: we can do *as if* that ontology exist, therefore introducing descriptions, concepts, situations, and collections only on the basis of DnS axioms. When used in this way, DnS "sketches" an ontology as an embedded context, which can be possibly *de-reified* by means of the converse methodology. We call this use *creative*.
As a matter of fact, after looking at the tables above, it results that reification can be a useful means for meta-level reasoning, but the example seems also pretty *ad hoc*. For example, the creation of collections for persons, food and time intervals in general is hardly very meaningful (although technically sound, for example when dealing with most description logics, which cannot express n-ary relations).
On the contrary, the creative use of embedded contexts is focused, since it only encodes the individuals and statements that are useful or known to the modeller e.g. for a special purpose, or when knowledge is fragmented.

3.3.3 A DOLCE-related axiomatization for the basic notions of DnS

In this section we introduce DnS as an augmentation to DOLCE, then assuming that the domain of DnS is a subset of DOLCE's.
The main intent of DnS is enabling the ontological talk about non-physical objects, especially social and knowledge objects. The rationale is that the properties that we attribute to entities are entities themselves, and that we can treat them as "knowledge" or "information" objects (see below).
In more detail, DnS is based on a fundamental distinction between **descriptions** (for instance, in the legal domain, *legal* descriptions, or *conceptualizations*, which encompass laws, norms,

regulations, crime types, etc.) and **situations** (again, in the legal domain, *legal facts* or *cases*, which encompass legal states of affairs, non-legal states of affairs that are relevant to the Law, and purely juridical states of affairs). This distinction may be used somewhat recursively (in the example of the legal domain, we may use the distinction to represent meta-juridical conceptualizations, i.e. *meta-norms*, or norms about norms).
Differently from most ontologies, DnS has been built in order to facilitate ontology-driven data entry to experts in knowledge-intensive domains. In fact, its very first formulation was a Design Pattern represented by means of a UML class diagram (Fig.3).

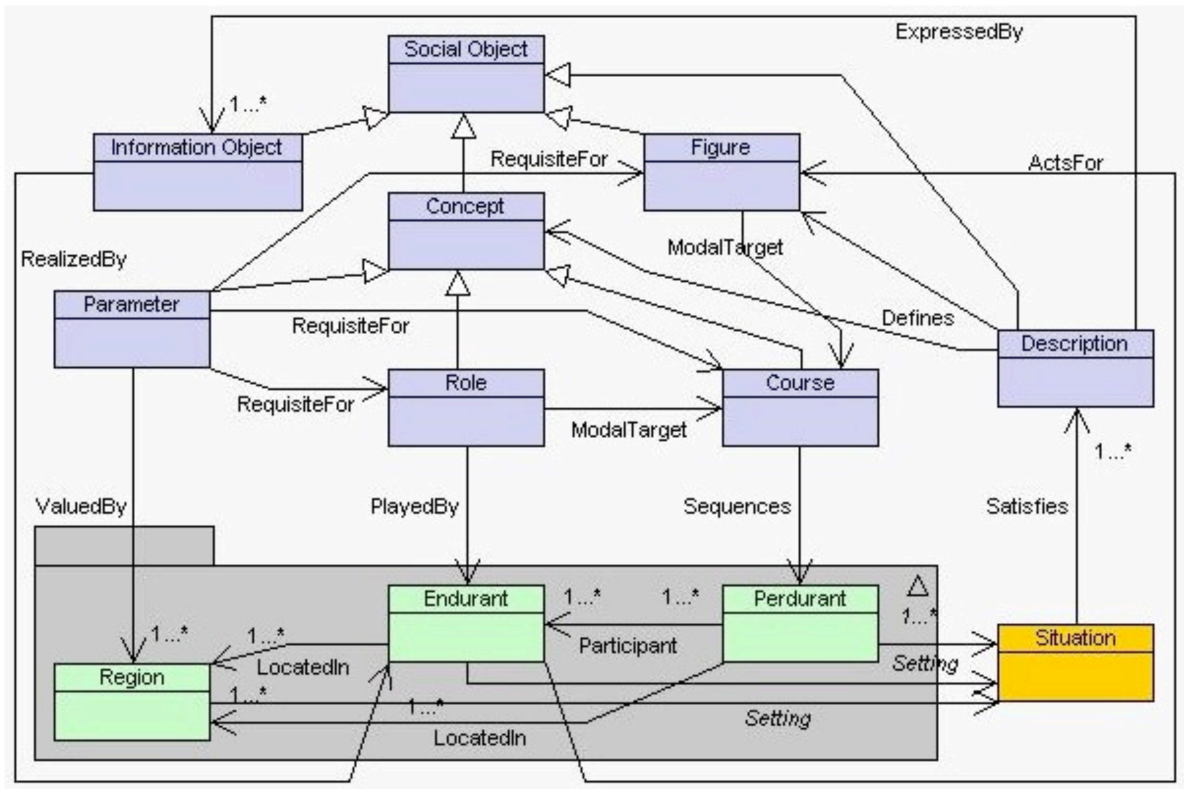

Fig. 3 The DnS Design Pattern as a UML class diagram. The lower part of the pattern is called the *ground ontology*, the higher is called the *descriptive ontology*; a situation satisfies a description if the two parts match according to specified rules. Part of the structure matching expected by situations satisfying descriptions appears symmetrically from the overall shape.

DnS basic predicates and axioms are presented in the following. Firstly, we introduce the signature for DnS in DOLCE (including both DOLCE predicates, which are underlined here, and predicates defined or axiomatized in DnS), divided here into a) unary predicates, b) binary predicates, and c) ternary predicates.

Unary predicates: { *Particular, Perdurant, Endurant, PhysicalObject. AgentivePhysicalObject, NonPhysicalObject, AgentiveSocialObject, NonAgentiveSocialObject, CognitiveAgentivePhysicalObject, Agent, CognitiveAgent, Description, Plan, Concept, Role, Course, Task, Parameter, Situation, Figure, AgentiveFigure, Collection, OrganizedCollection, Collective, Region, TimeInterval* }

Binary predicates: { *PresentAt, Part, ProperPart, Defines, Satisfies, P-Satisfies, R-Satisfies, C-Satisfies, Setting, SettingFor, Uses, SubconceptOf, Specializes, TemporalLocation, SpatialLocation, ReferencedTemporalLocation, ReferencedSpatialLocation, Covers, Characterizes, Unifies, SRD* }

Ternary predicates: { *SpecificallyDependsOn, GenericallyDependsOn, GenericConstituentOf, Participant, Conceives, Sit-Conceives, DirectlySit-Conceives, Classifies, PlayedBy, Sequences,* *ValuedBy, Adopts, Creates, DeputedBy, ModalTarget, AttitudeTowards, ActsFor, RequisiteFor, Membership, ExtensionallyEquivalent* }

For presentation reasons, sometimes we anticipate axioms and definitions for a predicate that uses other predicates that have not yet been axiomatized. This is not a good practice from the logical viewpoint, but helps maintaining an intuitive narrative.
The time indexing variables ("*t*") in the following formulas are not explicitly typed, but are to be assumed as typed by the predicate *TimeInterval*.

A **Description** is a (non-agentive) social object which represents a conceptualization, hence it is generically dependent on some agent and communicable [16]. Example of descriptions are regulations, plans, laws, diagnoses, projects, plots, techniques, etc.:

(A1) Description(x) → NonAgentiveSocialObject(x)
(A2) Description(x) → ∃y,t. AgentivePhysicalObject(y) ∧ GenericallyDependsOn(x,y,t)
(A3) Description(x) → ∀y. Part(x,y) → NonPhysicalObject(y)

Like physical objects, social ones have a lifecycle, can have parts, etc. Unlike physical objects, social (like all non-physical) ones are **generically dependent on** some agentive physical object. The relationship between a description and an agent is the following:

(A4) Conceives(x,y,t) → GenericallyDependsOn(y,x,t) ∧ Agent(x) ∧ Description(y)

Hence, a description generically depends on some agent which is (at some time)[6] able to **conceive** it. *Agent* is introduced here as a primitive (subclass of *Endurant*), and is axiomatized later here.

Agentivity in DOLCE is not (explicitly) defined, but by means of DnS we can now define it as follows:

(D1) AgentivePhysicalObject(x) $=_{df}$ PhysicalObject(x) ∧ ∃y,t. Description(y) ∧ Conceives(x,y,t)

In simple words, *agentivity* is defined here in a wide sense as implying *conception* (to be characterized in a dedicated – but not developed as yet – ontology of mind). A conception only requires *intentionality* in Brentano's terms [20], i.e., the ability to represent something to oneself.
Compliying i.e. with the BDI paradigm - when it attributes to **cognitive agents** not only the ability of self-representing descriptions like beliefs, desires, and intentions, but also of representing the actions schemata necessary to the achievement of their goals -, we can also identify a second, stronger sense of agentivity that involves the conceiving of descriptions *about descriptions* ("meta-description"):

(D2) CognitiveAgentivePhysicalObject(x) $=_{df}$ AgentivePhysicalObject(x) ∧ ∃y,z,c,t. Description(y) ∧ Description(z) ∧ Concept(c) ∧ Conceives(x,y,t) ∧ Defines(y,c) ∧ Classifies(c,z,t)

D2 says that a CognitiveAgentivePhysicalObject is able to conceive descriptions that define concepts that classify other descriptions; in practice a cognitive agent is able to compare, reuse, reason over alternative descriptions by framing them within a meta-level description.
Conceptions can be held by *agentive social objects* (e.g. organizations) as well, through the cognitive agentive physical objects they depend on:

[6] Notice, however, that since ternary relationships are not supported by OWL-DL, time is ignored in the OWL-DL version given in the Annex, and will have to be managed in an extrinsic way in practical applications. There is also the possibility of *reifying* temporalized relations by using DnS, but this has not been considered here.

(A5) $(\text{Conceives}(x,y,t) \wedge \text{AgentiveSocialObject}(x)) \rightarrow \exists z,t.\ \text{CognitiveAgentivePhysicalObject}(z) \wedge \text{GenericallyDependsOn}(x,z,t) \wedge \text{Conceives}(z,y,t)$

The way agents create, choose, or transform their conceptualizations, however, is extremely diversified. We do not enter here this difficult area, leaving it to future investigation. On the other hand, we need some preliminary distinction, in order to relate agents and descriptions that represent those conceptualizations.
In order to simplify our formulas and try to comply with the common-sense polysemy of "agent", we define it here as a catch-all class, encompassing either agentive physical objects or agentive social objects:

(D3) $\text{Agent}(x) =_{df} \text{AgentivePhysicalObject}(x) \vee \text{AgentiveSocialObject}(x)$

We also introduce a restricted class for *cognitive agents:*

(D4) $\text{CognitiveAgent}(x) =_{df} \text{CognitiveAgentivePhysicalObject}(x) \vee (\text{AgentiveSocialObject}(x) \wedge \exists y,t.\ \text{CognitiveAgentivePhysicalObject}(y) \wedge \text{GenericallyDependsOn}(x,y,t))$

An important relation between agents and descriptions is **creation**, implying that a description is specifically dependent on a cognitive agent that firstly conceives of it at some time interval (of a magnitude of choice in a given context):

(A6) $\text{Creates}(x,y) \rightarrow \text{CognitiveAgent}(x) \wedge \text{SpecificallyDependsOn}(y,x) \wedge \exists t.\ \text{Conceives}(x,y,t) \wedge \neg\exists x',t'.\ t'<t \wedge \text{Conceives}(x',y,t')$

Another important relation between agents and descriptions is **adoption** (requiring creation and previous conceiving):

(A7) $\text{Adopts}(x,y,t) \rightarrow \text{Conceives}(x,y,t) \wedge \text{CognitiveAgent}(x) \wedge \text{Description}(y) \wedge \exists z,t'.\ \text{CognitiveAgentivePhysicalObject}(z) \wedge \text{Creates}(z,y,t') \wedge t'<t$

(A8) $\text{Adopts}(x,y,t) \rightarrow \exists t_1.\ >(t_1,t) \wedge \text{Conceives}(x,y,t_1)$

Descriptions have typical components, called *concepts* (introduced later in some detail). Concept types can vary according to the ground ontology that is taken into account. This version of DnS takes DOLCE as its ground ontology.

A **situation** is a non-agentive social object which represents a state of affairs or relationship, or tuple, or fact (see DnS semantics in 3.3.2), under the assumption that its components 'carve up' a view (a *setting*) on the domain of an ontology by virtue of a description. A situation aims at representing the referent of a "cognitive disposition" towards a world, thus reflecting the willingness, expectation, desire, belief, etc. to carve up that world in a certain way. Consequently, a situation has to *satisfy* a description (see below).

(D5) $\text{Situation}(x) =df\ \text{NonAgentiveSocialObject}(x) \wedge (\exists y.\ \text{Description}(y) \wedge \text{Satisfies}(x,y)) \wedge (\exists z.\ \text{Particular}(z) \wedge \neg\text{Situation}(z) \wedge \text{Setting}(z,x))$

(A9) $\text{Situation}(x) \rightarrow \forall y.\ \text{Part}(x,y) \rightarrow \text{Situation}(y)$

The **setting** relation holds between situations and particulars from the ground ontology. At least a perdurant must exist in the situation setting:

(A10) $\text{SettingFor}(x,y) \rightarrow \text{Situation}(x) \wedge \text{Particular}(y) \wedge \neg\text{Situation}(y)$

(A11) $\text{SettingFor}(x,y) \rightarrow \exists z.\ \text{Perdurant}(z) \wedge \text{SettingFor}(x,z)$

(A12) Setting(x,y) =df SettingFor(y,x)

A situation can have both an ***internal*** and a ***referenced*** **time** or **space**. While its internal time/space is agent-dependent (it's the time/space of its conceiving by an agent), its referenced time/space is constituted by the time and space of the ("referenced") particulars in the setting[7]: more specifically, the referenced temporal location of a situation has only parts that are temporal locations of perdurants that are in the setting of the situation:

(A13) $\forall$s,t1,t2. [(Situation(s) $\wedge$ ReferencedTemporalLocation(s,t1) $\wedge$ Part(t1,t2)) $\rightarrow$ $\exists$p. (Perdurant(p) $\wedge$ SettingFor(s,p) $\wedge$ TemporalLocation(p,t2))]

and, conversely, if a situation has a perdurant in the setting, the temporal location of the perdurant is part of its referenced temporal location:

(A14) $\forall$s,p. [Situation(s) $\wedge$ Perdurant(p) $\wedge$ SettingFor(s,p)] $\rightarrow$ $\exists$t1,t2. ReferencedTemporalLocation(s,t1) $\wedge$ TemporalLocation(p,t2) $\wedge$ Part(t1,t2)

moreover, if a situation has an endurant in the setting, the temporal location of a perdurant in the same setting, in which that endurant participates, is also part of the temporal location of the situation:

(A15) $\forall$s,e. [Situation(s) $\wedge$ Endurant(e) $\wedge$ SettingFor(s,e)] $\rightarrow$ $\exists$t1,t2,p. ReferencedTemporalLocation(s,t1) $\wedge$ Perdurant(p) $\wedge$ TemporalLocation(p,t2) $\wedge$ Participant(p,e,t2) $\wedge$ Part(t1,t2)

The referenced spatial properties of a situation follow from similar axioms:

(A16) $\forall$s,r1,r2. [(Situation(s) $\wedge$ ReferencedSpatialLocation(s,r1) $\wedge$ Part(r1,r2)) $\rightarrow$ $\exists$e. (Endurant(p) $\wedge$ SettingFor(s,e) $\wedge$ SpatialLocation(p,r2))]

(A17) $\forall$s,e. [Situation(s) $\wedge$ Endurant(e) $\wedge$ SettingFor(s,e)] $\rightarrow$ $\exists$r1,r2. ReferencedSpatialLocation(s,r1) $\wedge$ SpatialLocation(e,r2) $\wedge$ Part(r1,r2)

(A18) $\forall$s,p. [Situation(s) $\wedge$ Perdurant(p) $\wedge$ SettingFor(s,p)] $\rightarrow$ $\exists$r1,r2,e. ReferencedSpatialLocation(s,r1) $\wedge$ Endurant(e) $\wedge$ SpatialLocation(e,t2) $\wedge$ Participant(p,e,t2) $\wedge$ Part(r1,r2)

Implicitly, (A13) to (A18) state that if a situation has an external temporal – respectively, spatial – location, that location is the mereological sum of the locations of the particulars in the setting. For example, the time of *World War II* might span from the German invasion of Poland in 1939 to the Yalta conference in 1945; its space might include most of the Earth surface. Hence, the setting relation is not temporalized, because the time of Setting(x,y) can be inferred from the previous axioms.
Examples of situations, related to the examples of descriptions above, are: facts, desired states, plan executions, legal cases, diagnostic cases, attempted projects, performances, technical actions, system functioning, ecosystems, finished working products, etc. (Tab.1).

Description	Situation	*Description*	Situation
Theory	Model	*Diagnosis*	Diagnostic case
Proposition	Fact	*Project*	Project undertaking
Relation	Relationship	*Play*	Performance
Belief	State of affairs	*Script*	Movie

[7] All 't'' variables in the formulas denote *time intervals*.

Desire	(Desired) state	*Coding system*	Information encoding
Plan	Plan execution	*Communication rules*	Communication setting
Workflow	Work being done	*Technique*	(Technical) activity
Legal Norm	Legal case	*Instruction*	(Guided) activity
Law of nature	Fact in nature	*Rules of game*	Play a game
Contract	Contract enforcement	*System specification*	System functioning
Product design	Finished working product	*Constraints in an ecosystem*	Ecosystem

Tab.1 Examples of classes of *descriptions* and corresponding classes of *situations*.

Similarly to the *conceives* relation holding between agents and descriptions, which represents intentionality in its largest, cognitive sense, a *sit-conceives* relation holding between agents and situations is introduced to represent two assumptions: i) an agent can perceive an observable entitiy only based on some (conscious or not) intentionality, and ii) agents can have a relation to a situation without considering themselves in the setting of the situation, even when there exists at least one viewpoint according to which they are in that setting.
In simpler terms, an agent can be in the setting of a situation, but the same agent can also be the conceptualizer of the situation based on its conceptualization skills.
Notice that the circumstances of perception are not necessarily bound to the same circumstances of the conceived situation, since an agent can use a prosthesis, a recorded report, a witness, etc. Of course, the evaluation of perception (an epistemological issue) requires to take into account both the circumstances of the conceived and the perceived situations.
The **sit-conceives** relation is introduced as follows:

(A19) Sit-Conceives(x,y,t) → Agent(x) ∧ Situation(y) ∧ ∃z. Description(z) ∧ Conceives(x,z) ∧ Satisfies(y,z)
(A20) Sit-Conceives(x,y,t) → SpecificallyConstantlyDependsOn(y,x) ∧ Agent(x) ∧ Situation(y)
(A21) Sit-Conceives(x,y,t) → ∃z. Situation(z) ∧ SettingFor(z,x)
(A22) DirectlySit-Conceives(x,y,t) → ∃z. Situation(z) ∧ SettingFor(z,x) ∧ Part(y,z)

The axioms for Sit-Conceives do not provide necessary and sufficient criteria. They state that: i) situation conceptualization is mediated by conception, ii) a conceived situation is then specifically constantly dependent on the conceiving agent, iii) there exists at least one situation that provides the setting for the conceiving agent, iv) a *direct sit-conceiving* occurs when a perceived situation is part of the conceived one. (A22) does not make any claim about what kind of involvement is required from the agent to state that the circumstances of perception are part of the conceived situation: as a matter of fact, in different domains and communities, different criteria may apply.

A **Concept**, like a description, is a non-agentive social object, which is **defined by** a description. Once defined, a concept can be **used in** other descriptions. We firstly introduce *defined by* as a subrelation of *uses*. **Uses** is the proper part relation holding between descriptions and concepts (A27) or figures (A30):

(D6) Uses(x,y) $=_{df}$ProperPart(x,y) ∧ Description(x) ∧ (Concept(y) ∨ Figure(y))

Defines is a subrelation of Uses. Defined concepts and figures specifically depend on defining descriptions:

(A23) $Defines(x,y) \rightarrow Uses(x,y) \land SpecificallyDependsOn(y,x)$
(A24) $Uses(x,y) \rightarrow \exists z. Description(z) \land Defines(z,y)$

For example, a *car design* can define an *engine role* that can be restricted to classify only instances from a specified class of *artifacts*.
The **classifies** relation relates concepts to particulars (and even concepts to concepts) at some time. There are several kinds of concepts reified in DnS, the primary ones (**role**, **course**, and **parameter**) being distinguished by the categories of entities they classify in DOLCE:

(A25) $Defines(x,y) \rightarrow Description(x) \land Concept(y)$
(A26) $Classifies(x,y,t) \rightarrow Concept(x) \land Particular(y) \land TimeInterval(t)$
(A27) $Concept(x) \rightarrow NonAgentiveSocialObject(x) \land \exists y. Defines(y,x) \land Description(y)$
(D7) $Role(x) =_{df} Concept(x) \land \forall y,t. Classifies(x,y,t) \rightarrow Endurant(y)$
(D8) $Course(x) =_{df} Concept(x) \land \forall y,t. Classifies(x,y,t) \rightarrow Perdurant(y)$
(D9) $Parameter(x) =df Concept(x) \land \exists y,t. Classifies(x,y,t) \land \forall y. Classifies(x,y,t) \rightarrow Region(y)$

Examples of roles[8] are: *manager, student, assistant, actuator, toxic agent,* etc. Examples of courses are *routes, pathways, tasks*, etc. Examples of parameters are: *speed limits, allowed colors* (e.g. for a certain book cover), *temporal constraints*, etc.
There are relations between concepts. For example, some concepts are apparently classified by other concepts, e.g. a *manager* that is also a *buyer*. In most cases, however, they are not classified, but they are actually **subconcepts** (e.g. it's not the role *manager* that actually plays the role *buyer*, but i'ts someone playing the role *manager* that is also playing the role *buyer*).
The subconcept relation holds between concepts:

(A28) $SubconceptOf(x,y) \rightarrow Concept(x) \land Concept(y)$

In particular, concepts can be **specialized by** other concepts, e.g. *president of the Italian republic* specializes *president of republic*[9]:

(A29) $Specializes(x,y) \rightarrow SubconceptOf(x,y)$
(T1) $\forall x,y,t.\exists z. (Classifies(x,y,t) \land Specializes(x,z) \land x \neq z) \rightarrow Classifies(z,y,t)$

Figures, or **social individuals** (either agentive or not), are other social objects defined by descriptions; differently from concepts, however, they do not classify any particular:

(A30) $Figure(x) \rightarrow SocialObject(x)$
(A31) $Figure(x) \rightarrow \exists y. Description(y) \land Defines(y,x)$
(A32) $Figure(x) \rightarrow \neg\exists y,t. Classifies(x,y,t)$

Examples of figures are organizations, political-geographic objects, sacred symbols, *personas*, personal or shared façades, etc.

Agentive figures are those which can conceive descriptions, by means of some agentive physical object that *acts for* the figure (for instance, as *representative* or *delegate*).

(D10) $AgentiveFigure(x) =_{df} Figure(x) \land AgentiveSocialObject(x) \land \exists y. Description(y) \land Conceives(x,y,t)$

[8] There are additional axioms to characterize roles as *anti-rigid* and *founded* concepts. For definitions of anti-rigidity and of foundation, see [16].
[9] Cf. [16] for more examples.

(A33) $(AgentiveFigure(x) \wedge Conceives(x,y,t)) \rightarrow \exists z,t.\ AgentivePhysicalObject(z) \wedge Conceives(z,y,t)$

Agentive figures are established by a society or community; hence, they can act like a physical agent, can play roles, etc.. In our ontology, this formally amounts to have at least two descriptions, one defining an agentive figure, and another defining a role played by that agentive figure:

(A34) $AgentiveFigure(x) \rightarrow Figure(x) \wedge \exists y,z,w,t.\ Description(y) \wedge Role(z) \wedge Description(w) \wedge y \neq w \wedge Defines(y,z) \wedge Defines(w,x) \wedge Classifies(z,x,t)$

Typical agentive figures are societies, organizations, and in general all socially constructed persons.
Figures are not dependent on roles defined or used in the same descriptions in which the figures themselves are defined or used, but they can act because they **depute** some tasks to some of those roles, which, in turn, must classify some individual agent. In other words, when a figure is classified by some agentive role, or participates in some event, it can be classified or participate because there is someone (or something) that is classified by other roles in the descriptions that define or use the figure. The relation is temporalized in order to suggest that a figure can preserve its identity despite changes of deputed roles (even though there are cases in which the identity of a figure is inextricably bound to one - or more - of its roles):

(A35) $DeputedBy(r,f,t) \rightarrow Role(r) \wedge Figure(f) \wedge TimeInterval(t) \wedge \exists c,d.\ Course(c) \wedge Description(d) \wedge Uses(d,r) \wedge Uses(d,f) \wedge Uses(d,c) \wedge ModalTarget(r,c,t)$
(A36) $DeputedBy(r,f,t) \rightarrow \exists r1,t1.\ Role(r1) \wedge Classifies(r1,f,t1)$

Those roles classify endurants, which result to **act for** the figure:

(A37) $ActsFor(e,f,t) \rightarrow \exists r,t1.\ Role(r) \wedge DeputedBy(r,f,t1) \wedge Classifies(r,e,t)$
(A38) $(ParticipatesIn(f,p,t) \wedge AgentiveFigure(f)) \rightarrow \exists e.\ ActsFor(e,f,t) \wedge Participant(p,e,t)$

For example, an employee *acts for* an organization that *deputes* the role (e.g. *turner*) that *classifies* the employee. Simply put, a guy working as a turner at FIAT acts for (or *on behalf of*) FIAT, so that in actions classified by turning tasks, if FIAT participates, so necessarily does the turner.
In complex figures, like organizations or institutions, a *total agency* is possible (usually limited to some actions), when an endurant plays a *delegate* or a *representative* role deputed by the figure[10]. Since figures are social objects, it can happen to find figures that *act for* other figures[11].

The *classifies* relation is specialized by three subrelations: **played by**, **sequences**, and **valued by**, which apply to three different categories in DOLCE (Endurant, Perdurant, and Region, from (D#-#))[12]:

(D11) $PlayedBy(x,y,t) =_{df} Role(x) \wedge Classifies(x,y,t)$
(D12) $Sequences(x,y,t) =_{df} Course(x) \wedge Classifies(x,y,t)$
(D13) $ValuedBy(x,y,t) =_{df} Parameter(x) \wedge Classifies(x,y,t)$

10 Cases of full delegation or representation, however, are quite unusual, and even prohibited in some legal contexts.
11 Indeed, this kind of situation is at work in many contemporary settings and can reach great complexity, as e.g. in financial *chinese boxes*, which can even create an *agency loop*.
12 Only three categories from DOLCE have been assigned a concept type at the descriptive layer, because the resulting pattern is simpler and there is no loss of relevant knowledge, at least in applications developed until now.

Roles or figures and courses are related by relations expressing the **modalities** that (players of) roles and figures can have towards a course. The relation is temporalized to suggest that a description can preserve its identity against changes of the structure among components (though there can be mandatory structures for description identity):

(A39) $\text{ModalTarget}(x,y,t) \rightarrow (\text{Role}(x) \vee \text{Figure}(x)) \wedge \text{Course}(y) \wedge \text{TimeInterval}(t)$

Modal target is the descriptive counterpart of the "participant-in" relation used in the ground ontology, i.e. modalities are *participation modes*. In other words, the ModalTarget relation can be used to reify, for instance, *alethic, epistemic*, or *deontic* operators. For example, a person is usually *obliged* to drive in a way that prevents her from hurting other people; or a person can have the *right* to express her ideas. A subclass of modal-target relations representing dispositional attitudes (such as beliefs or desires) towards courses is called *AttitudeTowards*, and it holds only when roles are played by cognitive agents:

(A40) $\text{AttitudeTowards}(x,y,t) \rightarrow \text{ModalTarget}(x,y,t) \wedge \text{Task}(y) \wedge \text{TimeInterval}(t) \wedge \forall e.\ \text{Classifies}(x,e) \rightarrow \text{CognitiveAgent}(e)$

Consider, for instance, the following, more complex example: a BDI application to a certain ordered set of tasks including initial conditions (beliefs), final conditions (desires), and ways to reach goals (intentions). In other words, moving from beliefs to goals is a way of bounding one or more agent(s) to a sequence of actions. In the plan ontology this intuition is deepened considerably.

Parameters, roles, figures or courses are related by a **requisite for** relation, expressing the kind of requisites that particulars which are classified by roles or courses should have. The relation is temporalized to suggest that a description can preserve its identity against changes of structuring among components (though there can be mandatory structures for description identity):

(A41) $\text{RequisiteFor}(x,y,t) \rightarrow \text{Parameter}(x) \wedge (\text{Role}(x) \vee \text{Figure}(x) \vee \text{Course}(y)) \wedge \text{TimeInterval}(t)$

Requisites are constraints over the values of the qualities of particulars. When a situation satisfies a description that uses parameters, endurants and perdurants that constitute the situation must have attributes that range between the boundaries stated by said parameters (in terms of DOLCE, particulars must have qualities that are mapped onto certain value ranges of regions). For example, if *speed limit of 50kmph* is a requisite for a *driving task*; a satisfying situation will have any *speed* of driving (e.g. in an instance of *driving in Rome by car*) to be less or equal to *50kmph*.

Collections are social objects (either agentive or not) which, although not defined by a description, (generically, one-sidedly, and temporarily) depend on member entities and (specifically, one-sidedly and constantly) depend on *concepts*, hence indirectly on descriptions; in some cases, collections can depend also on *figures*. While we could talk in general of collections of any kind of particulars (events, objects, abstracts, etc.), we focus here on collections of *endurants*, and therefore, on the concepts that classify them, i.e. *roles*.

(D14) $\text{Collection}(x) =_{df} \text{SocialObject}(x) \wedge \exists r.\ \text{Role}(r) \wedge \forall w,t.\ \text{GenericConstituentOf}(w,x,t) \rightarrow \text{Classifies}(r,w,t) \wedge \exists y,z,t1.\ \text{Endurant}(y) \wedge \text{Endurant}(z) \wedge y \neq z \wedge \text{GenericConstituentOf}(y,x,t1) \wedge \text{ConstituentOf}(z,x,t1) \wedge \text{Classifies}(r,y,t1) \wedge \text{Classifies}(r,z,t1)$

A **membership** relation is defined on collections:

(D15) $Membership(e,c,t) =_{df} GenericConstituentOf(e,c,t) \wedge Endurant(e) \wedge Collection(c) \wedge \exists r. Role(r) \wedge Classifies(r,e,t)$

In other words, a collection is a social object whose members are all classified by a same role, and which has at least two endurants as actual members.
Two or more collections can be *extensionally* equivalent and still not be the same collection. Each collection needs a unifying description which provides its *intensional* identity criterion:

(D16) $ExtensionallyEquivalent(x,y,t) =_{df} Collection(x) \wedge Collection(y) \wedge \forall z. Membership(z,x,t) \leftrightarrow Membership(z,y,t)$

The role shared by members has a **covering** relation towards the collection:

(D17) $Covers(r,c) =_{df} Role(r) \wedge Collection(c) \wedge \forall w,t. Membership(w,c,t) \rightarrow Classifies(r,w,t)$

Summing up, a *concept* is defined by a description and can classify some particular (a role being a concept that classifes only endurants), while a *figure* is defined by a description, but cannot classify any entity, and must act by means of something else. A *collection*, on the other hand, is not defined by a description, and cannot classify any particular, but has members that are classified by at least one and the same *role* (defined by some description). Figures and collections are social *individuals*, while concepts are not. We may say that collections are *emergent* social individuals because, unlike figures and concepts, they do not need to be explicitly *defined by* a description.

Organized collections, however, introduce a different unity criterion for collections. They can be conceived as *characterized* by further roles played by some (or all) members of the collection, and related among them through the social objects (figures, descriptions, collections) that either use or depute or are covered by them:

(D18) $Characterizes(r,c) =_{df} Role(r) \wedge Collection(c) \wedge \exists e,f,o,s,t. (Figure(o) \vee Description(o) \vee Collection(o)) \wedge Role(s) \wedge e \neq f \wedge r \neq s \wedge Membership(e,c,t) \wedge Membership(f,c,t) \wedge (Uses(o,r) \vee DeputedBy(r,o,t) \vee Covers(r,o)) \wedge (Uses(o,s) \vee DeputedBy(s,o,t) \vee Covers(s,o)) \wedge Classifies(r,e,t) \wedge Classifies(s,f,t)$

(T2) $Characterizes(r,c) \rightarrow \exists s. Role(s) \wedge r \neq s \wedge Characterizes(s,c)$

(D19) $OrganizedCollection(c) =_{df} Collection(c) \wedge \exists r,s. Characterizes(r,s) \wedge Characterizes(s,c)$

From previous definitions and theorems, we can claim that collections specifically depend on some description:

(A42) $Collection(c) \rightarrow \exists d. Description(d) \wedge SpecificallyDependsOn(c,d)$

We can therefore build a new relation of **unification** between collections and the descriptions on which they depend. *Unification* is axiomatized by means of *sufficient* conditions (A35-37), and is not temporalized, since changing the description (differently from changing some members) creates a new collection:

(A43) $Unifies(x,y) \rightarrow Description(x) \wedge Collection(y)$

(A44) $Covers(x,y) \rightarrow \exists d. Description(x) \wedge Defines(d,x) \wedge Unifies(d,y)$

(A45) $Characterizes(x,y) \rightarrow \exists d. Description(x) \wedge Defines(d,x) \wedge Unifies(d,y)$

(A46) $(Characterizes(x,y) \wedge \exists f. DeputedBy(x,f,t)) \rightarrow \exists d. Description(x) \wedge Uses(d,f) \wedge Unifies(d,y)$

From (A23), (D8), (D15), (D18) and (A40) we can derive that a collection must be unified by at least one description, which provides to said collection its unity criterion:

(T3) $\text{Collection}(c) \rightarrow \exists d.\ \text{Description}(d) \wedge \text{Unifies}(d,c)$

We can imagine roles that are used by, deputed by, or cover more than one description, figure, or collection[13]. In other words, characterizing roles can be related among them through some *composition* (or *bundle*) of descriptions, figures, or collections. We expect to extend our axiomatization to compositions and bundles in the near future.

A **collective** is a collection of *agents*:

(D20) $\text{Collective}(c) =_{df} \text{Collection}(c) \wedge \forall x,t.\ \text{Membership}(x,c,t) \rightarrow \text{Agent}(x)$

Similarly to all collections, collectives are covered or characterized by roles and eventually unified by some description. In collectives, roles are played by agents. Since agents can participate in, and/or conceive, plans, roles can be assigned *modalities* or *attitudes* (participation modes) *towards* courses that can sequence actions.
A typology of collectives will be introduced in next versions which mainly exploits the presence of a *plan* as the core unity criterion for a bundle of descriptions that originates collective action. The prior existence of this plan, its conceivability in the members of the collective, and the amount, the modes, and the types of existence and conceivability will be the criteria used to build our typology.

3.3.4 Reified satisfaction in DnS

In this section we try to answer the question: «*how to formally represent the (possible, actual, obliged, desired, etc.) correspondence between situations and descriptions*?». The basic semantics given in 3.3.2 simply states that reified tuples must satisfy reified relations. Here we build on that to see how that satisfaction can be introduced, checked, etc.

(a) satisfaction

The **satisfies** relation holds between situations and descriptions, and implies that at least some concept in a description must classify at least some particular in the situation setting (see 3.3.2):

(A47) $\text{Satisfies}(x,y) \rightarrow \text{Situation}(x) \wedge \text{Description}(y)$
(A48) $\text{Satisfies}(x,y) \rightarrow \exists z.\ \text{Concept}(z) \wedge \text{Uses}(y,z) \wedge \exists w,t.\ \text{SettingFor}(x,w) \wedge \text{Classifies}(z,w,t)$

For specialized descriptions additional constraints have to be given in order to reason over the satisfaction of candidate situations (see below the constraints for plans).

(b) redundant satisfaction

(A48) is quite generic and even counterintuitive from a logical viewpoint. This “relaxed” semantics for satisfaction needs explanation.
In general, DnS does not constrain situations to include *only* individuals classified by the concepts of a description. In other words, reified satisfaction admits *redundant* situations.

[13] Unifying descriptions of a collection can be: a) those which define covering or characterizing roles; and b) those which use said roles (defined elsewhere), but whose unifying function is explicitly stated.

Semantically, this means that a situation *s* can be a setting for individuals that are not constants from the tuple τ reified by *s*.
But if we assume that descriptions and situations are reifications of ≥2-ary predicates resp. tuples in those predicates, redundant situations appear to be logically rough, since non-reified tuples include only individuals belonging to classes defined for the predicate.
On the other hand, real world uses of DnS are showing that most situations derive from other legacy situations that already have an internal structure, and modifying them with the sole purpose of getting non-redundant situations seems a bad practice. For example, a *detective report* (a description) can depict a situation that may contain useless information from the point of view of a certain *legal rule* (another description) but, to a certain extent, it is important to preserve the *unity* of the reported situation, instead of "cleaning" it up and making a new entity out of it, for the sake of merely complying to the legal-rule description.
It is remarkable that the same practice usually applies to physical objects: provided that they respect some basic properties, any other property results acceptable. For instance, having *dust* on the *rooftop* is not usually relevant in order to recognize the model of a *car*, but it is nonetheless a property of the car, and it can be perceived and conceptualized by some persons ("*uh, it's a* dusty *1968 Triumph TR4!*").[14]
Under this assumption, the same situation can satisfy different descriptions that can even be unrelated. The formal consistency is given by the fact that legacy situations already satisfy other descriptions (e.g. they result fom the reification of a database view).

(c) qualified satisfaction
Moreover, DnS admits a *qualified satisfaction*: the set of concepts that "must" classify a particular in a situation can be explicitly stated by means of a set of axioms that specialize the *satisfies* relation for a certain domain.

Summing up, reified satisfaction in DnS allows for situations that can be redundant on one hand (the respective non-reified models would be undecidable), and more restricted on the other hand (only certain non-reified models would be acceptable). But since reification allows a common domain for both ground and descriptive parts of an ontology, reified satisfaction does not lead to undecidability, and allows a custom design of the satisfiability conditions.

The semantics of embedded contexts sketched in 3.3.2 is extended here to deal with redundant and qualified models.
Roughly speaking, when dealing with e.g. an axiomatic theory T, a model M of T must [satisfy] (in the logical sense) the axioms of T, so that each individual is an instance of a class defined in T, and each tuple must be allowed by constraints encoded in the axioms of T. If a tuple in M generates a contradiction in T, M is not a model of T. If an individual in M is not an instance of any class in T, or a tuple in M involves individuals for whose classes no relation is defined in T, then M is undecidable in T.
When moving to reification by extending an ontology augmented with DnS, *satisfaction* (in the reified sense) has some more possibilities:[15]

- **purely reified satisfaction**: in this simple case, each individual *i* in a situation *s* must be classified by a concept *c* used in a description *d*, and for each DnS relationship $dr(c_1,c_2)$ with c_1 and c_2 both used in *d,* no relationship $gr(i_1,i_2)$ with i_1 classified by c_1 and i_2 classified by i_2, and both in *s*, can be incompatible with $dr(c_1,c_2)$. No further individuals and relationships are allowed in *s* (Fig.4,5)

14 Similar remarks have been made in linguistics and philosophy works on *event* reification, starting at least from [Davidson 1968].
15 Actually, the same possibilities exist for [satisfies] in the logical sense, when a first-order theory is enriched with meta-level expressivity.

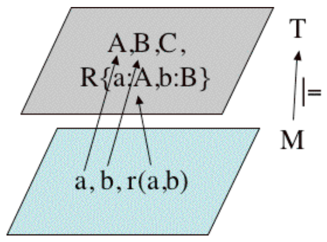

Fig. 4 Purely reified satisfaction is the reification of being a (valid) model M for a theory T: e.g. a is an instance of A, b is an instance of B, and the tuple a,b for the relation r conforms to the domain and range of R.

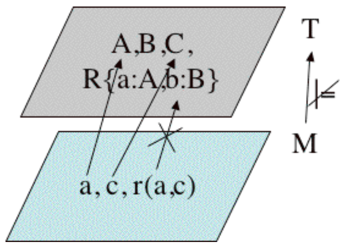

Fig. 5 Purely reified satisfaction failing is the reification of failing to be a (valid) model M for a theory T: e.g. a is an instance of A, c is an instance of C, but the tuple a,c for the relation r does not conform to the range of R (B).

- **redundant reified satisfaction**: in this case it is acceptable that there are individuals in *s* that are not classified by concepts in *d,* and/or relationships in *s* which do not correspond to DnS relationships between concepts in *d* (Fig.6)

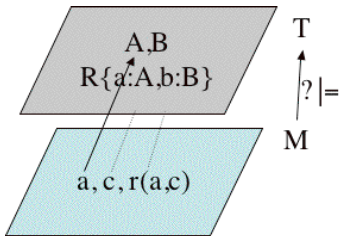

Fig. 6 Redundant reified satisfaction is the reification of being an undecidable model M for a theory T: e.g. a is an instance of A, but c and the tuple a,c for the relation r are undecidable within T, because the predicate C is not within its vocabulary.

- **structure matching satisfaction**: in this case, each concept *c* used in a description *d* classifies an individual *i* in a situation *s*, and for each DnS relationship *dr*(c_1,c_2) with c_1 and c_2 both used in *d,* there exists a relationship *gr*(i_1,i_2) with i_1 classified by c_1 and i_2 classified by i_2, and both are in *s*, which is compatible with *dr*(c_1,c_2). No further individuals and relationships are allowed in *s* (Fig.7)

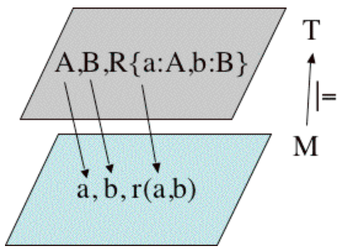

Fig. 7 Structure matching satisfaction is the reification of being a (valid) model M for a theory T: e.g. a is an instance of A, b is an instance of B, and the tuple a,b for the relation r conforms to the domain and range of R. But, differently from purely reified satisfaction, here *each* element in T has a correspondent in M.

- **qualified satisfaction**: in this case, there is an explicit set of axioms that specifies what concepts $c_{1...n}$ used in a description *d* must classify individuals $i_{1...n}$ in a situation *s*, and for which DnS relationships $dr_1(c_1,c_2),..., dr_n(c_n,c_m)$ with c_n and c_m both used in *d* there must be a relationship $gri(i_1,i_2)$ with i_1 classified by c_1 and i_2 classified by c_2, and both are in *s*, which is compatible with $dr_1(c_1,c_2),..., dr_n(c_n,c_m)$. Further individuals and relationships are allowed in *s*. In other words, some concepts and relationships in *d* are considered either **optional**, or **discarded** by decision (Fig.8)

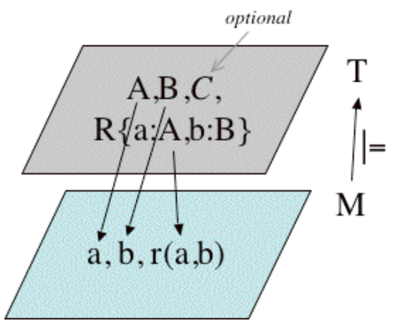

Fig. 8 Qualified satisfaction is similar to structure matching satisfaction, but some *specified* (e.g. optional tasks) elements in T have no necessary correspondence in M.

With reference to these cases of satisfaction, DnS relies in general on *redundant satisfaction* between situations and descriptions, and also on *qualified satisfaction* for specialised descriptions, such as plans or diagnoses.
Although this apparently allows the reification of undecidable models in a purely model-theoretical sense, it is not actually so, because the parts of the models that do not correspond to any concepts or axioms in the description are nonetheless decidable within the so-called *ground ontology*.
For example, if we have a model of the ground ontology that represents *some guy with a red jacket driving his car at an excessive speed*, and we add a *legal regulation* to the ontology, defining concepts for *speed limits*, *driving*, *vehicles*, and *drivers*, a situation can be constructed from that model whose individuals are classified by those concepts, except for *red jacket*, which is anyway decidable within the ground ontology. Therefore, no redundant entity or tuple in a situation can lead to undecidability, provided that those entities and tuples are decidable in the ground ontology.

3.3.5 Satisfaction types

The following (still preliminary) definitions introduce a basic typology for the satisfaction relation between situations and descriptions, leveraging on the semantical distinctions provided in 3.3.2 and 3.3.4.
The three types introduced are: **proactively satisfies** (*P-Satisfies*), meaning that the situation has a *pre hoc* description, **retrospectively satisfies** (*R-Satisfies*), meaning that the situation has a *post hoc* description, and **constructively satisfies** (*C-Satisfies*) meaning that the situation has an *ad hoc* description:

(A49) P-Satisfies(x,y) → Satisfies(x,y)
(A50) R-Satisfies(x,y) → Satisfies(x,y)
(A51) C-Satisfies(x,y) → Satisfies(x,y)

Provided that the space-time of a situation is the mereological sum of the spatial and temporal regions of the particulars in the setting of a situation, we can figure out the following axioms for the *-Satisfies relations:

P-Satisfies assumes two of the satisfaction semantics presented above: *redundant satisfaction* and *qualified satisfaction*. In order to allow for a correct implementation of the qualified satisfaction, P-Satisfies requires that the description exists prior to at least some of the particulars in the setting of the satisfying situation. Ontologically, it results that P-Satisfies also implies a specific dependency of the situation on its description.
P-Satisfies typically applies to plans, projects, designs, methods, techniques, rules of game, instructions, punishment rules, constitutive descriptions, sanctions, strategies, etc.:

(A52) $\text{P-Satisfies}(x,y) \rightarrow \exists e.\ \text{Particular}(e) \wedge \text{SettingFor}(x,e) \wedge \exists t1,t2.\ \text{PresentAt}(e,t1) \wedge \text{PresentAt}(y,t2) \wedge t2<t1$

(A53) $\text{P-Satisfies}(x,y) \rightarrow \text{SpecificallyDependsOn}(x,y)$

and, from the axioms for the situation space-time:

(T4) $\text{P-Satisfies}(x,y) \rightarrow \exists t1,t2.\ \text{PresentAt}(x,t1) \wedge \text{PresentAt}(y,t2) \wedge t2<t1$

R-Satisfies also assumes *redundant satisfaction* and *qualified satisfaction*, but further assumes that the particulars in the situation entirely exist prior to the description. This seems paradoxical, since a description hardly motivates what happens if it is not conceived by any agent involved in things happening. For this reason, we postulate a so-called **specific retrospective dependency** (SRD), meaning that the creator of the description is willing to attribute the status of a (scientific or anyway well-founded) law to that description, despite the latter was not present before the situation.
R-Satisfies typically applies to explanations that are considered as well-founded in science (physical, social, or cognitive), reverse engineering, criminal investigation, etc. Consider that the actual validity of the explanation is not addressed by the description, but by external evaluation descriptions:

(A54) $\text{R-Satisfies}(x,y) \rightarrow \forall e,t1.\ (\text{Particular}(e) \wedge \text{SettingFor}(x,e) \wedge \text{PresentAt}(y,t1)) \rightarrow \exists t2.\ \text{PresentAt}(e,t2) \wedge t2<t1$

(D21) $\text{SRD}(x,y) =_{df} \text{SpecificallyDependsOn}(x,y) \wedge \exists t1,t2.\ \text{PresentAt}(x,t1) \wedge \text{PresentAt}(y,t2) \wedge t1<t2$

(A55) $\text{R-Satisfies}(x,y) \rightarrow \text{SRD}(x,y)$

C-Satisfies - like R-Satisfies - concerns particulars that exist in a situation entirely prior to the description. Moreover, it assumes *redundant satisfaction*. But, differently from P-Satisfies and R-Satisfies, no qualified satisfaction is assumed. In fact, C-Satisfies implies no dependency of a situation on its description.
C-Satisfies typically applies to different views of existing situations, as for regulative descriptions (disclaimer: the situation can be already created by complying to the regulation, e.g. executing it as a plan, but in this case there actually exists a plan that has the regulation as part), narratives, symbolic interpretations, etc.:

(A56) $\text{C-Satisfies}(x,y) \rightarrow \forall e,t1.\ (\text{Particular}(e) \wedge \text{SettingFor}(x,e) \wedge \text{PresentAt}(y,t1)) \rightarrow \exists t2.\ \text{PresentAt}(e,t2) \wedge t2<t1$

(A57) $\text{C-Satisfies}(x,y) \rightarrow \neg\text{DependsOn}(x,y)$

While each *-Satisfies subrelation has applications on typical kinds of descriptions, in principle they can apply also to less typical description types, for example a regulation can be used strictly as a plan, thus a situation P-Satisfies it.

3.3.6 Execution

Satisfies subrelations have interesting relationships to the *execution* of ontologies that contain descriptions of behaviors, methods, actions, etc. An ontology is executed when it is able to change or produce entities or data structures (in an open domain of discourse).
Execution is related to the preliminary intuition of DnS as being the abstract specification of a system (*system_as_description*) and of its realizations (*system_as_situation*), while the real components and operations of a system ground those specifications into a substrate (physical, social, mental, computational).
For example, the grounding of a *plan* into the components of a system will take into account that those components should allow for the *enactment* of a *p-satisfying* situation at runtime: in a physical system this amounts to have e.g. correct actuators; in a computational system it amounts to implement production rules, etc.
A different example: the grounding for a *regulation* should allow for *checking* a *c-satisfying* situation: in a legal system, it amounts e.g. to compare social behaviors to required ones; in a computational system, it amounts to check the compliance between schemata and data structures.
Still another different example: the grounding for an *explanation* should allow for either *retrieving* or *simulating* (previewing) an *r-satisfying* situation: in a physical system, it amounts e.g. to create the conditions for something to happen, and to check the reproducibility of a behaviour; in a computational system, it amounts to retrieving or mining data structures, or to create simulations.
Another interesting application of *-Satisfies relations concerns the production of optimal descriptions according to available resources, a task addressed by either classical planning or problem-solving methods. In these cases, a *relatively unordered* legacy situation exists (it is presented by legacy systems or just by listing the elements), and an optimal plan should be produced that exploits the legacy situation according to a goal and some additional constraints and preferences. This could be described as a case of R-Satisfies from given elements of potential situations, which aims to discover a description equivalent to the best plan that can be *p-satisfied* by any new situation that includes those elements.
In the following table, the *satisfies* types are summarized, with their typical applications and intuititive uses in the physical, social, and computational worlds:

Description	**satisfaction**	**Situation**	***Physical***	***Social***	***Computational***
Plan	p-satisfied by	Execution	*Actuator*	*Intention*	*Production rule*
Regulation	c-satisfied by	Conformance	*Compliance*	*Compliance*	*Compliance*
Explanation	r-satisfied by	Recognition	*Phenomenon*	*Behaviour*	*Simulation*

3.4 The Plan Ontology

The plan ontology depends on the DnS ontology, and specializes it with tasks, goals, P-Satisfies rules, etc.
Our intention is providing an ontology for the specification of social or cognitive plans, while we are not trying to characterize *computationally executable* plans. Neither we attempt at

characterizing the *planning* task. Executable plans, as well as the planning task are e.g. characterized in the abovementioned "ontology of the planning task" [Rajpathak&Motta 2004].
In order to provide a minimal intuition of the relation between social plans/tasks, and computational plans/tasks, we provide here a simple example. Suppose a student has to deliver a report on the Canto I in the Inferno part of Dante's Comedy: he can decide a plan to produce the report, based on the reading of essays and critiques, direct reading of the Canto, consulting with teachers and friends, etc. The plan, once formalized with appropriate time sequences, roles, and parameters, will be a descriptive counterpart of the *intention* of the student to produce the report in a certain way.
If the actions carried out by the student based on that plan will be compliant to it, such actions will constitute a plan execution (that can success or fail, be suspended, postponed, aborted, etc.).
If we want to use that plan in an information system, it can be viewed as a set of *requirements* from the student. Now, a software engineer can take the requirements, and look for or implement software components that attempt at *operationalizing* the requirements.
The first analysis from the software engineer could be: what requirements can be effectively supported computationally? If this analysis leads to assessing the operationalization as feasible, and relevantly so, the problem is now to integrate the software components so that they make up a useful and robust system.
The second analysis could be: what resources can be found that can actually play the roles and tasks, according to parameters, of that plan? and are they computationally implementable?
Once components and resources are integrated/collected, a more specific, *circumstantiated* plan can be produced by a planner.
Back to the example, if an appropriate electronic agent acts for the student, if all documents are reachable by that agent, and if each action to produce a report can be executed by an electronic agent, such agents will be able to completely operationalize the student's requirements.

Bottom-line: software components, resources, and planners live in the world of *computational plans*, while social/cognitive requirements live in the *cognitive agents'* world. In the ideal case, for each part of a requirement, a specific component implements the required functionality, finds all the resources needed, and a planner is able to compute a circumstantiated plan. The ideal case is not realistic, unless the social/cognitive plan is equivalent to an abstract algorithm to be applied on purely computational entities (with no intended reference to the social/cognitive world). For non-purely computational plans/entities, requirements will be overdetermined with respect to the abstract computational plan, and underdetermined with respect to the circumstantiated computational plan, but such cognitive plans will be anyway useful to characterize a plan independently from its (current) operationalization.
This is also a disclaimer: our ontology of plans is not intended to substitute in any way the existing planning frameworks; on the contrary, it is complementary to them, because it aims at explicitating the dependencies between a set of domain-related assumptions, and the computational operationalization/optimization of a requirement-driven system.

3.4.1 Plans and goals

A **plan** is a description that is conceived by a cognitive agent, defines or uses at least one *task* (a kind of course of *actions*) and one role (played by agents), and has at least one goal as a proper part:

(A58) $\mathrm{Plan}(x) \rightarrow \mathrm{Description}(x)$
(A59) $\mathrm{Plan}(x) \rightarrow \exists y,t.\ \mathrm{Conceives}(y,x,t) \wedge \mathrm{CognitiveAgent}(y)$
(A60) $\mathrm{Plan}(x) \rightarrow \exists y.\ \mathrm{Task}(y) \wedge \mathrm{Uses}(x,y)$
(A61) $\mathrm{Plan}(x) \rightarrow \exists c.\ (\mathrm{Role}(c) \wedge \forall a,t.\ \mathrm{Classifies}(c,a,t) \rightarrow \mathrm{Agent}(a)) \wedge \mathrm{Uses}(x,c)$
(A62) $\mathrm{Plan}(x) \rightarrow \exists g.\ \mathrm{Goal}(g) \wedge \mathrm{ProperPart}(x,g)$

Examples of plans include: *the way to prepare an espresso in the next five minutes*, a *company's business plan*, a *military air campaign*, a *car maintenance routine*, a *plan to start a relationship*, etc.
A plan can have several proper parts (regulations, goals, laws), including other plans:

(D22) $\mathrm{Subplan}(x) =_{df} \mathrm{Plan}(x) \wedge \exists y.\ \mathrm{Plan}(y) \wedge \mathrm{ProperPart}(y,x)$

If a plan uses a figure without defining it, and that figure is defined by a constitutive description, the plan specifically depends on the constitutive description:[16]

(A63) $\mathrm{ConstitutiveDescription}(x) \rightarrow \mathrm{Description}(x) \wedge \exists y.\ \mathrm{Defines}(x,y) \wedge \mathrm{Figure}(y)$
(A64) $[\mathrm{Plan}(x) \wedge \mathrm{Figure}(f) \wedge \mathrm{Uses}(x,f) \wedge \exists y.\ \mathrm{ConstitutiveDescription}(y) \wedge \mathrm{Defines}(y,f)] \rightarrow \exists t.\ \mathrm{GenericallyDependsOn}(x,y,t)$

Of course, plans can directly define figures, e.g. some plans define *temporary* figures, such as *teams* or *task forces*, whose lifecycle starts and ends within the plan lifecycle.
The notion of **Goal** is more complicated, due to the widespread polysemy it suffers from. Here a goal is considered to be a desire (another kind of description) that is a part of a plan. **Desires** in general are characterized as defining or using at least one role classifying an agent, and at least one course. The role is played by the agent in a *desire mode* towards the course:

(A65) $\mathrm{Desire}(x) \rightarrow \mathrm{Description}(x)$
(A66) $\mathrm{DesireTowards}\ (x,y) \rightarrow \mathrm{AttitudeTowards}(x,y)$
(A67) $\mathrm{Desire}(x) \rightarrow \exists y,t.\ \mathrm{Conceives}(y,x,t) \wedge \mathrm{CognitiveAgent}(y)$
(A68) $\mathrm{Desire}(x) \rightarrow \exists y,z.\ (\mathrm{Role}(y) \wedge \forall a,t.\ \mathrm{Classifies}(y,a,t) \rightarrow \mathrm{Agent}(a)) \wedge \mathrm{Course}(z) \wedge \mathrm{Uses}(x,y) \wedge \mathrm{Uses}(x,z) \wedge \mathrm{DesireTowards}(y,z,t)$

For example, a *desire to start a relationship* can become a *goal to start a relationship* if someone *takes action* – or lets someone else take action on her behalf – with the purpose of starting the relationship.
We are proposing here a restrictive notion of **goal** that relies upon its desirability by some agent, which not necessarily plays a role in the execution of the plan the goal is part of. For example, an agent can have an attitude towards some course defined in a plan, e.g. *duty towards*, which is different from desiring it (*desire towards*). We might say that a goal is usually desired by the creator or beneficiary of a plan. The minimal constraint for a goal is that it is a proper part of a plan:

(D23) $\mathrm{Goal}(x) =_{df} \mathrm{Desire}(x) \wedge \exists p.\ \mathrm{Plan}(p) \wedge \mathrm{ProperPart}(p,x)$

A **subgoal** (relative to a plan) is a goal that is a proper part of a subplan:

(D24) $\mathrm{Subgoal}(x,y) =_{df} \mathrm{Part}(x,y) \wedge \mathrm{Goal}(y) \wedge \mathrm{Plan}(x) \wedge \exists z.\ \mathrm{Plan}(z) \wedge \mathrm{ProperPart}(z,x)$

[16] This is a special case of the dependence of a description d on another description d' that *defines* concepts or figures *used* in d. It is still under discussion is such dependence is specific or generic: for example, if a theory changes its identity, do its dependent theories change their identity as well? if no, dependence must be generic.

A **main goal** can be defined as a goal that is part of a plan but not of one of its subplans (i.e. it is a goal, but not a subgoal in that plan):

(D25) MainGoal(p1,x) $=_{df}$ ProperPart(p1,x) ∧ Plan(p1) ∧ Goal(x) ∧ ¬∃p2. Plan(p2) ∧ ProperPart(p1,p2) ∧ ProperPart(p2,x)

A goal is not necessarily a part of the main goal of the plan it is a subgoal of. E.g. consider the goal: *being satiated*; *eating food* can be a subgoal of the plan having *being satiated* as its main goal (see below), but it is not a part of *being satiated.*
Nonetheless, we can also conceive of an **influence** relation between a goal and the main goal of the plan the first goal is a subgoal of:

(D26) InfluenceOn(x,y) $=_{df}$ Goal(x) ∧ Goal(y) ∧ ∃z. Plan(z) ∧ Subgoal(z,x) ∧ MainGoal(z,y)

It is also possible to represent the maximal goal of a plan, i.e. the sum of all goals that are parts of a plan:

(D27) MaximalGoal(x,y) $=_{df}$ Plan(x) ∧ Goal(y) ∧ ∀z. [Goal(z) ∧ ProperPartOf(z,x)] → PartOf(z,y)

By using the previous definitions, we can also define a **disposition** relation between the roles used in a plan having a main goal, and the influenced goal:

(D28) DispositionTo(x,y) $=_{df}$ (Role(x) ∧ ∀a,t. Classifies(x,a,t) → Agent(a)) ∧ Goal(y) ∧ ∃p,g. Plan(p) ∧ Goal(g) ∧ ProperPart(p,g) ∧ Uses(p,x) ∧ InfluenceOn(g,y)

For example, the role *eater* can have a disposition to *being satiated*, meaning that a person playing the role of *eater* that adopts that plan can act in order to be satiated.
Disposition relation is useful to account for those cases in which a task addressed by a role is not *internal* to the plan, but the plan is a subplan of another one in which that course is represented as a full-fledged goal.

In interesting cases, supergoals can be created in order to support the adoption of a subgoal. In order to describe these cases, we need to specialize the adoption relation. Goals and plans can be in fact adopted with different constraints:

(D29) AdoptsGoal(x,y,t) $=_{df}$ Adopts(x,y,t) ∧ CognitiveAgent(x) ∧ Goal(y) ∧ ∀z. (Course(z) ∧ Uses(y,z)) → DesireTowards(x,z,t)

(D30) AdoptsPlan(x,y,t) $=_{df}$ Adopts(x,y,t) ∧ CognitiveAgent(x) ∧ Plan(y)

In those interesting cases, given a plan and its *main* goal, e.g. some service to be delivered, it is a common practice to envisage the *super*goals of the main goal that can be more clearly desirable from e.g. prospective users of a service (for example, a claim like the following generates a supergoal for the service's goal: *our service will improve your life*). In these cases, goal adoption and plan adoption are taken *as if* the following theorem would be undebatably sustainable, i.e. that goal adoption implies adopting all its subgoals:

(T5) ? (AdoptsGoal(x,y,t) ∧ Subgoal(y,z)) → AdoptsGoal(x,z,t)

Amother disclaimer should be made on other apparently sensible axioms, e.g. that plan adoption implies the adoption of its main goal or of its subgoals. In particular the first one seems plausible in most cases, but in general social conditions, it is debatable:

(A69) *? $AdoptsPlan(x,y) \rightarrow \exists z. MainGoal(y,z) \wedge AdoptsGoal(x,z)$
(A70) *? $AdoptsPlan(x,y) \rightarrow \forall z. SubGoal(y,z) \rightarrow AdoptsGoal(x,z)$

Alternatively, goals can be directly represented by means of a relation: Goal(x,y) ranging on plans and desires. This solution would be formally closer to the classical BDI paradigm [19], by which, given a set of *beliefs* about a world (preconditions), and a *desire* towards another world (the goal state), an *intention* connects possible means to the desired world through a path (the *plan*), developed by following the so-called *means-end reasoning*. In other words, in BDI (but also in some PSMs) plans are *tailored* to the available entities in the world, according to some explicit constraints, preferences, optimal conditions, cost functions, etc.
In DDPO there are similarities as well, because tailoring a plan in DDPO is equivalent to classifying entities from the ground ontology before starting an actual plan execution.
On the other hand, DDPO implements the BDI paradigm differently, because DDPO reifies logical constraints, classes, and relations, and is therefore able to represent various level of abstractions.
A DDPO plan can consist only of the constraints, preferences, cost functions, and restrictions over the *classes* of entities that can be classified in the plan execution: this is what we call an *abstract plan* (see below).
While a *tailored plan* is equivalent in DDPO to a so-called *circumstantial plan*, namely a plan that specifies each entity that can be classified in the plan execution (together with the relations among those entities).
Finally, a plan that specifies its spatio-temporal execution is a *saturated plan*. This is close to what classical planning calls *schedule*.

These distinctions can be formalised as a typology of plans built according to their *situatedness*, i.e. according to how many variables are left open in the class of situations that can satisfy the plan.
For example, an **abstract plan** is a plan whose roles and tasks only specify *classes* of entities that can be included in a plan execution. In other words, a component from an abstract plan does not classify any *named entity*. This condition cannot be formalized in FOL, since the following axiom:

(A71) $AbstractPlan(x) \rightarrow Plan(x) \wedge \forall yz. ((Role(y) \wedge Uses(x,y)) \rightarrow (Endurant(z) \wedge Classifies(y,z))) \wedge$
$\forall wk. ((Course(w) \wedge Uses(x,w)) \rightarrow (Perdurant(k) \wedge Classifies(w,k)))$

only states general restrictions over plan components and situation elements. We need to express a condition by which an instance of an abstract plan specifies instances of plan components, but no instances of situation elements, e.g. that *manager* classifies *some (if any) instance of person*.

A **circumstantial plan** has all components classifying named individuals from the ground ontology (e.g. only specific persons, specified resources, a finite number of time intervals and space regions, etc.):

(D31) $CircumstantialPlan(x) =_{df} Plan(x) \wedge \forall y. (Concept(y) \wedge Uses(x,y)) \rightarrow \exists z. Particular(z) \wedge$
$Classifies(y,z)$*

*provided that *z* is a *named entity*, and not a skolemized individual (this is relevant only for the languages allowing skolemization btw).

A **saturated plan** is a plan that cannot be executed twice, since it defines spatio-temporal parameters restricted to one value, e.g. one of its courses classifies an event that is valued by a definite temporal value in a definite space region:

(D32) $\mathrm{SaturatedPlan}(x) =_{df} \mathrm{Plan}(x) \wedge \exists y,z.\ \mathrm{Parameter}(y) \wedge \mathrm{Parameter}(z) \wedge \mathrm{Uses}(x,y) \wedge \mathrm{Uses}(x,z) \wedge \exists t,s.\ \mathrm{ValuedBy}(y,t) \wedge \mathrm{TimeInterval}(t) \wedge \neg\exists t1.\ \mathrm{TimeInterval}(t1) \wedge \mathrm{ValuedBy}(y,t1) \wedge t\neq t1 \wedge \mathrm{ValuedBy}(z,s) \wedge \mathrm{SpaceRegion}(s) \wedge \neg\exists s1.\ \mathrm{SpaceRegion}(s1) \wedge \mathrm{ValuedBy}(y,s1) \wedge s\neq s1$

Of course, in the case of maximal spatio-temporal regions, a saturated plan tends to approximate an abstract plan from the execution viewpoint, but these worst cases are unavoidable when dealing with maximality.[17]

Plan executions are situations that proactively satisfy a plan, meaning that the plan anticipates its execution (cf. definition of P-Satisfies above):

(D33) $\mathrm{PlanExecution}(x) =_{df} \mathrm{Situation}(x) \wedge \exists y.\ \mathrm{Plan}(y) \wedge \mathrm{Satifies}(x,y) \wedge \exists t.\ \mathrm{PresentAt}(y,t) \wedge \neg\mathrm{PresentAt}(x,t)$

Subplan executions are parts of the whole plan execution:

(A72) $\forall p1,p2,s1,s2.\ (\mathrm{Plan}(p1) \wedge \mathrm{Plan}(p2) \wedge \mathrm{ProperPart}(p1,p2) \wedge \mathrm{Satifies}(s1,p1) \wedge \mathrm{Satifies}(s2,p2)) \rightarrow \mathrm{ProperPart}(s1,s2)$

A **goal situation** is a situation that satisfies a goal:

(D34) $\mathrm{GoalSituation}(x) =_{df} \mathrm{Situation}(x) \wedge \exists y.\ \mathrm{Goal}(y) \wedge \mathrm{Satifies}(x,y)$

Contrary to the case of subplan executions, a goal situation is not part of a plan execution:

(A73) $\mathrm{GoalSituation}(x) \rightarrow \forall y,p,s.\ (\mathrm{Goal}(y) \wedge \mathrm{Satifies}(x,y) \wedge \mathrm{Plan}(p) \wedge \mathrm{ProperPart}(p,y) \wedge \mathrm{Satifies}(s,p)) \rightarrow \neg\mathrm{ProperPart}(s,x)$

In other words, it is not true in general that any situation satisfying a part of a description is also part of the situation that satisfies the whole description:

(T6) $\forall d1,d2,s1\neg\forall s2.\ (\mathrm{Description}(d1) \wedge \mathrm{Description}(d2) \wedge \mathrm{ProperPart}(d1,d2) \wedge \mathrm{Satifies}(s1,d1) \wedge \mathrm{Satifies}(s2,d2)) \rightarrow \mathrm{ProperPart}(s1,s2)$

This helps to account for the following cases:

- Execution of plans containing *abort* or *suspension* conditions (the plan would be satisfied even if the goal has not been reached, see below)
- *Incidental* satisfaction, as when a situation satisfies a goal without being intentionally planned (but anyway desired).

A **precondition** for a plan can be defined as a relation between a situation and a plan, implying that, for all plan executions of that plan to occur, a situation should preliminarily satisfy some description as well:

[17] Suppose someone makes a plan of her life by stating a generic maxim, like in traditional wisemen's suggestions.

(A74) Precondition(p,s) → Plan(p) ∧ Situation(s)
(A75) Precondition(p,s) → ∀s1. (PlanExecution(s1) ∧ P-Satisfies(s1,p)) → (∃d. Satifies(s,d) ∧ Precedes(s,s1))

Notice that we do not exclude that *s1* and *s* could have the same minimally common type (i.e. that they satisfy the same plan, i.e. that in practice we are characterising a cyclical plan).

A **postcondition** for a plan can be defined as a relation between a situation and a plan, implying that, after plan executions of that plan occur, a situation should satisfy some description as well:

(A76) Postcondition(p,s) → Plan(x) ∧ Situation(s)
(A77) Postcondition(p,s) → ∀s1. (PlanExecution(s1) ∧ P-Satisfies(s1,p)) → (∃d. Satifies(s,d) ∧ Precedes(s1,s))

It often holds that the main goal situation is a postcondition of plans, but this is not mandatory.

An **accompanying condition** (sometimes called ‘constraint’ in the planning literature) for a plan can be defined as a relation between a situation and a *task* (see below), implying that, for all plan executions of that plan to occur, a situation should satisfy some description as well, at the time of some specified perdurant that is sequenced by a task defined in the plan:

(A78) AccompanyingCondition(p,s,t) → Plan(p) ∧ Situation(s) ∧ Task(t)
(A79) AccompanyingCondition(p,s,t) → Defines(p,t)
(A80) AccompanyingCondition(p,s,t) → ∀s1. (PlanExecution(s1) ∧ P-Satisfies(s1,p) ∧ s≠s1) → (∃d,e,i. Satifies(s,d) ∧ Perdurant(e) ∧ TimeInterval(i) ∧ Sequences(t,e,i) ∧ Setting(e,s1) ∧ Precedes(s,e))

3.4.2 Tasks

Tasks are courses that are (mostly) used to sequence activities, or other perdurants that can be under the control of a planner. They are defined by a plan, but can be used by other kinds of descriptions.
Tasks can be considered as *shortcuts* for plans, since at least one role played by agents has a “desire attitude” towards them (possibly different from the one that puts the task into action):

(A81) DesireTowards(x,y,t) → AttitudeTowards(x,y,t) ∧ ∃e,d,t. Agent(e) ∧ Classifies(x,e,t) ∧ Uses(d,x) ∧ Uses(d,y) ∧ Conceives(e,d,t)
(D35) Task(x) $=_{df}$ Course(x) ∧ ∃y,z. Plan(y) ∧ Defines(y,x) ∧ (Role(z) ∧ ∀a,t. Classifies(z,a,t) → Agent(a)) ∧ Uses(y,z) ∧ DesireTowards(z,x,t)

Tasks can be complex, and ordered according to an abstract succession relation. Tasks can relate to concrete actions or decision making; the latter deals with typical flowchart content. A task is different both from a flowchart node, and from an action or a class of actions.

A **scheduled task** is a task that cannot be executed twice, since it has a temporal parameter restricted to one value, e.g. it classifies an event that is valued by a definite temporal value:

(D36) ScheduledTask(x) $=_{df}$ Task(x) ∧ ∃y,i. Parameter(y) ∧ TimeInterval(i) ∧ RequisiteFor(y,x,i) ∧ ∃t. ValuedBy(y,t) ∧ TimeInterval(t) ∧ ¬∃t1. TimeInterval(t1) ∧ ValuedBy(y,t1) ∧ t≠t1

For example, "*pick me up at 3pm today*" is a scheduled task.

A **complex task** is a task that has at least two other tasks as components:

(D37) $\text{ComplexTask}(x) =_{df} \text{Task}(x) \wedge \exists y,z.\ \text{Task}(y) \wedge \text{Task}(z) \wedge y \neq z \wedge \text{Component}(x,y) \wedge \text{Component}(x,z)$

The primary ordering relation for tasks is **direct successor**; its transitive version is called **successor**. Notice that *successor* relations are abstract, and do not include a temporal ordering, although the usual correspondence within sequenced perdurants is a *temporal* relation (*precedes* or *overlaps*), and sometimes a *causal* relation. The distinction is clear when we consider two tasks having a direct successor relation holding for them, while the actions sequenced by them could temporally overlap:

(A82) $\text{DirectSuccessor}(x,y) \rightarrow \text{Particular}(x) \wedge \text{Particular}(y)$

(A83) $\text{Successor}(x,y) \rightarrow \text{Particular}(x) \wedge \text{Particular}(y) \wedge \forall z,w,k.\ (\text{DirectSuccessor}(z,w) \wedge \text{DirectSuccessor}(w,k)) \rightarrow \text{Successor}(z,k)$

DirectSuccessor is irreflexive, antisymmetric, and intransitive. Successor is irreflexive, antisymmetric, and transitive.

Before analyzing types of complex tasks, we need to introduce the notion of *control task*, *action task*, etc. On the other hand, we make here forward references for the sake of homogeneity.

A **sequential task** is a complex task that includes a successor relation among any two component tasks, and does not contain any control task (cf. (D44)):

(D38) $\text{SequentialTask}(x) =_{df} \text{ComplexTask}(x) \wedge (\forall y,z.\ (\text{Component}(x,y) \wedge \text{Component}(x,z) \wedge y \neq z) \rightarrow (\text{Successor}(y,z) \vee \text{Successor}(z,y))) \wedge (\neg\exists w.\ \text{Component}(x,w) \wedge \text{ControlTask}(w))$

For example, "*eat your watermelon slice, then go playing games*" is a sequential task.

A **hybrid task** is a complex task that has at least one control task and one action task (cf. (D43)) as components:

(D39) $\text{HybridTask}(x) =_{df} \text{ComplexTask}(x) \wedge \exists y,z.\ \text{Component}(x,y) \wedge \text{Component}(x,z) \wedge y \neq z \wedge \text{ControlTask}(y) \wedge \text{ActionTask}(z)$

For example, "*eat your watermelon slice, then - if you find a friend on the beach - go playing games*" is a hybrid task.

A **bag task** is a complex task that does not include neither a control task, nor a successor relation among any two component tasks:

(D40) $\text{BagTask}(x) =_{df} \text{ComplexTask}(x) \wedge (\neg\exists y,z.\ (\text{Component}(x,y) \wedge \text{Component}(x,z) \wedge y \neq z) \rightarrow \text{Successor}(y,z)) \wedge (\neg\exists w.\ \text{Component}(x,w) \wedge \text{ControlTask}(w))$

For example, "*eat your watermelon slice, your cheese, look out that bee, take care to keep your shirt clean, stop jumping*" is a bag task (said by an anxious parent …), since no particular ordering can be guessed, probably neither a concurrency.

A **maximal task** is a complex task that has all the tasks defined in a plan as components:

(D41) $\text{MaximalTask}(x) =_{df} \text{ComplexTask}(x) \wedge \forall y. (\text{Task}(y) \wedge \text{Component}(p,y)) \rightarrow \text{Part}(x,y)$

An **elementary task** is a an atomic task:

(D42) $\text{ElementaryTask}(x) =_{df} \neg\exists y. \text{Component}(x,y) \wedge \text{Task}(y)$

An **action task** is an elementary task that sequences non-planning activities, like: moving, exercising forces, gathering information, etc. Planning activites are mental events involving some *rational* event:

(D43) $\text{ActionTask}(x) =_{df} \neg\exists y,i. \text{Sequences}(x,y,i) \wedge \text{TimeInterval}(i) \wedge \text{PlanningActivity}(y)$

For example, "*eat your watermelon slice*" is an action task.

A **control task** is an elementary task that sequences a planning activity, e.g. an activity aimed at (cognitively or via simulation) anticipating other activities. Therefore, control tasks have usually at least one direct successor task (the *controlled* one), with the exception of *ending tasks* (see below):

(D44) $\text{ControlTask}(x) =_{df} \text{Task}(x) \wedge (\forall y,i. \text{Sequences}(x,y,i) \rightarrow \text{PlanningActivity}(y) \wedge \text{TimeInterval}(i)) \wedge \exists z. \text{Task}(z) \wedge \text{DirectSuccessor}(x,z)$

The reification of control constructs allows to represent procedural knowledge into the same ontology including controlled action. Besides cognitive transparency and independency from a particular grounding system, a further advantage offered by reification is to enable the representation of *coordination* tasks and their relation to roles defined in the same plan. For example, a *manager* that coordinates the execution of several related activities can be represented as a role with a *responsibility* (defined as a combination of duties and rights) towards a control task that has some complex task as a direct successor (or that is a component of a complex task).

Individual control tasks are differentiated by their interrelations and application to actions they sequence.
Loop task is a control task that has as successor an action (or complex) task that sequences at least two distinct activities sharing a minimal common set of properties (i.e., the activities have a **minimal common type**):

(A84) $\text{ControlTask}(\text{LoopTask}) \rightarrow \exists y,z,w,i,j,\mathcal{P}. \text{Task}(y) \wedge \text{Action}(z) \wedge \text{Action}(w) \wedge z \neq w \wedge \text{TimeInterval}(i) \wedge \text{DirectSuccessor}(\text{LoopTask},y) \wedge \text{Sequences}(y,z,i) \wedge \text{Sequences}(y,w,j) \wedge \text{Precedes}(z,w) \wedge \text{MinimalCommonType}(z,w,\mathcal{P})$

For example, "*repeat the poem on and on*" is a complex task controlled by the loop task. Notice that *MinimalCommonType* cannot be formalized as a first-order predicate, hence the more so in OWL-DL. It can, however, be considered as a basic guideline: «when sequencing looped actions, choose a definite action class from the ground ontology».
Some relations typically hold when the loop task is used. **Exit condition** can be used to state what *decision state* (see below) causes to exit the cycle; **iteration interval** can be used to state how much time should be taken by each iteration of the looped activity; **iteration cardinality** can be used to state how many times the action should be repeated:

(A85) ExitCondition(LoopTask,y) → DecisionState(y)
(A86) IterationInterval(LoopTask,y) → TimeInterval(y)
(A87) IterationInterval(LoopTask,y) → ∀z. (Action(z) ∧ Successor(LoopTask,z)) → TemporalLocation(z,y)
(A88) IterationCardinality(LoopTask,y) → Integer(y)

Cyclical task is a complex task that is composed by the loop task and the case task (cf. (A91)). The case task specifies the exit condition(s) of the cyclical task indirectly (only the decisions that do not have the cyclical task as successor are exit conditions), while loop task specifies which is the exit condition:

(A89) ComplexTask(CyclicalTask) → DirectSuccessor(LoopTask,CyclicalTask) ∧ DirectSuccessor(CaseTask,LoopTask) ∧ Component(CyclicalTask,CaseTask) ∧ DirectSuccessor(CaseTask,DeliberationTask) ∧ Component(CyclicalTask, DeliberationTask) ∧ ∃w. DecisionState(w) ∧ Sequences(DeliberationTask,w) ∧ ExitCondition(LoopTask,w)

For example, "*repeat the poem until you remember it smoothlessly*" is a cyclical task.

Branching task is a control task that articulates a complex task into an ordered set of tasks:

(A90) ControlTask(BranchingTask) → ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(BranchingTask,y) ∧ DirectSuccessor(BranchingTask,z)

Case task is a specialization of a branching task branched to a set of tasks that are not executable concurrently. In order to choose the task to be executed, the preliminary deliberation task should be executed. The case task sequences a decision activity (a kind of mental event involving rationality) that has a decision state as outcome (sequenced by the **deliberation task**):

(A91) ControlTask(CaseTask) → Specializes(CaseTask,BranchingTask) ∧ (∀y,i. Sequences(CaseTask,y,i) → DecisionActivity(y) ∧ TimeInterval(i)) ∧ DirectSuccessor(CaseTask,DeliberationTask)
(A92) ControlTask(CaseTask) → ∃y,z,w,k. DecisionState(y) ∧ DecisionState(z) ∧ y≠z ∧ DirectSuccessor(CaseTask,DeliberationTask) ∧ Sequences(DeliberationTask,y) ∧ Sequences(DeliberationTask,z) ∧ ActionTask(w) ∧ ActionTask(k) ∧ w≠k ∧ DirectSuccessor(DeliberationTask,w) ∧ DirectSuccessor(DeliberationTask,k) ∧ ∀e1,e2,i,j. Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ TimeInterval(i) ∧ TimeInterval(j) ∧ Sequences(w,e1,i) ∧ Sequences(k,e2,j) ∧ ¬Overlaps(e1,e2)
(A93) ControlTask(DeliberationTask) → (∀y,i. (Sequences(DeliberationTask,y,i) → DecisionState(y) ∧ TimeInterval(i)) ∧ DirectSuccessor(CaseTask,DeliberationTask)

For example, "*if you find a friend on the beach, go playing games*" are action tasks controlled by a case task and two deliberation tasks.

Alternate task specializes the case task to exactly two executions of the deliberation task:

(A94) ControlTask(AlternateTask) → Specializes(AlternateTask,CaseTask) ∧ ∃y,z. DecisionState(y) ∧ DecisionState(z) ∧ y≠z ∧ DirectSuccessor(CaseTask,DeliberationTask) ∧ Sequences(DeliberationTask,y) ∧ Sequences(DeliberationTask,z) ¬∃w. w≠y ∧ w≠z ∧ DecisionState(w) ∧ Sequences(AlternateTask,w)

Concurrency task is a a specialization of a branching task branched to a set of tasks executable concurrently (the sequenced perdurants can overlap), which means that no deliberation task is executed in order to choose among them. Concurrency task has successor the **synchronization (synchro) task**, which is aimed at waiting for the execution of all (except the optional ones) tasks direct successor to the concurrent (or *any order*, see below) one:

(A95) ControlTask(ConcurrencyTask) → Specializes(ConcurrencyTask,BranchingTask) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(ConcurrencyTask,y) ∧ DirectSuccessor(ConcurrencyTask,z) ∧ ∀e1,e2,i,j. Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ TimeInterval(i) ∧ TimeInterval(j) ∧ Sequences(y,e1,i) ∧ Sequences(z,e2,j)) → Overlaps(e1,e2)

(A96) ControlTask(ConcurrencyTask) → Successor(ConcurrencyTask,SynchroTask)

For example, "*eat your watermelon slice, your cheese, but look out that bee*" are action tasks controlled by a concurrency task.

(A97) ControlTask(SynchroTask) → ∃t1,t2,t3. (t1=ConcurrencyTask ∨ t1=AnyOrderTask) ∧ Successor(t1,x) ∧ (ComplexTask(t2) ∨ ActionTask(t2)) ∧ (ComplexTask(t3) ∨ ActionTask(t3)) ∧ DirectSuccessor(t2,SynchroTask) ∧ DirectSuccessor(t3,SynchroTask)

For example, "*after you've eaten your watermelon slice and also talked to me frankly, we can go home*" are action tasks controlled by a concurrency task and a synchronization one.

Parallel task is a specialization of a concurrent task branching to at least two tasks that sequence temporally coinciding perdurants:

(A98) ControlTask(ParallelTask) → Specializes(ParallelTask,ConcurrencyTask) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(ParallelTask,y) ∧ DirectSuccessor(ParallelTask,z) ∧ ∀e1,e2,i,j. Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ TimeInterval(i) ∧ TimeInterval(j) ∧ Sequences(y,e1,i) ∧ Sequences(z,e2,j)) → Coincides(e1,e2)

Any-order task is a specialization of the branching task that defines no order in the successor tasks. This is another way of introducing a *bag task*,[18] because any temporal relation can be expected between any two perdurants sequenced by the tasks that are direct successors to the any-order task:

(A99) ControlTask(AnyOrderTask) → Specializes(AnyOrderTask,BranchingTask) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(AnyOrderTask,y) ∧ DirectSuccessor(AnyOrderTask,z) ∧ ∀e1,e2,i,j. Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ TimeInterval(i) ∧ TimeInterval(j) ∧ Sequences(y,e1,i) ∧ Sequences(z,e2,j)) → TemporalRelation(e1,e2)

(A100) ControlTask(AnyOrderTask) → Successor(AnyOrderTask,SynchroTask)

Beginning task is a control task that is the predecessor of all tasks defined in the plan:

(A101) ControlTask(BeginningTask) → ∀y,p. (Task(y) ∧ Plan(p) ∧ Component(p,BeginningTask) ∧ Component(p,y) ∧ BeginningTask≠y) → Successor(BeginningTask,y)

Ending task is a control task that has no successor tasks defined in the plan:

18 The difference is that a bag task is composed of any kind of tasks, while any-order task is a compound control task.

(A102) $\text{ControlTask(EndingTask)} \rightarrow \forall p \neg \exists y. (\text{Task}(y) \wedge \text{Plan}(p) \wedge \text{Component(p,EndingTask)} \wedge \text{Component}(p,y) \wedge \text{EndingTask} \neq y) \rightarrow \text{Successor(EndingTask},y)$

3.4.3 Satisfaction in DDPO

The *satisfies* relation (as well as its subrelations) is usually constrained for special classes of descriptions ("qualified" satisfaction), in order to have necessary and sufficient conditions to infer it between a certain situation and a description. For plans, a preliminary axiomatization for P-Satisfies is provided here:

(A103) (P-Satisfies(x,y) ∧ Plan(y)) ↔
{
[∀p. (Parameter(p) ∧ ∃t,i. Task(p) ∧ TimeInterval(i) ∧ RequisiteFor(p,t,i)) →
∃r. ValuedBy(p,r) ∧ Region(r) ∧ SettingFor(x,r)] ∧
[∃c,o. (Role(c) ∧ PlayedBy(c,o)) ∧ (Figure(c) ∧ ActedBy(c,o)) ∧ Endurant(o) ∧
SettingFor(x,o)] ∧
[∀t. ControlTask(t) →
∃a,i. Sequences(t,a,i) ∧ Perdurant(a) ∧ TimeInterval(i) ∧ Setting(y,a)] ∧
[∀t. (ActionTask(t) ∧ ¬∃z. ControlTask(z) ∧ DirectSuccessor(z,t)) →
∃a,i. Sequences(t,a,i) ∧ TimeInterval(i) ∧ Perdurant(a) ∧ SettingFor(x,a)] ∧
[∀t. (ActionTask(t) ∧ DirectSuccessor(t,SynchroTask)) →
∃a,i. Sequences(t,a,i) ∧ TimeInterval(i) ∧ Perdurant(a) ∧ SettingFor(x,a) ∧
¬OptionallyUsedBy(t,y) ∧ ¬DiscardedWithin(t,y)] ∧
[∃t,a. ActionTask(t) ∧ DirectSuccessor(BranchingTask,t) ∧ ∃a,i. Sequences(t,a,i) ∧
TimeInterval(i) ∧ Perdurant(a) ∧ SettingFor(x,a) ∧ ¬OptionallyUsedBy(t,y) ∧
¬DiscardedWithin(t,y)] ∧
[∃a,i. Sequences(EndingTask,a,i) ∧ Perdurant(a) ∧ TimeInterval(i) ∧ SettingFor(x,a)]
}

Intuitively, we are suggesting that for a plan to be satisfied, we require that the following concepts classify some particular in the situation setting: *i)* all parameter for tasks, *ii)* at least one role, *iii)* all control tasks, *iv)* all action tasks that are not bound by a control task, *v)* all action tasks bound by the synchronization task, except optional or discarded tasks, *vi)* at least one action task from any set bound by the branching task, *vii)* the ending task,

Notice that we are not including the satisfaction of the plan's goal among the P-Satisfies constraints for the plan (see above for the asymmetry between goal description and situation). This means that a plan can be satisfied even when its execution is aborted or suspended, provided that at least the ending task classifies a perdurant (e.g. decision to abort, to suspend, etc.). An additional model for control tasks can catch these notions:

(S1) ControlTask(AbortionTask)
(S2) Specializes(AbortionTask,EndingTask)
(S3) ControlTask(SuspensionTask)
(S4) Specializes(SuspensionTask,EndingTask)
(S5) ControlTask(CompletionTask)
(S6) Specializes(CompletionTask,EndingTask)

Completion task requires that the goal of the plan has been satisfied, while in abortion and suspension tasks it is not required:

(A104) ∀p,e. (Plan(p) ∧ Uses(p,CompletionTask) ∧ PlanExecution(e) ∧ P-Satisfies(e,p)) → ∃g,s. GoalSituation(s) ∧ Goal(g) ∧ Satisfies(s,g) ∧ ProperPart(p,g)

Of course, a plan execution can be aborted or suspended independently of an existing specific task ruling for that: these are properties of the situation, and usually prevent the plan execution to satisfy the plan (because the ending task couldn't be reached).

Additional notions and axioms catch stricter conditions expressible in plans:

A task (as any other concept) can be **optional** within some plan (or any description). In this case, it can be ignored in plan execution without affecting the satisfaction of the plan:

(A105) $\text{OptionallyUsedBy}(x,y) \rightarrow \text{UsedBy}(x,y) \wedge \text{Concept}(x) \wedge \text{Description}(y)$

Within plans, an task said to be optional should be placed in a way that preserves the topology (the connectedness) of the maximal task, except for sequential tasks, where it can be skipped without affecting the control structure. In fact, an optional task must either be component of a bag or sequential task, or have the concurrent task or the any-order task as a direct predecessor:

(A106) $(\text{OptionallyUsedBy}(x,y) \wedge \text{Task}(x)) \rightarrow \exists z. [(\text{BagTask}(z) \vee \text{SequentialTask}(z)) \wedge \text{Uses}(y,z) \wedge \text{Component}(z,x)] \vee [(z=\text{ConcurrencyTask} \vee z=\text{AnyOrderTask}) \wedge \text{Uses}(y,z) \wedge \text{DirectSuccessor}(z,x)]$

For example, "*eat your watermelon slice anyway you like, possibly without using your hands at all*" are action tasks controlled by the concurrency task, and one of them is optional.

A task can be **discarded** within some plan. In this case, it is ignored in plan execution without affecting the satisfaction of the plan. A discarded task can appear only as a direct successor to a deliberation task:

(A107) $\text{DiscardedWithin}(x,y) \rightarrow \text{UsedBy}(x,y) \wedge \text{Task}(x) \wedge \text{Plan}(y)$

(A108) $\text{DiscardedWithin}(x,y) \rightarrow \text{Component}(y,\text{DeliberationTask}) \wedge \text{DirectSuccessor}(\text{DeliberationTask},x)$

For example, "*eat your watermelon slice, but if it stinks, stop*" are action tasks controlled by a case task, and one of them leads to discarding the other one.

A taxonomy of the tasks defined in the OWL-DL version of DDPO is shown in Fig.9.

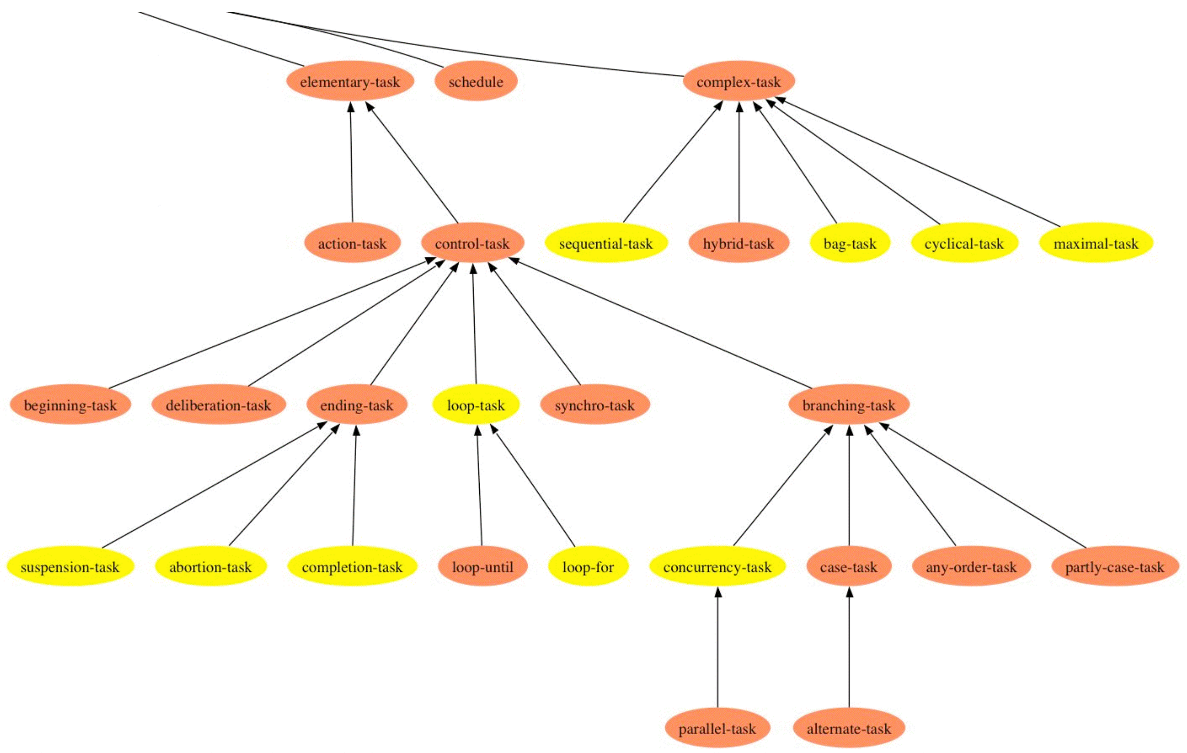

Fig. 9 Taxonomy of tasks in DDPO. Orange nodes represent *completely defined* OWL classes (having necessary and sufficient conditions). Yellow nodes represent *partially defined* OWL classes (having only necessary conditions). Arrows represent either *subclass-of* (IS_A) relations, or *instance-of* relations for control tasks.

3.4.4 Plan composition

It is possible to represent **plan composition**, when two plans which are defined separately should be joined under a common superplan. The primary components to be merged in plan composition are tasks. When the maximal tasks of two plans within the same plan are merged, the following operations can be performed: **juxtaposition**, **sum**, and **product**. Juxtaposition consists only in declaring which maximal task should be executed first:

(D45) TaskJuxtaposition(x,y,z) $=_{df}$ MaximalTask(x) ∧ MaximalTask(y) ∧ x≠y ∧ DirectSuccessor(x,y) ∧ ProperPart(z,x) ∧ ProperPart(z,y)

Sum consists in creating a task containing all the subtasks of the maximal tasks that are summed (no ordering is derivable from this operation):

(D46) TaskSum(x,y,z) $=_{df}$ ∀w,k. (MaximalTask(x) ∧ MaximalTask(y) ∧ x≠y ∧ Task(w) ∧ Task(k) ∧ w≠k ∧ Component(x,w) ∧ Component(y,k)) → Component(z,w) ∧ Component(z,k)

Product consists in adding some ordering to the sum: it results that, necessarily, each pair of action tasks from the two maximal tasks are either in a succession relation, or have a concurrency or any-order task as a direct predecessor:

(D47) TaskProduct(x,y,z) $=_{df}$ ∀w,k. (MaximalTask(x) ∧ MaximalTask(y) ∧ x≠y ∧ ActionTask(w) ∧ ActionTask(k) ∧ w≠k ∧ Component(x,w) ∧ Component(y,k)) → (Component(z,w) ∧ Component(z,k) ∧ ((Successor(w,k) ∨ Successor(k,w)) ∨ (DirectSuccessor(ConcurrencyTask,w) ∧ DirectSuccessor(ConcurrencyTask,k)) ∨ (DirectSuccessor(AnyOrderTask,w) ∧ DirectSuccessor(AnyOrderTask,k)))

Once an operation on maximal tasks has been performed, further operations can be performed on roles and parameters.

Concepts and figures can be **refined** by adding components, e.g. an elementary task can become complex, a complex task can increase its complexity, maximal tasks can be composed, etc.:

(A109) $\mathrm{Refines}(x,y) \rightarrow \mathrm{ProperPart}(y,x) \wedge (\mathrm{Concept}(x) \wedge \mathrm{Concept}(y)) \vee (\mathrm{Figure}(x) \wedge \mathrm{Figure}(y))$

(A110) $\mathrm{TaskJuxtaposition}(x,y,z) \rightarrow (\mathrm{Refines}(z,x) \wedge \mathrm{Refines}(z,y))$

(A111) $\mathrm{TaskSum}(x,y,z) \rightarrow (\mathrm{Refines}(z,x) \wedge \mathrm{Refines}(z,y))$

(A112) $\mathrm{TaskProduct}(x,y,z) \rightarrow (\mathrm{Refines}(z,x) \wedge \mathrm{Refines}(z,y))$

Consequently, descriptions can be **expanded** either by adding other descriptions as parts, or by refining their concepts or figures:

(A113) $\mathrm{Expands}(x,y) \rightarrow \mathrm{ProperPart}(y,x) \wedge \mathrm{Description}(x) \wedge \mathrm{Description}(y)$

(A114) $\mathrm{Expands}(x,y) \rightarrow (\exists z.\ \mathrm{Description}(z) \wedge \mathrm{PropertPart}(x,z) \wedge \leftarrow\mathrm{PropertPart}(y,z)) \vee \exists w,k.\ \mathrm{Concept}(w) \wedge \mathrm{Concept}(k) \wedge \mathrm{Refines}(k,w) \wedge \mathrm{UsedBy}(k,x) \wedge \mathrm{UsedBy}(w,y) \wedge \leftarrow\mathrm{UsedBy}(k,y) \wedge \leftarrow\mathrm{UsedBy}(w,x)$

3.4.5 Further work

Further work in DDPO will concentrate in the following areas:

- Organizational concepts (e.g. from the Enterprise Ontology and from the Business Modelling literature): role hierarchies, types of figures, statuses, missions, etc.
- On-the-fly definition of temporary figures and collectives (e.g. teams from the Klett case)
- Optimality conditions for plans, when dealing with sparse resources
- Strategies for plan accommodation (to circumstances) or adaptation (accommodation for reusability)
- Cost-functions as definable within ontologies
- …

4 Reengineering Metadata Structures by means of DDPO

This section illustrates a examples of how to use DDPO, the formal-ontological structure introduced in section 3. The main point is to illustrate for each example the threefold transition from a general description of a workflow, to an informal schematic representation of any given part of such workflow and, finally, to its formal characterization.

Section 4.1 provides a first intuitive presentation and partial restructuring of the material about the KLETT case study. The original material underlying this section is contained in the following three source documents: “Business Models” by W. Volz, W. Maas et al. (Doc-1, in the following); “First Sketch” by W. Volz & J. Schmidt (Doc-2, in the following); “E-learning Task Analysis” by Motti Benari (Doc-3, in the following). Section 4.1.1 presents an informal schema of both the Concept Design step and the Concept Development step in Klett’s workflow. This schema is then used in section 4.1.2 as a basis for the definition of a formal model, in terms of DDPO predicates and relations.

Section 4.2 provides a first intuitive presentation and partial restructuring of the material about the Templeton Oxford Retail Futures Group (ORFG) case study. The original material is contained in the following source documents: “Use Case: Templeton Oxford Retail Futures Group (ORFG)” by P. Young (Doc-4, in the following); “Discussion Templeton College Business Models Templeton” by M. Schäfer (Doc-5, in the following); “It was agreed at the April review meeting” by. P. Young (Doc-6, in the following); “Minutes: Application Development Meeting” by P. Young (Doc-7, in the following); “Taxonomy Diagrams” by P. Young (Doc-8, in the following). Section 4.2.1 presents an informal schema of the plan Agenda. This schema as well as the general material presented in the introductory part to section 4.2 are used in section 4.2.2 as a basis for the definition of a formal model, in terms of DDPO predicates and relations.

Section 4.3 provides a first intuitive presentation and partial restructuring of the material about the Clinical Trials case study. The original material is contained in the following source documents:… Section 4.3.1 presents…

4.1 The KLETT case study

KLETT Verlag is a German publishing house offering course material for different classes; it has 3 main products – schoolbooks for each school year; accompanying teaching material, e.g. history CD-ROMs; online learning material or programs - but the producing procedure and idea are nearly the same for all of them.
As reported in Doc-1, the Situations Design Methodology has been applied to an analysis and description of the most typical situations involved in KLETT Verlag business. An important caveat is immediately needed here: the meaning of the term ‘situation’ as used in Doc-1 is different from the meaning given to this term in DDPO. In order to adhere to the terminology used in Doc-1 and, at same time, in order to avoid possible ambiguities with DDPO’s situations, from now on we write *SITUATION* for whatever in Doc-1 is referred to with the term ‘situation’.

Now, according to Doc-1 *SITUATIONS* are settings into the business environment (which includes all the relevant stakeholders for a given business); they represent the business. *SITUATIONS* informally refer to and are connoted by definitions given in three areas[19]:

1. Social Sciences, where situations are human interaction patterns, by which persons judge and evaluate "the meaning of encounters" (Miller 1995) and have a clear view of their roles, rights and obligations.
2. From Epistemology, where situations are common patterns which enable interacting people to share meaningful information and knowledge (because they participate in a "community of thought", Fleck 1979).
3. From Artificial Intelligence, where situations, as formalized in the Situation Calculus are "snapshots of the world at some instant" (McCarthy 2000), (the world being the environment of the business at hand).

SITUATIONS are derived by developing a role-play for the business at hand; the key "actors" (i.e. stakeholders) are identified, then their roles are specified (in terms of duties, goals/motivations, rights, and obligations), and finally the participants "enact" the roles, rotating through each of them, so that they can understand the relative goals, etc. From this role-play, and the way the participants have described the interactions performed in order to achieve their goals, specific patterns and phases of interaction are derived.
Typical *SITUATIONS* consist of:

1. The *environment* the persons are acting in.
2. The *logical space* (i.e., the structure of the content exchanged between agents, its syntax and semantics).
3. The *channels of interaction* (telephone, e-mail, meetings, etc.).
4. The *organization/community* (roles and artifacts involved).

Other elements may be included, like e.g. the description of generic services supporting situations and interaction.

In Doc-1 the following sequence of 5 *SITUATIONS* is identified, which describes *in general* how to develop new course material and provide it to schools:

concept development → data collection → editorial work → course preparation → mediation and e-learning

Each *SITUATION* is described in terms of its key actor(s), their roles, duties, rights, obligations and tasks. It should be noticed that, as put in Doc-1, there are some overlaps and/or unclear distribution of information between these entries and that some of the entries are "nested". For instance, the *SITUATIONS* Concept Development and Data Collection are organized as follows:

SITUATION: **Concept Development**
Actor 1: Project Manager
Actor 2: Author(s)
Where:
Actor 1:

[19]From a DDPO point of view, 1 and 2 are better characterized as *descriptions* while 3 are *situations*.

Duties: co-ordinate the work with author(s) of educational material and check that material fits the syllabus of the different German Laender; responsible for pedagogical quality of final product and for marketing concept.
Rights: request info about needs of the Laender; co-ordinate author(s); assign tasks to author(s); ask them status reports.
Obligations: set the goals of training material (e.g. maths book 7th class).
Tasks: decision on orientation of material; development of conceptions for material; elaboration of complete implementation concept (including marketing).
Actor 2:
see: *SITUATION*: Data Collection

SITUATION: **Data Collection**
Actor 1: Author
Where:
Actor 1:
Duties: find and/or develop data for teaching materials
Rights: access various information and data
Obligations: deliver ready to use teaching material fitting the needs of the Laender
Tasks: assembling of information; enrichment with pedagogical knowledge and methods according to the requirements set by the Project Manager

During the Stuttgart Meeting with KLETT (see Doc-1), the typical *internal* workflow of KLETT's production was roughly modelled as follows (note that this only partially matches the general workflows described above):

idea to develop course material → concept design → concept development → production → sales

Two of the above *SITUATIONS* are of particular interest to Metokis: **Concept Design** and **Concept Development**. They occur only once the decision to develop new course material has been taken. Note that the steps in the complete workflows and the relative terminology are different in the 3 source documents and that they only partially match one another. The *SITUATIONS* Concept Design and Concept Development seem to correspond to Use Case 1 (Business Plan Creation) and, respectively, Use Case 2 (Editorial Support) in Doc-2. On the other hand, the first three *SITUATIONS* correspond to the "Creation stage" as analysed in Doc-3.

Section 4.1.1 presents an informal model of both the Concept Design and the Concept Development.

INTERACTION PHASES CONCEPT DESIGN	INTERACTION PHASES CONCEPT DEVELOPMENT
concept outline → technical description → production planning → financial planning → release	project setup → content creation → technical implemetation → marketing concept → production launch → project finalization

4.1.1 Schemas for Concept Design and Concept Development

CONCEPT DESIGN:	CONCEPT DEVELOPMENT:
It is a proposal-generation situation, where an idea of developing new material is at hand and a development plan has to be elaborated, upon which a decision about actual development can be made.	If the business plan is accepted, a pilot version of the new learning material is developed which can be directly produced.

ROLES:	**ROLES:**
• PROJECT MANAGER (PM): <u>Duties/Responsibility</u>: o compile the development plan <u>Rights</u>: o request information (from various other roles) o staff the project (involving other roles) o taking decisions on content and strategy <u>Obligations</u>: o compile a <u>directly applicable</u> development plan o organize and co-ordinate all other involved roles <u>Goals/Motivation:</u> o acquire a concept development project that leads to high profits for KLETT and thus (being then run by him) strengthens his position within KLETT) • TECHNICAL PROJECT MANAGER (TPM): <u>Duties/Responsibility</u>: o evaluate the technical perspective of the PM concept o represent the technical strategy of KLETT <u>Rights:</u> o be involved in development of new technical concepts, i.e. make suggestions on, and enjoin in, themes concerning technical details <u>Obligations</u>: o support the PM with technical decisions, i.e. provide info so that the right standards can be met and	• PROJECT MANAGER (PM): <u>Duties/Responsibility:</u> o co-ordinate and plan steps of compilation of the new material <u>Rights</u>: o arrange duties with editors and authors o arrange duties and deliverables with programmers and technical project manager o request information about the status of work from involved staff o request support from assistant(s) concerning administration <u>Obligations</u>: o co-ordinate the compilation of new material - set tasks and relative deadlines to the editors and authors - control the project and adjust planning if delays occur - provide a ready-to-use version of material and deliver it to production <u>Goals/Motivation:</u> o produce high-quality material, because it is upon the performance of this material (once launched) that he will be measured • TECHNICAL PROJECT MANAGER (TPM): <u>Duties/Responsibility</u>: o supervise the production of new material from a technical point of view <u>Rights:</u> o be involved in all technical decisions concerning new material o declare technical standards for KLETT o share decisions with the PM on kind of technology used <u>Obligations</u>:

To be included in Concept Design and Concept Development are the **teams** formed by some of the agents playing the roles.

ORGANIZATION (TEAMS): The PM is responsible for the whole Concept Design. He forms several teams and subteams.	**ORGANIZATION (TEAMS):** The PM is responsible for the whole Concept Develpment. He forms several teams and subteams.

TEAM TYPES	**TEAM TYPES**
• (EXECUTIVE) TEAM^{1} Roles: o PM, Assistant(s) Functions: o organize Concept Design o elaborate business plan	• PROJECT MANAGEMENT TEAM^{1} Roles: o PM + Assistant(s) Functions: o organize Concept Development o elaborate new material
• (EDITORIAL) TEAM^{2} Roles: o PM, 2 to 10 Editors Functions: o make all decisions regarding content of new material (implies responsibility for content)	• CONCEPT DEVELOPMENT TEAM^{2} Roles = o PM, 2 to 10 Editors Functions: o make all decisions regarding development of content of new material (implies responsibility for content developed)
. . . .	• TEAM^{3} Roles: o PM, Technical PM Functions: o check concept development with technical details
. . . .	• TEAM^{4} Roles: o PM, Production Manager [undefined within *this* plan] Functions: o countercheck concept development with production possibilities and deadlines for market launch
• SUBTEAM^{1} Roles: o TEAM^{1}, Organizational, Consultants(Controllers) [?] Functions: o [control tasks?]	• SUBTEAM^{1} Roles: o TEAM^{1}, Organizational, Consultants(Controllers) [?] Functions: o [control tasks?]
• SUBTEAM^{n} Roles: o PM, each Editor, whole of TEAM^{2,} Author(s), Consultant(s) Functions: o collaborate on decisions regarding content of new material	• SUBTEAM^{n} Roles: o PM, each Editor, whole of TEAM^{2,} Author(s), Consultant(s) Functions: o collaborate on decisions regarding content of new material

4.1.2 Formal Model

The following is a model of a fragment of the examples introduced. Six plans with some of their defined tasks, roles, and parameters are formalized by means of DDPO classes and relations.
The original material did not include clear decisions on task ordering (at the release time of version 1.0, Klett experts are still producing a revision including an ordering), then this model does not make a heavy use of DDPO predicates.
Here we present the model in a FOL model syntax, while the Annex contains the model in OWL-RDF.

```
Plan(producing_1_piece_of_new_learning_material)
Plan(acquire_idea)
Plan(concept_design)
Plan(concept_development)
Plan(production)
Plan(sales)

ProperPart(producing_1_piece_of_new_learning_material, acquire_idea)
ProperPart(producing_1_piece_of_new_learning_material, concept_design)
ProperPart(producing_1_piece_of_new_learning_material, concept_development)
ProperPart(producing_1_piece_of_new_learning_material, production)
ProperPart(producing_1_piece_of_new_learning_material, sales)

Goal(bring_high_profits_to KLETT)
ProperPart(producing_1_piece_of_new_learning_material, bring_high_profits_to KLETT)
Goal(acquire_a_development_plan)
ProperPart(concept_design, acquire_a_development_plan)
Goal(develop_a_pilot_version_of_new_learning_material)
ProperPart(concept_development, develop_a_pilot_version_of_new_learning_material)

Task(decide_on_content)
Task(provide_info_on_technical_standards)
Task(provide_info_on_administrative_issues_regarding_content)
Task(compile_development_plan)
Task(disburden_project_manager)
Task(set_deadlines_to_authors)
Task(provide_content)
Task(coordinate_compilation_of_new_learning_material)

Component(concept_design, decide_on_content)
Component(concept_design, provide_info_on_technical_standards)
Component(concept_design, provide_info_on_administrative_issues_regarding_content)
Component(concept_design, compile_ development_plan)
Component(concept_design, disburden_project_manager)
Component(concept_development, set_deadlines_to_authors)
Component(concept_development, provide_info_on_technical_standards)
Component(concept_development, provide_content)
Component(concept_development, coordinate_compilation_of_new_learning_material)
Component(concept_development, disburden_project_manager)

AgentDrivenRole(project_manager)
AgentDrivenRole(technical_project_manager)
AgentDrivenRole(author)
AgentDrivenRole(assistant)
Role(standard)

Component(concept_design, project_manager)
Component(concept_design, technical_project_manager)
Component(concept_design, author)
Component(concept_design, assistant)
Component(concept_design, standard)
Component(concept_development, project_manager)
```

Component(concept_development, technical_project_manager)
Component(concept_development, author)
Component(concept_development, assistant)
Component(concept_development, standard)

RightTo(project_manager, decide_on_content)
ObligationTo(project_manager, set_deadlines_to_authors)
ObligationTo(project_manager, coordinate_compilation_of_new_learning_material)
ObligationTo(technical_project_manager, provide_info_on_technical_standards)
ObligationTo(author, provide_info_on_administrative_issues_regarding_content)
ObligationTo(author, provide_content)
DutyTo(assistant, disburden_project_manager)
DutyTo(project_manager, compile_ development_plan)

Parameter(right)
RequisiteFor(right, standard)

4.2 The Templeton Oxford Retail Futures Group case study

The Templeton Oxford Retail Futures Group (ORFG) is a group of senior executives with links to retail that meets six times a year. The group is hosted by Templeton College. According to Doc-4 each meeting typically consists of: a presentation by an authority on a topic of interest to the group, a discussion on the presentation, a dinner and a chance to network and, if the presentation is off-site/not at Templeton, a guided tour. Furthermore, each year has one special CEO meeting, where a high-level (and possibly international) executive talks to the group.

Members of the ORFG fall into 4 camps: Retail Executives, Non-Retail Executives, Retail Associations, Templeton & Oxford Institute of Retail Management (OXIRM).
Others involved in the ORFG are: Speakers (who give presentations at the meetings. These may or may not be current ORFG members), Support (such as staff involved in organising the events), Potential New Members.

Members of the ORFG join to get a better Individual understanding of and ability to advocate the future of retailing over the next 3-7 years. Moreover, each member has its own particular pay-off from participating in ORFG: Templeton improves its research and credibility, retail members improve their organisations operations and improve/sustain their credibility, Non-retail members gain business development opportunities, Organisations gain more knowledgeable employees better equipped to plan and implement the organisation's future retail-related strategy and operations.

According to Doc-4, the ORFG system can be outlined as below. Notice that this scheme (as most of those picturing ORFG's workflow in Docs-5to8) is very rich -- it provides valuable information -- but still too generic -- for instance: arrows are used homogeneously, i.e. ambiguously. Therefore, ontological analysis should in the first place be used to support disambiguation.

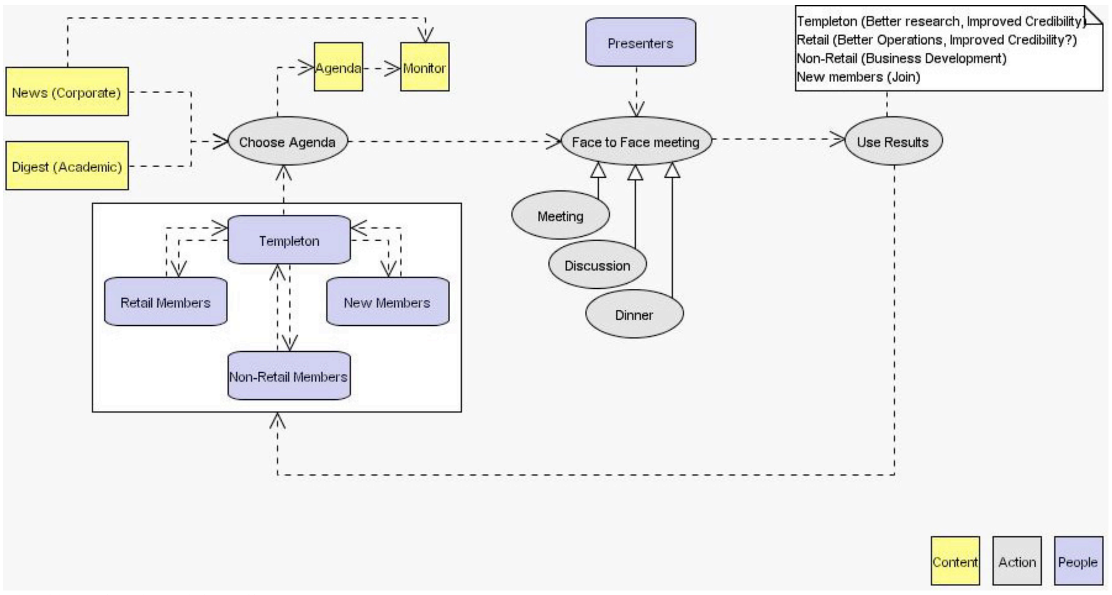

Fig. 10 Outline of ORFG system

According to Doc-4, there are three key (interrelated) areas:
1. Content, including agenda, monitor, blackboard and access to presentation. Content supports:
2. Actions, usually related to the organization of an event (a meeting), where one or more of the following are combined: speakers (for the presentation), venues (for the tour and/or presentation and/or dinner), invites. Actions support:
3. People, are involved in one of the following ways: recruitment, monitoring and maintenance, member loss. People support content.

This way of describing things is a little shallow, especially for what concerns the use of the "support" relation, which has arguments that are ontologically disparate.

In order to structure, according to DDPO, the knowledge presented so far, one should proceed top-down and give an explicit definition of ORFG's Plan, Sub-plans, Roles.
One way of doing this is by simply defining ORFG's plan as to meet 6 times a year. Retail and non-retail executives, Oxirm, retail associations, speakers, supporting stuff and potential new members pursue, each according to their role, the goal of this plan: enhance members' understanding of the retail sector. At same time, by playing a role in ORFG's plan, each member takes care of its own backyard: for instance, Oxirm has a disposition to improve research and credibility; retail associations have a disposition to improve operations and credibility; etc. ORFG's plan entails at least the following four plans: make agenda, monitor literature, manage blackboard, organize single event, keep relations with people. Each of these sub-plans has, besides its own sub-plans and roles, a number of tasks, i.e. a number of actions to be performed in order to get the job done. In the following section, an example is provided of the task structure of the plan make agenda, as presented in Doc-4. The three other sub-plans are schematized in a similar fashion.

4.2.1 Schema for Agenda

The goal of the plan make agenda is to have a decided agenda for the next year's 6 meetings. The following tasks are involved: identify known events, choose topics to fit members' interests, validate topics, identify speakers, choose speakers to fit topics, validate speakers, publish agenda.

1. **Identify known events**. Retail sector events in the coming year that are known to be happening are identified. Examples may be “academic”, such as key research publication dates (e.g. DTI & Productivity report), or “industry”, such as the opening of the Birmingham Bull Ring. The final meeting, in which a CEO gives a talk at Templeton, is also marked.
2. **Choose topics of interest**. Informal discussions are held between Templeton and Retail members to gauge what the main topics of interest are. This is supported by an informal discussion document. Once agreed a formal “yes/no/what else” document is sent to all ORFG members for their feedback. Once feedback has been received, an outcome discussion document is written detailing the chosen topics.
3. **Validate topics**. At the same time as the topics are being chosen, Templeton and the ORFG validates the topics to make sure that the final list conforms to various constraints. These constraints are: Community management (ensuring members are happy with topics), Breadth of focus (ensuring all topics are equally addressed), Breadth of location (ensuring meetings are split between Templeton and other venues) That previous or “old” topics are readdressed as major changes occur.
4. **Choose speakers**. Once the topics have been chosen, Templeton searches for speakers to talk on each topic. Speakers are found through searching: The ORFG community (who they know) news & media.
5. **Validate speakers**. As with the topics, at the same time as the speakers are being chosen, Templeton validates the choices to make sure that the final list conforms to various constraints. These constraints are: Focus (ensuring that the speakers are mainly from retail and that, where possible, there are no suppliers as speakers). Breadth of background (ensuring a counterbalance between academic and corporate views)
6. **Publish agenda**. Once topics and speakers have been finalised, the agenda is sent to all ORFG members.

4.2.2 Formal Model

The following is a model of a fragment of the material introduced above. Both ORFG general plan and the plan make agenda are modelled by means of DDPO classes and relations. Here we present the model in a FOL model syntax, while the Annex contains the model in OWL-RDF.

Plan(meet_6_times_a_year)
Plan(make_agenda)
Plan(monitor_literature)
Plan(manage_blackboard)
Plan(organize_single_event)
Plan(keep_relations_with_people)

ProperPart(meet_6_times_a_year, make_agenda)
ProperPart(meet_6_times_a_year, monitor_literature)
ProperPart(meet_6_times_a_year, organize_single_event)
ProperPart(meet_6_times_a_year, keep_relations_with_people)

Goal(enhance_members’_understanding_retail_sector)
ProperPart(meet_6_times_a_year, enhance_members’_understanding_retail_sector)
Goal(improve_research_and_credibility)
InfluenceOn(enhance_members’_understanding_retail_sector,
improve_research_and_credibility)
Goal(improve_operations_and_credibility)

InfluenceOn(enhance_members'_understanding_retail_sector,
improve_operations_and_credibility)
Goal(gain_business_opportunities)
InfluenceOn(enhance_members'_understanding_retail_sector, gain_business_opportunities)
Goal(gain_more_knowledgeable_employees)
InfluenceOn(enhance_members'_understanding_retail_sector,
gain_more_knowledgeable_employees)

AgentDrivenRole(retail_executive)
AgentDrivenRole(non_retail_executive)
AgentDrivenRole(speaker)
AgentDrivenRole(supporting_staff)
AgentDrivenRole(potential_new_member)
AgentDrivenRole(representative)
AgentDrivenRole(oxirm_representative)
Specializes(representative, oxirm_representative)
Institution(oxirm)
Deputes(oxirm, oxirm_representative)
$\forall$x,y. (RetailAssociation(x) $\wedge$ Deputes(x,y)) $\rightarrow$ y = retail_association_representative)
AgentDrivenRole(retail_association_representative)
Specializes(representative, retail_association_representative)
RetailAssociation(x) $\rightarrow$ Institution(x)

Component(meet_6_times_a_year, retail_executive)
Component(meet_6_times_a_year, non_retail_executive)
Component(meet_6_times_a_year, retail_association_representative)
Component(meet_6_times_a_year, oxirm_representative)
Component(meet_6_times_a_year, speaker)
Component(meet_6_times_a_year, supporting_staff)

DispositionTo(oxirm, improve_research_and_credibility)
DispositionTo(retail_associtation, improve_operations_and_credibility)
DispositionTo(non_retail_executive, gain_business_opportunities)
DispositionTo(oxirm, gain_more_knowledgeable_employees)
DispositionTo(retail_associtation, gain_more_knowledgeable_employees)

Goal(having_a_decided_agenda)
ProperPart(make_agenda, having_a_decided_agenda)

Component(make_agenda, oxirm)
Task(identify_known_events)
Task(choose_topics_to_fit_members_interest)
Task(validate_topics)
Task(identify_speakers)
Task(choose_speakers_to_fit_topics)
Task(validate_speakers)
Task(publish_agenda)

Defines(make_agenda, identify_known_events)
Defines(make_agenda, choose_topics_to_fit_members_interest)
Defines(make_agenda, validate_topics)

Defines(make_agenda, identify_speakers)
Defines(make_agenda, choose_speakers_to_fit_topics)
Defines(make_agenda, validate_speakers)
Defines(make_agenda, publish_agenda)

Role(known_event)
Subject(known_event, identify_known_event)
Parameter(current)
RequisiteFor(current, known_event)

5 An ontology of information objects

Currently, the proposed ontology for information objects is reused from an extension of DOLCE, but future versions of the deliverable will customize it to the needs of Metokis use cases. Part of the reused ontology has been developed within the WonderWeb EU project [15][17][14][10].

5.1 The basic IO design pattern

According to the reused ontologies mentioned in previous sections, a content (information) transferred in any modality is assumed to be equivalent to a kind of social object called **information object** (IO). Information objects are *spatio-temporal reifications* of pure (abstract) information as described e.g. in Shannon's communication theory, hence they are assumed to be in time, and realized by some entity.
Information objects are the core notion of a *semiotic ontology design pattern*, which employs typical *semiotic relations*, as explained here.
An IO has the following properties (Fig.11): a *support* that **realizes** the IO, one or more *combinatorial structure(s)* (or **information encoding system**), according to which IO is **ordered**, a *meaning* (or *conceptualization*) that an IO **expresses,** a *reference* an IO **is about**, and one or more agents that **interpret** the IO (see [20] for a review of the relations between ontology and semiotics, and a similar account of semiotic relations):

(A115) InformationEncodingSystem(x) → Description(x)
(A116) InformationObject(x) → SocialObject(x)
(A117) InformationObject(x) → ∃y. InformationEncodingSystem(y) ∧ OrderedBy(x,y)
(A118) InformationObject(x) → ∃y. Particular(y) ∧ RealizedBy(x,y,t)
(A119) InformationObject(x) → ∀y. Expresses(x,y,t) → Description(y)
(A120) InformationObject(x) → ∀y. About(x,y,t) → Particular(y)
(A121) InformationObject(x) → ∀y. Interprets(y,x,t) → Agent(y)

For example, Dante's Divine Comedy is an IO, it is *ordered* by Middle Age Italian language (the information encoding system), is *realized* by e.g. a paper copy of the 1861 edition with Doré's illustrations, *expresses* a certain plot and its related meanings (literal or metaphorical), as *interpreted* by an agent with an average knowledge of MA Italian and literary criticism, and it is *about* certain entities and facts (Fig. 11).

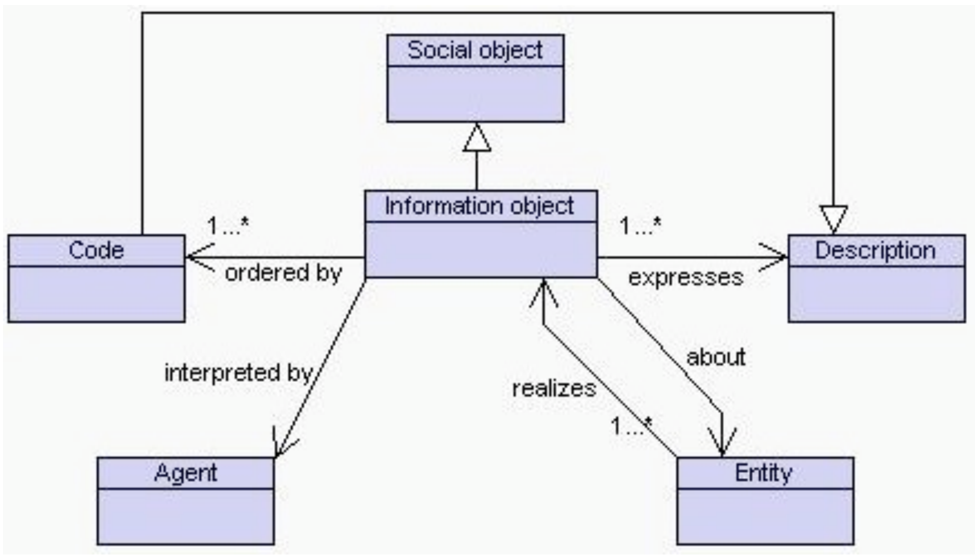

Fig. 11 The basic IO design pattern

Therefore, we are assuming the following functional reductions: *meanings* (independently of how a meaning is theorized: as a mental content, intertextual reference, propositional content, etc.) can be partly formalized as *descriptions*; *supports* are *particulars* of some kind (e.g. a paper sheet, a sound, a sequence of bits or pulses, etc.); *referenced particulars* are particulars whatsoever (usually *set* in *situations* that satisfy an expressed description); interpretations of IOs are confined to endurants that can (directly or indirectly) *conceive* descriptions: *agents*. IOs are necessarily *encoded* by some information encoding system, and must be *realized* by some particular. IOs can *express* a description, and if that description is satisfied by a situation, IOs can be *about* it. Finally, IOs can be *interpreted* by agents that can the description expressed by said IOs.
This theory is compliant with so-called *communication elements* (roles) defined by Jakobson [21]: an information object works as a *message*, an information encoding system as a *code*, a supporting entity acts as *channel*, an interpreting agent works as an *encoder* or *decoder*, and an expressed description as a *context*.
Among the semiotic relations used above to create the IO pattern are the following:

(A122) Orders(x,y) → InformationEncodingSystem(x)
(A123) Realizes(x,y,t) → Particular(x)
(A124) Expresses(x,y,t) → Description(y)
(A125) About(x,y,t) → Particular(x)
(A126) Interprets(x,y,t) → Agent(x)

The relations *realizes*, *expresses*, *about*, and *interprets* must be taken as *temporally indexed*.

5.2 Advanced paths in the IO pattern

These *semiotic* relations constitute a typical *ontology design pattern*, so that any composition of relations can be built starting from any node in the pattern or in an application of the pattern.
The pattern has also some required paths (Fig. 12):

i) for any description, it is mandatory to have at least one IO that *expresses* it:

(A127) Description(x) → ∃y. InformationObject(y) ∧ ExpressedBy(x,y,t)

ii) *interprets* implies that an expressed description is *conceived* by the agent (i.e., when an agent interprets an IO, it conceives the description expressed by the IO; of course two agents can conceive different descriptions, then resulting in different interpretations):

(A128) Interprets(x,y,t) → ∃d. Description(d) ∧ Expresses(y,d,t) ∧ Conceives (x,d,t)

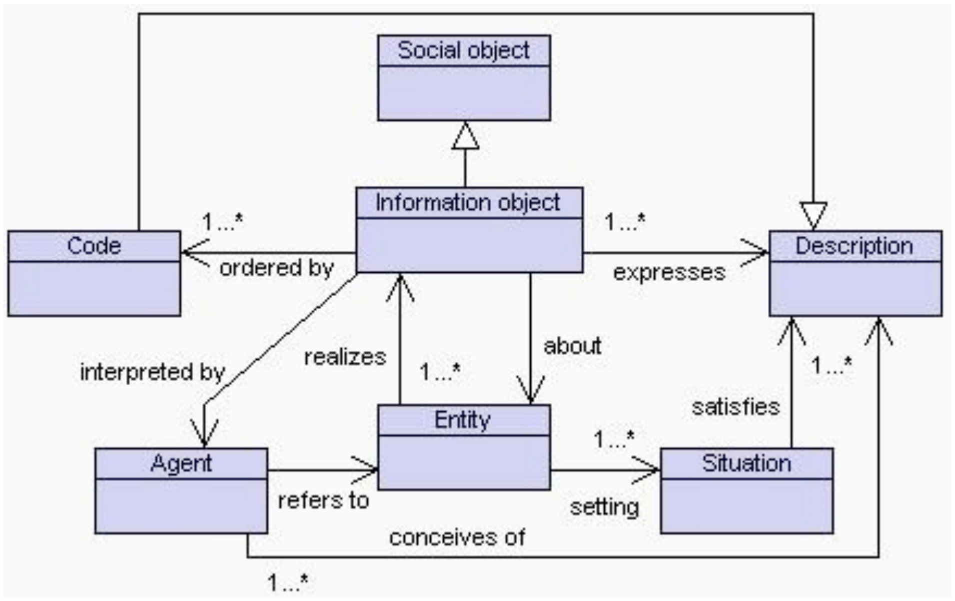

Fig. 12 The IO pattern with some implied paths: situations exist for the setting of *realization*, as well as *aboutness*; agents *refer* to particulars that interpreted IOs are about, etc.

iii) another axiom can be proposed about the need of one description and (at least) one situation so that an IO be *about* something. Given that descriptions are expressed by at least one IO, and that interpretations of IOs require both conceiving a description and the (plausible) claim that being about something can only be done *in context*, i.e. within a situation), we can propose that the conceived description is satisfied by the situation (the context) of the particular the IO is about:

(A129) About(x,y,t) ↔ ∃d,s. Description(d) ∧ Expresses(x,d,t) ∧ Situation(s) ∧ SettingFor(s,y) ∧ SAT(s,d)

iv) the previous axiom makes a move to "negotiated reference", i.e. agents **refer to** particulars by conceiving a description appropriate to the given context:

(A130) RefersTo(x,y,t) → Agent(x) ∧ Particular(y) ∧ ∃z,d. InformationObject(z) ∧ Description(d) ∧ Interprets(x,z,t) ∧ Expresses(z,d,t) ∧ About(z,y,t)

v) IOs can be *realized* by any sort of entities, provided that the structure of the entity is such that the ordering of an IO is properly mapped. On the other hand, an IO can be *about* any sort of entities, provided that the entity can be situated into a situation that satisfies a conceivable description.[20] Possibly, this twofold nature of entities wrt to semiotic properties accounts for the enormous expressive power of e.g. human information encoding systems: humans use the world to map (represent) the world, including themselves, and the "legacy" structure of the world is exploited or modified in order to gain even more power, in a game that leads to many layers of abstraction.
Entities (may) reveal information realized by their own, as well as other information realized by them, but not *proper* of their own. The second revealing requires a *mapping* (or representation) context, the first does not. For example, a (traditional, figurative) painting of a landscape realizes a picture, which is about that landscape, but it also "exhibits" its own structure (information), which can be appreciated. Hence, there is a sense in which any entity that realizes an IO also realizes an IO about itself:

[20] In principle, any IO can be about any entity, but social conventions, usage history, and various *iconical* or *economical* reasons actually limit conceivability and expressibility. Most of the literature in philosophy of language and semiotics accounts for these issues; Eco 1997 has a long section precisely on these limits (cf. [21].

(A131) Realizes(x,y,t) → ∃z. About(z,x,t) ∧ Realizes(x,z,t)

For example, a painting realizing information about a woman also realizes information about itself.
If one wants to consider an entity as inherently semiotic, when in the domain of an ontology, then the axiom above needs to be complemented by y≠z.
Of course, the converse of the previous axiom does not hold in general:

(A132) * About(x,y,t) → ∃z. Realizes(z,x,t) ∧ About(x,z,t) [∧ y≠z]

For example, the information about a woman can be realized by entities that are different from the woman in question (as when referring to an absent woman).
In other words, an entity (when considered in a semiotic perspective) always realizes two information objects: one about itself, and another about something else. In the non-semiotic cases, the two information objects are identical (an entity only realizes information about itself).
We could propose that entities, once they have a relevance in a society, can have semiotic properties. Even physical artifacts that are not built primarily for communicative purposes – e.g. a chair – can be considered as realizing some IO that expresses a *design* description, and that is about a context (situation) of *use*, *fruition*, or just *affordance* that satisfies the design.
Of course, the aboutness of IOs realized by physical artifacts is peculiar, since they are more evidently about the *artifacts themselves* than about the context; while, on the contrary, the "intrinsic" aboutness of IOs realized by typical semiotic artifacts (texts, pictures, voice) is less evident, with the notable exception of artistic realizations.
The *Comedy* example sketched above can be refined with the paths embedded in the IO design pattern (Fig. 13).

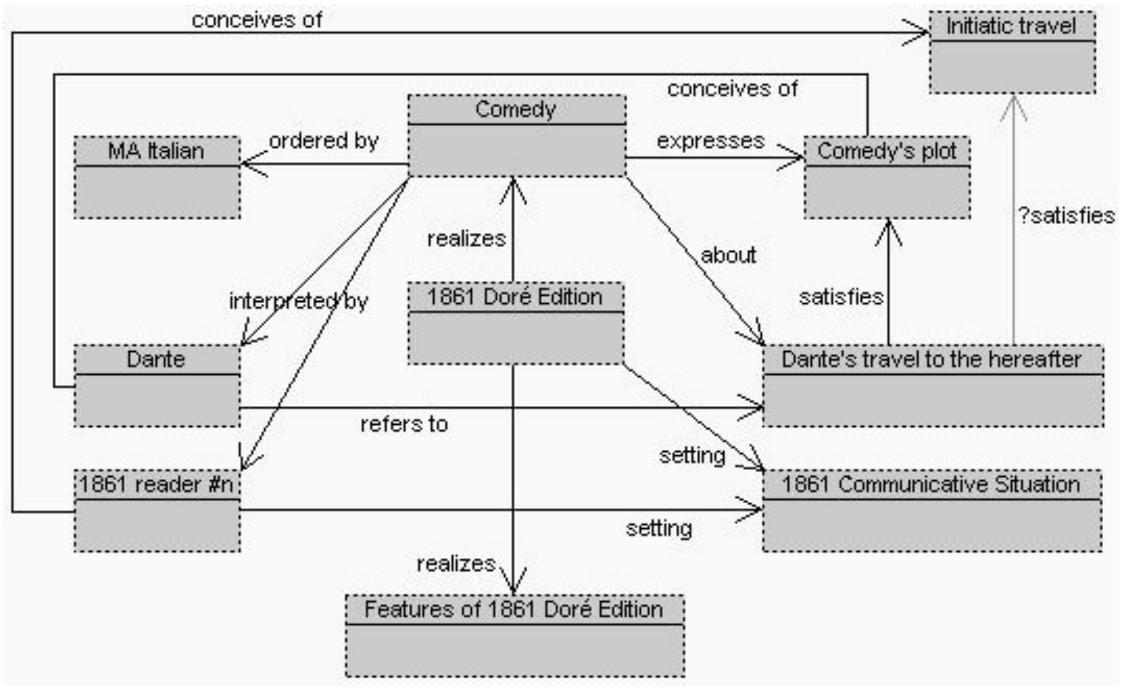

Fig. 13 A model about *Comedy* using the extended IO design pattern.

5.3 Using the IO design pattern

As a summary, we can conceive of a maximal semiotic relation **S**, and its projections:

(D48) S(e,io,c,a,d,n,t) $=_{df}$ Particular(e) ∧ InformationObject(io) ∧ InformationEncodingSystem(c) ∧ Agent(a) ∧ Description(d) ∧ Particular(n) ∧ Realizes(e,io,t) ∧ OrderedBy(io,c) ∧ Interprets(a,io,t) ∧ Expresses(io,d,t) ∧ About(io,n,t)

In practice, the maximal semiotic relation is defined after the axioms provided for its projections.

An interesting case of semiotic entrenchment has appeared in (computational) *knowledge representation* (KR). Computational (software) domain has a reality of its own, consisting of symbols (that are *abstract regions* in DOLCE) that are manipulated (ordered) by programs that can implement algorithms.
KR techniques have introduced the practice of distinguishing the symbols (called *concrete data types*, or simply *data types*, e.g. in OWL) that can be computed *as such* by the program or by additional programs, and the symbols (called *abstract data types*, or *objects*, e.g. in OWL) that are invariant across the logical computation, and whose lifecycle depends on entities that are not computed, because external to the computation world. For example, the notion of *30 ducks* can be equally represented by the expressions:

(S1) ∃x. Duck(x) ∧ Numerosity(x,30)
(S2) ∃x. Duck(x) ∧ Numerosity(x, =(* 3 10))

because *30* is computationally equivalent to *(*3 10)*, i.e. there is a procedure to derive 30 from (* 3 10), while *duck* is a symbol of a predicate that semantically extends on all the ducks assumed to be represented in the domain of the theory (and ducks cannot be computed[21]).
This distinction is convenient to extend the domain of KR theories to entities that cannot be efficiently manipulated by inference engines (and unnecessarily so), e.g. classification algorithms.
Ontologically, data types are regions: concrete data types can be used as *values* of e.g. metric relations of measurement, temporal and spatial location, etc., while abstract data types are used as *names* for logical entities (values of a *name* relation ranging on predicates, constants, etc.).
On the other hand, when considered in a semiotical perspective, data types are information objects, the only ones that have a computational life (they are realized in machines), are about regions (concrete data types) or other entities (abstract data types).
Data types are actually manipulated, exchanged, etc. No other entities are manipulated in electronic services. We can therefore exploit the IO pattern to create/reclassify metadata about 'content', which enhance content manipulation technically, economically, legally, and from the usability viewpoint.

In order to put metadata on content, we need to know what kind of entities those metadata are talking about.

[21] In principle, ducks could be *simulated*, but this is another story.

6 Annex

6.1 OWL-DL abstract syntax of DDPO (complete)

The following is the complete DDPO code as included in the DOLCE-Lite-Plus library (version 3955) in the OWL abstract syntax form. The concrete syntax version has been validated for OWL-DL, and checked for consistency and classified with FaCT++ and RACER.

OWL Species Validation Report

Conclusion

DL: YES

Abstract Syntax Form

```
Namespace(rdf    = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(xsd    = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs   = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(owl    = <http://www.w3.org/2002/07/owl#>)
Namespace(a      = <http://www.loa-cnr.it/ontologies/CommonSenseMapping#>)
Namespace(b      = <http://www.loa-cnr.it/ontologies/TemporalRelations#>)
Namespace(c      = <http://www.loa-cnr.it/ontologies/DOLCE-Lite#>)
Namespace(d      = <http://www.loa-cnr.it/ontologies/DnS#>)
Namespace(e      = <http://www.loa-cnr.it/ontologies/Plans#>)
Namespace(f      = <http://www.loa-cnr.it/ontologies/InformationObjects#>)
Namespace(g      = <http://www.loa-cnr.it/ontologies/SemioticCommunicationTheory#>)
Namespace(h      = <http://www.loa-cnr.it/ontologies/ModalDescriptions#>)
Namespace(i      = <http://www.loa-cnr.it/ontologies/SocialUnits#>)
Namespace(j      = <http://www.loa-cnr.it/ontologies/Collections#>)
Namespace(k      = <http://www.loa-cnr.it/ontologies/Actions#>)
Namespace(l      = <http://www.loa-cnr.it/ontologies/Collectives#>)
Namespace(m      = <http://www.loa-cnr.it/ontologies/FunctionalParticipation#>)
Namespace(n      = <http://www.loa-cnr.it/ontologies/SpatialRelations#>)
Namespace(o      = <http://www.loa-cnr.it/ontologies/Systems#>)
Namespace(p      = <http://www.loa-cnr.it/ontologies/Causality#>)

Ontology( <http://www.loa-cnr.it/ontologies/DOLCE-Lite>

 Annotation(rdfs:label "DOLCE-Lite-Plus")
 Annotation(owl:versionInfo "3.9.4.1”)
 Annotation(rdfs:comment "The version 3.9 of DOLCE-Lite (updated to D18 of DOLCE-Full) with  some basic extensions, called DOLCE-Lite-Plus, or DLP.  The ontology graph in this version is the following:
------Backbone:
http://www.loa-cnr.it/ontologies/DOLCE-Lite#
 http://www.loa-cnr.it/ontologies/TemporalRelations#
 http://www.loa-cnr.it/ontologies/SpatialRelations#
 http://www.loa-cnr.it/ontologies/DnS#
------Basic extensions:
  http://www.loa-cnr.it/ontologies/InformationObjects#
  http://www.loa-cnr.it/ontologies/Actions#
   http://www.loa-cnr.it/ontologies/SocialUnits#
    http://www.loa-cnr.it/ontologies/Plans#
     http://www.loa-cnr.it/ontologies/FunctionalParticipation#
     http://www.loa-cnr.it/ontologies/Collections#
      http://www.loa-cnr.it/ontologies/Collectives#
       http://www.loa-cnr.it/ontologies/CommonSenseMapping#
-----Experimental extensions:
  http://www.loa-cnr.it/ontologies/Systems#
  http://www.loa-cnr.it/ontologies/SemioticCommunicationTheory#
  http://www.loa-cnr.it/ontologies/Causality#
```

http://www.loa-cnr.it/ontologies/ModalDescriptions#
The backbone of the library is constituted by
(1) DOLCE-Lite
(2) two sets of temporal relations defined over perdurants which are adapted from Allen's temporal calculus, and of spatial relations that simplify the expression of places and locations from particulars to regions
(3) the DnS (Descriptions and Situations) ontology, which provides a vocabulary to talk of reified entities such as relations, roles, contexts, situations, parameters, etc. Appropriate relations link DnS reifications to DOLCE-Lite non-reified entities.

Based on that backbone, other wide-scoping ontologies are provided:
(4) ontology of information objects, based on semiotics, which provides a vocabulary to talk of languages, expressions vs. meaning, logical vs. physical documents, reference, etc.
(5) a still preliminary and rough vocabulary for actions, agents and social units (persons, organizations)
(6) a well-developed ontology of plans and tasks, containing also a set of
Individual tasks that provide grounded primitives to specify process types
(7) a preliminary ontology of functional participation relations, which provide a vocabulary for event-oriented relations encoded by linguistic verbs (in Western languages), like 'performs' or 'makes'
(8) an ontology of collections and collectives
(9) a set of common sense mappings, introduced to support a mapping to WordNet (contained in another file).

Besides these basic extensions, which are currently exploited in several application domains, and are actively under development, there are also some less developed ontologies, all bases on the backbone, but still at a preliminary and debatable stage. Some of them are included here as placeholders, and are used by some applications, but they are not yet stable.

*******Scope of DOLCE-Lite-Plus*******
The lite versions of DOLCE are simplified translations of DOLCE into various logical languages. They are maintained for several reasons:
1. allowing the implementation of DOLCE-based ontologies in languages that are less expressive than FOL. In particular, DOLCE-Lite does not make use of S5 modalities and of some temporally-indexed relations. Modal operators are not heavily exploited in DOLCE, then the consequences are not very harmful for most uses. Temporal indexing is partly supported by composing originally indexed relations with temporal location relations. Even this support is not provided for description logic versions of DOLCE-Lite like DAML+OIL, OWL-DL, etc.
2. allowing a description-logic-like naming policy for DOLCE signature. In many cases, different names are adopted for relations that have the same name but different arities in the FOL version, or for relations that have polymorphic domains
3. allowing extensions of DOLCE that do not have a detailed axiomatization yet, and modularizing them (placeholders)
4. taking benefit of the services of certain implemented languages -specially the classification services provided by description logics- in order to support domain applications The DLP ontology library is currently maintained in two languages: a dialect of KIF3.0 (PL), and DAML+OIL. The first one contains a complete code for the library, including theWordNet alignment modules. The second one contains the library (according to available costructs of DAML+OIL) without the WordNet code, since it is very simple and takes much space. DLP+KIF is currently used in some applications that need deep inferences, which can only be provided by expressive, logic-programming-enabled languages. DLP+DAML is currently used in Semantic Web applications, for example in the Core Ontology for Services (COS). The extensions to DOLCE presented in the library are work in progress, and although some of them have been tested in realistic applications, they should be taken cautiously from the viewpoint of rigorous formal ontology.")

```
ObjectProperty(<k:adopted-by>
 inverseOf(k:adopts)
 domain(c:description)
 range(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>)))
ObjectProperty(k:adopts
 inverseOf(<k:adopted-by>)
 domain(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>))
 range(c:description))
ObjectProperty(<k:co-participates-with>
 inverseOf(<k:co-participates-with>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<k:created-by>
 inverseOf(k:creates)
 domain(c:description)
 range(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>)))
ObjectProperty(k:creates
 inverseOf(<k:created-by>)
 domain(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>))
 range(c:description))
ObjectProperty(<k:empowered-for>
 inverseOf(<k:empowered-to>)
 domain(c:task)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<k:empowered-to>
 inverseOf(<k:empowered-for>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:task))
ObjectProperty(<k:exploited-by>
 inverseOf(k:exploits)
 domain(d:endurant)
 range(c:method))
ObjectProperty(k:exploits
 inverseOf(<k:exploited-by>)
 domain(c:method)
 range(d:endurant))
ObjectProperty(<k:has-method>
 inverseOf(<k:method-of>)
 domain(k:activity)
 range(c:method))
ObjectProperty(<k:made-by>
 inverseOf(k:makes)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(k:makes
 inverseOf(<k:made-by>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<k:method-of>
 inverseOf(<k:has-method>)
 domain(c:method)
 range(k:activity))
ObjectProperty(<k:obligation-for>
 inverseOf(<k:obliged-to>)
 domain(c:task)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<k:obliged-to>
 inverseOf(<k:obligation-for>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:task))
ObjectProperty(k:postcondition
 inverseOf(<k:postcondition-of>)
 domain(c:description)
 range(c:situation))
ObjectProperty(<k:postcondition-of>
 inverseOf(k:postcondition)
 domain(c:situation)
 range(c:description))
ObjectProperty(k:precondition
 inverseOf(<k:precondition-of>)
 domain(c:description)
 range(c:situation))
ObjectProperty(<k:precondition-of>
 inverseOf(k:precondition)
 domain(c:situation)
 range(c:description))
ObjectProperty(<k:regulated-by>
 inverseOf(k:regulates)
 domain(c:situation)
 range(c:regulation))
ObjectProperty(k:regulates
 inverseOf(<k:regulated-by>)
 domain(c:regulation)
 range(c:situation))
ObjectProperty(<k:right-task-for>
 inverseOf(<k:right-to>)
 domain(c:task)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<k:right-to>
 inverseOf(<k:right-task-for>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:task))
ObjectProperty(<k:used-by>
 inverseOf(k:uses)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(k:uses
 inverseOf(<k:used-by>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<p:causally-follows>
 inverseOf(<p:causally-precedes>)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(<p:causally-precedes>
 inverseOf(<p:causally-follows>)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(<j:characterized-by>
 inverseOf(j:characterizes)
 domain(j:collection)
 range(c:role))
ObjectProperty(j:characterizes
 inverseOf(<j:characterized-by>)
 domain(c:role)
 range(j:collection))
ObjectProperty(<j:covered-by>
 inverseOf(j:covers)
 domain(j:collection)
 range(c:role))
ObjectProperty(j:covers
 inverseOf(<j:covered-by>)
 domain(c:role)
 range(j:collection))
ObjectProperty(<j:extensionally-equivalent>
 inverseOf(<j:extensionally-equivalent>)
 domain(j:collection)
 range(j:collection))
ObjectProperty(j:member
 inverseOf(<j:member-of>)
 domain(j:collection)
 range(d:endurant))
ObjectProperty(<j:member-of>
 inverseOf(j:member)
 domain(d:endurant)
 range(j:collection))
ObjectProperty(<j:unified-by>
 inverseOf(j:unifies)
 domain(j:collection)
 range(c:description))
ObjectProperty(j:unifies
 inverseOf(<j:unified-by>)
 domain(c:description)
 range(j:collection))
ObjectProperty(a:duration
 inverseOf(<a:duration-of>)
 domain(d:perdurant)
 range(<d:time-interval>))
ObjectProperty(<a:duration-of>
 inverseOf(a:duration)
 domain(<d:time-interval>)
 range(d:perdurant))
ObjectProperty(<a:geographic-part>
 inverseOf(<a:geographic-part-of>)
 domain(<a:political-geographic-object>)
 range(<a:political-geographic-object>))
ObjectProperty(<a:geographic-part-of>
 inverseOf(<a:geographic-part>)
 domain(<a:political-geographic-object>)
 range(<a:political-geographic-object>))
ObjectProperty(<a:happens-at>
 inverseOf(<a:time-of-happening-of>)
 domain(d:perdurant)
 range(<d:time-interval>))
ObjectProperty(<a:time-of-happening-of>
 inverseOf(<a:happens-at>)
 domain(<d:time-interval>)
 range(d:perdurant))
ObjectProperty(a:unit
 inverseOf(<a:unit-of>)
 domain(d:region)
 range(<d:measurement-unit>))
ObjectProperty(<a:unit-of>
 inverseOf(a:unit)
 domain(<d:measurement-unit>)
 range(d:region))
ObjectProperty(<d:abstract-location>
 inverseOf(<d:abstract-location-of>)
 domain(<d:non-physical-endurant>)
 range(<d:abstract-region>))
ObjectProperty(<d:abstract-location-of>
 inverseOf(<d:abstract-location>)
 domain(<d:abstract-region>)
 range(<d:non-physical-endurant>))
ObjectProperty(<d:atomic-part>
 inverseOf(<d:atomic-part-of>)
 domain(d:particular)
 range(d:atom))
ObjectProperty(<d:atomic-part-of>
 inverseOf(<d:atomic-part>)
 domain(d:atom)
 range(d:particular))
ObjectProperty(d:boundary
 inverseOf(<d:boundary-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:boundary-of>
 inverseOf(d:boundary)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:constant-participant>
 inverseOf(<d:constant-participant-in>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<d:constant-participant-in>
 inverseOf(<d:constant-participant>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<d:direct-predecessor>
 inverseOf(<d:direct-successor>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:direct-successor>
 inverseOf(<d:direct-predecessor>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:exact-location>
 inverseOf(<d:exact-location-of>)
 domain(d:particular)
 range(d:region))
ObjectProperty(<d:exact-location-of>
 inverseOf(<d:exact-location>)
 domain(d:region)
 range(d:particular))
ObjectProperty(<d:generic-constituent>
 inverseOf(<d:generic-constituent-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:generic-constituent-of>
 inverseOf(<d:generic-constituent>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:generic-dependent>
 inverseOf(<d:generically-dependent-on>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:generic-location>
 inverseOf(<d:generic-location-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:generic-location-of>
 inverseOf(<d:generic-location>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:generically-dependent-on>
 inverseOf(<d:generic-dependent>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:has-quale>
 inverseOf(<d:quale-of>)
 domain(d:quality)
 range(d:quale))
ObjectProperty(<d:has-quality>
 inverseOf(<d:inherent-in>)
 domain(d:particular)
 range(d:quality))
ObjectProperty(<d:has-t-quality>
 inverseOf(<d:t-inherent-in>)
 domain(d:particular)
 range(d:quality))
ObjectProperty(d:host
 inverseOf(<d:host-of>)
 domain(d:feature)
 range(<d:physical-endurant>))
ObjectProperty(<d:host-of>
 inverseOf(d:host)
 domain(<d:physical-endurant>)
 range(d:feature))
ObjectProperty(<d:identity-c> Transitive
 inverseOf(<d:identity-c>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:identity-n> Transitive
 inverseOf(<d:identity-n>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:immediate-relation>
 inverseOf(<d:immediate-relation-i>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:immediate-relation-i>
 inverseOf(<d:immediate-relation>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:inherent-in>
 inverseOf(<d:has-quality>)
 domain(d:quality)
 range(d:particular))
ObjectProperty(<d:intensionally-referenced-by>
 inverseOf(<d:intensionally-references>)
 domain(d:particular)
 range(<d:non-physical-object>))
ObjectProperty(<d:intensionally-references>
 inverseOf(<d:intensionally-referenced-by>)
 domain(<d:non-physical-object>)
 range(d:particular))
ObjectProperty(d:life Functional
 inverseOf(<d:life-of>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<d:life-of> Functional
 inverseOf(d:life)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<d:mediated-relation>
 inverseOf(<d:mediated-relation-i>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:mediated-relation-i>
 inverseOf(<d:mediated-relation>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:mereologically-coincides>
 inverseOf(<d:mereologically-coincides>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(d:overlaps
 inverseOf(d:overlaps)
 domain(d:particular)
 range(d:particular))
ObjectProperty(d:part
 inverseOf(<d:part-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:part-of>
 inverseOf(d:part)
 domain(d:particular)
 range(d:particular))
ObjectProperty(d:participant
 inverseOf(<d:participant-in>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<d:participant-in>
 inverseOf(d:participant)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<d:partly-compresent>
 inverseOf(<d:partly-compresent>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:physical-location>
 inverseOf(<d:physical-location-of>)
 domain(<d:physical-endurant>)
 range(<d:physical-region>))
ObjectProperty(<d:physical-location-of>
 inverseOf(<d:physical-location>)
 domain(<d:physical-region>)
 range(<d:physical-endurant>))
ObjectProperty(d:predecessor
 inverseOf(d:successor)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:proper-part>
 inverseOf(<d:proper-part-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:proper-part-of>
 inverseOf(<d:proper-part>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:q-location>
 inverseOf(<d:q-location-of>)
 domain(d:quality)
 range(d:region))
ObjectProperty(<d:q-location-of>
 inverseOf(<d:q-location>)
 domain(d:region)
 range(d:quality))
ObjectProperty(<d:q-present-at>
 inverseOf(<d:time-of-q-presence-of>)
 domain(<d:physical-quality>)
 range(<d:time-interval>))
ObjectProperty(<d:quale-of>
 inverseOf(<d:has-quale>)
 domain(d:quale)
 range(d:quality))
ObjectProperty(<d:r-location>
 inverseOf(<d:r-location-of>)
 domain(d:region)
 range(d:region))
ObjectProperty(<d:r-location-of>
 inverseOf(<d:r-location>)
 domain(d:region)
 range(d:region))
ObjectProperty(<d:sibling-part>
 inverseOf(<d:sibling-part>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:spatio-temporal-presence-of>
 inverseOf(<d:spatio-temporally-present-at>)
 domain(<d:spatio-temporal-region>)
 range(d:particular))
ObjectProperty(<d:spatio-temporally-present-at>
 inverseOf(<d:spatio-temporal-presence-of>)
 domain(d:particular)
 range(<d:spatio-temporal-region>))
ObjectProperty(<d:specific-constant-constituent>
 inverseOf(<d:specific-constant-constituent-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:specific-constant-constituent-of>
 inverseOf(<d:specific-constant-constituent>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:specific-constant-dependent>
 inverseOf(<d:specifically-constantly-dependent-on>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:specifically-constantly-dependent-on>
 inverseOf(<d:specific-constant-dependent>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:strong-connection>
 inverseOf(<d:strong-connection>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(d:successor
 inverseOf(d:predecessor)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<d:t-inherent-in>
 inverseOf(<d:has-t-quality>)
 domain(d:quality)
 range(d:particular))
ObjectProperty(<d:temporary-atomic-part>
 inverseOf(<d:temporary-atomic-part-of>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<d:temporary-atomic-part-of>
 inverseOf(<d:temporary-atomic-part>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<d:temporary-part>
 inverseOf(<d:temporary-part-of>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<d:temporary-part-of>
 inverseOf(<d:temporary-part>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<d:temporary-participant>
 inverseOf(<d:temporary-participant-in>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<d:temporary-participant-in>
 inverseOf(<d:temporary-participant>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<d:temporary-proper-part>
 inverseOf(<d:temporary-proper-part-of>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<d:temporary-proper-part-of>
 inverseOf(<d:temporary-proper-part>)
 domain(d:endurant)
 range(d:endurant))
ObjectProperty(<d:time-of-q-presence-of>
 inverseOf(<d:q-present-at>)
 domain(<d:time-interval>)
 range(<d:physical-quality>))
ObjectProperty(<d:total-constant-participant>
 inverseOf(<d:total-constant-participant-in>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<d:total-constant-participant-in>
 inverseOf(<d:total-constant-participant>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<d:total-temporary-participant>
 inverseOf(<d:total-temporary-participant-in>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<d:total-temporary-participant-in>
 inverseOf(<d:total-temporary-participant>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<d:weak-connection>
 inverseOf(<d:weak-connection>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(c:about
 inverseOf(<c:aboutness-of>)
 domain(<c:information-object>)
 range(d:particular))
ObjectProperty(<c:aboutness-of>
 inverseOf(c:about)
 domain(d:particular)
 range(<c:information-object>))
ObjectProperty(<c:acted-by>
 inverseOf(<c:acts-for>)
 domain(c:figure)
 range(unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
ObjectProperty(<c:acts-for>
 inverseOf(<c:acted-by>)
 domain(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))
 range(c:figure))
ObjectProperty(c:admits
 inverseOf(<c:admitted-by>)
 domain(c:description)
 range(d:region))
ObjectProperty(<c:admitted-by>
 inverseOf(c:admits)
 domain(d:region)
 range(c:description))
ObjectProperty(<c:attitude-target-of>
 inverseOf(<c:attitude-towards>)
 domain(c:course)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<c:attitude-towards>
 inverseOf(<c:attitude-target-of>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:course))
ObjectProperty(<c:c-sat>
 inverseOf(<c:c-sat-by>)
 domain(c:situation)
 range(c:description))
ObjectProperty(<c:c-sat-by>
 inverseOf(<c:c-sat>)
 domain(c:description)
 range(c:situation))
ObjectProperty(<c:classified-by>
 inverseOf(c:classifies)
 domain(d:particular)
 range(c:concept))
ObjectProperty(c:classifies
 inverseOf(<c:classified-by>)
 domain(c:concept)
 range(d:particular))
ObjectProperty(c:component
 inverseOf(<c:component-of>)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<c:component-of>
 inverseOf(c:component)
 domain(d:particular)
 range(d:particular))
ObjectProperty(<c:conceived-by>
 inverseOf(c:conceives)
 domain(c:description)
 range(unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
ObjectProperty(c:conceives
 inverseOf(<c:conceived-by>)
 domain(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))
 range(c:description))
ObjectProperty(<c:d-used-by>
 inverseOf(<c:d-uses>)
 domain(unionOf(c:figure c:concept))
 range(c:description))
ObjectProperty(<c:d-uses>
 inverseOf(<c:d-used-by>)
 domain(c:description)
 range(unionOf(c:figure c:concept)))
ObjectProperty(<c:defined-by>
 inverseOf(c:defines)
 domain(unionOf(c:figure c:concept))
 range(c:description))
ObjectProperty(c:defines
 inverseOf(<c:defined-by>)
 domain(c:description)
 range(unionOf(c:figure c:concept)))
ObjectProperty(<c:deputed-by>
 inverseOf(c:deputes)
 domain(c:role)
 range(c:figure))
ObjectProperty(c:deputes
 inverseOf(<c:deputed-by>)
 domain(c:figure)
 range(c:role))
ObjectProperty(<c:expanded-by>
 inverseOf(c:expands)
 domain(c:description)
 range(c:description))
ObjectProperty(c:expands
 inverseOf(<c:expanded-by>)
 domain(c:description)
 range(c:description))
ObjectProperty(<c:expected-by>
 inverseOf(c:expects)
 domain(d:perdurant)
 range(c:description))
ObjectProperty(<c:expected-setting>
 inverseOf(<c:expected-setting-for>)
 domain(unionOf(c:role c:course c:parameter))
 range(c:situation))
ObjectProperty(<c:expected-setting-for>
 inverseOf(<c:expected-setting>)
 domain(c:situation)
 range(unionOf(c:role c:course c:parameter)))
ObjectProperty(c:expects
 inverseOf(<c:expected-by>)
 domain(c:description)
 range(d:perdurant))
ObjectProperty(<c:expressed-by>
 inverseOf(c:expresses)
 domain(<d:non-physical-object>)
 range(<c:information-object>))
ObjectProperty(c:expresses
 inverseOf(<c:expressed-by>)
 domain(<c:information-object>)
 range(<d:non-physical-object>))
ObjectProperty(<c:has-in-scope>
 inverseOf(<c:in-scope-of>)
 domain(c:description)
 range(c:situation))
ObjectProperty(<c:in-scope-of>
 inverseOf(<c:has-in-scope>)
 domain(c:situation)
 range(c:description))
ObjectProperty(<c:interpreted-by>
 inverseOf(c:interprets)
 domain(<c:information-object>)
 range(unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
ObjectProperty(c:interprets
 inverseOf(<c:interpreted-by>)
 domain(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))
 range(<c:information-object>))
ObjectProperty(<c:involved-in>
 inverseOf(c:involves)
 domain(d:endurant)
 range(c:description))
ObjectProperty(c:involves
 inverseOf(<c:involved-in>)
 domain(c:description)
 range(d:endurant))
ObjectProperty(<c:metaphorically-played-by>
 inverseOf(<c:metaphorically-plays>)
 domain(c:role)
 range(d:endurant))
ObjectProperty(<c:metaphorically-plays>
 inverseOf(<c:metaphorically-played-by>)
 domain(d:endurant)
 range(c:role))
ObjectProperty(<c:modal-target>
 inverseOf(<c:modal-target-of>)
```

```
  domain(unionOf(c:role c:figure))
  range(c:course))
 ObjectProperty(<c:modal-target-of>
  inverseOf(<c:modal-target>)
  domain(c:course)
  range(unionOf(c:role c:figure)))
 ObjectProperty(<c:optionally-used-by>
  inverseOf(<c:optionally-uses>)
  domain(c:concept)
  range(c:description))
 ObjectProperty(<c:optionally-uses>
  inverseOf(<c:optionally-used-by>)
  domain(c:description)
  range(c:concept))
 ObjectProperty(<c:p-sat>
  inverseOf(<c:p-sat-by>)
  domain(c:situation)
  range(c:description))
 ObjectProperty(<c:p-sat-by>
  inverseOf(<c:p-sat>)
  domain(c:description)
  range(c:situation))
 ObjectProperty(<c:parametrized-by>
  inverseOf(c:parametrizes)
  domain(d:particular)
  domain(complementOf(c:situation))
  range(c:parameter))
 ObjectProperty(c:parametrizes
  inverseOf(<c:parametrized-by>)
  domain(c:parameter)
  range(complementOf(c:situation))
  range(d:particular))
 ObjectProperty(<c:played-by>
  inverseOf(c:plays)
  domain(c:role)
  range(d:endurant))
 ObjectProperty(c:plays
  inverseOf(<c:played-by>)
  domain(d:endurant)
  range(c:role))
 ObjectProperty(<c:r-sat>
  inverseOf(<c:r-sat-by>)
  domain(c:situation)
  range(c:description))
 ObjectProperty(<c:r-sat-by>
  inverseOf(<c:r-sat>)
  domain(c:description)
  range(c:situation))
 ObjectProperty(<c:realized-by>
  inverseOf(c:realizes)
  domain(<d:non-physical-object>)
  range(unionOf(<d:physical-region> <d:physical-endurant> <d:physical-quality> intersectionOf(c:situation
restriction(<c:setting-for> someValuesFrom(unionOf(<d:physical-region> restriction(d:participant
someValuesFrom(<d:physical-endurant>)) <d:physical-endurant> <d:physical-quality>))))
intersectionOf(restriction(d:participant someValuesFrom(<d:physical-endurant>)) d:perdurant))))
 ObjectProperty(c:realizes
  inverseOf(<c:realized-by>)
  domain(unionOf(<d:physical-region> <d:physical-endurant> <d:physical-quality> intersectionOf(restriction(d:participant
someValuesFrom(<d:physical-endurant>)) d:perdurant) intersectionOf(c:situation restriction(<c:setting-for>
someValuesFrom(unionOf(<d:physical-region> <d:physical-endurant> <d:physical-quality> restriction(d:participant
someValuesFrom(<d:physical-endurant>))))))))
  range(<d:non-physical-object>))
 ObjectProperty(<c:referenced-by>
  inverseOf(c:references)
  domain(d:particular)
  range(<d:non-physical-object>))
 ObjectProperty(c:references
  inverseOf(<c:referenced-by>)
  domain(<d:non-physical-object>)
  range(d:particular))
 ObjectProperty(<c:refined-by>
  inverseOf(c:refines)
  domain(unionOf(c:figure c:concept))
  range(unionOf(c:figure c:concept)))
 ObjectProperty(c:refines
  inverseOf(<c:refined-by>)
  domain(unionOf(c:figure c:concept))
  range(unionOf(c:figure c:concept)))
 ObjectProperty(<c:required-by>
  inverseOf(c:requires)
  domain(<d:social-object>)
  range(<d:social-object>))
 ObjectProperty(c:requires
  inverseOf(<c:required-by>)
  domain(<d:social-object>)
  range(<d:social-object>))
 ObjectProperty(c:requisite
  inverseOf(<c:requisite-for>)
  domain(unionOf(c:role c:course <c:information-object> c:figure))
  range(c:parameter))
 ObjectProperty(<c:requisite-for>
  inverseOf(c:requisite)
  domain(c:parameter)
  range(unionOf(c:role c:course <c:information-object> c:figure)))
 ObjectProperty(<c:satisfied-by>
  inverseOf(c:satisfies)
  domain(c:description)
  range(c:situation))
 ObjectProperty(c:satisfies
  inverseOf(<c:satisfied-by>)
  domain(c:situation)
  range(c:description))
 ObjectProperty(<c:sequenced-by>
  inverseOf(c:sequences)
  domain(d:perdurant)
  range(c:course))
 ObjectProperty(c:sequences
  inverseOf(<c:sequenced-by>)
  domain(c:course)
  range(d:perdurant))
 ObjectProperty(c:setting
  inverseOf(<c:setting-for>)
  domain(complementOf(c:situation))
  domain(d:particular)
  range(c:situation))
 ObjectProperty(<c:setting-for>
  inverseOf(c:setting)
  domain(c:situation)
  range(complementOf(c:situation))
  range(d:particular))
 ObjectProperty(<c:specialized-by>
  inverseOf(c:specializes)
  domain(<d:social-object>)
  range(<d:social-object>))
 ObjectProperty(c:specializes
  inverseOf(<c:specialized-by>)
  domain(<d:social-object>)
  range(<d:social-object>))
 ObjectProperty(<c:temporary-component>
  inverseOf(<c:temporary-component-of>)
  domain(d:endurant)
  range(d:endurant))
 ObjectProperty(<c:temporary-component-of>
  inverseOf(<c:temporary-component>)
  domain(d:endurant)
  range(d:endurant))
```

```
ObjectProperty(<c:value-for>
 inverseOf(<c:valued-by>)
 domain(d:region)
 range(c:parameter))
ObjectProperty(<c:valued-by>
 inverseOf(<c:value-for>)
 domain(c:parameter)
 range(d:region))
ObjectProperty(<m:functional-participant>
 inverseOf(<m:functional-participant-in>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<m:functional-participant-in>
 inverseOf(<m:functional-participant>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<m:generic-target>
 inverseOf(<m:generic-target-of>)
 domain(k:activity)
 range(d:endurant))
ObjectProperty(<m:generic-target-of>
 inverseOf(<m:generic-target>)
 domain(d:endurant)
 range(k:activity))
ObjectProperty(<m:has-state>
 inverseOf(<m:state-of>)
 domain(d:endurant)
 range(d:state))
ObjectProperty(m:instrument
 inverseOf(<m:instrument-of>)
 domain(k:activity)
 range(<d:physical-object>))
ObjectProperty(<m:instrument-of>
 inverseOf(m:instrument)
 domain(<d:physical-object>)
 range(k:activity))
ObjectProperty(m:patient
 inverseOf(<m:patient-of>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<m:patient-of>
 inverseOf(m:patient)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(<m:performed-by>
 inverseOf(m:performs)
 domain(k:action)
 range(unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
ObjectProperty(m:performs
 inverseOf(<m:performed-by>)
 domain(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))
 range(k:action))
ObjectProperty(<m:prescribed-by>
 inverseOf(m:prescribes)
 domain(k:action)
 range(unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
ObjectProperty(m:prescribes
 inverseOf(<m:prescribed-by>)
 domain(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))
 range(k:action))
ObjectProperty(m:product
 inverseOf(<m:product-of>)
 domain(k:activity)
 range(d:endurant))
ObjectProperty(<m:product-of>
 inverseOf(m:product)
 domain(d:endurant)
 range(k:activity))
```

```
ObjectProperty(m:resource
 inverseOf(<m:resource-for>)
 domain(k:activity)
 range(<d:amount-of-matter>))
ObjectProperty(<m:resource-for>
 inverseOf(m:resource)
 domain(<d:amount-of-matter>)
 range(k:activity))
ObjectProperty(m:result
 inverseOf(<m:result-of>)
 domain(k:activity)
 range(d:perdurant))
ObjectProperty(<m:result-of>
 inverseOf(m:result)
 domain(d:perdurant)
 range(k:activity))
ObjectProperty(<m:state-of>
 inverseOf(<m:has-state>)
 domain(d:state)
 range(d:endurant))
ObjectProperty(m:substrate
 inverseOf(<m:substrate-of>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<m:substrate-of>
 inverseOf(m:substrate)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(m:target
 inverseOf(<m:target-of>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<m:target-of>
 inverseOf(m:target)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(m:theme
 inverseOf(<m:theme-of>)
 domain(d:perdurant)
 range(<c:information-object>))
ObjectProperty(<m:theme-of>
 inverseOf(m:theme)
 domain(<c:information-object>)
 range(d:perdurant))
ObjectProperty(<m:use-of>
 inverseOf(<m:used-in>)
 domain(k:action)
 range(d:endurant))
ObjectProperty(<m:used-in>
 inverseOf(<m:use-of>)
 domain(d:endurant)
 range(k:action))
ObjectProperty(<f:ordered-by>
 inverseOf(f:orders)
 domain(<c:information-object>)
 range(<f:information-encoding-system>))
ObjectProperty(f:orders
 inverseOf(<f:ordered-by>)
 domain(<f:information-encoding-system>)
 range(<c:information-object>))
ObjectProperty(<f:q-represented-by>
 inverseOf(<f:q-represents>)
 domain(d:region)
 range(<c:information-object>))
ObjectProperty(<f:q-represents>
 inverseOf(<f:q-represented-by>)
 domain(<c:information-object>)
 range(d:region))
```

```
ObjectProperty(<f:referred-by>
 inverseOf(<f:refers-to>)
 domain(d:particular)
 range(unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
ObjectProperty(<f:refers-to>
 inverseOf(<f:referred-by>)
 domain(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))
 range(d:particular))
ObjectProperty(h:bdi
 inverseOf(<h:bdi-target-of>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:task))
ObjectProperty(<h:bdi-target-of>
 inverseOf(h:bdi)
 domain(c:task)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<h:desire-target-of>
 inverseOf(<h:desire-towards>)
 domain(c:course)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<h:desire-towards>
 inverseOf(<h:desire-target-of>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:course))
ObjectProperty(<h:subject-target-of>
 inverseOf(<h:subjected-to>)
 domain(c:task)
 range(unionOf(<k:agent-driven-role> <i:agentive-figure>)))
ObjectProperty(<h:subjected-to>
 inverseOf(<h:subject-target-of>)
 domain(unionOf(<k:agent-driven-role> <i:agentive-figure>))
 range(c:task))
ObjectProperty(<e:achievable-through>
 domain(e:goal)
 range(c:task))
ObjectProperty(<e:adopts-goal>
 domain(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>))
 range(e:goal))
ObjectProperty(<e:adopts-plan>
 domain(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>))
 range(c:plan))
ObjectProperty(<e:discarded-within>
 inverseOf(e:discards)
 domain(c:task)
 range(c:plan))
ObjectProperty(e:discards
 inverseOf(<e:discarded-within>)
 domain(c:plan)
 range(c:task))
ObjectProperty(<e:disposition-to>
 domain(c:role)
 range(e:goal))
ObjectProperty(<e:exit-condition>
 inverseOf(<e:exit-condition-of>)
 domain(<e:control-task>)
 range(<e:control-task>))
ObjectProperty(<e:exit-condition-of>
 inverseOf(<e:exit-condition>)
 domain(<e:control-task>)
 range(<e:control-task>))
ObjectProperty(<e:influenced-by>
 inverseOf(e:influences)
 domain(e:goal)
 range(e:goal))
ObjectProperty(e:influences
 inverseOf(<e:influenced-by>)
 domain(e:goal)
 range(e:goal))
```

```
ObjectProperty(<e:iteration-interval>
 inverseOf(<e:iteration-interval-of>)
 domain(c:task)
 range(<d:time-interval>))
ObjectProperty(<e:iteration-interval-of>
 inverseOf(<e:iteration-interval>)
 domain(<d:time-interval>)
 range(c:task))
ObjectProperty(<e:main-goal>
 inverseOf(<e:main-goal-of>)
 domain(c:plan)
 range(e:goal))
ObjectProperty(<e:main-goal-of>
 inverseOf(<e:main-goal>)
 domain(e:goal)
 range(c:plan))
ObjectProperty(<e:sibling-task>
 inverseOf(<e:sibling-task>)
 domain(c:task)
 range(c:task))
ObjectProperty(e:subgoal
 inverseOf(<e:subgoal-of>)
 domain(c:plan)
 range(e:goal))
ObjectProperty(<e:subgoal-of>
 inverseOf(e:subgoal)
 domain(e:goal)
 range(c:plan))
ObjectProperty(<e:task-postcondition>
 inverseOf(<e:task-postcondition-of>)
 domain(c:task)
 range(c:situation))
ObjectProperty(<e:task-postcondition-of>
 inverseOf(<e:task-postcondition>)
 domain(c:situation)
 range(c:task))
ObjectProperty(<e:task-precondition>
 inverseOf(<e:task-precondition-of>)
 domain(c:task)
 range(c:situation))
ObjectProperty(<e:task-precondition-of>
 inverseOf(<e:task-precondition>)
 domain(c:situation)
 range(c:task))
ObjectProperty(<i:enforced-by>
 inverseOf(i:enforces)
 domain(c:regulation)
 range(i:institution))
ObjectProperty(i:enforces
 inverseOf(<i:enforced-by>)
 domain(i:institution)
 range(c:regulation))
ObjectProperty(<i:ruled-by>
 inverseOf(i:rules)
 domain(unionOf(c:role c:figure))
 range(<i:socially-constructed-person>))
ObjectProperty(i:rules
 inverseOf(<i:ruled-by>)
 domain(<i:socially-constructed-person>)
 range(unionOf(c:role c:figure)))
ObjectProperty(<n:approximate-location>
 inverseOf(<n:approximate-location-of>)
 domain(intersectionOf(complementOf(d:region) d:particular))
 range(intersectionOf(d:particular complementOf(d:region))))
ObjectProperty(<n:approximate-location-of>
 inverseOf(<n:approximate-location>)
 domain(intersectionOf(d:particular complementOf(d:region)))
 range(intersectionOf(complementOf(d:region) d:particular)))
```

```
ObjectProperty(<n:d-spatial-location>
 inverseOf(<n:d-spatial-location-of>)
 domain(<d:non-physical-endurant>)
 range(<d:space-region>))
ObjectProperty(<n:d-spatial-location-of>
 inverseOf(<n:d-spatial-location>)
 domain(<d:space-region>)
 range(<d:non-physical-endurant>))
ObjectProperty(<n:descriptive-origin>
 inverseOf(<n:descriptive-origin-of>)
 domain(d:endurant)
 range(<d:non-physical-endurant>))
ObjectProperty(<n:descriptive-origin-of>
 inverseOf(<n:descriptive-origin>)
 domain(<d:non-physical-endurant>)
 range(d:endurant))
ObjectProperty(<n:descriptive-place>
 inverseOf(<n:descriptive-place-of>)
 domain(d:endurant)
 range(<d:non-physical-endurant>))
ObjectProperty(<n:descriptive-place-of>
 inverseOf(<n:descriptive-place>)
 domain(<d:non-physical-endurant>)
 range(d:endurant))
ObjectProperty(<n:material-place>
 inverseOf(<n:material-place-of>)
 domain(<d:physical-endurant>)
 range(<d:physical-endurant>))
ObjectProperty(<n:material-place-of>
 inverseOf(<n:material-place>)
 domain(<d:physical-endurant>)
 range(<d:physical-endurant>))
ObjectProperty(n:origin
 inverseOf(<n:origin-of>)
 domain(<d:physical-endurant>)
 range(<d:physical-endurant>))
ObjectProperty(<n:origin-of>
 inverseOf(n:origin)
 domain(<d:physical-endurant>)
 range(<d:physical-endurant>))
ObjectProperty(<n:p-spatial-location>
 inverseOf(<n:p-spatial-location-of>)
 domain(d:perdurant)
 range(<d:space-region>))
ObjectProperty(<n:p-spatial-location-of>
 inverseOf(<n:p-spatial-location>)
 domain(<d:space-region>)
 range(d:perdurant))
ObjectProperty(<n:participant-place>
 inverseOf(<n:participant-place-of>)
 domain(d:perdurant)
 range(d:endurant))
ObjectProperty(<n:participant-place-of>
 inverseOf(<n:participant-place>)
 domain(d:endurant)
 range(d:perdurant))
ObjectProperty(n:place
 inverseOf(<n:place-of>)
 domain(d:endurant)
 range(<d:physical-endurant>))
ObjectProperty(<n:place-of>
 inverseOf(n:place)
 domain(<d:physical-endurant>)
 range(d:endurant))
ObjectProperty(<n:situation-place>
 inverseOf(<n:situation-place-of>)
 domain(c:situation)
 range(d:endurant))
```

```
ObjectProperty(<n:situation-place-of>
 inverseOf(<n:situation-place>)
 domain(d:endurant)
 range(c:situation))
ObjectProperty(<n:spatial-location>
 inverseOf(<n:spatial-location-of>)
 domain(<d:physical-endurant>)
 range(<d:space-region>))
ObjectProperty(<n:spatial-location-of>
 inverseOf(<n:spatial-location>)
 domain(<d:space-region>)
 range(<d:physical-endurant>))
ObjectProperty(<o:functionally-unified-by>
 inverseOf(<o:functionally-unifies>)
 domain(<d:physical-object>)
 range(c:description))
ObjectProperty(<o:functionally-unifies>
 inverseOf(<o:functionally-unified-by>)
 domain(c:description)
 range(<d:physical-object>))
ObjectProperty(<b:concluded-by>
 inverseOf(b:concludes)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(b:concludes
 inverseOf(<b:concluded-by>)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(<b:e-temporal-location>
 inverseOf(<b:e-temporal-location-of>)
 domain(d:endurant)
 range(<d:temporal-region>))
ObjectProperty(<b:e-temporal-location-of>
 inverseOf(<b:e-temporal-location>)
 domain(<d:temporal-region>)
 range(d:endurant))
ObjectProperty(b:follows Transitive
 inverseOf(b:precedes)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(b:meets
 inverseOf(<b:met-by>)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(<b:met-by>
 inverseOf(b:meets)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(b:precedes Transitive
 inverseOf(b:follows)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(<b:present-at>
 inverseOf(<b:time-of-presence-of>)
 domain(d:endurant)
 range(<d:time-interval>))
ObjectProperty(<b:started-by>
 inverseOf(b:starts)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(b:starts
 inverseOf(<b:started-by>)
 domain(d:perdurant)
 range(d:perdurant))
ObjectProperty(<b:temporal-location>
 inverseOf(<b:temporal-location-of>)
 domain(d:perdurant)
 range(<d:temporal-region>))
 ObjectProperty(<b:temporal-location-of>
  inverseOf(<b:temporal-location>)
  domain(<d:temporal-region>)
  range(d:perdurant))
 ObjectProperty(<b:temporal-relation>
  inverseOf(<b:temporal-relation-i>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:temporal-relation-i>
  inverseOf(<b:temporal-relation>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:temporally-coincides>
  inverseOf(<b:temporally-coincides>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:temporally-connected>
  inverseOf(<b:temporally-connected>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:temporally-included-in>
  inverseOf(<b:temporally-includes>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:temporally-includes>
  inverseOf(<b:temporally-included-in>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:temporally-overlaps>
  inverseOf(<b:temporally-overlaps>)
  domain(d:perdurant)
  range(d:perdurant))
 ObjectProperty(<b:time-of-presence-of>
  inverseOf(<b:present-at>)
  domain(<d:time-interval>)
  range(d:endurant))

 DatatypeProperty(<a:counted-by>
  domain(d:region)
  range(xsd:integer))
 DatatypeProperty(<a:has-informal-description>
  domain(d:particular)
  range(xsd:string))
 DatatypeProperty(f:title
  domain(<c:information-object>)
  range(xsd:string))
 DatatypeProperty(<e:iteration-cardinality>
  domain(c:task)
  range(xsd:integer))

 Class(k:action partial
  restriction(<d:generically-dependent-on> someValuesFrom(<k:cognitive-state>))
  d:accomplishment
  restriction(d:participant someValuesFrom(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))))
 Class(k:action partial
  annotation(rdfs:comment "A Perdurant that exemplifies the intentionality of an agent. Could it be aborted, incomplete,
mislead, while remaining a (potential) accomplishment ... The point here is that having a result depends on a method, then an
action remains an action under incomplete results. As a matter of fact, if we neutralize intentionality, a purely topological,
post-hoc view is at odds with the notion of incomplete accomplishments.")
)
 Class(k:activity partial
  restriction(<d:generically-dependent-on> someValuesFrom(c:course))
  k:action
  restriction(<c:sequenced-by> someValuesFrom(c:course)))
 Class(k:activity partial
  annotation(rdfs:comment "In dependency terms, an activity is an action that is generically constantly dependent on a
conventional, shared description (course) adopted by participants. Intuitively, activities are complex actions that are at least
partly conventionally planned.")
```

```
)
 Class(<k:agent-driven-role> complete
  intersectionOf(c:role restriction(<c:played-by> allValuesFrom(unionOf(<d:agentive-social-object> <d:agentive-physical-
object>)))))
 Class(<k:agent-driven-role> partial
  annotation(rdfs:comment "AKA Agentive-role.

A role that can only be played by agents.")
)
 Class(<k:cognitive-agentive-physical-object> complete
  intersectionOf(restriction(c:conceives someValuesFrom(c:plan)) <d:agentive-physical-object>))
 Class(<k:cognitive-agentive-physical-object> partial
  annotation(rdfs:comment "An agentive physical object that is able to have desires and intentions, besides beliefs. In this
ontology, this is encoded as having the ability to conceive plans.")
)
 Class(<k:cognitive-event> partial
  d:event
  restriction(m:substrate someValuesFrom(<d:natural-person>)))
 Class(<k:cognitive-event> partial
  annotation(rdfs:comment "An event occurring in the (embodied) mind.")
)
 Class(<k:cognitive-state> partial
  d:state
  restriction(m:substrate someValuesFrom(<d:natural-person>)))
 Class(<k:cognitive-state> partial
  annotation(rdfs:comment "A state of the (embodied) mind")
)
 Class(k:flux complete
  intersectionOf(d:process restriction(<d:specific-constant-constituent> someValuesFrom(d:accomplishment))))
 Class(k:flux partial
  annotation(rdfs:comment "Fluxes are processes that (also) contain accomplishments as constituents. In other words, fluxes
emerge out of accomplishments.")
)
 Class(k:indicator partial
  c:parameter)
 Class(k:indicator partial
  annotation(rdfs:comment "A parameter valued by regions that are used asindicators for some behaviour or event to be
checked.")
)
 Class(<k:life-cycle> complete
  intersectionOf(c:course restriction(c:sequences allValuesFrom(restriction(<d:life-of> someValuesFrom(d:endurant))))))
 Class(<k:life-cycle> partial
  annotation(rdfs:comment "The course of events typical of the life of an object (kind).")
)
 Class(<k:reconstructed-flux> complete
  intersectionOf(restriction(d:part allValuesFrom(d:accomplishment)) k:flux))
 Class(<k:reconstructed-flux> partial
  annotation(rdfs:comment "Reconstructed fluxes are fluxes that only contain  accomplishments as members.")
)
 Class(k:status partial
  <c:social-role>)
 Class(k:status partial
  annotation(rdfs:comment "A role that involves responsibility, e.g. both duties and rights, in order to perform some task. It
usually involves additional rights and/or powers in contexts (descriptions) different from the one that defines the status.")
)
 Class(j:collection complete
  intersectionOf(<d:social-object> restriction(j:member allValuesFrom(d:endurant)) restriction(<j:covered-by>
someValuesFrom(c:role)) restriction(j:member minCardinality(2)) restriction(j:member someValuesFrom(d:endurant))))
 Class(j:collection partial
  annotation(rdfs:comment "Collections are social objects (either agentive or not), which are not defined by a description, but
they depend both on member entities and on some concepts, figures, and indirectly on descriptions. While we could talk in
general of collections of any kind of entities (events, objects, abstracts, etc.), we restrict here our attention to collections of
endurants, and therefore to their roles (not to concepts whatsoever).")
)
 Class(<j:non-physical-collection> complete
  intersectionOf(restriction(j:member someValuesFrom(<d:non-physical-object>)) j:collection restriction(j:member
allValuesFrom(<d:non-physical-object>)) restriction(j:member minCardinality(2))))
 Class(<j:non-physical-collection> partial
  annotation(rdfs:comment "A collection of non-physical objects that is characterized by a conventional or emergent property,
e.g. a corpus, a legal body, etc.
A non-physical collection only has non-physical endurants as members.")
)
 Class(<j:organized-collection> complete
  intersectionOf(restriction(<j:characterized-by> minCardinality(2)) j:collection restriction(<j:characterized-by>
someValuesFrom(c:role))))
 Class(<j:organized-collection> partial
  annotation(rdfs:comment "Organized collections introduce a different unity criterion for collections. They can be conceived
as characterized by further roles played by some (or all) members of the collection, and related among them through the
social objects (figures, descriptions, collections) that either use or depute or are covered by them.")
)
 Class(<j:parametrized-collection> complete
  intersectionOf(<j:simple-collection> restriction(j:member allValuesFrom(restriction(<d:generic-location>
someValuesFrom(restriction(<c:value-for> someValuesFrom(restriction(<c:requisite-for> someValuesFrom(c:role))))))))))
 Class(<j:parametrized-collection> partial
  annotation(rdfs:comment "A type of simple collections are parametrized collections, whose members must have a quality
constrained by some parameter that is a requisite of their covering role(s).

For example, a crowd of people has members that have spatial positions in a range that makes them proximal (a condition
traditionally used to distinguish so-called aggregates (King 2004)).

On the other hand, if positions are reciprocally relevant (as, for instance, in a living chess setting) according to multiple roles
defined by some plan or design, the collection becomes organized.")
)
 Class(<j:physical-plurality> complete
  intersectionOf(restriction(<d:proper-part> allValuesFrom(restriction(<j:member-of> cardinality(1)))) <d:physical-object>))
 Class(<j:physical-plurality> partial
  annotation(rdfs:comment "a.k.a. unitary collection in D18.

The physical counterpart (realization) of a collection.
A collection (see) is characterized by a conventional or emergent property.
Physical pluralities have as *proper parts* only physical objects that are *members* of a same collection.")
)
 Class(<j:simple-collection> complete
  intersectionOf(restriction(<j:characterized-by> allValuesFrom(complementOf(c:role))) j:collection))
 Class(<j:simple-collection> partial
  annotation(rdfs:comment "A simple collection (for instance, a collection of saxophones, or a mass of lymphocytes ) is a
collection having only covering roles.")
)
 Class(<j:taxonomic-collection> partial
  <j:simple-collection>)
 Class(<j:taxonomic-collection> partial
  annotation(rdfs:comment "A simple collection covered by roles corresponding to natural science properties ascribed to
members.")
)
 Class(<l:biological-collective> partial
  <l:type-based-collective>)
 Class(<l:biological-collective> partial
  annotation(rdfs:comment "Biological collectives are type-based collectives that are *covered* by roles typical of the
biological world.
They can be divided into various kinds (genetic, taxonomic, epidemiological, etc.).
Biological properties produce either crisp or fuzzy/probabilistic types.")
)
 Class(l:collective complete
  intersectionOf(<d:agentive-social-object> j:collection restriction(j:member allValuesFrom(unionOf(<d:agentive-social-
object> <d:agentive-physical-object>)))))
 Class(l:collective partial
  annotation(rdfs:comment "A collection whose members are only agents.")
)
 Class(<l:ecological-collective> partial
  <l:organized-collective>)
 Class(<l:ecological-collective> partial
  annotation(rdfs:comment "An organized collective that receives its organization from the characterizing roles of social
interaztion between organisms in a niche.")
)
 Class(<l:genetic-collective> partial
  <l:biological-collective>)
 Class(<l:genetic-collective> partial
  annotation(rdfs:comment "A biological collective covered by genetic roles (whose members are identified by means of the
genetic properties ascribed to them).")
)
 Class(<l:intentional-collective> partial
  restriction(<j:unified-by> someValuesFrom(intersectionOf(restriction(<c:d-uses>
someValuesFrom(restriction(j:characterizes someValuesFrom(<l:intentional-collective>)))) c:plan)))
  <l:organized-collective>)
 Class(<l:intentional-collective> partial
  annotation(rdfs:comment "We use the presence and structure of a unifying plan in order to characterize kinds of collectives.
A preliminary consideration is that plan unification can have two senses.
The first one only takes into account the action schemas executed by the members, who do not necessarily interact in a
?global? way. In other words, the roles played by members cover the collective, because they are (dispositionally) played by
each member.
The second sense is richer, and assumes that the unifying (maximal) plan (d-)uses roles that characterize (are played by some
members, and related between them in a typical way) the collective.
The first sense of plan unification is applicable to a subclass of simple collectives that we call here 'simple-planned-
collectives'.
The second sense of plan unification applies to intentional collectives proper.

An intentional collective acts intentionally because its members act, and because it is unified by a plan that is conceived by
some cognitive agent. Therefore, there is nothing special in a collective being intentional: it is just a matter of having a plan
and agentive members playing its characterizing roles. What is special is the distinction between the diversified ways of
acting collectively (see subclasses).")
)
 Class(<l:organized-collective> complete
  intersectionOf(l:collective <j:organized-collection>))
 Class(<l:organized-collective> partial
  annotation(rdfs:comment "An organized collection whose only members are agents.")
)
 Class(<l:simple-collective> complete
  intersectionOf(restriction(<j:unified-by> allValuesFrom(complementOf(intersectionOf(c:plan restriction(<c:d-uses>
someValuesFrom(restriction(j:characterizes someValuesFrom(<l:intentional-collective>)))))))) <j:simple-collection>
l:collective))
 Class(<l:simple-collective> partial
  annotation(rdfs:comment "A simple collection whose members are only agents.")
)
 Class(<l:simple-planned-collective> partial
  <l:simple-collective>)
 Class(<l:simple-planned-collective> partial
  annotation(rdfs:comment "We use the presence and structure of a unifying plan in order to characterize kinds of collectives.
A preliminary consideration is that plan unification can have two senses.
The first one only takes into account the action schemas executed by the members, who do not necessarily interact in a
?global? way. In other words, the roles played by members cover the collective, because they are (dispositionally) played by
each member.
The second sense is richer, and assumes that the unifying (maximal) plan (d-)uses roles that characterize the collective.
The first sense of plan unification is applicable to a subclass of simple collectives that we call here 'simple-planned-
collectives'.")
)
 Class(<l:social-type-collective> partial
  <l:type-based-collective>
  restriction(<j:covered-by> someValuesFrom(<c:social-role>))
  restriction(<j:unified-by> someValuesFrom(unionOf(<c:social-relationship> c:practice))))
 Class(<l:social-type-collective> partial
  annotation(rdfs:comment "Social type-based collectives are type-based collectives that are *covered* by roles typical of the
social world.
Social collectives are usually based on action schemas (practices, rather than plans, which are typical of intentional
collectives).
They can be distinguished into neighborhood, geographic (at various granularities), ethnic, linguistic, commercial, industrial,
scientific, political, religious, institutional, administrative, professional, sportive, interest-based, stylistic, devotional, etc.

WordNet contains an impressive set of social-type-based-collectives, which are encoded in the lexicon.")
)
 Class(<l:taxonomic-collective> complete
  intersectionOf(<l:biological-collective> <j:taxonomic-collection>))
 Class(<l:taxonomic-collective> partial
  annotation(rdfs:comment "A simple collective covered by roles corresponding to natural science properties ascribed to
members.")
)
 Class(<l:type-based-collective> partial
  <l:simple-collective>)
 Class(<l:type-based-collective> partial
  annotation(rdfs:comment "Collectives can be classified according to different property kinds. The first one is the type of
members (e.g. physical persons, boys, cows, left-handers, etc.). Types are used in traditional classifications.
For example, biological collectives can be distinguished from social collectives, based on the (biological or social) properties
ascribed to members.")
)
 Class(<a:biological-object> partial
  <a:physical-body>
  restriction(<d:generic-constituent> someValuesFrom(<a:chemical-object>)))
 Class(<a:biological-object> partial
  annotation(rdfs:comment "Any physical body at the biological granularity level. They are (generically) constituted by
chemical objects.")
)
 Class(<a:causal-role> partial
  c:role)
 Class(<a:causal-role> partial
  annotation(rdfs:comment "A role defined (not just used!) by a causal description, and exploited to conceptualize some
causation invariants.
Causal notions are still primitive in this version of DLP.")
)
 Class(<a:chemical-object> partial
  <a:physical-body>)
 Class(<a:chemical-object> partial
  annotation(rdfs:comment "Any physical body at the chemical granularity level.")
)
 Class(<a:collection-role> complete
  intersectionOf(restriction(<c:played-by> allValuesFrom(j:collection)) c:role))
 Class(<a:collection-role> partial
  annotation(rdfs:comment "A role only played by collections.")
)
 Class(<a:commerce-role> partial
  <c:social-role>)
 Class(<a:commerce-role> partial
  annotation(rdfs:comment "A role played by some substance or object within a commercial transaction description.")
)
 Class(a:contract partial
  c:regulation
  restriction(d:part someValuesFrom(h:promise)))
 Class(a:contract partial
  annotation(rdfs:comment "A binding agreement that is possibly enforceable by law.")
)
 Class(a:country partial
  <a:political-geographic-object>
  restriction(<d:generically-dependent-on> someValuesFrom(<a:physical-place>)))
 Class(a:country partial
  annotation(rdfs:comment "A political geographic object that is (generically) dependent on some physical place (in principle,
countries can change their borders).")
)
 Class(<a:creative-object> partial
  <c:information-object>)
 Class(<a:creative-object> partial
  annotation(rdfs:comment "The information realized by an entity for creative purposes. Here mainly for mapping purpose
from WordNet.")
)
 Class(<a:description-role> complete
  intersectionOf(c:role restriction(<c:played-by> allValuesFrom(c:description))))
 Class(<a:description-role> partial
  annotation(rdfs:comment "A role played by descriptions only. Usable for metalinguistic notions, like those that deal with
granular partitions of knowledge, strata of reality, argumentation, etc.")
)
 Class(<a:feature-role> partial
  c:role)
 Class(<a:feature-role> partial
  annotation(rdfs:comment "A role played by some feature of a physical object.")
)
 Class(<a:functional-matter> complete
  intersectionOf(restriction(c:plays someValuesFrom(c:role)) <d:amount-of-matter>))
 Class(<a:functional-matter> partial
  restriction(<m:used-in> someValuesFrom(k:activity)))
 Class(<a:functional-matter> partial
  annotation(rdfs:comment "Amount of matter playing a typically 'functional' role at some time in some situation.")
)
 Class(<a:geographical-object> partial
  <a:physical-place>)
 Class(<a:geographical-object> partial
  annotation(rdfs:comment "A physical place whose spatial quality is q-located in geographical coordinates.")
)
 Class(<a:geographical-place> partial
  restriction(<d:generically-dependent-on> someValuesFrom(<a:geographical-object>))
  <a:non-physical-place>)
 Class(<a:geographical-place> partial
  annotation(rdfs:comment "A non-physical place, generically dependent on some (physical) geographical object.")
)
 Class(<a:legal-possession-entity> partial
  <c:social-role>)
 Class(<a:legal-possession-entity> partial
  annotation(rdfs:comment "A role played by assets involved in a legal possession description.")
)
 Class(<a:locative-role> partial
  restriction(<c:played-by> allValuesFrom(unionOf(<a:non-physical-place> <d:physical-object>)))
  c:role)
 Class(<a:locative-role> partial
  annotation(rdfs:comment "This is a role (e.g. closed area) for places. Locative roles are played by physical objects (in
locational cases, physical places), as well as non-physical places (individual places depending on a physical object).")
)
 Class(<a:logical-role> partial
  <a:description-role>)
 Class(<a:logical-role> partial
  annotation(rdfs:comment "A role used to express logical levels within some layering description or granular partition. A
typical example is the Linnean taxonomic ordering, where Phylum or Species are hierarchical roles.")
)
 Class(<a:material-artifact> partial
  restriction(<d:proper-part> someValuesFrom(restriction(<j:member-of> someValuesFrom(intersectionOf(j:collection
restriction(<j:unified-by> someValuesFrom(unionOf(c:project c:plan))))))))
  <d:non-agentive-physical-object>)
 Class(<a:material-artifact> partial
  annotation(rdfs:comment "No easy definition of artifactual properties is possible, hence it is better to rely on alternative
descriptions and roles: a physical object that shows or is known to have an artifactual origin that counts in the tasks an
ontology is supposed to support, will be a material artifact.
On the other hand, physical objects that do not show that origin, or that origin is unimportant for the task of the ontology, will
be physical bodies.
Formally, a restriction is provided here that requires that the collection whose members are (at least some of the) proper parts
of a material artifact is *unified* by a plan or project.")
)
 Class(<a:non-physical-place> partial
  c:figure
  restriction(<d:generically-dependent-on> someValuesFrom(<a:physical-place>)))
 Class(<a:non-physical-place> partial
  annotation(rdfs:comment "A figure (e.g. Italy) for non-physical (i.e. socially- or cognitively-constructed) places.
Non-physical places generically depend on physical places.")
)
 Class(a:norm partial
  restriction(c:involves someValuesFrom(d:agent))
  c:regulation)
 Class(a:norm partial
  annotation(rdfs:comment "A regulation having an agent obligation as part.")
)
 Class(a:path partial
  c:course
  restriction(c:sequences someValuesFrom(a:phenomenon)))
 Class(a:path partial
  annotation(rdfs:comment "A course used to sequence phenomena (non-intentional processes).")
)
 Class(<a:phase-role> partial
  c:role)
 Class(<a:phase-role> partial
  annotation(rdfs:comment "Formerly: (non-) agentive temporary role.
A role for talking of someone or something at certain phases of own life. It can be used also to map temporal parts of
agentive objects from a 4D ontology.")
)
 Class(a:phenomenon partial
  d:accomplishment)
 Class(a:phenomenon partial
  annotation(rdfs:comment "A phenomenon is basically a process that does not include any intentional active participation.
It can be seen as an accomplishment when some intentionality puts boundaries on it (although it is not claimed to be
inherently  intentional). On the other hand, a purely physical phenomenon does not seem to have inherent boundaries either ...
and also for biological processes as well as economic processes this seems to be disputable. If the boundary hypothesis is
discarded, phenomenon should migrate under process.")
)
 Class(<a:physical-body> partial
  <d:non-agentive-physical-object>
  restriction(<d:proper-part> allValuesFrom(restriction(<j:member-of>
someValuesFrom(intersectionOf(restriction(<j:unified-by> allValuesFrom(complementOf(unionOf(c:project c:plan))))
j:collection))))))
 Class(<a:physical-body> partial
  annotation(rdfs:comment "A physical body is a non-agentive physical object whose primary identity criterion is not given
by its artefactual origin, if any. For example, a rock or a tree can be considered physical bodies unless or until they are not
viewed as artifacts.
As a matter of fact, no easy definition of artifactual properties is possible, hence it is better to rely on alternative descriptions
and roles: a physical object that shows or is known to have an artifactual origin that counts in the tasks an ontology is
supposed to support, will be a material artifact.
On the other hand, physical objects that do not show that origin, or that origin is unimportant for the task of the ontology, will
be physical bodies.
Formally, a restriction is provided here that requires that the collection whose members are proper parts of a physical body is
not *unified* by a plan or project.
BTW, a physical body can still be a *device*, can be 'used' and have 'functions' (roles), e.g. a stone used as a weapon, but it
plays no role like being produced, as material artifacts do. Moreover, a collection whose members are proper parts of a
physical body can still be unified by a description (e.g. a biochemical model).

Physical bodies can have several granularity levels: geological, chemical, physical, biological, etc.")
)
 Class(<a:physical-phenomenon> partial
  a:phenomenon
  restriction(d:participant someValuesFrom(<d:physical-endurant>)))
 Class(<a:physical-phenomenon> partial
  annotation(rdfs:comment "A phenomenon having a physical endurant as participant.")
)
 Class(<a:physical-place> partial
  <d:non-agentive-physical-object>)
 Class(<a:physical-place> partial
  annotation(rdfs:comment "A placeholder for physical objects that are conceived primarily as places, e.g. wrt their spatial
quality.")
)
 Class(<a:political-geographic-object> partial
  <a:geographical-place>)
 Class(<a:political-geographic-object> partial
  annotation(rdfs:comment "A geographical place, conventionally accepted by a community.")
)
 Class(<a:qualitative-role> partial
  c:role)
 Class(<a:qualitative-role> partial
  annotation(rdfs:comment "A placeholder for some roles in common sense that do not easily map to other types of roles.
More work is needed here.")
)
 Class(<a:spatial-feature> partial
  <d:relevant-part>)
 Class(<a:spatial-feature> partial
  annotation(rdfs:comment "A feature related to spatial properties.")
)
```

```
 Class(<a:substance-role> partial
  c:role
  restriction(<c:played-by> allValuesFrom(<d:amount-of-matter>)))
 Class(<a:substance-role> partial
  annotation(rdfs:comment "A role played by some substance.")
)
 Class(d:abstract partial
  restriction(<d:has-quality> allValuesFrom(complementOf(<d:temporal-location_q>)))
  d:particular
  restriction(<d:has-quality> allValuesFrom(complementOf(<d:spatial-location_q>))))
 Class(d:abstract partial
  annotation(rdfs:comment "The main characteristic of abstract entities is that  they do not have spatial nor temporal qualities,
and they are not qualities themselves.  The only class of abstract entities we consider in the present version of the upper
ontology is that of quality regions (or simply regions). Quality spaces are special  kinds of quality regions, being
mereological sums of all the regions related to a certain  quality type. The other examples of abstract entities (sets and facts)
are only  indicative.")
)
 Class(<d:abstract-quality> partial
  restriction(<d:inherent-in> allValuesFrom(<d:non-physical-endurant>))
  restriction(<d:inherent-in> someValuesFrom(<d:non-physical-endurant>))
  restriction(<d:q-location> allValuesFrom(<d:abstract-region>))
  d:quality
  restriction(<d:has-quality> allValuesFrom(<d:abstract-quality>)))
 Class(<d:abstract-quality> partial
  annotation(rdfs:comment "A quality inherent in a non-physical endurant.")
)
 Class(<d:abstract-region> partial
  restriction(d:part allValuesFrom(<d:abstract-region>))
  d:region
  restriction(<d:q-location-of> allValuesFrom(<d:abstract-quality>)))
 Class(<d:abstract-region> partial
  annotation(rdfs:comment "A region at which only abstract qualities can be directly located. It assumes some metrics for
abstract (neither physical nor temporal) properties.")
)
 Class(d:accomplishment partial
  d:event)
 Class(d:accomplishment partial
  annotation(rdfs:comment "Eventive occurrences (events) are called achievements if they are atomic, otherwise they are
accomplishments.
Further developments: being 'achievement', 'accomplishment', 'state', 'event', etc. can be also considered 'aspects' of processes
or of parts of them.
For example, the same process 'rock erosion in the Sinni valley' can be seen as an accomplishment (what has brought the
current state that e.g. we are trying to explain), as an achievement (the erosion process as the result of a previous
accomplishment), as a state (collapsing the time interval of the erosion into a time point), as an event (what has changed our
focus from a state to another).
In the erosion case, we could have good motivations to shift from one aspect to another: a) causation focus, b) effectual
focus, c) condensation d) transition (causality).")
)
 Class(d:achievement partial
  d:event)
 Class(d:achievement partial
  annotation(rdfs:comment "Eventive occurrences (events) are called achievements  if they are atomic, otherwise they are
accomplishments.
Further developments: being 'achievement', 'accomplishment', 'state', 'event', etc. can be also considered 'aspects' of processes
or of parts of them.
For example, the same process 'rock erosion in the Sinni valley' can be seen as an accomplishment (what has brought the
current state that e.g. we are trying to explain), as an achievement (the erosion process as the result of a previous
accomplishment), as a state (collapsing the time interval of the erosion into a time point), as an event (what has changed our
focus from a state to another).
In the erosion case, we could have good motivations to shift from one aspect to another: a) causation focus, b) effectual
focus, c) condensation d) transition (causality).")
)
 Class(d:agent complete
  intersectionOf(d:endurant unionOf(<d:agentive-social-object> <d:agentive-physical-object>)))
 Class(d:agent partial
  annotation(rdfs:comment "A dummy class used to join agentive objects (either physical or social).
Agents are dispositionally so, in the sense that they are able to conceive descriptions and possible actions, but they do not
necessarily act.
```

```
In everyday language, agent is used in this sense, but also to tell that something has acted in a certain way, or to say that something has an initiator or leading role in some action. In DLP, the performs relation encodes these notions.")
)
 Class(<d:agentive-physical-object> complete
  intersectionOf(restriction(c:conceives someValuesFrom(c:description)) d:agent <d:physical-object>))
 Class(<d:agentive-physical-object> partial
  annotation(rdfs:comment "Within Physical objects, a special place have  those to which we ascribe generic intentionality (compatibly to Brentano's distinction, the ability to conceive a description). These are called Agentive,  as opposite to Non-agentive.
In general, we assume that agentive objects are constituted by non-agentive objects: a person is constituted by an organism, a robot is constituted by some machinery, and  so on. Among non-agentive physical objects we have for example houses, body organs,  pieces of wood, etc.
Generic agentivity is defined here in a wide sense as implying conception (to be characterized in a dedicated ? but not developed as yet ? ontology of mind). A conception only requires intentionality in Brentano?s terms (i.e., the ability to represent something to oneself).
See also 'cognitive agentive physical object'.")
)
 Class(<d:agentive-social-object> complete
  intersectionOf(<d:social-object> restriction(c:conceives someValuesFrom(c:description)) d:agent))
 Class(<d:agentive-social-object> partial
  annotation(rdfs:comment "A social object that is assumed to have intentionality (in the wider sense of conceiving some description).
Since a social object is dependent on physical ones, it is not trivial to interpret the local sense in which a social object 'conceives' a description.
For example, an institution can have the belief in the existence of some physical person, but this is possible by means of the powers conferred by some legal system, through its representatives, and that belief has to verified or 'used' by means of the physical agents that 'act for' the institution.
A different sense of social object conceiving descriptions holds for collectives, which ground the overall conception on either a shared, or distributed, or external description conceived by either members of the collective, or by some non-member agent.")
)
 Class(<d:amount-of-matter> partial
  <d:physical-endurant>)
 Class(<d:amount-of-matter> partial
  annotation(rdfs:comment "The common trait of amounts of matter is that they are endurants with no unity (according to Gangemi et a. 2001 none of them is an essential  whole). Amounts of matter - 'stuffs' referred to by mass nouns like 'gold', 'iron', 'wood',  'sand', 'meat', etc. - are mereologically  invariant, in the sense that they change their  identity when they change some parts.")
)
 Class(<d:arbitrary-sum> partial
  d:endurant
  restriction(d:part someValuesFrom(d:endurant))
  restriction(d:part minCardinality(2)))
 Class(<d:arbitrary-sum> partial
  annotation(rdfs:comment "AKA arbitrary-collection.
The mereological sum of any two or more endurants (physical or not). Arbitrary sums have no unity criterion (they are 'extensional').")
)
 Class(d:atom complete
  intersectionOf(d:particular restriction(<d:proper-part> allValuesFrom(complementOf(d:particular)))))
 Class(d:atom partial
  annotation(rdfs:comment "A particular with no proper parts.")
)
 Class(<d:atomic-interval> complete
  intersectionOf(<d:time-interval> restriction(d:part allValuesFrom(complementOf(<d:time-interval>)))))
 Class(<d:atomic-interval> partial
  annotation(rdfs:comment "A time interval with no proper parts (within the clocktick chosen for the time-interval quality space).")
)
 Class(<d:communication-event> partial
  d:accomplishment)
 Class(<d:communication-event> partial
  annotation(rdfs:comment "Here communication is taken in a rather wide sense, being possible as an (intentional) activity as well as a phenomenon.")
)
 Class(<d:dependent-place> partial
  d:feature)
 Class(<d:dependent-place> partial
  annotation(rdfs:comment "A feature that is not part of its host, like a hole in a piece of cheese, the underneath of a table, the front of a house, or the shadow of a tree.")
)
 Class(d:endurant partial
  restriction(<d:participant-in> allValuesFrom(d:perdurant))
  restriction(<d:participant-in> someValuesFrom(d:perdurant))
  restriction(<d:specific-constant-constituent> allValuesFrom(d:endurant))
  d:particular
  restriction(d:part allValuesFrom(d:endurant)))
 Class(d:endurant partial
  annotation(rdfs:comment "The main characteristic of endurants is that all of them are independent essential wholes. This does not mean that the corresponding property (being an endurant) carries proper unity, since there is  no common unity criterion for endurants. Endurants can 'genuinely' change in time,  in the sense that the very same endurant as a whole can have incompatible properties  at different times. To see this, suppose that an endurant say 'this paper' has a  property at a time t 'it's white', and a different, incompatible property at time t'  'it's yellow': in both cases we refer to the whole object, without picking up any  particular part of it. Within endurants, we distinguish between physical and non-physical  endurants, according to whether they have direct spatial qualities. Within physical  endurants, we distinguish between amounts of matter, objects, and features.")
)
 Class(d:event partial
  d:perdurant)
 Class(d:event partial
  annotation(rdfs:comment "An occurrence-type is stative or eventive according  to whether it holds of the mereological sum of two of its instances, i.e. if it is cumulative or not. A sitting occurrence is stative since the sum of two sittings is still a sitting occurrence.

In general, events differ from situations because they are not assumed to have a description from which they depend. They can be sequenced by some course, but they do not require a description as a unifying criterion.
On the other hand, at any time, one can conceive a description that asserts the constraints by which an event of a certian type is such, and in this case, it becomes a situation.
Since the decision of designing an explicit description that unifies a perdurant depends on context, task, interest, application, etc., when aligning an ontology do DLP, there can be indecision on where to align an event-oriented class.
For example, in the WordNet alignment, we have decided to put only some physical events under 'event', e.g. 'discharge', in order to stress the social orientedness of DLP. But whereas we need to talk explicitly of the criteria by which we conceive discharge events, these will be put under 'situation'.
Similar considerations are made for the other types of perdurants in DOLCE.

A different notion of event (dealing with change) is currently investigated for further developments: being 'achievement', 'accomplishment', 'state', 'event', etc. can be also considered 'aspects' of processes or of parts of them.
For example, the same process 'rock erosion in the Sinni valley' can be conceptualized as an accomplishment (what has brought the current state that e.g. we are trying to explain), as an achievement (the erosion process as the result of a previous accomplishment), as a state (if we collapse the time interval of the erosion into a time point), or as an event (what has changed our focus from a state to another).
In the erosion case, we could have good motivations to shift from one aspect to another: a) causation focus, b) effectual focus, c) condensation d) transition (causality).

If we want to consider all the aspects of a process together, we need to postulate a unifying descriptive set of criteria (i.e. a 'description'), according to which that process is circumstantiated in a 'situation'. The different aspects will arise as a parts of a same situation.")
)
 Class(d:feature partial
  <d:physical-endurant>
  restriction(d:host someValuesFrom(<d:physical-endurant>)))
 Class(d:feature partial
  annotation(rdfs:comment "Features are 'parasitic entities', that exist insofar their host exists. Typical examples of features are holes, bumps, boundaries, or spots of color. Features may be relevant parts of their host, like a bump or an edge, or dependent regions like a hole in a piece of cheese, the underneath of a table, the front of a house, or the shadow of a tree, which are not parts of their host. All features are essential wholes, but no common unity criterion may exist for all of them. However, typical features have a topological unity, as they are singular entities.

Here only features of physical endurants are considered.")
)
 Class(<d:measurement-unit> partial
  <d:abstract-region>)
 Class(<d:measurement-unit> partial
  annotation(rdfs:comment "A quality space used as a reference metrics (\"measurement space\") for other spaces. It is usually
\"counted by\" some number.")
)
 Class(<d:mental-object> complete
  intersectionOf(<d:non-physical-object> restriction(<d:specifically-constantly-dependent-on>
someValuesFrom(<d:agentive-physical-object>))))
 Class(<d:mental-object> partial
  annotation(rdfs:comment "AKA \"internal description\". Mental objects are dependent on an intentional agent. This class is
just a pointer to a complex ontology of mental entities that is currently under development.")
)
 Class(<d:natural-person> partial
  <d:agentive-physical-object>)
 Class(<d:natural-person> partial
  annotation(rdfs:comment "An agentive physical object, capable of 'acting for' a social individual.
Socially-constructed persons, like legal ones, can be acted by natural persons, but are never identical to them, since social and
physical objects are disjoint in DOLCE. As a consequence e.g. someone after death can no longer be a natural person, but in
some legal systems, the legal counterpart (a socially-constructed person) can still exist for some legal contexts, e.g. for
hereditary issues.")
)
 Class(<d:non-agentive-physical-object> partial
  restriction(c:conceives cardinality(0))
  <d:physical-object>)
 Class(<d:non-agentive-physical-object> partial
  annotation(rdfs:comment "Within Physical objects, a special place have those  those to which we ascribe intentions, beliefs,
and desires. These are called Agentive,  as opposite to Non-agentive. Intentionality is understood here as the capability of
heading for/dealing with objects or states of the world. This is an important area  of ontological investigation we haven't
properly explored yet, so our suggestions are  really very preliminary. A possible modelling of case roles has been started
within the descriptions plugin  that could be embedded within basic DOLCE. In general, we assume that agentive objects are
constituted by non-agentive objects: a  person is constituted by an organism, a robot is constituted by some machinery, and so
on.  Among non-agentive physical objects we have for example houses, body organs, pieces of wood,  etc.")
)
 Class(<d:non-agentive-social-object> partial
  <d:social-object>
  restriction(c:conceives cardinality(0)))
 Class(<d:non-agentive-social-object> partial
  annotation(rdfs:comment "A social object that is not assumed to have intentionality (in the wider sense of conceiving some
description). Since a social object is dependent on physical ones, it is not trivial to interpret the local sense in which a social
object 'conceives' a description. See 'agentive-social-object' for some discussion.")
)
 Class(<d:non-physical-endurant> partial
  d:endurant
  restriction(d:part allValuesFrom(<d:non-physical-endurant>))
  restriction(<d:has-quality> allValuesFrom(<d:abstract-quality>)))
 Class(<d:non-physical-endurant> partial
  annotation(rdfs:comment "An endurant with no mass, generically constantly depending on some intentional agent.
Non-physical endurants can have physical constituents (e.g. in the case of members of a collection).")
)
 Class(<d:non-physical-object> partial
  restriction(<d:generically-dependent-on> someValuesFrom(<d:physical-endurant>))
  restriction(d:part allValuesFrom(<d:non-physical-object>))
  <d:non-physical-endurant>)
 Class(<d:non-physical-object> partial
  annotation(rdfs:comment "Formerly known as description.
A unitary endurant with no mass (non-physical), generically constantly depending on some intentional agent, on some
communication act, and indirectly on some agent participating in that act.
Either descriptions (in the current sense), and concepts are non-physical objects.")
)
 Class(d:particular partial
  annotation(rdfs:comment "AKA 'entity'.
Any individual in the DOLCE domain of discourse. The extensional coverage of DOLCE is as large as possible, since it
ranges on 'possibilia', i.e all possible individuals that can be postulated by means of DOLCE axioms. Possibilia include
physical objects, substances, processes, qualities, conceptual regions, non-physical objects, collections and even arbitrary
sums of objects. Extensions of DOLCE included in this ontology also feature 'situations' (qualified reifications of states of
affairs).")
)
 Class(d:perdurant partial
  restriction(<d:specific-constant-constituent> allValuesFrom(d:perdurant))
  restriction(<d:has-quality> someValuesFrom(<d:temporal-location_q>))
  restriction(d:part allValuesFrom(d:perdurant))
  restriction(d:participant someValuesFrom(d:endurant))
  restriction(d:participant allValuesFrom(d:endurant))
  restriction(<d:has-quality> allValuesFrom(<d:temporal-quality>))
  d:particular)
 Class(d:perdurant partial
  annotation(rdfs:comment "Perdurants (AKA occurrences) comprise what are  variously called events, processes,
phenomena, activities and states. They can have  temporal parts or spatial parts. For instance, the first movement of (an
execution of)  a symphony is a temporal part of it. On the other side, the play performed by the left  side of the orchestra is a
spatial part. In both cases, these parts are occurrences  themselves. We assume that objects cannot be parts of occurrences,
but rather they  participate in them. Perdurants extend in time by accumulating different temporal parts,  so that, at any time
they are present, they are only partially present, in the sense that  some of their proper temporal parts (e.g., their previous or
future phases) may be not  present. E.g., the piece of paper you are reading now is wholly present, while some temporal  parts
of your reading are not present any more. Philosophers say that endurants are  entities that are in time, while lacking however
temporal parts (so to speak, all their  parts flow with them in time). Perdurants, on the other hand, are entities that happen  in
time, and can have temporal parts (all their parts are fixed in time).")
)
 Class(<d:physical-endurant> partial
  restriction(d:part allValuesFrom(<d:physical-endurant>))
  restriction(<d:has-quality> someValuesFrom(<d:physical-quality>))
  d:endurant
  restriction(<d:has-quality> someValuesFrom(<d:spatial-location_q>))
  restriction(<d:specific-constant-constituent> allValuesFrom(<d:physical-endurant>))
  restriction(<d:has-quality> allValuesFrom(<d:physical-quality>)))
 Class(<d:physical-endurant> partial
  annotation(rdfs:comment "An endurant having a direct physical (at least spatial) quality.")
)
 Class(<d:physical-object> partial
  <d:physical-endurant>)
 Class(<d:physical-object> partial
  annotation(rdfs:comment "The main characteristic of physical objects is that  they are endurants with unity. However, they
have no common unity criterion, since  different subtypes of objects may  have different unity criteria. Differently from
aggregates, (most) physical objects change some of their parts while keeping their  identity, they can have therefore
temporary parts. Often physical objects (indeed,  all endurants) are ontologically independent from occurrences (discussed
below).  However, if we admit that every object has a life, it is hard to exclude a mutual  specific constant dependence
between the two. Nevertheless, we may still use the  notion of dependence to (weakly) characterize objects as being not
specifically  constantly dependent on other objects.")
)
 Class(<d:physical-quality> partial
  restriction(<d:q-location> allValuesFrom(<d:physical-region>))
  restriction(<d:inherent-in> allValuesFrom(<d:physical-endurant>))
  restriction(<d:has-quality> allValuesFrom(<d:physical-quality>))
  d:quality
  restriction(<d:inherent-in> someValuesFrom(<d:physical-endurant>)))
 Class(<d:physical-quality> partial
  annotation(rdfs:comment "A quality inherent in a physical endurant.")
)
 Class(<d:physical-region> partial
  restriction(d:part allValuesFrom(<d:physical-region>))
  d:region
  restriction(<d:q-location-of> allValuesFrom(<d:physical-quality>)))
 Class(<d:physical-region> partial
  annotation(rdfs:comment "A region at which only physical qualities can be  directly located. It assumes some metrics for
physical properties.")
)
 Class(d:process partial
  d:stative)
 Class(d:process partial
  annotation(rdfs:comment "Within stative occurrences, we distinguish between  states and processes according to
homeomericity: sitting is classified as a state  but running is classified as a process, since there are (very short) temporal parts
of a running that are not themselves runnings.
```

In general, processes differ from situations because they are not assumed to have a description from which they depend. They can be sequenced by some course, but they do not require a description as a unifying criterion.
On the other hand, at any time, one can conceive a description that asserts the constraints by which a process of a certian type is such, and in this case, it becomes a situation.
Since the decision of designing an explicit description that unifies a perdurant depends on context, task, interest, application, etc., when aligning an ontology do DLP, there can be indecision on where to align a process-oriented class.

For example, in the WordNet alignment, we have decided to put only some physical processes under 'process', e.g. 'organic process', in order to stress the social orientedness of DLP. But whereas we need to talk explicitly of the criteria by which we conceive organic processes, these will be put under 'situation'.
Similar considerations are made for the other types of perdurants in DOLCE.

A different notion of event (dealing with change) is currently investigated for further developments: being 'achievement', 'accomplishment', 'state', 'event', etc. can be also considered 'aspects' of processes or of parts of them.
For example, the same process 'rock erosion in the Sinni valley' can be conceptualized as an accomplishment (what has brought the current state that e.g. we are trying to explain), as an achievement (the erosion process as the result of a previous accomplishment), as a state (if we collapse the time interval of the erosion into a time point), or as an event (what has changed our focus from a state to another).
In the erosion case, we could have good motivations to shift from one aspect to another: a) causation focus, b) effectual focus, c) condensation d) transition (causality).

If we want to consider all the aspects of a process together, we need to postulate a unifying descriptive set of criteria (i.e. a 'description'), according to which that process is circumstantiated in a 'situation'. The different aspects will arise as a parts of a same situation.")
)
 Class(d:proposition partial
  d:abstract)
 Class(d:proposition partial
  annotation(rdfs:comment "The abstract content of a proposition. Abstract content is purely combinatorial: from this viewpoint, any content that can be generated by means of combinatorial rules is assumed to exist in the domain of quantification (reified abstracts).")
)
 Class(d:quale complete
  intersectionOf(d:region d:atom restriction(<d:proper-part> allValuesFrom(complementOf(d:particular)))))
 Class(d:quale partial
  annotation(rdfs:comment "An atomic region.")
)
 Class(d:quality partial
  restriction(<d:inherent-in> someValuesFrom(d:particular))
  restriction(<d:has-quality> allValuesFrom(d:quality))
  d:particular
  restriction(<d:q-location> allValuesFrom(d:region)))
 Class(d:quality partial
  annotation(rdfs:comment "Qualities can be seen as the basic entities we can perceive or measure: shapes, colors, sizes, sounds, smells, as well as weights, lengths, electrical charges... 'Quality' is often used as a synonymous of 'property', but this is not the case in this upper ontology: qualities are particulars, properties are universals. Qualities inhere to entities: every entity (including qualities themselves) comes with certain qualities, which exist as long as the entity exists.")
)
 Class(<d:quality-space> complete
  intersectionOf(d:region restriction(d:overlaps allValuesFrom(complementOf(<d:quality-space>)))))
 Class(<d:quality-space> partial
  annotation(rdfs:comment "A quality space is a topologically maximal region. The constraint of maximality cannot be given completely in OWL, but a constraint is given that creates a partition out of all quality spaces (e.g. no two quality spaces can overlap mereologically).")
)
 Class(d:region partial
  d:abstract
  restriction(<d:q-location-of> allValuesFrom(d:quality))
  restriction(d:part allValuesFrom(d:region)))
 Class(d:region partial
  annotation(rdfs:comment "We distinguish between a quality (e.g., the color of a specific rose), and its value (e.g., a particular shade of red). The latter is called quale, and describes the position of an individual quality within a certain conceptual space (called here quality space) Gardenfors (2000). So when we say that two roses have (exactly) the same color, we mean that their color qualities, which are distinct, have the same position in the color space, that is they have the same color quale.")
)
 Class(<d:relevant-part> partial
  d:feature)
 Class(<d:relevant-part> partial
  annotation(rdfs:comment "Features that are relevant parts of their host, like a bump or an edge.")
)
 Class(d:set partial
  d:abstract)
 Class(d:set partial
  annotation(rdfs:comment "A mathematical set.")

```
)
 Class(<d:social-object> partial
  restriction(<d:generically-dependent-on> someValuesFrom(<d:agentive-physical-object>))
  restriction(<d:generically-dependent-on> someValuesFrom(<d:communication-event>))
  <d:non-physical-object>)
 Class(<d:social-object> partial
  annotation(rdfs:comment "A catch-all class for entities from the social world. It includes agentive and non-agentive
socially-constructed objects: descriptions, concepts, figures, collections, information objects.
It could be equivalent to 'non-physical object', but we leave open the possibility of 'private' non-physical objects.")
)
 Class(<d:space-region> partial
  <d:physical-region>
  restriction(<d:q-location-of> allValuesFrom(<d:spatial-location_q>))
  restriction(d:part allValuesFrom(<d:space-region>)))
 Class(<d:space-region> partial
  annotation(rdfs:comment "An ordinary space: geographical, cosmological, anatomical, topographic, etc.")
)
 Class(<d:spatial-location_q> partial
  <d:physical-quality>)
 Class(<d:spatial-location_q> partial
  annotation(rdfs:comment "A physical quality, q-located in (whose value is given within) ordinary spaces (geographical
coordinates, cosmological positions, anatomical axes, etc.).")
)
 Class(<d:spatio-temporal-region> partial
  <d:space-region>)
 Class(<d:spatio-temporal-region> partial
  annotation(rdfs:comment "Any region resulting from the composition of a space region with a temporal region, i.e. being
present in region r at time t.")
)
 Class(d:state partial
  d:stative)
 Class(d:state partial
  annotation(rdfs:comment "Within stative occurrences, we distinguish between  states and processes according to
homeomericity: sitting is classified as a state  but running is classified as a process, since there are (very short) temporal parts
of a running that are not themselves runnings.

In general, states differ from situations because they are not assumed to have a description from which they depend. They can
be sequenced by some course, but they do not require a description as a unifying criterion.
On the other hand, at any time, one can conceive a description that asserts the constraints by which a state of a certian type is
such, and in this case, it becomes a situation.
Since the decision of designing an explicit description that unifies a perdurant depends on context, task, interest, application,
etc., when aligning an ontology do DLP, there can be indecision on where to align a state-oriented class.
For example, in the WordNet alignment, we have decided to put only some physical states under 'state', e.g. 'turgor', in order
to stress the social orientedness of DLP. But whereas we need to talk explicitly of the criteria by which we conceive turgor
states, these will be put under 'situation'.
Similar considerations are made for the other types of perdurants in DOLCE.

A different notion of event (dealing with change) is currently investigated for further developments: being 'achievement',
'accomplishment', 'state', 'event', etc. can be also considered 'aspects' of processes or of parts of them.
For example, the same process 'rock erosion in the Sinni valley' can be conceptualized as an accomplishment (what has
brought the current state that e.g. we are trying to explain), as an achievement (the erosion process as the result of a previous
accomplishment), as a state (if we collapse the time interval of the erosion into a time point), or as an event (what has
changed our focus from a state to another).
In the erosion case, we could have good motivations to shift from one aspect to another: a) causation focus, b) effectual
focus, c) condensation d) transition (causality).

If we want to consider all the aspects of a process together, we need to postulate a unifying descriptive set of criteria (i.e. a
'description'), according to which that process is circumstantiated in a 'situation'. The different aspects will arise as a parts of
a same situation.")
)
 Class(d:stative partial
  d:perdurant)
 Class(d:stative partial
  annotation(rdfs:comment "An occurrence-type is stative or eventive according  to whether it holds of the mereological sum
of two of its instances, i.e. if it is  cumulative or not. A sitting occurrence is stative since the sum of two sittings  is still a
sitting occurrence.")
)
```

```
 Class(<d:temporal-location_q> partial
  <d:temporal-quality>)
 Class(<d:temporal-location_q> partial
  annotation(rdfs:comment "A temporal location quality.")
)
 Class(<d:temporal-quality> partial
  restriction(<d:q-location> allValuesFrom(<d:temporal-region>))
  restriction(<d:inherent-in> someValuesFrom(d:perdurant))
  restriction(<d:inherent-in> allValuesFrom(d:perdurant))
  d:quality
  restriction(<d:has-quality> allValuesFrom(<d:temporal-quality>)))
 Class(<d:temporal-quality> partial
  annotation(rdfs:comment "A quality inherent in a perdurant.")
)
 Class(<d:temporal-region> partial
  d:region
  restriction(<d:q-location-of> allValuesFrom(<d:temporal-quality>))
  restriction(d:part allValuesFrom(<d:temporal-region>)))
 Class(<d:temporal-region> partial
  annotation(rdfs:comment "A region at which only temporal qualities can be  directly located. It assumes a metrics for
time.")
)
 Class(<d:temporary-atom> complete
  intersectionOf(d:endurant restriction(<d:temporary-proper-part> cardinality(0))))
 Class(<d:temporary-atom> partial
  annotation(rdfs:comment "An endurant that is an atom at time t.")
)
 Class(<d:time-interval> partial
  <d:temporal-region>)
 Class(<d:time-interval> partial
  annotation(rdfs:comment "A temporal region, measured according to a calendar.")
)
 Class(c:concept partial
  <d:non-agentive-social-object>
  restriction(<c:defined-by> someValuesFrom(c:description))
  restriction(<c:refined-by> allValuesFrom(c:concept)))
 Class(c:concept partial
  annotation(rdfs:comment "AKA C-Description.
A non-physical object that is defined by a description s, and whose function is classifying entities from a ground ontology in
order to build situations that can satisfy s.")
)
 Class(<c:constitutive-description> partial
  c:description
  restriction(c:defines someValuesFrom(c:figure)))
 Class(<c:constitutive-description> partial
  annotation(rdfs:comment "A description whose main purpose is defining a figure.")
)
 Class(c:course partial
  restriction(d:part allValuesFrom(c:course))
  restriction(<c:defined-by> someValuesFrom(c:description))
  restriction(c:sequences allValuesFrom(d:perdurant))
  c:concept
  restriction(<c:modal-target-of> allValuesFrom(unionOf(c:role c:figure))))
 Class(c:course partial
  annotation(rdfs:comment "A concept that selects (in particular, it 'sequences') perdurants (processes, events, or states), as a
component of some description.
Courses are the descriptive counterpart of perdurants, and, as perdurants have endurants as participants, they are usually the
function of some role.")
)
 Class(c:description partial
  <d:non-agentive-social-object>
  restriction(<c:expressed-by> someValuesFrom(<c:information-object>))
  restriction(<c:d-uses> someValuesFrom(unionOf(c:figure c:concept)))
  restriction(c:defines allValuesFrom(unionOf(c:figure c:concept)))
  restriction(<c:conceived-by> someValuesFrom(<d:agentive-physical-object>)))
 Class(c:description partial
  annotation(rdfs:comment "A description is a non-physical object, which represents a conceptualization (as a mental object
or state), hence generically dependent on some agent, and which is also social, i.e. communicable.
```

```
Descriptions define or use concepts or figures, and can be satisfied by situations.
The typology of descriptions is still preliminary.")
)
 Class(c:figure partial
  <d:social-object>
  restriction(<c:refined-by> allValuesFrom(c:figure))
  restriction(<c:defined-by> someValuesFrom(<c:constitutive-description>)))
 Class(c:figure partial
  annotation(rdfs:comment "a.k.a. 'social individual'.
Figures are social objects defined or used by descriptions, but differently from concepts, they do not classify entities.
Examples of figures are organisations, political geographic objects, sacred symbols, etc.")
)
 Class(c:gestalt partial
  c:theory)
 Class(c:gestalt partial
  annotation(rdfs:comment "A perceptual structure, from the descriptive viewpoint. In other words, this encodes the
conditions by which a configuration, structure, or arrangement is perceived by a cognitive agent.")
)
 Class(<c:information-object> complete
  intersectionOf(<d:social-object> restriction(<c:interpreted-by> allValuesFrom(unionOf(<d:agentive-social-object>
<d:agentive-physical-object>))) restriction(<c:realized-by> someValuesFrom(<c:physical-realization>)) restriction(c:about
allValuesFrom(d:particular)) restriction(c:expresses allValuesFrom(c:description))))
 Class(<c:information-object> partial
  restriction(<f:ordered-by> someValuesFrom(<f:information-encoding-system>))
  restriction(d:part allValuesFrom(<c:information-object>)))
 Class(<c:information-object> partial
  annotation(rdfs:comment "Information objects are social objects. They are realized by some entity. They are ordered
(expressed according to) by some system for information encoding. Consequently, they are dependent from an encoding as
well as from a concrete realization.
They can express a description (the ontological equivalent of a meaning/conceptualization), can be about any entity, and can
be interpreted by an agent.
From a communication perspective, an information object can play the role of \"message\". From a semiotic perspective, it
playes the role of \"expression\".")
)
 Class(c:method partial
  c:description)
 Class(c:method partial
  annotation(rdfs:comment "A description that contains a specification to do, realize, behave, etc. Subclasses are plan,
technique, practice, project, etc.")
)
 Class(<c:modal-description> partial
  restriction(<d:temporary-part-of> someValuesFrom(c:description))
  restriction(<c:d-uses> someValuesFrom(c:course))
  restriction(<c:d-uses> someValuesFrom(intersectionOf(c:role restriction(<c:attitude-towards>
someValuesFrom(c:course)))))
  c:description)
 Class(<c:modal-description> partial
  annotation(rdfs:comment "A modal description is any part of a description that has a unity criterion consisting in the
specification of a modal target (some course), and it can be a right, power, duty, etc.
Notice that modal descriptions can appear in conventionalized descriptions as well as in idiosyncratic assessements,
narratives, promises, etc.")
)
 Class(c:parameter partial
  restriction(<c:valued-by> allValuesFrom(d:region))
  restriction(<c:valued-by> someValuesFrom(d:region))
  restriction(<c:requisite-for> allValuesFrom(unionOf(c:role c:course c:figure)))
  restriction(<c:defined-by> someValuesFrom(c:description))
  c:concept)
 Class(c:parameter partial
  annotation(rdfs:comment "A concept that classifies (in particular, it is 'valued by') regions, as defined by some description.
Parameters are the descriptive counterpart of regions, and, as regions represent the qualities of perdurants or endurants, they
can be requisites for some role or course.
A parameter has at least one region that is a value for it.")
)
 Class(<c:physical-realization> complete
  intersectionOf(unionOf(intersectionOf(c:situation restriction(<c:setting-for> someValuesFrom(unionOf(<d:physical-
region> <d:physical-endurant> <d:physical-quality> restriction(d:participant someValuesFrom(<d:physical-endurant>))))))
<d:physical-region> <d:physical-endurant> <d:physical-quality> intersectionOf(restriction(d:participant
someValuesFrom(<d:physical-endurant>)) d:perdurant)) d:particular restriction(c:realizes someValuesFrom(<d:non-
physical-object>))))
 Class(<c:physical-realization> partial
  annotation(rdfs:comment "Any physical particular that realizes a non-physical endurant. Such physical particulars can be
either physical endurants, physical qualities, physical regions, perdurants with at least one physical participant, or a situation
with one physical entity in its setting.
Ultimately, a physical realization depends on at least one physical endurant (each of the others physical entity types depend
on a physical endurant to be considered as such).")
)
 Class(c:plan partial
  restriction(<c:d-uses> someValuesFrom(intersectionOf(c:role restriction(<c:played-by>
allValuesFrom(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))))))
  c:method
  restriction(<d:proper-part> someValuesFrom(e:goal))
  restriction(<c:d-uses> someValuesFrom(c:task)))
 Class(c:plan partial
  annotation(rdfs:comment "A plan is a method for executing or  performing a procedure or a stage of a procedure.
A plan must use both at least one role played by an agent, and at least one task.
Finally, a plan has a goal as proper part, and can also have regulations and other descriptions as proper parts.")
)
 Class(c:practice partial
  restriction(<c:conceived-by> someValuesFrom(<d:agentive-social-object>))
  c:method)
 Class(c:practice partial
  annotation(rdfs:comment "A social method carried out explicitly or by tradition, spontaneously emerged, or moderately or
strongly regulated.")
)
 Class(c:project partial
  c:method)
 Class(c:project partial
  annotation(rdfs:comment "A project is a proactively satisfied method. Differently from a plan, a project includes at least one
'product' role to be played by some endurant (e.g. a house), or one 'result' role played by a perdurant with a definite
participant (e.g. a restored state of a house).")
)
 Class(c:regulation partial
  <c:social-description>)
 Class(c:regulation partial
  annotation(rdfs:comment "A description usually requiring a C-SAT satisfaction for a situation. Norms, codes of practice,
etc. are examples.")
)
 Class(c:relation partial
  c:theory)
 Class(c:relation partial
  annotation(rdfs:comment "A non-social relation(ship): formal, linguistic, etc. It is considered here a theory, because
relations are established in order to give an ordering to some reality.")
)
 Class(c:role partial
  restriction(c:requisite allValuesFrom(c:parameter))
  restriction(<c:modal-target> allValuesFrom(c:course))
  restriction(<c:defined-by> someValuesFrom(c:description))
  restriction(<c:played-by> allValuesFrom(d:endurant))
  c:concept)
 Class(c:role partial
  annotation(rdfs:comment "Also known as 'functional role'.
A concept that classifies (in particular, it is  'played by') endurants, as used in some description.  Roles are the descriptive
counterpart of endurants, and, as endurants participate in perdurants, they usually have courses as modal targets (see).
The typology of roles is still preliminary.")
)
 Class(c:situation complete
  intersectionOf(<d:non-agentive-social-object> restriction(<c:setting-for>
someValuesFrom(intersectionOf(complementOf(c:situation) d:particular))) restriction(c:satisfies
someValuesFrom(c:description))))
 Class(c:situation partial
  restriction(d:part allValuesFrom(c:situation)))
 Class(c:situation partial
  annotation(rdfs:comment "A situation is a social object that appears in the domain of an ontology only because there is a
description whose components can ?carve up? a view (setting) on that domain. A situation has to satisfy a description (see
below for ways of defining the satisfies relation), and it has to be setting for at least one entity.
```

In other words, it is the ontological counterpart (with due local differences or restrictions) of settings (situations from SC, contexts, episodes, states of affairs, structures, configurations, cases, etc.).
A perdurant is usually the only mandatory constituent of a setting.
Two descriptions of a same situation are possible, otherwise we would result in a solipsistic ontology.
The time and space (and possibly other qualities) of a situation are the time and space of the perdurants in the setting.")
)
Class(<c:social-description> complete
intersectionOf(restriction(<d:generically-dependent-on> someValuesFrom(l:collective)) c:description))
Class(<c:social-description> partial
annotation(rdfs:comment "Examples of Social Descriptions are laws, norms, shares, peace treaties, etc., which are generically dependent on societies.
Social descriptions are dependent on a community of agents.")
)
Class(<c:social-relationship> partial
restriction(<c:d-uses> someValuesFrom(intersectionOf(c:role restriction(<c:played-by> allValuesFrom(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-object>))))))
<c:social-description>)
Class(<c:social-relationship> partial
annotation(rdfs:comment "A social description defining roles for the interaction of cognitive agents.")
)
Class(<c:social-role> partial
c:role)
Class(<c:social-role> partial
annotation(rdfs:comment "A role created and maintained by a society.")
)
Class(<c:symmetric-role> partial
c:role)
Class(<c:symmetric-role> partial
annotation(rdfs:comment "A role played by each of two entities at the same time and with the same parameters: e.g. equivalent, neighbor, father.")
)
Class(c:task complete
intersectionOf(restriction(<h:desire-target-of> someValuesFrom(unionOf(<k:agent-driven-role> <i:agentive-figure>))) c:course restriction(<c:defined-by> someValuesFrom(c:method))))
Class(c:task partial
annotation(rdfs:comment "A course used to sequence activities or other controllable perdurants (some states, processes), usually within methods.
They must be defined by a method, but can be *used* by other kinds of descriptions.
They are desire targets of some role played by an agent.

Tasks can be complex, and ordered according to an abstract succession relation. Tasks can relate to ground activities or decision making; the last kind deals with typical flowchart content. A task is different both from a flowchart node, and from an action or action type.

Tasks can be considered shortcuts for plans, since at least one role played by an agent has a desire attitude towards them (possibly different from the one that puts the task into action). In principle, tasks could be transformed into explicit plans.")
)
Class(c:technique partial
c:method)
Class(c:technique partial
annotation(rdfs:comment "A technique is a practical method to obtain some modification in the environment (or evaluation of an environment) that fulfils some task.")
)
Class(c:theory partial
c:description)
Class(c:theory partial
annotation(rdfs:comment "This is used in a wide cultural sense: a theory about something, expressed in a rather systematic way, but not necessarily public (although communicable in principle). An axiomatic theory is not a theory in this sense, although we can expect an axiomatic theory to be the formal representation of a generic theory.")
)
Class(<f:classification-system> partial
<f:information-encoding-system>
restriction(c:involves someValuesFrom(<c:information-object>)))
Class(<f:classification-system> partial
annotation(rdfs:comment "An information encoding system that provides rules for (ev. ordered) lists of information objects, e.g terminologies, subjects, knowledge domains.")
)
Class(<f:combinatorial-system> partial

```
  <f:information-encoding-system>
  restriction(f:orders allValuesFrom(<c:information-object>)))
 Class(<f:combinatorial-system> partial
  annotation(rdfs:comment "An information encoding system that provides roles and operations to create valid information
objects (e.g. grammars, templates, codes).")
)
 Class(<f:diagrammatic-object> partial
  <c:information-object>)
 Class(<f:diagrammatic-object> partial
  annotation(rdfs:comment "An information object ordered by a shematic iconic code")
)
 Class(<f:formal-expression> partial
  <c:information-object>
  restriction(<f:ordered-by> someValuesFrom(<f:formal-system>)))
 Class(<f:formal-system> partial
  <f:information-encoding-system>
  restriction(f:orders allValuesFrom(<f:formal-expression>)))
 Class(<f:formal-system> partial
  annotation(rdfs:comment "A code that orders the generation of information objects according to formally defined
vocabulary, axioms, rules, etc.")
)
 Class(f:grammar partial
  <f:combinatorial-system>)
 Class(f:grammar partial
  annotation(rdfs:comment "A set of rules for the generation of a (closed or open set of) information objects.")
)
 Class(<f:iconic-object> partial
  <c:information-object>)
 Class(<f:iconic-object> partial
  annotation(rdfs:comment "An information object ordered by a visual code.")
)
 Class(<f:information-collection> partial
  <j:non-physical-collection>
  restriction(j:member allValuesFrom(f:text))
  restriction(j:member someValuesFrom(f:text))
  restriction(j:member minCardinality(2)))
 Class(<f:information-collection> partial
  annotation(rdfs:comment "A collection of texts.")
)
 Class(<f:information-encoding-system> partial
  c:description
  restriction(c:involves someValuesFrom(<c:information-object>)))
 Class(<f:information-encoding-system> partial
  annotation(rdfs:comment "An information encoding system is a description that involves information objects. They can be
divided into 1) axiomatic systems, which provide roles and operations to define formal descriptions (e.g. theories), 2)
combinatorial systems, which provide roles and operations to create valid information objects (e.g. grammars),  3)
classification systems, which are contexts of (ev.  ordered) lists of information objects, and 4) informal encoding systems,
which provide roles  and operations to define informal descriptions (e.g.  narratives).")
)
 Class(<f:information-realization> complete
  intersectionOf(<c:physical-realization> restriction(c:realizes someValuesFrom(<c:information-object>))))
 Class(<f:information-realization> partial
  annotation(rdfs:comment "Any physical entity that realizes an information object.")
)
 Class(<f:linguistic-object> partial
  <c:information-object>
  restriction(<f:ordered-by> someValuesFrom(g:language)))
 Class(<f:linguistic-object> partial
  annotation(rdfs:comment "An information object ordered by (encoded according to) a language.")
)
 Class(f:morpheme partial
  <f:linguistic-object>
  restriction(<d:part-of> someValuesFrom(f:word)))
 Class(f:morpheme partial
  annotation(rdfs:comment "A part of a word that can express a meaning.")
)
 Class(f:narrative partial
  restriction(<c:expressed-by> someValuesFrom(f:text))
  c:description)
 Class(f:narrative partial
  annotation(rdfs:comment "A description expressed by a text, and ordered by additional semiotic codes (narratological
structures).")
)
 Class(f:phoneme partial
  restriction(<d:part-of> someValuesFrom(f:word))
  <f:linguistic-object>)
 Class(f:phoneme partial
  annotation(rdfs:comment "A part of a word that is assumed to be sensible to speakers when physically realized by voice.
A phoneme is not necessarily able to express a meaning (description), although it can in principle (e.g. 'a' in English).")
)
 Class(f:text partial
  <f:linguistic-object>
  restriction(<f:ordered-by> someValuesFrom(g:language)))
 Class(f:text partial
  annotation(rdfs:comment "A complex linguistic object,  expressed according to a language and still independent from  a
particular physical support.")
)
 Class(<f:text-repository> partial
  restriction(j:member someValuesFrom(f:text))
  <f:information-collection>
  restriction(j:member minCardinality(2))
  restriction(j:member allValuesFrom(f:text)))
 Class(<f:text-repository> partial
  annotation(rdfs:comment "A collection having only texts as members.")
)
 Class(f:word partial
  <f:linguistic-object>)
 Class(f:word partial
  annotation(rdfs:comment "A linguistic object consisting of a string (independently of its physical realization). Its
topological unity can change according to its physical realization: as a written realization, its boundaries are blank spaces, as
a spoken realization, sometimes is silence, sometimes not, and higher order features intervene.")
)
 Class(<h:cognitive-modal-description> partial
  <c:modal-description>)
 Class(<h:cognitive-modal-description> partial
  annotation(rdfs:comment "The modal descriptions depending on some mental attitude, represented here by means of a
relation between roles and tasks.")
)
 Class(h:commitment partial
  <h:cognitive-modal-description>)
 Class(h:commitment partial
  annotation(rdfs:comment "A commitment is a cognitive modal description, characterized by certain obligations and rights
targeted by at least one of its roles.")
)
 Class(h:desire partial
  <h:cognitive-modal-description>
  restriction(<c:conceived-by> someValuesFrom(unionOf(<d:agentive-social-object> <k:cognitive-agentive-physical-
object>))))
 Class(h:desire partial
  annotation(rdfs:comment "Desires in general are characterised by defining or using at least one intentional agentive role or
figure, and at least one course towards which the role or figure has a desire.
The coreference between the two axioms cannot be represented in OWL-DL.")
)
 Class(h:promise partial
  h:commitment)
 Class(h:promise partial
  annotation(rdfs:comment "A commitment in which an obligation to some future result is expressed.")
)
 Class(h:responsibility partial
  h:commitment
  restriction(<c:d-uses> someValuesFrom(c:task))
  restriction(<c:d-uses> someValuesFrom(k:status)))
 Class(h:responsibility partial
  annotation(rdfs:comment "Responsibility is preliminarily described here as a commitment that includes a status, which has
some rights and duties towards some task (see related axioms).")
)
```

```
 Class(<e:abstract-plan> partial
  c:plan)
 Class(<e:abstract-plan> partial
  annotation(rdfs:comment "An abstract plan is a plan whose roles and tasks only specify classes of entities that can be
included in a plan execution. In other words, a component from an abstract plan does not select any named entity.
This condition cannot be formalized in FOL, since we would like to express a condition by which an instance of an abstract
plan specifies instances of plan components, but no instances of situation elements, e.g. that 'manager' selects some (if any)
instance of person, but not a specified (named) person.")
)
 Class(<e:action-task> complete
  intersectionOf(restriction(c:sequences allValuesFrom(complementOf(<e:planning-activity>))) <e:elementary-task>))
 Class(<e:action-task> partial
  annotation(rdfs:comment "An action task is an elementary task that sequences non-planning activities, like: moving,
exercising forces, gathering information, etc. Planning activites are mental events involving some rational event.")
)
 Class(<e:bag-task> partial
  <e:complex-task>
  restriction(c:component allValuesFrom(complementOf(<e:control-task>))))
 Class(<e:bag-task> partial
  annotation(rdfs:comment "A bag task is a complex task that does not include either a control task, or a successor relation
among any two component tasks.
The last condition cannot be stated in OWL-DL, because it needs a coreference.")
)
 Class(<e:circumstantial-plan> complete
  intersectionOf(restriction(<c:d-uses> allValuesFrom(intersectionOf(restriction(c:classifies someValuesFrom(d:particular))
c:concept))) c:plan))
 Class(<e:circumstantial-plan> partial
  annotation(rdfs:comment "A circumstantial plan has all components selecting named individuals from the ground ontology
(e.g. only specific persons, specified resources, a finite number of time intervals and space regions, etc.).

This condition cannot be formalized in FOL, since we would like to express a condition by which an instance of an
circumstantial plan specifies both instances of plan components, and instances of situation elements, e.g. that 'manager'
selects a specified (named) person.")
)
 Class(<e:complex-task> complete
  intersectionOf(restriction(c:component minCardinality(2)) restriction(c:component allValuesFrom(c:task))
restriction(c:component someValuesFrom(c:task)) c:task))
 Class(<e:complex-task> partial
  annotation(rdfs:comment "A task that has at least two other tasks as components.")
)
 Class(<e:control-task> complete
  intersectionOf(<e:elementary-task> restriction(c:sequences allValuesFrom(unionOf(<e:planning-activity> <e:decision-
state>)))))
 Class(<e:control-task> partial
  annotation(rdfs:comment "A control task is an elementary task that sequences a planning activity, e.g. an activity aimed at
(cognitively or via simulation) anticipating other activities. Therefore, control tasks have usually at least one direct successor
task (the controlled one), with the exception of ending tasks.

The reification of control constructs allows to represent procedural knowledge into the same ontology including controlled
action. Besides cognitive transparency and independency from a particular grounding system, a further advantage is enable
the representation of coordination tasks. For example, a manager that coordinates the execution of several related activities
can be represented as a role with a responsibility (duty+right) towards some complex task.")
)
 Class(<e:decision-activity> partial
  <e:planning-activity>
  restriction(<c:sequenced-by> someValuesFrom(oneOf(<e:case-task>))))
 Class(<e:decision-activity> partial
  annotation(rdfs:comment "An activity related to planning. It is sequenced by 'case task', and can contain an information
gathering activity.")
)
 Class(<e:decision-state> complete
  intersectionOf(d:state restriction(b:follows someValuesFrom(<e:decision-activity>)) restriction(<c:sequenced-by>
someValuesFrom(oneOf(<e:deliberation-task>)))))
 Class(<e:decision-state> partial
  annotation(rdfs:comment "A state related to planning. It is sequenced by 'deliberation task', and is preceded by a decision
activity.")
)
 Class(<e:elementary-task> complete
  intersectionOf(restriction(c:component allValuesFrom(complementOf(c:task))) c:task))
 Class(<e:elementary-task> partial
  annotation(rdfs:comment "An atomic task.")
)
 Class(e:goal complete
  intersectionOf(restriction(<d:proper-part-of> someValuesFrom(c:plan)) h:desire))
 Class(e:goal partial
  annotation(rdfs:comment "We are proposing here a restrictive notion of goal that relies upon its desirability by some agent, which does not necessarily play a role in the execution of the plan the goal is a part of. For example, an agent can have an attitude towards some task defined in a plan, e.g. duty towards, which is different from desiring it (desire towards). We might say that a goal is usually desired by the creator or beneficiary of a plan. The minimal constraint for a goal is that it is a proper part of a plan.
For example, a desire to start a relationship can become a goal if someone takes action (or lets someone else take it for her sake) to obtain it.")
)
 Class(<e:goal-qua-main> complete
  intersectionOf(e:goal restriction(<e:main-goal-of> someValuesFrom(c:plan))))
 Class(<e:goal-qua-main> partial
  annotation(rdfs:comment "A main goal can be defined as a goal that is part of a plan but not of one of its subplans.
The characteristic axiom cannot be formalized in OWL-DL (it requires coreference).")
)
 Class(<e:goal-situation> complete
  intersectionOf(c:situation restriction(c:satisfies someValuesFrom(e:goal))))
 Class(<e:goal-situation> partial
  annotation(rdfs:comment "A goal situation is a situation that satisfies a goal.

Opposite to the case of subplan executions, a goal situation is not part of a plan execution.

In other words, it is not true in general that any situation satisfying a part of a description, is also part of the situation that satisfies the whole description.

This helps to account for the following cases:

? Execution of plans containing abort or suspension conditions (the plan would be satisfied even if the goal has not been reached, see below)
? Incidental satisfaction, like when a situation satisfies a goal without being intentionally planned (but anyway desired).")
)
 Class(<e:hybrid-task> complete
  intersectionOf(<e:complex-task> restriction(c:component someValuesFrom(<e:control-task>)) restriction(c:component someValuesFrom(<e:action-task>))))
 Class(<e:hybrid-task> partial
  annotation(rdfs:comment "A complex task that has at least one control task (and then, at least one action task as well) as component.")
)
 Class(<e:information-gathering> partial
  k:activity)
 Class(<e:information-gathering> partial
  annotation(rdfs:comment "An activity aimed at gathering information for some purpose. It is typically sequenced by case tasks for taking decisions (can be part of decision activities).")
)
 Class(<e:maximal-task> partial
  <e:complex-task>)
 Class(<e:maximal-task> partial
  annotation(rdfs:comment "A maximal task is a complex task that has all the tasks defined in a plan as components.

In OWL-DL the axiom is defined as a concept axiom over plan component  task.")
)
 Class(<e:plan-assessment> partial
  restriction(<c:has-in-scope> allValuesFrom(<e:plan-execution>))
  c:technique)
 Class(<e:plan-assessment> partial
  annotation(rdfs:comment "A technique to evaluate a plan execution.")
)
 Class(<e:plan-assessment-task> complete
  intersectionOf(restriction(<c:defined-by> someValuesFrom(<e:plan-assessment>)) <e:control-task>))
 Class(<e:plan-assessment-task> partial
  annotation(rdfs:comment "A task defined in a plan assessment.")
```

```
)
 Class(<e:plan-execution> complete
  intersectionOf(c:situation restriction(<c:p-sat> someValuesFrom(c:plan))))
 Class(<e:plan-execution> partial
  annotation(rdfs:comment "Plan executions are situations that proactively satisfy a plan (cf. definition of P-SAT above).
Subplan executions are proper parts of the whole plan execution.")
)
 Class(<e:planning-activity> partial
  restriction(m:product allValuesFrom(c:plan))
  k:activity)
 Class(<e:planning-activity> partial
  annotation(rdfs:comment "The activity to generate a plan.")
)
 Class(<e:saturated-plan> complete
  intersectionOf(restriction(<c:d-uses> someValuesFrom(intersectionOf(restriction(<c:valued-by>
someValuesFrom(<d:space-region>)) c:parameter))) c:plan restriction(<c:d-uses>
someValuesFrom(intersectionOf(restriction(<c:valued-by> someValuesFrom(<d:time-interval>)) c:parameter)))))
 Class(<e:saturated-plan> partial
  annotation(rdfs:comment "A saturated plan is a plan that cannot be executed twice, since it defines spatio-temporal
parameters restricted to one value, e.g. one of its tasks selects an event that is valued by a definite temporal value in a definite
space region.

Of course, in the case of maximal spatio-temporal regions, a saturated plan tends to approximate an abstract plan from the
execution viewpoint, but these worst cases are unavoidable when dealing with maximality.")
)
 Class(e:schedule complete
  intersectionOf(restriction(c:requisite someValuesFrom(intersectionOf(restriction(<c:valued-by> someValuesFrom(<d:time-
interval>)) c:parameter))) c:task))
 Class(e:schedule partial
  annotation(rdfs:comment "A scheduling is a task that cannot be executed twice, since it has a temporal parameter restricted
to one value, e.g. it selects an event that is valued by a definite temporal value.")
)
 Class(<e:sequential-task> partial
  restriction(c:component minCardinality(2))
  <e:complex-task>
  restriction(c:component allValuesFrom(complementOf(<e:control-task>)))
  restriction(c:component someValuesFrom(<e:action-task>)))
 Class(<e:sequential-task> partial
  annotation(rdfs:comment "A sequential task is a complex task that includes a successor relation among any two component
tasks, and does not contain any control task.

The first condition cannot be stated in OWL-DL, because it needs coreference.")
)
 Class(e:subplan complete
  intersectionOf(restriction(<d:proper-part-of> someValuesFrom(c:plan)) c:plan))
 Class(e:subplan partial
  annotation(rdfs:comment "A proper part of a plan.")
)
 Class(<g:communication-role> partial
  restriction(<c:d-used-by> someValuesFrom(oneOf(<g:s-communication-theory>)))
  c:role)
 Class(<g:communication-role> partial
  annotation(rdfs:comment "The roles employed to characterize communication. E.g. the roles from Jakobson's theory of
communication.")
)
 Class(<g:communication-situation> complete
  intersectionOf(c:situation restriction(c:satisfies someValuesFrom(oneOf(<g:s-communication-theory>)))))
 Class(<g:communication-situation> partial
  annotation(rdfs:comment "Any situation that satisfies Jakobson's communication theory.")
)
 Class(<g:interpretation-situation> complete
  intersectionOf(c:situation restriction(c:satisfies someValuesFrom(oneOf(<g:semiotic-interpretation-function>)))))
 Class(<g:interpretation-situation> partial
  annotation(rdfs:comment "The class of situations that satisfy the semiotic interpretation function (given an expression and a
context, a meaning is provided).")
)
 Class(g:language partial
  <g:semiotic-code>)
 Class(<g:semiotic-code> partial
  <f:combinatorial-system>)
 Class(<g:semiotic-code> partial
  annotation(rdfs:comment "A combinatorial code intended to ordering of information objects involved in the semiotic
'interpretation function'.")
)
 Class(<g:semiotic-role> partial
  c:role
  restriction(<c:specialized-by> someValuesFrom(<g:communication-role>)))
 Class(<g:semiotic-role> partial
  annotation(rdfs:comment "A semiotic role is a non-agentive role defined by the interpretation function.
It should be specialized within a communication setting by a role that is played by some entity in a communication situation.
Semiotic roles are used to fill the universe of the so-called 'interpretation function'.
Two of them are specialized by two communication roles (message and context).")
)
 Class(<i:agentive-figure> complete
  intersectionOf(<d:agentive-social-object> restriction(c:conceives someValuesFrom(c:description)) restriction(<c:acted-by>
allValuesFrom(unionOf(<d:agentive-social-object> <d:agentive-physical-object>))) restriction(c:plays
someValuesFrom(c:role)) c:figure restriction(c:deputes allValuesFrom(c:role))))
 Class(<i:agentive-figure> partial
  annotation(rdfs:comment "Agentive figures are those which are assigned (agentive) roles from a society or community;
hence, they can act like a physical agent.

Typical agentive figures are societies, organizations, and in general all socially constructed persons.
Figures are not dependent on roles defined or used in the same descriptions they are defined or used, but they can act because
they depute some powers to some of those roles. In other words, a figure selected by some agentive role can play that role
because there are other roles in the descriptions that define or use the figure. Those roles select endurants that result to act for
the figure.

For example, an employee acts for an organization that deputes the role (e.g. turner) that classifies the employee. Simply put,
a guy working as a turner at FIAT acts for (or on behalf of) FIAT.
In complex figures, like organizations or societies, a total agency is possible when an endurant plays a delegate, or
representative role of the figure.
Since figures are social objects, it is conceivable to find figures that act for other figures.")
)
 Class(i:institution partial
  i:organization)
 Class(i:institution partial
  annotation(rdfs:comment "An organization bearing a legal status and having powers conferred by Law.")
)
 Class(i:organization partial
  <i:socially-constructed-person>)
 Class(i:organization partial
  annotation(rdfs:comment "A socially-constructed person with a complex articulation of tasks, roles and figures.")
)
 Class(<i:social-individual> complete
  intersectionOf(restriction(<c:acted-by> someValuesFrom(unionOf(<d:natural-person> l:collective))) <i:agentive-figure>))
 Class(<i:social-individual> partial
  annotation(rdfs:comment "a.k.a. social agent.
a.k.a. social figure.
An agentive figure created and maintained by a society (a collective).")
)
 Class(<i:social-unit> partial
  <i:social-individual>)
 Class(<i:social-unit> partial
  annotation(rdfs:comment "A social individual that promotes a collective to a definite social recognition.
It is usually acted by a collective, but there can be exceptions (e.g. mononuclear families).")
)
 Class(<i:socially-constructed-person> partial
  <i:social-individual>)
 Class(<i:socially-constructed-person> partial
  annotation(rdfs:comment "A definite social figure that is constructed and acted by other previously existing persons
(socially constructed or naturally born). A person in general is not characterized in this ontology.
In a legal extension, it could be reasonable to create a class of legal persons, defined by legal constitutive descriptions, which
includes the legal figures related to both natural and socially-constructed persons.")
)
 Class(<o:design-object-materialization> complete
  intersectionOf(<o:system-as-situation> restriction(<d:generically-dependent-on> someValuesFrom(<o:production-
workflow-execution>)) restriction(c:satisfies someValuesFrom(<o:system-design>))))
 Class(<o:design-object-materialization> partial
  annotation(rdfs:comment "A situation in which an object exists that has been produced according to a system design
specification.")
)
 Class(<o:production-workflow-execution> complete
  intersectionOf(<o:system-as-situation> restriction(c:satisfies someValuesFrom(<o:system-production-workflow>))))
 Class(<o:production-workflow-execution> partial
  annotation(rdfs:comment "A situation satisfying the production workflow of a system.")
)
 Class(<o:system-as-artifact> complete
  intersectionOf(restriction(<d:proper-part> someValuesFrom(intersectionOf(restriction(<j:member-of>
someValuesFrom(intersectionOf(j:collection restriction(<j:unified-by> someValuesFrom(unionOf(c:project c:plan))))))
<d:physical-object>))) <a:material-artifact> restriction(<d:proper-part> allValuesFrom(intersectionOf(restriction(<j:member-
of> someValuesFrom(intersectionOf(restriction(<j:unified-by> someValuesFrom(unionOf(c:project c:plan))) j:collection)))
<d:physical-object>)))))
 Class(<o:system-as-artifact> partial
  annotation(rdfs:comment "A material artifact whose proper parts ('components') are physical objects, members of a
collection unified by a project or plan.")
)
 Class(<o:system-as-description> partial
  c:description
  restriction(<c:satisfied-by> allValuesFrom(<o:system-as-situation>)))
 Class(<o:system-as-description> partial
  annotation(rdfs:comment "The descriptive, unifying aspect of a system (usually it includes at least a design, or project, plan,
etc.).")
)
 Class(<o:system-as-situation> complete
  intersectionOf(c:situation restriction(c:satisfies someValuesFrom(<o:system-as-description>))))
 Class(<o:system-as-situation> partial
  annotation(rdfs:comment "The realization aspect of a system, satisfying the descriptive aspect.
If the descriptive part only includes a design, it can be a situation in which that design has been realized (e.g. consisting
essentially of a system-as-artifact as a design object).
If the descriptive part includes a project, it can be a workflow situation resulting in the production of e.g. a system-as-artifact.
If the descriptive part includes a set of instructions, it can be a situation in which e.g. a system-as-artifact interacts with the
environment effectively (according to some evaluation criteria).")
)
 Class(<o:system-design> partial
  <o:system-as-description>)
 Class(<o:system-design> partial
  annotation(rdfs:comment "The description of a system from the design viewpoint (how it is structured, but also including
possible aesthetic or functional descriptions).")
)
 Class(<o:system-functionality> partial
  <o:system-as-description>)
 Class(<o:system-functionality> partial
  annotation(rdfs:comment "The description of a system from the functional viewpoint (how it works).")
)
 Class(<o:system-production-workflow> complete
  intersectionOf(restriction(<d:specifically-constantly-dependent-on> someValuesFrom(<o:system-functionality>))
<o:system-as-description> restriction(<d:specifically-constantly-dependent-on> someValuesFrom(<o:system-design>))))
 Class(<o:system-production-workflow> partial
  annotation(rdfs:comment "The description of how a system is produced.")
)
 Class(<o:working-system-situation> complete
  intersectionOf(<o:system-as-situation> restriction(<d:generically-dependent-on> someValuesFrom(<o:design-object-
materialization>)) restriction(c:satisfies someValuesFrom(<o:system-functionality>))))
 Class(<o:working-system-situation> partial
  annotation(rdfs:comment "The situation in which a working system interacts with its environment according to its
functionality description.")
)
 Class(owl:Thing partial)


 AnnotationProperty(rdfs:comment)
 AnnotationProperty(owl:versionInfo)


 Individual(<e:abandonment-task>
  annotation(rdfs:comment "A specialization of ending-task, aimed at sequencing events that end a plan execution without
having reached its main-goal, and with no intention to resurrect the plan.")

  type(<e:control-task>)
  value(c:specializes <e:ending-task>))
 Individual(<e:abortion-task>
  annotation(rdfs:comment "A specialization of ending-task, aimed at sequencing events that end a plan execution without
having reached its main-goal, but with the possibility or resurrecting the plan.")

  type(<e:control-task>)
  value(c:specializes <e:ending-task>))
 Individual(<e:acceptation-task>
  annotation(rdfs:comment "The task sequencing a positive decision to adopt a plan for execution.")

  type(owl:Thing)
  type(restriction(d:predecessor someValuesFrom(oneOf(<e:consideredness-task>))))
  type(<e:plan-assessment-task>)
  value(c:specializes <e:deliberation-task>))
 Individual(<e:activation-task>
  annotation(rdfs:comment "A control task aimed at starting an activity. It is specialized either by a beginning task or a
reactivation task.")

  type(restriction(<d:direct-predecessor> someValuesFrom(oneOf(<e:readiness-task>))))
  type(<e:control-task>)
  type(restriction(<d:direct-predecessor> allValuesFrom(complementOf(<e:action-task>))))
  value(<c:specialized-by> <e:reactivation-task>)
  value(<c:specialized-by> <e:beginning-task>))
 Individual(<e:alternate-task>
  annotation(rdfs:comment "A case task branched to exactly 2 tasks, not executable in  parallel.")

  type(restriction(<d:direct-successor> cardinality(2)))
  type(<e:control-task>)
  value(c:specializes <e:case-task>)
  value(<d:direct-successor> <e:deliberation-task>))
 Individual(<e:any-order-task>
  annotation(rdfs:comment "An any order task is a branching task that defines no order in the successor tasks. It?s another
way of defining a bag task, because any temporal relation can be expected between any two perdurants sequenced by the
tasks that are direct successor to the any order task.")

  type(owl:Thing)
  type(restriction(<d:direct-successor> someValuesFrom(intersectionOf(restriction(c:sequences
allValuesFrom(intersectionOf(restriction(<b:temporal-relation> someValuesFrom(intersectionOf(restriction(<c:sequenced-
by> someValuesFrom(c:task)) d:perdurant))) d:perdurant))) c:task))))
  type(<e:control-task>)
  value(c:specializes <e:branching-task>)
  value(d:successor <e:synchro-task>))
 Individual(<e:beginning-task>
  annotation(rdfs:comment "A beginning task is a control task that is the predecessor of all tasks defined in the plan.")

  type(restriction(d:successor someValuesFrom(c:task)))
  type(<e:control-task>)
  type(restriction(d:predecessor allValuesFrom(complementOf(c:task))))
  value(<d:specifically-constantly-dependent-on> <e:readiness-task>)
  value(c:specializes <e:activation-task>))
 Individual(<e:branching-task>
  annotation(rdfs:comment "A task that articulates the plan into an ordered set of tasks.")

  type(restriction(<d:direct-successor> someValuesFrom(c:task)))
  type(restriction(c:sequences allValuesFrom(<e:planning-activity>)))
  type(<e:control-task>)
  type(restriction(<d:direct-successor> minCardinality(2))))
 Individual(<e:case-task>
  annotation(rdfs:comment "A case task is a task branched to a set of tasks that are not executable concurrently.
In order to choose the task to be executed, preliminary deliberation tasks should be executed.
A case task sequences a decision activity (a kind of mental event involving rationality) that has a deliberation state as
outcome (sequenced by a deliberation task).")
```

type(restriction(c:sequences allValuesFrom(<e:decision-activity>)))
type(owl:Thing)
type(<e:control-task>)
value(c:specializes <e:branching-task>)
value(<d:direct-successor> <e:deliberation-task>))
Individual(<e:completion-task>
annotation(rdfs:comment "A specialization of ending-task, aimed at sequencing events that end a plan execution having reached its main-goal.")

type(<e:control-task>)
value(c:specializes <e:ending-task>))
Individual(<e:concurrency-task>
annotation(rdfs:comment "A concurrent task is a task branched to a set of tasks executable concurrently (the sequenced perdurants can overlap), which means that no deliberation task is performed in order to choose among them. A concurrent task has at least one successor synchronization task, which is aimed at waiting for the execution of all (except the optional ones) tasks direct successor to the concurrent (or any order, see below) one.

The axioms cannot be expressed fully in OWL-DL (no value mapping available).")

type(owl:Thing)
type(<e:control-task>)
type(restriction(<d:direct-successor> someValuesFrom(intersectionOf(restriction(c:sequences allValuesFrom(intersectionOf(d:perdurant restriction(d:overlaps someValuesFrom(intersectionOf(restriction(<c:sequenced-by> someValuesFrom(c:task)) d:perdurant)))))) c:task))))
value(c:specializes <e:branching-task>)
value(d:successor <e:synchro-task>))
Individual(<e:consideredness-task>
annotation(rdfs:comment "The task sequencing a decision activity, aiming at if action has to be taken in order to start a plan execution.")

type(owl:Thing)
type(restriction(<d:direct-predecessor> someValuesFrom(oneOf(<e:possibility-task>))))
type(<e:plan-assessment-task>)
value(c:specializes <e:case-task>))
Individual(<e:decidedness-task>
annotation(rdfs:comment "The task sequencing a decision to take action in order to start a plan execution.")

type(owl:Thing)
type(restriction(<d:direct-predecessor> someValuesFrom(oneOf(<e:acceptation-task>))))
type(<e:plan-assessment-task>)
value(c:specializes <e:deliberation-task>))
Individual(<e:deliberation-task>
annotation(rdfs:comment "A deliberation task is a control task that sequences deliberation states (decisions taken after a case task execution).")

type(owl:Thing)
type(<e:control-task>)
type(restriction(c:sequences allValuesFrom(<e:decision-state>)))
value(<d:direct-predecessor> <e:case-task>))
Individual(<e:ending-task>
annotation(rdfs:comment "An ending task is a control task that has no successor tasks defined in the plan.")

type(<e:control-task>)
type(restriction(d:predecessor someValuesFrom(c:task)))
type(restriction(d:successor allValuesFrom(complementOf(c:task)))))
Individual(<e:loop-for>
annotation(rdfs:comment "A loop task with a defined number (and possibly frequency) of iterations.")

type(<e:control-task>)
value(c:specializes <e:loop-task>))
Individual(<e:loop-task>
annotation(rdfs:comment "A loop task is a control task that has as successor an action (or complex) task that sequences at least two distinct activities sharing a minimal common set of properties (they have a minimal common type).

Notice that MinimalCommonType cannot be formalised as a first-order predicate, and then neither in OWL-DL. It can be considered a trivial guideline: ?when sequencing looped actions, choose a definite action class from the ground ontology?. Some relations typically hold for loop tasks. Exit condition can be used to state what deliberation task (see below) causes to exit the cycle; iteration interval can be used to state how much time should be taken by each iteration of the looped activity; iteration cardinality can be used to state how many times the action should be repeated.")

```
  type(<e:control-task>))
 Individual(<e:loop-until>
  annotation(rdfs:comment "A loop task, which specifies when a certain condition becomes true for a cyclical task to exit.")

  type(<e:control-task>)
  value(<e:exit-condition> <e:deliberation-task>)
  value(c:specializes <e:loop-task>))
 Individual(<e:parallel-task>
  annotation(rdfs:comment "A task for parallel concurrent activities.")

  type(restriction(<d:direct-successor> someValuesFrom(intersectionOf(restriction(<e:sibling-task>
someValuesFrom(c:task)) c:task))))
  type(restriction(<d:direct-successor> someValuesFrom(intersectionOf(restriction(<e:sibling-task>
someValuesFrom(c:task)) restriction(c:sequences allValuesFrom(intersectionOf(restriction(<b:temporally-coincides>
someValuesFrom(intersectionOf(restriction(<c:sequenced-by> allValuesFrom(c:task)) k:activity))) k:activity))) c:task))))
  type(restriction(<d:direct-successor> minCardinality(2)))
  type(<e:control-task>)
  value(c:specializes <e:concurrency-task>))
 Individual(<e:partly-case-task>
  annotation(rdfs:comment "A control task that directly precedes both a case task and some other task.
It specializes the branching task.")

  type(restriction(<d:direct-successor> someValuesFrom(intersectionOf(complementOf(oneOf(<e:case-task>)) c:task))))
  type(<e:control-task>)
  value(c:specializes <e:branching-task>)
  value(<d:direct-successor> <e:case-task>))
 Individual(<e:possibility-task>
  annotation(rdfs:comment "The task sequencing an activity from which the possibility is raised to execute a plan.")

  type(owl:Thing)
  type(<e:plan-assessment-task>))
 Individual(<e:preparedness-task>
  annotation(rdfs:comment "The task sequencing an assessment that the activities aiming at creating the prerequisites to start
a plan execution are completed.")

  type(owl:Thing)
  type(<e:plan-assessment-task>)
  type(restriction(<d:direct-predecessor> someValuesFrom(oneOf(<e:decidedness-task>)))))
 Individual(<e:reactivation-task>
  annotation(rdfs:comment "An activation task to start a plan execution after it has been suspended.")

  type(restriction(<d:direct-predecessor> someValuesFrom(oneOf(<e:suspension-task>))))
  type(<e:control-task>)
  value(c:specializes <e:activation-task>))
 Individual(<e:readiness-task>
  annotation(rdfs:comment "The task joining the decision and preparation phases of the plan assessment, with the activation
phases of the plan.")

  type(owl:Thing)
  type(restriction(<d:direct-predecessor> someValuesFrom(oneOf(<e:preparedness-task>))))
  type(<e:plan-assessment-task>))
 Individual(<e:rejectedness-task>
  annotation(rdfs:comment "The task sequencing a negative decision to adopt a plan execution.")

  type(restriction(d:predecessor someValuesFrom(oneOf(<e:consideredness-task>))))
  type(<e:plan-assessment-task>)
  value(c:specializes <e:deliberation-task>))
 Individual(<e:suspension-task>
  annotation(rdfs:comment "A specialization of ending-task, aimed at sequencing events that end a plan execution without
having reached its main-goal for a certain time")

  type(owl:Thing)
  type(<e:control-task>)
  value(c:specializes <e:ending-task>))
 Individual(<e:synchro-task>
  annotation(rdfs:comment "A task that joins a set a tasks after a branching.
In particular, a synchronization task is aimed at waiting for the execution of all (except the optional ones) tasks that are direct
successor to a concurrent or any order task.")

  type(restriction(d:predecessor someValuesFrom(oneOf(<e:concurrency-task> <e:any-order-task>))))
  type(<e:control-task>)
  type(restriction(<d:direct-predecessor> someValuesFrom(intersectionOf(restriction(<d:direct-predecessor>
someValuesFrom(oneOf(<e:concurrency-task> <e:any-order-task>))) unionOf(<e:complex-task> <e:action-task>))))))
 Individual(<g:c-context>
  annotation(rdfs:comment "The context role in Jakobson's theory of communication.")

  type(<a:description-role>)
  type(<g:communication-role>)
  type(restriction(<c:played-by> allValuesFrom(c:description)))
  value(c:specializes <g:s-context>)
  value(<c:defined-by> <g:s-communication-theory>))
 Individual(<g:channel-role>
  annotation(rdfs:comment "The channel role in Jakobson's theory of communication.")

  type(<g:communication-role>)
  type(restriction(<c:played-by> allValuesFrom(<d:physical-endurant>)))
  value(<c:defined-by> <g:s-communication-theory>))
 Individual(<g:code-role>
  annotation(rdfs:comment "The code role in Jakobson's theory of communication, which should be played by an
information-encoding-system.")

  type(<a:description-role>)
  type(<g:communication-role>)
  type(restriction(<c:played-by> allValuesFrom(<f:information-encoding-system>)))
  value(<c:defined-by> <g:s-communication-theory>))
 Individual(<g:decoder-role>
  annotation(rdfs:comment "A specialization of the interpreter role, played by the agents trying to conceive the description
expressed by some information object created by agents playing the encoder role.")

  type(<g:communication-role>)
  type(<k:agent-driven-role>)
  value(c:specializes <g:encoder-role>))
 Individual(<g:encoder-role>
  annotation(rdfs:comment "A specialization of the interpreter role, played by creators of information objects expressing some
description.")

  type(<g:communication-role>)
  type(<k:agent-driven-role>)
  value(c:specializes <g:interpreter-role>))
 Individual(g:expression
  annotation(rdfs:comment "Expression is a semiotic role played by information objects.
It is used to fill the first domain of the  so-called 'interpretation function'. It can be considered equivalent to the 'message'
communication role, but since communication theory and semiotic theories are different, it is more correct to say that a
message role specializes an expression role.")

  type(<g:semiotic-role>)
  type(restriction(<c:played-by> allValuesFrom(<c:information-object>)))
  value(<c:defined-by> <g:semiotic-interpretation-function>))
 Individual(<g:interpreter-role>
  annotation(rdfs:comment "A generalization of the encoder and decoder roles in Jakobson's theory of communication, which
should be played by an agent.")

  type(<g:communication-role>)
  type(<k:agent-driven-role>)
  value(<c:defined-by> <g:s-communication-theory>))
 Individual(g:meaning
  annotation(rdfs:comment "Meaning is a semiotic role played by descriptions whatsoever.
It is used to fill the range of the  so-called 'interpretation function'.
It is not equivalent to any communication function.")

  type(<a:description-role>)
  type(<g:semiotic-role>)
  type(restriction(<c:played-by> allValuesFrom(c:description)))
  value(<c:defined-by> <g:semiotic-interpretation-function>))
 Individual(<g:message-role>
  annotation(rdfs:comment "The message role in Jakobson's theory of communication, played by information objects. It
specializes the expression role from semiotic interpretation theory.")

  type(<g:communication-role>)
  type(restriction(<c:played-by> allValuesFrom(<c:information-object>)))
  value(c:specializes g:expression)
  value(<c:defined-by> <g:s-communication-theory>))
 Individual(<g:s-communication-theory>
  annotation(rdfs:comment "Jakobson defined six functions of communication that are compatible with Shannon's theory of
information. They are the 'message', here covered by 'Message-Role', the context, covered here by 'C-Context', the code,
covered by 'Code', plus 'Channel', 'Encoder', and 'Decoder', which are introduced below. Message-Role, C-Context, and Code
can also be viewed as playing a semiotic role (Expression, S-Context, Semiotic-Code).
For a communication theory in general, we also need other components that are not specified in Jakobson's theory', e.g. 'turn-
taking', governing the sequence of a communication process,  'communication parameters', governing the values that
participants and events of a communication should have in order for the communication to be  successful (i.e. for the
communication method to be satisfied), 'conversational maxims' (superordered theories) that provide guidelines for
communication to be successful, etc.")

  type(owl:Thing)
  type(c:theory)
  value(c:defines <g:message-role>)
  value(c:defines <g:c-context>)
  value(c:defines <g:channel-role>)
  value(c:defines <g:code-role>)
  value(c:defines <g:interpreter-role>))
 Individual(<g:s-context>
  annotation(rdfs:comment "S-context (semiotic context) is played by descriptions and is a semiotic role. It is used to fill the
second domain of the  so-called 'interpretation function'.
It may be equivalent to the 'c-context' communication role, but since communication theory and semiotic theories are
different, it is more correct to say that c-context (communication context) specializes s-context.")

  type(<a:description-role>)
  type(<g:semiotic-role>)
  type(restriction(<c:played-by> allValuesFrom(c:description)))
  value(<c:defined-by> <g:semiotic-interpretation-function>))
 Individual(<g:semiotic-interpretation-function>
  annotation(rdfs:comment "Interpretation functions are descriptions that can include roles either for semiotics or for formal
semantics.

Here we only characterize a basic, simple theory of semiotic interpretation. Three semiotic roles are defined: s-context
(semiotic context), expression, and meaning.

It has complex dependencies to mental objects, social objects, as well as references to entities as such, but we currently prefer
to put it here as a placeholder (a forthcoming ontology of mind should give some more detail on those issues). See semiotic
roles for further comments.")

  type(c:relation)
  type(owl:Thing)
  type(restriction(c:defines someValuesFrom(<g:semiotic-role>)))
  value(c:defines g:meaning)
  value(c:defines g:expression)
  value(c:defines <g:s-context>))

 DisjointClasses(c:description c:concept)
 DisjointClasses(d:endurant d:quality)
 DisjointClasses(d:endurant d:perdurant)
 DisjointClasses(<d:amount-of-matter> <d:physical-object>)
 DisjointClasses(<d:abstract-quality> <d:physical-quality>)
 DisjointClasses(d:abstract d:perdurant)
 DisjointClasses(c:description c:figure)
 DisjointClasses(<d:physical-object> d:feature)
 DisjointClasses(<c:information-object> c:concept)
 DisjointClasses(j:collection c:figure)
 DisjointClasses(d:abstract d:endurant)
 DisjointClasses(j:collection c:description)
 DisjointClasses(<d:physical-region> <d:temporal-region>)
 DisjointClasses(<d:abstract-region> <d:temporal-region>)
 DisjointClasses(c:role c:parameter)
 DisjointClasses(c:situation <c:information-object>)
 DisjointClasses(c:figure c:concept)
 DisjointClasses(<c:information-object> j:collection)
 DisjointClasses(d:abstract d:quality)
 DisjointClasses(c:role c:course)
 DisjointClasses(<d:physical-endurant> <d:arbitrary-sum>)
 DisjointClasses(<d:non-agentive-physical-object> <d:agentive-physical-object>)
 DisjointClasses(<d:amount-of-matter> d:feature)
 DisjointClasses(c:situation c:concept)
 DisjointClasses(c:situation c:description)
 DisjointClasses(d:quality d:perdurant)
 DisjointClasses(<d:physical-region> <d:abstract-region>)
 DisjointClasses(<d:temporal-quality> <d:abstract-quality>)
 DisjointClasses(<d:arbitrary-sum> <d:non-physical-endurant>)
 DisjointClasses(c:situation j:collection)
 DisjointClasses(<d:temporal-quality> <d:physical-quality>)
 DisjointClasses(c:situation c:figure)
 DisjointClasses(<e:action-task> <e:control-task>)
 DisjointClasses(<d:physical-endurant> <d:non-physical-endurant>)
 DisjointClasses(j:collection c:concept)
 DisjointClasses(<c:information-object> c:description)
 DisjointClasses(c:course c:parameter)

 SubPropertyOf(<e:sibling-task> <d:mediated-relation>)
 SubPropertyOf(<c:requisite-for> <d:immediate-relation>)
 SubPropertyOf(c:realizes <c:referenced-by>)
 SubPropertyOf(<d:specific-constant-constituent> <d:immediate-relation>)
 SubPropertyOf(<d:generically-dependent-on> <d:immediate-relation-i>)
 SubPropertyOf(<c:refined-by> <d:proper-part-of>)
 SubPropertyOf(<d:strong-connection> <d:mediated-relation>)
 SubPropertyOf(<m:generic-target-of> <m:functional-participant-in>)
 SubPropertyOf(<c:temporary-component> c:component)
 SubPropertyOf(<b:temporally-overlaps> <b:temporal-relation>)
 SubPropertyOf(b:precedes <b:temporal-relation>)
 SubPropertyOf(<d:temporary-proper-part> <d:proper-part>)
 SubPropertyOf(<c:involved-in> <d:mediated-relation-i>)
 SubPropertyOf(<k:obliged-to> <c:attitude-towards>)
 SubPropertyOf(<n:material-place-of> <n:place-of>)
 SubPropertyOf(<d:weak-connection> <d:immediate-relation>)
 SubPropertyOf(<e:task-postcondition-of> <d:mediated-relation-i>)
 SubPropertyOf(k:uses <k:co-participates-with>)
 SubPropertyOf(e:influences <d:specific-constant-dependent>)
 SubPropertyOf(<d:t-inherent-in> <d:inherent-in>)
 SubPropertyOf(<c:expected-setting> <d:mediated-relation-i>)
 SubPropertyOf(<d:temporary-participant> d:participant)
 SubPropertyOf(<k:regulated-by> c:satisfies)
 SubPropertyOf(<c:expanded-by> <d:proper-part-of>)
 SubPropertyOf(<c:d-used-by> <c:temporary-component-of>)
 SubPropertyOf(<d:r-location-of> <d:immediate-relation-i>)
 SubPropertyOf(<d:temporary-part> d:part)
 SubPropertyOf(<i:ruled-by> <d:mediated-relation-i>)
 SubPropertyOf(<a:unit-of> <d:r-location-of>)
 SubPropertyOf(<d:exact-location-of> <d:generic-location-of>)
 SubPropertyOf(<e:influenced-by> <d:specifically-constantly-dependent-on>)
 SubPropertyOf(k:postcondition <d:mediated-relation>)
 SubPropertyOf(<d:abstract-location-of> <d:exact-location-of>)
 SubPropertyOf(<m:theme-of> <m:patient-of>)
 SubPropertyOf(<c:c-sat-by> <c:satisfied-by>)
 SubPropertyOf(<c:metaphorically-played-by> <c:played-by>)
 SubPropertyOf(<a:geographic-part> <n:descriptive-place-of>)
 SubPropertyOf(<c:has-in-scope> c:references)
 SubPropertyOf(<b:temporal-location> <d:exact-location>)
 SubPropertyOf(<d:generic-dependent> <d:immediate-relation>)
 SubPropertyOf(<b:temporally-coincides> <b:temporal-relation>)
 SubPropertyOf(<c:temporary-component-of> <d:partly-compresent>)
 SubPropertyOf(<n:spatial-location-of> <d:physical-location-of>)
 SubPropertyOf(<m:patient-of> <m:functional-participant-in>)
 SubPropertyOf(<d:abstract-location> <d:exact-location>)
 SubPropertyOf(<b:temporally-includes> <b:temporal-relation>)
 SubPropertyOf(<c:specialized-by> <d:immediate-relation-i>)
 SubPropertyOf(<d:part-of> <d:immediate-relation-i>)
 SubPropertyOf(n:place <n:approximate-location>)
 SubPropertyOf(<c:optionally-used-by> <c:d-used-by>)
 SubPropertyOf(<d:generic-constituent> <d:immediate-relation>)
 SubPropertyOf(<m:generic-target> <m:functional-participant>)
 SubPropertyOf(<d:quale-of> <d:q-location-of>)
 SubPropertyOf(<n:p-spatial-location-of> <d:exact-location-of>)
 SubPropertyOf(b:concludes <b:temporally-included-in>)
 SubPropertyOf(n:origin <n:material-place>)
 SubPropertyOf(<c:metaphorically-plays> c:plays)
 SubPropertyOf(<e:task-precondition> <d:mediated-relation>)
 SubPropertyOf(<c:attitude-target-of> <c:modal-target-of>)
 SubPropertyOf(<c:modal-target> <d:immediate-relation>)
 SubPropertyOf(<c:deputed-by> <d:immediate-relation-i>)
 SubPropertyOf(<k:method-of> c:expects)
 SubPropertyOf(<c:realized-by> c:references)
 SubPropertyOf(<n:origin-of> <n:material-place-of>)
 SubPropertyOf(c:conceives <d:immediate-relation>)
 SubPropertyOf(<e:achievable-through> c:references)
 SubPropertyOf(k:makes <k:co-participates-with>)
 SubPropertyOf(<j:unified-by> <c:referenced-by>)
 SubPropertyOf(<k:precondition-of> <d:mediated-relation-i>)
 SubPropertyOf(m:substrate <m:functional-participant>)
 SubPropertyOf(<d:temporary-part-of> <d:partly-compresent>)
 SubPropertyOf(<m:target-of> <m:patient-of>)
 SubPropertyOf(c:satisfies <d:intensionally-referenced-by>)
 SubPropertyOf(<d:generic-location-of> <d:mediated-relation-i>)
 SubPropertyOf(<d:temporary-atomic-part> <d:temporary-proper-part>)
 SubPropertyOf(m:product <m:functional-participant>)
 SubPropertyOf(<e:main-goal> <d:proper-part>)
 SubPropertyOf(<b:met-by> <b:temporally-connected>)
 SubPropertyOf(<e:iteration-interval> <d:mediated-relation>)
 SubPropertyOf(m:result <p:causally-precedes>)
 SubPropertyOf(<n:spatial-location> <d:physical-location>)
 SubPropertyOf(<k:right-to> <c:attitude-towards>)
 SubPropertyOf(c:plays <c:classified-by>)
 SubPropertyOf(<c:setting-for> c:references)
 SubPropertyOf(k:exploits c:involves)
 SubPropertyOf(c:setting <c:referenced-by>)
 SubPropertyOf(<d:physical-location> <d:exact-location>)
 SubPropertyOf(m:substrate <d:total-constant-participant>)
 SubPropertyOf(c:requisite <d:immediate-relation-i>)
 SubPropertyOf(<k:exploited-by> <c:involved-in>)
 SubPropertyOf(<e:adopts-goal> k:adopts)
 SubPropertyOf(<m:result-of> <p:causally-follows>)
 SubPropertyOf(<c:acts-for> <d:mediated-relation>)
 SubPropertyOf(<d:sibling-part> <d:mediated-relation>)
 SubPropertyOf(<k:co-participates-with> <d:mediated-relation>)
 SubPropertyOf(<f:q-represented-by> <d:mediated-relation-i>)
 SubPropertyOf(<c:r-sat-by> <c:satisfied-by>)
 SubPropertyOf(<c:acted-by> <d:mediated-relation-i>)
 SubPropertyOf(<m:resource-for> <m:used-in>)
 SubPropertyOf(<b:temporally-included-in> <b:temporal-relation-i>)
 SubPropertyOf(<b:concluded-by> <b:temporally-includes>)
 SubPropertyOf(<h:subject-target-of> <c:attitude-target-of>)
 SubPropertyOf(<f:ordered-by> <d:mediated-relation>)
 SubPropertyOf(c:classifies <d:intensionally-references>)
 SubPropertyOf(<d:temporary-proper-part> <d:temporary-part>)
 SubPropertyOf(d:host <d:specifically-constantly-dependent-on>)
 SubPropertyOf(b:meets <b:temporally-connected>)
 SubPropertyOf(<n:situation-place-of> <n:approximate-location-of>)
 SubPropertyOf(<b:temporally-connected> <b:temporal-relation>)
 SubPropertyOf(<c:satisfied-by> <d:intensionally-references>)
 SubPropertyOf(c:involves <d:mediated-relation>)
 SubPropertyOf(c:refines <d:proper-part>)
 SubPropertyOf(<k:empowered-to> <c:attitude-towards>)
 SubPropertyOf(<b:temporal-location-of> <d:exact-location-of>)
 SubPropertyOf(<b:time-of-presence-of> <b:e-temporal-location-of>)
 SubPropertyOf(<c:conceived-by> <d:immediate-relation-i>)
 SubPropertyOf(<m:product-of> <m:functional-participant-in>)
 SubPropertyOf(<d:temporary-part-of> <d:part-of>)
 SubPropertyOf(<o:functionally-unified-by> <c:referenced-by>)
 SubPropertyOf(e:subgoal d:part)
 SubPropertyOf(<k:made-by> <k:co-participates-with>)
 SubPropertyOf(<n:situation-place> <n:approximate-location>)
 SubPropertyOf(<e:iteration-interval-of> <d:mediated-relation-i>)
 SubPropertyOf(k:precondition <d:mediated-relation>)
 SubPropertyOf(<c:valued-by> c:classifies)
 SubPropertyOf(m:prescribes m:performs)
 SubPropertyOf(<o:functionally-unifies> c:references)
 SubPropertyOf(<c:d-uses> <c:temporary-component>)
 SubPropertyOf(c:about c:references)
 SubPropertyOf(<n:participant-place> <n:approximate-location>)
 SubPropertyOf(<d:proper-part> d:part)
 SubPropertyOf(<a:geographic-part-of> <n:descriptive-place>)
 SubPropertyOf(<n:descriptive-place> <n:approximate-location>)
 SubPropertyOf(<m:instrument-of> <m:used-in>)
 SubPropertyOf(m:instrument <m:use-of>)
 SubPropertyOf(<c:played-by> c:classifies)
 SubPropertyOf(<j:member-of> <d:generic-constituent-of>)
 SubPropertyOf(<d:q-location> <d:immediate-relation>)
 SubPropertyOf(<k:used-by> <k:co-participates-with>)
 SubPropertyOf(<c:parametrized-by> <d:mediated-relation-i>)
 SubPropertyOf(<k:created-by> <c:conceived-by>)
 SubPropertyOf(<k:postcondition-of> <d:mediated-relation-i>)
 SubPropertyOf(<d:temporary-part> <d:partly-compresent>)
 SubPropertyOf(<d:q-present-at> <d:mediated-relation>)
 SubPropertyOf(<d:time-of-q-presence-of> <d:mediated-relation-i>)
 SubPropertyOf(<c:required-by> <d:immediate-relation-i>)
 SubPropertyOf(<d:atomic-part-of> <d:part-of>)
 SubPropertyOf(<m:use-of> <m:functional-participant>)
 SubPropertyOf(<c:c-sat> c:satisfies)
 SubPropertyOf(<k:obligation-for> <c:attitude-target-of>)
 SubPropertyOf(<c:component-of> <d:proper-part-of>)
 SubPropertyOf(<d:total-temporary-participant-in> <d:temporary-participant-in>)
 SubPropertyOf(<c:p-sat-by> <c:satisfied-by>)
 SubPropertyOf(<d:constant-participant-in> <d:participant-in>)
 SubPropertyOf(i:rules <d:mediated-relation>)
 SubPropertyOf(<c:r-sat> c:satisfies)
 SubPropertyOf(<h:subjected-to> <c:attitude-towards>)
 SubPropertyOf(<e:disposition-to> <k:used-by>)
 SubPropertyOf(<e:discarded-within> <c:d-used-by>)
 SubPropertyOf(<d:total-temporary-participant> <d:temporary-participant>)
 SubPropertyOf(<d:constant-participant> d:participant)
 SubPropertyOf(<c:sequenced-by> <c:classified-by>)
 SubPropertyOf(<b:temporal-relation-i> <d:mediated-relation-i>)
 SubPropertyOf(<d:spatio-temporal-presence-of> <d:exact-location-of>)
 SubPropertyOf(<e:task-precondition-of> <d:mediated-relation-i>)
 SubPropertyOf(b:starts <b:temporally-included-in>)
 SubPropertyOf(<b:temporal-relation> <d:mediated-relation>)
 SubPropertyOf(<k:right-task-for> <c:attitude-target-of>)
 SubPropertyOf(j:characterizes <d:immediate-relation>)
 SubPropertyOf(<h:desire-target-of> <c:attitude-target-of>)
 SubPropertyOf(<j:extensionally-equivalent> <d:immediate-relation>)
 SubPropertyOf(<h:bdi-target-of> <c:attitude-target-of>)
 SubPropertyOf(<e:subgoal-of> <d:part-of>)
 SubPropertyOf(<b:present-at> <b:e-temporal-location>)
 SubPropertyOf(<c:referenced-by> <d:immediate-relation-i>)
 SubPropertyOf(<n:descriptive-origin> <n:descriptive-place>)
 SubPropertyOf(<d:exact-location> <d:generic-location>)
 SubPropertyOf(<c:defined-by> <c:d-used-by>)
 SubPropertyOf(c:expresses c:references)
 SubPropertyOf(c:interprets <d:immediate-relation>)
 SubPropertyOf(<c:attitude-towards> <c:modal-target>)
 SubPropertyOf(<d:life-of> <d:constant-participant>)
 SubPropertyOf(<b:e-temporal-location-of> <d:exact-location-of>)
 SubPropertyOf(j:unifies c:references)
 SubPropertyOf(<e:adopts-plan> k:adopts)
 SubPropertyOf(<m:state-of> m:substrate)
 SubPropertyOf(<n:place-of> <n:approximate-location-of>)
 SubPropertyOf(a:unit <d:r-location>)
 SubPropertyOf(<m:functional-participant-in> <d:participant-in>)
 SubPropertyOf(<c:in-scope-of> <c:referenced-by>)
 SubPropertyOf(<d:intensionally-referenced-by> <c:referenced-by>)
 SubPropertyOf(<m:prescribed-by> <m:performed-by>)
 SubPropertyOf(m:theme m:patient)
 SubPropertyOf(<m:has-state> <m:substrate-of>)
 SubPropertyOf(<n:d-spatial-location> <d:exact-location>)
 SubPropertyOf(<f:q-represents> <d:mediated-relation>)
 SubPropertyOf(<b:started-by> <b:temporally-includes>)
 SubPropertyOf(<e:task-postcondition> <d:mediated-relation>)
 SubPropertyOf(<d:direct-predecessor> d:predecessor)
 SubPropertyOf(<e:main-goal-of> <d:proper-part-of>)
 SubPropertyOf(<d:mereologically-coincides> <d:partly-compresent>)
 SubPropertyOf(<d:total-constant-participant-in> <d:constant-participant-in>)
 SubPropertyOf(<c:temporary-component-of> <c:component-of>)
 SubPropertyOf(j:member <d:generic-constituent>)
 SubPropertyOf(<c:classified-by> <d:intensionally-referenced-by>)
 SubPropertyOf(f:orders <d:mediated-relation-i>)
 SubPropertyOf(d:life <d:constant-participant-in>)
 SubPropertyOf(d:predecessor <d:immediate-relation-i>)
 SubPropertyOf(<c:admitted-by> <d:mediated-relation-i>)
 SubPropertyOf(<n:material-place> n:place)
 SubPropertyOf(<c:p-sat> c:satisfies)
 SubPropertyOf(<d:specific-constant-constituent-of> <d:immediate-relation-i>)
 SubPropertyOf(<d:boundary-of> <d:proper-part-of>)
 SubPropertyOf(e:discards <c:d-uses>)
 SubPropertyOf(<c:expected-by> <d:mediated-relation-i>)
 SubPropertyOf(c:expands <d:proper-part>)
 SubPropertyOf(<d:temporary-participant-in> <d:participant-in>)
 SubPropertyOf(<d:host-of> <d:specific-constant-dependent>)
 SubPropertyOf(m:target m:patient)
 SubPropertyOf(c:admits <d:mediated-relation>)
 SubPropertyOf(<c:optionally-uses> <c:d-uses>)
 SubPropertyOf(<c:expected-setting-for> <d:mediated-relation>)
 SubPropertyOf(<n:descriptive-place-of> <n:approximate-location-of>)
 SubPropertyOf(i:enforces <c:involved-in>)
 SubPropertyOf(<d:mereologically-coincides> <d:temporary-part>)
 SubPropertyOf(k:creates c:conceives)
 SubPropertyOf(<d:q-location-of> <d:immediate-relation-i>)
 SubPropertyOf(<d:proper-part-of> <d:part-of>)
 SubPropertyOf(<c:expressed-by> <c:referenced-by>)
 SubPropertyOf(<n:descriptive-origin-of> <n:descriptive-place-of>)
 SubPropertyOf(<d:r-location> <d:immediate-relation>)
 SubPropertyOf(<j:covered-by> <d:immediate-relation-i>)
 SubPropertyOf(<d:participant-in> <d:immediate-relation-i>)
 SubPropertyOf(<d:specifically-constantly-dependent-on> <d:immediate-relation-i>)
 SubPropertyOf(m:patient <m:functional-participant>)
 SubPropertyOf(d:participant <d:immediate-relation>)
 SubPropertyOf(<m:used-in> <m:functional-participant-in>)
 SubPropertyOf(<c:modal-target-of> <d:immediate-relation-i>)
 SubPropertyOf(<p:causally-follows> b:follows)
 SubPropertyOf(a:duration <b:temporal-location>)
 SubPropertyOf(c:deputes <d:immediate-relation>)
 SubPropertyOf(k:regulates <c:satisfied-by>)
 SubPropertyOf(<d:specific-constant-dependent> <d:immediate-relation>)
 SubPropertyOf(j:covers <d:immediate-relation>)
 SubPropertyOf(<d:partly-compresent> <d:mediated-relation>)
 SubPropertyOf(<d:identity-n> <d:immediate-relation>)
 SubPropertyOf(<m:substrate-of> <d:total-constant-participant-in>)
 SubPropertyOf(<d:physical-location-of> <d:exact-location-of>)
 SubPropertyOf(<i:enforced-by> c:involves)
 SubPropertyOf(<a:time-of-happening-of> <b:time-of-presence-of>)
 SubPropertyOf(<d:intensionally-references> c:references)
 SubPropertyOf(c:references <d:immediate-relation>)
 SubPropertyOf(<n:participant-place-of> <n:approximate-location-of>)
 SubPropertyOf(<c:temporary-component> <d:partly-compresent>)
 SubPropertyOf(<j:characterized-by> <d:immediate-relation-i>)
 SubPropertyOf(h:bdi <c:attitude-towards>)
 SubPropertyOf(<p:causally-precedes> b:precedes)
 SubPropertyOf(<k:has-method> <c:expected-by>)
 SubPropertyOf(<h:desire-towards> <c:attitude-towards>)
 SubPropertyOf(c:sequences c:classifies)
 SubPropertyOf(<d:generic-constituent-of> <d:immediate-relation-i>)
 SubPropertyOf(<a:happens-at> <b:present-at>)
 SubPropertyOf(<a:duration-of> <b:temporal-location-of>)
 SubPropertyOf(<d:generic-location> <d:mediated-relation>)
 SubPropertyOf(<e:exit-condition> d:successor)
 SubPropertyOf(m:performs <m:functional-participant-in>)
 SubPropertyOf(<d:temporary-proper-part-of> <d:proper-part-of>)
 SubPropertyOf(<f:refers-to> <d:mediated-relation>)
 SubPropertyOf(<n:d-spatial-location-of> <d:exact-location-of>)
 SubPropertyOf(<d:atomic-part> d:part)
 SubPropertyOf(<m:functional-participant> d:participant)
 SubPropertyOf(<k:adopted-by> <c:conceived-by>)
 SubPropertyOf(<c:interpreted-by> <d:immediate-relation-i>)
 SubPropertyOf(<d:temporary-proper-part-of> <d:temporary-part-of>)
 SubPropertyOf(<c:aboutness-of> <c:referenced-by>)
 SubPropertyOf(<d:has-quale> <d:q-location>)
 SubPropertyOf(c:component <d:proper-part>)
 SubPropertyOf(<f:referred-by> <d:mediated-relation-i>)
 SubPropertyOf(b:follows <b:temporal-relation-i>)
 SubPropertyOf(<n:approximate-location-of> <d:generic-location-of>)
 SubPropertyOf(k:adopts c:conceives)
 SubPropertyOf(c:defines <c:d-uses>)
 SubPropertyOf(c:requires <d:immediate-relation>)
 SubPropertyOf(c:specializes <d:immediate-relation>)
 SubPropertyOf(m:resource <m:use-of>)
 SubPropertyOf(<d:temporary-atomic-part-of> <d:temporary-proper-part-of>)
 SubPropertyOf(<d:spatio-temporally-present-at> <d:exact-location>)
 SubPropertyOf(<k:empowered-for> <c:attitude-target-of>)
 SubPropertyOf(<d:has-t-quality> <d:has-quality>)
 SubPropertyOf(<b:e-temporal-location> <d:exact-location>)
 SubPropertyOf(<e:exit-condition-of> d:predecessor)
 SubPropertyOf(<n:p-spatial-location> <d:exact-location>)
 SubPropertyOf(<d:inherent-in> <d:immediate-relation>)
 SubPropertyOf(<d:has-quality> <d:immediate-relation-i>)
 SubPropertyOf(<d:total-constant-participant> <d:constant-participant>)
 SubPropertyOf(d:successor <d:immediate-relation>)
 SubPropertyOf(c:expects <d:mediated-relation>)
 SubPropertyOf(<d:direct-successor> d:successor)
 SubPropertyOf(<c:value-for> <c:classified-by>)
 SubPropertyOf(d:boundary <d:proper-part>)
 SubPropertyOf(<m:performed-by> <m:functional-participant>)
 SubPropertyOf(d:part <d:immediate-relation>)
 SubPropertyOf(d:overlaps <d:mediated-relation>)
```

```
 SubPropertyOf(<n:approximate-location> <d:generic-location>)
 SubPropertyOf(<d:identity-c> <d:immediate-relation>)
 SubPropertyOf(c:parametrizes <d:mediated-relation>)

)
```

6.2 OWL-RDF abstract syntax of the sample Klett model

```
 Individual(Klett:acquire_a_development_plan
  type(e:goal))
 Individual(Klett:acquire_idea
  type(d:plan))
 Individual(a:assistant
  type(k:agent-driven-role)
  value(k:duty-to Klett:disburden_project_manager))
 Individual(a:author
  type(k:agent-driven-role)
  value(k:obligation-to Klett:provide_content)
  value(k:obligation-to Klett:providing_info_on_administrative_issues_regarding_content))
 Individual(Klett:bring_high_profits_to_Klett
  type(e:goal))
 Individual(Klett:compilation_of_new_learning_material
  type(e:complex-task))
 Individual(Klett:compile_development_plan
  type(e:action-task))
 Individual(Klett:concept_design
  type(d:plan)
  value(d:d-uses Klett:technical_project_manager)
  value(d:d-uses Klett:project_manager)
  value(d:d-uses Klett:assistant)
  value(d:d-uses Klett:author)
  value(d:d-uses Klett:standard)
  value(c:proper-part Klett:acquire_a_development_plan)
  value(d:defines Klett:compile_development_plan)
  value(d:defines Klett:providing_info_on_administrative_issues_regarding_content)
  value(d:defines Klett:disburden_project_manager)
  value(d:defines Klett:provide_info_on_technical_standards)
  value(d:defines Klett:decide_on_content))
 Individual(Klett:concept_development
  type(d:plan)
  value(d:d-uses Klett:technical_project_manager)
  value(d:d-uses Klett:project_manager)
  value(d:d-uses Klett:assistant)
  value(d:d-uses Klett:disburden_project_manager)
  value(d:d-uses Klett:author)
  value(d:d-uses Klett:provide_info_on_technical_standards)
  value(d:d-uses Klett:standard)
  value(c:proper-part Klett:develop_a_pilot_version_of_new_learning_material)
  value(d:defines Klett:provide_content)
  value(d:defines Klett:coordinate_compilation_of_new_learning_material)
  value(d:defines Klett:set_deadlines_to_authors))
 Individual(Klett:coordinate_compilation_of_new_learning_material
  type(e:any-order-task)
  value(a:direct-successor Klett:compilation_of_new_learning_material))
 Individual(Klett:decide_on_content
  type(e:action-task))
 Individual(Klett:develop_a_pilot_version_of_new_learning_material
  type(e:goal))
 Individual(Klett:disburden_project_manager
  type(e:action-task))
 Individual(Klett:producing_1_piece_of_new_learning_material
  type(d:plan)
  value(c:proper-part a:sales)
  value(c:proper-part a:production)
  value(c:proper-part Klett:bring_high_profits_to_Klett)
  value(c:proper-part Klett:concept_design)
  value(c:proper-part Klett:concept_development)
  value(c:proper-part Klett:acquire_idea))
 Individual(Klett:production
  type(d:plan))
 Individual(Klett:project_manager
  type(k:agent-driven-role)
  value(k:obligation-to Klett:coordinate_compilation_of_new_learning_material)
  value(k:obligation-to Klett:set_deadlines_to_authors)
  value(k:right-to Klett:decide_on_content)
  value(k:duty-to Klett:compile_development_plan))
 Individual(Klett:provide_content
  type(e:action-task))
 Individual(Klett:provide_info_on_technical_standards
  type(e:action-task))
 Individual(Klett:providing_info_on_administrative_issues_regarding_content
  type(e:action-task))
 Individual(Klett:right_as_value
  type(d:parameter)
  value(a:requisite-for a:standard))
 Individual(Klett:sales
  type(d:plan))
 Individual(Klett:set_deadlines_to_authors
  type(e:action-task))
 Individual(Klett:standard
  type(d:role))
 Individual(Klett:technical_project_manager
  type(k:agent-driven-role)
  value(k:obligation-to Klett:provide_info_on_technical_standards))
```