METOKIS
Methodology and Tools Infrastructure for the Creation of Knowledge Units

DELIVERABLE

Information Society Technologies

Task Taxonomies for Knowledge Content
D07

Task Number - Name	**T31, T21, T22, T23, T24**
Issuing Date	**June 30th, 2004**
Distribution	**Public**
Document Web Link	
File name	**D07_v11.doc**
Author/Editor	**Aldo Gangemi, Stefano Borgo, Carola Catenacci, Jos Lehmann**
QA Mentor	
Lead/Contact Person	**Aldo Gangemi**

Change History

Version	Status	Comments / Changes
	final	Signed off for delivery to project officer
	draft	
1.1	wip	Intermediate version

Table of Content

1 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 can be initially defined as any information object - as a whole or explicitly wrt to its parts - that can be tagged by means of metadata. More or less, most of the *Semantic Web* programme deals with knowledge content, but a clearcut understanding of the nature and types of content, and of what metadata should represent, is still faraway.
In Metokis, the two concepts match for the scope of the project: building a demonstration platform that allows a formal definition of certain types of content (news, clinical data, management documents, etc.), in conjunction with a formal definition of the action types (and their ordering) that involve that kind of 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 contents 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 D&S 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, 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].
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. Planners constructed through Problem Solving Methods.
b. the Act Formalism.
c. the Shared Planning and Activity Representation.
d. the Core Plan Representation.
e. the Process Specification Language.
f. the PLAN semantic NET.
g. the Enterprise Ontology.
h. the Ontology with Polymorphic Types.
i. the Core Ontology of Services.
j. 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+D&S 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.

2.3 DDPO: DOLCE+D&S Plan Ontology

DOLCE+D&S Plan Ontology (DDPO) specializes the concepts and relations defined in DOLCE, and some of its extensions, notably the Ontology of Descriptions and Situations (D&S). DDPO, like D&S, 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-physical object.
2. A **situation** is a **setting** for any number of entities.
3. The **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 **selects** 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 select in DOLCE.
7. **Figures** are concepts that do not select 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 roles or 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 selected 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 select at least some entity in the situation setting. There exist a basic typology for the satisfaction relation between situations and descriptions: P-SAT means **proactively satisfies**, R-SAT means **retroactively satisfies**, and C-SAT 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 intentional agentive role or figure, and at least one course towards which the 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 selects 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: **scheduling**, **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] Eco, U., Kant e l'ornitorinco, Milano, Bompiani, 1997.

[21] Jakobson, R.: Linguistics and Poetics: Closing Statement. In: Style in Language. MIT Press, Cambridge, MA (1960).

3 Basic axiomatization for the plan ontology

3.1 Introduction

The DOLCE+D&S Plan Ontology (DDPO) specializes the concepts and relations defined in DOLCE [14], and some of its extensions, notably the Ontology of Descriptions and Situations (D&S).
DDPO, like D&S, 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 D&S extension of DOLCE
- the D&S 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.

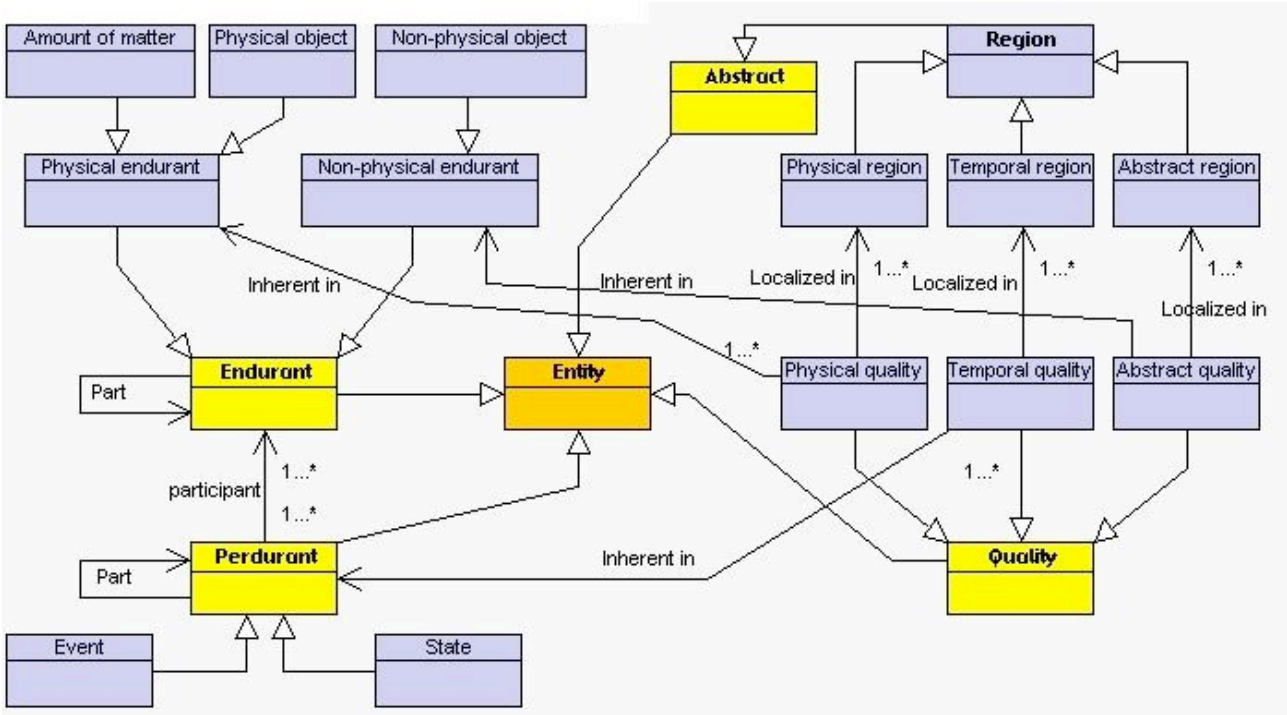


Fig. 1 A UML class diagram with top-level concepts and some relations defined in the DOLCE foundational ontology. Yellow nodes represent the categories. Unlabelled arrows are IS_A (subclass-of) relationships.

3.2 The DOLCE foundational ontology

DOLCE (Descriptive Ontology for Language and Cognitive Engineering) has been developed in the context of the WonderWeb project [17] by the Laboratory for Applied Ontology [18]. 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. **Entities** are the individuals of the DOLCE ontology domain. Entities can be as varied as possible: particular (e.g. *Italy*) or abstract (e.g. *empty set*), 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 categories of DOLCE entities are: **Endurant**, **Perdurant**, **Quality**, and **Abstract**.
Endurants are entities in space, which participate at least in one perdurant (e.g. substances, objects, social entities, concepts).
Perdurants are entities in time, which have at least one participant (e.g. events, states, processes, phenomena).
Qualities are dependent entities, “inherent” in either endurants or perdurants (e.g. actual colors, weights, speeds, etc.).
Abstracts are entities 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 here is the 370).

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

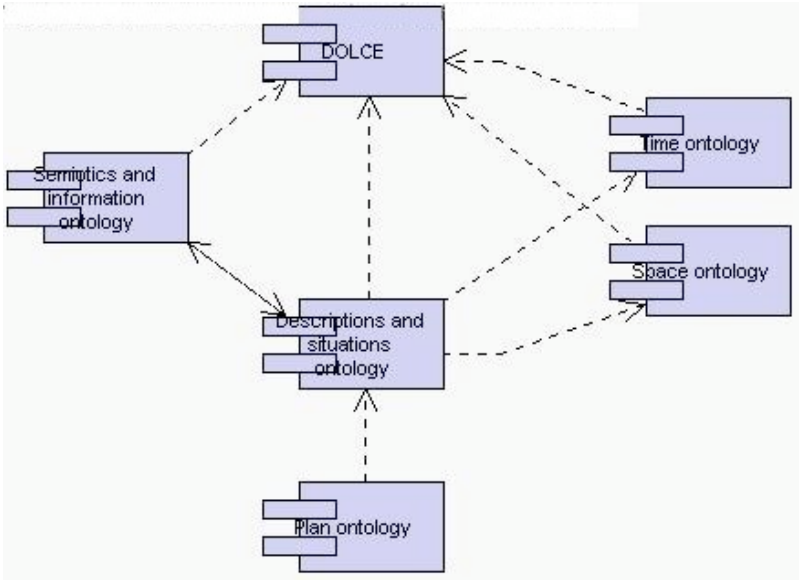


Fig. 2 A UML component diagram showing the main modules of DOLCE-Lite-Plus. Arrows represent dependencies.

3.3 Basic concepts and relations of the D&S Ontology

Since DDPO heavily relies on D&S, we present it here with some detail (but refer to [16] and to the OWL annex for completeness).

3.3.1 Basic notions

Descriptions and Situations (D&S) is an extension of DOLCE whose main intent is enabling the ontological talk about non-physical, social and especially *knowledge* objects. The rationale is that the properties that we attribute to entities are entities as well, and we can treat them as “knowledge” or “information” objects.
D&S has been built (differently from most ontologies) 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).
The lower part of the pattern is called the *ground ontology*, the higher is called the *descriptive ontology*: a *situation* constituted by entities in the ground ontology satisfies a *description* if the two parts of the pattern match, according to specified matching rules.
In the following, D&S is explained wrt to DOLCE concepts.

A **description** is a non-physical object (in particular, it is a non-agentive social object), which represents a conceptualization, hence it is *generically dependent* (GD) on some agent, and which is also social, i.e. communicable [16]:

$Description(x) \rightarrow NonAgentiveSocialObject(x)$
$Description(x) \rightarrow \exists y.\ AgentivePhysicalObject(y) \wedge GD(x,y)$
$Description(x) \rightarrow \forall y.\ Part(x,y) \rightarrow NonPhysicalObject(y)$

Example of descriptions are beliefs, desires, plans, laws, diagnoses, projects, plots, techniques, system specifications, ecosystem rules, product designs, etc.
Like physical objects, non-physical ones have a lifecycle, can have parts, etc. Differently from physical objects, non-physical ones are dependent on some agentive physical object that is able to **conceive** them.
Descriptions are **generically dependent on** (GD) objects that **conceive** them.

$ConceivesOf(x,y) \rightarrow GD(y,x) \wedge Object(x) \wedge Description(y)$

Agentivity in DOLCE is not defined, but in D&S we can add an axiom to characterize it:

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

Simply put, agentivity is taken in a wide sense as implying *conception* (to be characterized in a dedicated – but not yet developed – ontology of mind). An internal representation only requires *intentionality* (capacity to represent something to oneself).
A stronger sense of agentivity involves conceiving of *plans* (see below). This is compliant to e.g. BDI paradigm when attributes to **cognitive agents** the ability of self representing beliefs, desires, and intentions:

CognitiveAgentivePhysicalObject(x) $=_{df}$ AgentivePhysicalObject(x) ∧ ∃y. Plan(y) ∧ ConceivesOf(x,y)

Conception can be held by *agentive social objects* as well, through the cognitive agentive physical objects they depend on:

(ConceivesOf(x,y) ∧ AgentiveSocialObject(x)) → ∀z. (CognitiveAgentivePhysicalObject(z) ∧ GD(x,z)) → ConceivesOf(z,y) ∧ ∃z. CognitiveAgentivePhysicalObject(z)

On the other hand, the way agents create, choose, or transform their conceptualizations (the nature of intentionality) 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. An important relation between agents and descriptions is **creation**, implying that the description is specifically dependent (SD) on the agent:

Creates(x,y) → ConceivesOf(x,y) ∧ SD(y,x) ∧ CognitiveAgentivePhysicalObject(x) ∧ Description(y)

Another important relation between agents and descriptions is **adoption**:

Adopts(x,y) → ConceivesOf(x,y) ∧ Agent(x) ∧ Description(y) ∧ ∃z. CognitiveAgentivePhysicalObject(z) ∧ Creates(z,y)

Descriptions have typical components, called *concepts* (see below). Concept types can vary according to the ground ontology that is taken into account. This version of D&S takes DOLCE as its ground ontology.

A **situation** is an entity 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 aims at representing the referent of a “cognitive disposition” towards a world, i.e. 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 for the ways of defining the *satisfies* relation), and it has to be the *setting* for at least one entity from the ground ontology:

Situation(x) $=_{df}$ Entity(x) ∧ (∃y. Description(y) ∧ Satisfies(x,y)) ∧ (∃z. Entity(z) ∧ Setting(z,x))
Situation(x) → ∀y. Part(x,y) → Situation(y)

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).
All the remaining machinery of D&S tries to answer the question: «*how to formally represent the (possible, actual, obliged, desired, etc.) correspondence between situations and descriptions*?».

Description	***Situation***
Theory	Model
Proposition	Fact
Relation	Relationship
Belief	State of affairs
Desire	(Desired) state
Plan	Plan execution
Workflow	Work being done
Legal Norm	Legal case

Law of nature	Fact in nature
Contract	Contract enforcement
Diagnosis	Diagnostic case
Project	Project undertaking
Play	Performance
Script	Movie
Coding system	Information encoding
Communication rules	Communication setting
Technique	(Technical) activity
Instruction	(Guided) activity
Rules of game	Play a game
System specification	System functioning
Constraints in an ecosystem	Ecosystem
Product design	Finished working product

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

A situation is a **setting** for any number of entities, and at least one perdurant:

Setting(x,y) → Entity(x) ∧ Situation(y)
SettingFor(x,y) → ∃z. Perdurant(z) ∧ SettingFor(x,z)

The **time and space of a situation** are the time and space of the entities in the setting:

∀p,s,t1,t2. (Perdurant(p) ∧ TimeInterval(t1) ∧ TimeInterval(t2) ∧ TemporalLocation(p,t1) ∧ TemporalLocation(s,t2) ∧ Setting(p,s)) → Part(t2,t1)
∀e,s,r1,r2. (Endurant(e) ∧ SpaceRegion(r1) ∧ SpaceRegion(r2) ∧ SpatialLocation(e,r1) ∧ SpatialLocation(s,t2) ∧ Setting(e,s)) → Part(r2,r1)

Implicitly, the previous axioms state that a situation has a temporal (resp. spatial) location that is the mereological sum of the locations of the entities in the setting. For example, the time of *World War II* might span from its originating events to Yalta conference; 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 theorems.

A **concept** is also a non-agentive social object, which is "defined by" (see below) a description:

Concept(x) → NonAgentiveSocialObject(x)
Concept(x) → ∃y. Description(y) ∧ Defines(y,x)

Examples of concepts are *manager*, *employee*, *student*, *driver*, a *routine*, a *task*, a *schedule*, a *speed limit*, an *age restriction*, etc.

The **selects** relation relates concepts and entities (then possibly even concepts). In [16] it is called "classifies":

Selects(x,y) → Concept(x) ∧ Entity(y)
Selects(x,y) → ∃z. TimeInterval(z) ∧ Selects(x,y,t)

The Selects relation should be *temporalized*, but ternary relationships are not supported by OWL-DL, then we assume that time will be managed in an extrinsic way in practical applications.[1]
There are relations between concepts. For example, some concepts are apparently selected by other concepts, e.g. a *manager* that plays the role of *buyer*. In most cases they are not selected, but they are actually **subconcepts**. The subconcept relation holds between concepts:

SubconceptOf(x,y) → Concept(x) ∧ Concept(y)

[1] Temporal indexing should apply on many relations defined here, but since time must be ignored in the OWL-DL version, we skip it here.

A special case is **specialization**, which applies to the abovementioned concepts: *manager specializes buyer* (cf. [16] for more examples):

Specializes(x,y) → SubconceptOf(x,y)
Specializes(x,y) → ∀z. (Entity(z) ∧ Selects(x,z)) → Selects(y,z)
Specializes(x,y) → ¬(x=y)

There are several kinds of concepts reified in D&S, the primary ones (**role**, **course**, and **parameter**) being distinguished by the categories they select in DOLCE:

Role(x) $=_{df}$ Concept(x) ∧ ∀y. Selects(x,y) → Endurant(y)
Course(x) $=_{df}$ Concept(x) ∧ ∀y. Selects(x,y) → Perdurant(y)
Parameter(x) $=_{df}$ Concept(x) ∧ ∃y. Selects(x,y) ∧ ∀y. Selects(x,y) → Region(y)

Examples of roles[2] 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.

Figures are social objects (either agentive or not), defined or used by descriptions, but differently from concepts, they do not select entities:

Figure(x) → SocialObject(x)
Figure(x) → ∃y. Description(y) ∧ Defines(y,x)
Figure(x) → ¬∃y. Selects(x,y)

Examples of figures are organisations, political geographic objects, sacred symbols, etc.
Agentive figures are those which are assigned (agentive) roles from a society or community; hence, they can *act* like a physical agent:

AgentiveFigure(x) → Figure(x) ∧ ∃y,z,w. Description(y) ∧ Role(z) ∧ Description(w) ∧ y≠w ∧ Defines(y,z) ∧ Defines(w,x) ∧ Selects(z,x)

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:

DeputedBy(r,f) → Role(r) ∧ Figure(f) ∧ ∃d. Description(d) ∧ Uses(d,r) ∧ Uses(d,f)
DeputedBy(r,f) → ∃r1. Role(r1) ∧ Selects(r1,f)
ActsFor(e,f) → ∃r. Role(r) ∧ DeputedBy(r,f) ∧ Selects(r,e)

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.[3]
Since figures are social objects, it is conceivable to find figures that *act for* other figures.[4]
Since descriptions and concepts are (social) objects (hence *endurants*), they can be classified by a role in another description. This recursivity allows to manage meta-level descriptions in D&S (e.g. a *norm* for enforcing norms will define a role that can classify the *enforced norm*).

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).

[2] There are additional axioms to characterize roles as *anti-rigid* and *founded* concepts. For definitions of anti-rigidity and of foundation, see [16].
[3] Although the cases of full delegation or representation are quite unusual, and even prohibited in some legal contexts.
[4] Indeed this situation is at work in many contemporary settings, and can arrive at a great complexity, e.g. in financial *chinese boxes*, which can even create an *agency loop*.

In order to introduce collections, we need a **membership** relation:

Membership(e,o) $=_{df}$ Endurant(e) ∧ SocialObject(o) ∧ Constituent(o,e) ∧ ∃r. Role(r) ∧ Selects(r,e)
Collection(x) $=_{df}$ SocialObject(x) ∧ ∀w. Membership(w,x) → ∃r. Role(r) ∧ Selects(r,w) ∧ ∃y,z. Endurant(y) ∧ Endurant(z) ∧ y≠z ∧ Membership(y,x) ∧ Membership(z,x) ∧ Selects(r,y) ∧ Selects(r,z)

In other words, a collection is a social object whose members are all classified by a same role, and at least two endurants are actually members.
The role shared by members has therefore a **covering** relation towards the collection:

Covers(r,c) $=_{df}$ Role(r) ∧ Collection(c) ∧ ∀w. Membership(w,c) → Selects(r,w)

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

Organized collections can be conceived that are *characterized* by other roles that are played by some (or all) members of the collection, and are related among them through the social objects (figures, descriptions, collections) that either use or depute or are covered by them:

Characterizes(r,c) $=_{df}$ Role(r) ∧ Collection(c) ∧ ∃e,f,o,s. (Figure(o) ∨ Description(o) ∨ Collection(o)) ∧ Role(s) ∧ e≠f ∧ r≠s ∧ Membership(e,c) ∧ Membership(f,c) ∧ (Uses(o,r) ∨ Deputes(o,r) ∨ CoveredBy(o,r)) ∧ (Uses(o,s) ∨ Deputes(o,s) ∨ CoveredBy(o,s)) ∧ Classifies(r,e,t) ∧ Classifies(s,f,t)
Characterizes(r,c) → ∃s. Role(s) ∧ r≠s ∧ Characterizes(s,c)
OrganizedCollection(c) $=_{df}$ Collection(c) ∧ ∃r,s. Characterizes(r,s) ∧ Characterizes(s,c)

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

A **collective** is a collection of *agents* (either agentive physical objects or agentive figures or even collectives, recursively):

Agent(x) $=_{df}$ AgentivePhysicalObject(x) ∨ AgentiveFigure(x) ∨ Collective(x)
Collective(c) $=_{df}$ Collection(c) ∧ ∀x. (Membership(x,c) → Agent(x)

If a role of a collection member is deputed by a figure, that member can *act for* that figure. For each agentive figure, a collection can be conceived as the (reification of the) maximal set of agents that act for the figure. We can then introduce the notion of **maximal agency collection**:

MaximalAgencyCollection(c) $=_{df}$ Collective(c) ∧ (∀x. Membership(x,c) → ∃f. Figure(f) ∧ ActsFor(x,f)) ∧ ∃y,z. Membership(y,c) ∧ Membership(z,c)

The definition says that a maximal agency collection is a collection that has only members that act for a same figure, and at least two of them exist.
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.

A **component** relation (to be read: “x has component y”) is a proper part relation qualified by a description in which the proper parts are involved. In other words, *component* may be equivalent to *functional part*:

Component(x,y) $=_{df}$ ProperPart(x,y) ∧ ∃d,z,w. Description(d) ∧ Role(z) ∧ Role(w) ∧ Uses(d,z) ∧ Uses(d,w) ∧ Selects(z,x) ∧ Selects(w,y)

We can say e.g. that an *engine* is a component of a *car*, because there is a design or a system specification that defines roles for the car and the engine. Nonetheless, if an ontology does not have any commitment on car designs (e.g. a traffic norm about the admitted quality of engine smoke emission), we can still say that an engine is just a part of a car, or we can use *component* by just "postulating" that design (this is feasible in so-called *description logics*, like OWL-DL).

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 on its turn is an object with functional parts, etc.):

Uses(x,y) ↔ Component(x,y) ∧ Description(x) ∧ Concept(y)

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

Defines(x,y) → Uses(x,y) ∧ SD(y,x)
Uses(x,y) → ∃z. Description(z) ∧ Defines(z,y)

For example, a *car design* can define an *engine role* that can select a specified class of *artifacts*. Notice that, since descriptions and concepts are (non-physical) objects, they can also be selected by a role in another description. This recursivity allows to manage meta-level descriptions in D&S (e.g. a *norm* for enforcing norms will define a role that can select the *enforced norm*).

Selects is specialized by three subrelations: **played by**, **sequences**, and **valued by**, for three different categories in DOLCE (Endurant, Perdurant, and Region)[5]:

PlayedBy(x,y) $=_{df}$ Role(x) ∧ Endurant(y) ∧ Selects(x,y)
Sequences(x,y) $=_{df}$ Course(x) ∧ Perdurant(y) ∧ Selects(x,y)
ValuedBy(x,y) $=_{df}$ Parameter(x) ∧ Region(y) ∧ Selects(x,y)

Roles or figures, and courses are related by relations expressing the **attitudes** that roles or figures can have **towards** a course:

AttitudeTowards(x,y) → (Role(x) ∨ Figure(x)) ∧ Course(y)

Attitude towards is the descriptive counterpart of the 'participant-in' relation used in the ground ontology, i.e. attitudes are *participation modes*.
In other words, the AttitudeTowards relation can be used to state *beliefs*, *attitudes*, *attention* or *subjection* that an object can have wrt an action or process.
For example, a person is usually *obliged* to drive in a way that prevents hurting otherpersons. Or a person can have the *right* to express her ideas. Another, 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 and roles, figures, or courses are related by a **requisite for** relation, expressing the kind of requisites entities that are selected by roles or courses should have:

RequisiteFor(x,y) → Parameter(x) ∧ (Role(x) ∨ Figure(x) ∨ Course(y))

Requisites are constraints over the attributes of entities.
When a situation satisfies a description that uses parameters, endurants and perdurants that constitute the situation must have attributes that range within the boundaries stated by parameters (in DOLCE terms, entities must have qualities that are mapped to certain value ranges of regions).

[5] Only three categories from DOLCE have been assigned a concept type at the descriptive layer, because the resulting design pattern is simpler, and no relevant knowledge seems to be lost, at least in applications developed until now.

For example, a *speed limit of 60kmph* can be a requisite for a *driving* task; the satisfying situation will have to constrain any *speed* of e.g. *travelling within Rome by car* to be less or equal to *60kmph*.

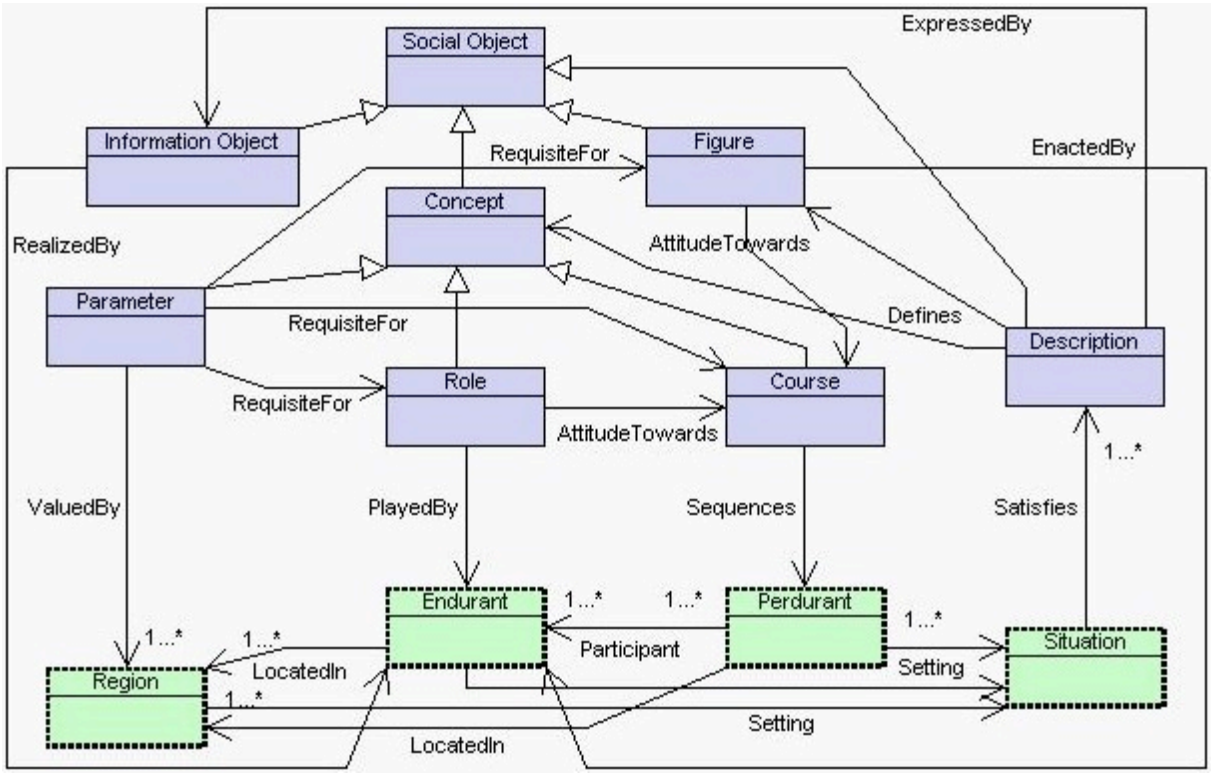


Fig. 3 The D&S Design Pattern as a UML class diagram. The lower part of the pattern is called the *ground ontology*, the higher is called the *descriptive ontology*; a situation satisfies a description if the two parts match according to specified rules. Part of the structure matching expected by situations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

Information objects are other social objects, **encoded by** special descriptions called *combinatorial systems* (or *codes*), which are able to **express** descriptions and other social objects. Information objects are **realized by** entities whose properties match those required by the combinatorial system (see section 5 for a more detailed ontology of information objects):

InformationObject(x) → SocialObject(x)
InformationObject(x) → ∃y. Description(y) ∧ EncodedBy(x,y)
InformationObject(x) → ∃y. Entity(y) ∧ RealizedBy(x,y)
Description(x) → ∃y. InformationObject(y) ∧ ExpressedBy(x,y)

Role-playing endurants are **involved** in descriptions:

Involves(x,y) $=_{df}$ Description(x) ∧ Endurant(y) ∧ ∃z. Role(z) ∧ Uses(x,z) ∧ Selects(z,y)

3.3.2 Satisfaction in D&S

The **satisfies** (SAT) relation holds between situations and descriptions, and implies that at least some components in a description must select at least some entity in the situation setting:

SAT(x,y) → Situation(x) ∧ Description(y)
SAT(x,y) → ∃z. Component(y,z) ∧ ∃w. Setting(w,x) ∧ Selects(z,w)

This constraint is very poor, and for specialised descriptions additional constraints should be given in order to reason over the satisfaction of candidate situations (see below the constraints for plans).
In general, D&S does not constrain situations to include *only* entities selected by description components.
This assumption may seem rough (redundant situations will be acceptable), but real world uses of D&S have shown that most situations derive from pre-existing situations that already have an internal

structure, and depriving them under the sole purpose of getting non-redundant situations seems a bad practice. For example, a *detective report* may contain useless informations from the point of view of a certain *legal rule*, but to a certain extent, it is important to preserve the *unity* of the report, instead of "cleaning" it up to a new entity that merely satisfies the legal rule description.
In addition, the same practice usually applies to physical objects: provided that they respect some basic properties, any other property results acceptable: 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!*").
Under this assumption, the same situation can satisfy different descriptions that can even be unrelated.

D&S can be applied as an ontology of **systems**.
As a matter of fact, a system builds upon existing structures, be it in the physical (*natural systems*, *material artifacts*), social (*organizations*, *societies*), or cognitive (*knowledge*) worlds.
Therefore, in order to talk about systems, we need one or more reference layer(s) that represent those worlds, which entities are assumed to belong to, each coming with their own structural (and even functional) organization. The entities from a layer are usually assembled according to the system's functional constraints at the next layer.
The many-layered nature of D&S seems to fit this necessity, specially since the elements belonging to different layers are allowed to coexist in the same ontological domain, and one can reason on them in a first-order logic.

In order to get a clear semantical foundation of these (functional) constraints - when they appear in a domain together with physical objects, events, and qualities - D&S exploits the logical mechanism of **reification**.
The semantics of reified theories and models that we are willing to accept for D&S models can be summarized as follows.
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 (Fig.4). If a tuple in M generates a contradiction in T, M is not a model of T (Fig.5). 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 (Fig.6).
When moving to D&S, more possibilities arise:

- **purely reified satisfaction**: this simple case corresponds to axiomatic theory satisfaction: each entity in a situation S must be selected by a concept in a description D, and each tuple asserted between the entities of S must correspond to some relation between concepts (attitude towards, requisite for, successor, etc.) in D. No further individuals and tuples are allowed in S

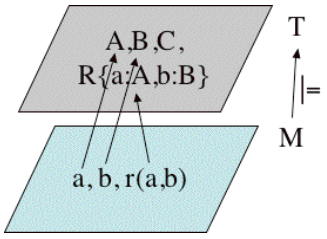


Fig. 4 Purely reified satisfaction is equivalent to being a (valid) model M for a theory T: e.g. a is an instance of A, b is an instance of B, and the tuple a,b for the relation r conforms to the domain and range of R.

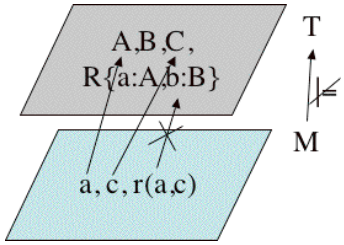


Fig. 5 Purely reified satisfaction failing is equivalent to failing to be a (valid) model M for a theory T: e.g. a is an instance of A, c is an instance of C, but the tuple a,c for the relation r does not conform to the range of R (B).

- **redundant reified satisfaction**: in this case one may accept that there are entities and tuples in S which do not correspond to concepts and relations in D; in practice, we accept to have undecidable models for a theory

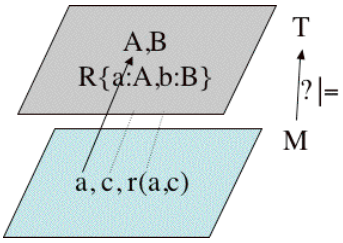


Fig. 6 Redundant reified satisfaction is equivalent to being an undecidable model M for a theory T: e.g. a is an instance of A, but c and the tuple a,c for the relation r are undecidable within T, because the predicate C is not within its vocabulary.

- **structure matching satisfaction**: in this case, each concept in D must select an entity in S, and each relation between concepts in D corresponds to appropriate relations in S; in practice, we use a theory as a protocol instead of a set of constraints over possible world models (Fig.7): the world in S must conform to D

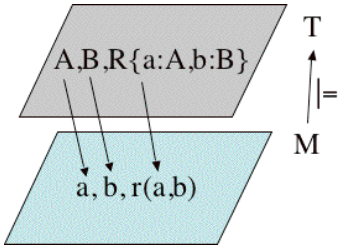


Fig. 7 Structure matching satisfaction is equivalent to being a (valid) model M for a theory T: e.g. a is an instance of A, b is an instance of B, and the tuple a,b for the relation r conforms to the domain and range of R. But, differently from purely reified satisfaction, here *each* element in T *must*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

- **qualified satisfaction**: in this case a set of axioms is provided that specifies what part of the structure in D must be matched by S; in practice, this is also a structure matching, but some concepts and relations in D are considered either **optional**, or **discarded** by decision (Fig.8)

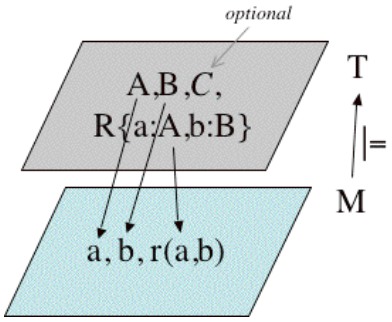


Fig. 8 Qualified satisfaction is similar to structure matching satisfaction, but some *specified* (e.g. optional tasks) elements in T have no necessary correspondence in M.

With reference to these cases of satisfaction, D&S 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 seems to give room for 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 elements are selected by those concepts, except for *red jacket*, which is anyway already decided 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.

3.3.3 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. The three types introduced are: **proactively satisfies** (*P-SAT*), meaning that the situation has a *pre hoc* description, **retroactively satisfies** (*R-SAT*), meaning that the situation has a *post hoc* description, and **constructively satisfies** (*C-SAT*) meaning that the situation has an *ad hoc* description:

P-SAT(x,y) → SAT(x,y)
R-SAT(x,y) → SAT(x,y)
C-SAT(x,y) → SAT(x,y)

Provided that the space-time of a situation is the mereological sum of the spatial and temporal regions of the entities in the setting of a situation, we can figure out the following axioms for the SAT subrelations:

P-SAT 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-SAT requires that the description exists prior to at least some of the entities in the setting of the satisfying situation. Ontologically, it results that P-SAT also implies a specific dependency of the situation on its description.
P-SAT typically applies to plans, projects, designs, methods, techniques, rules of game, instructions, punishment rules, constitutive descriptions, sanctions, strategies, etc.:

P-SAT(x,y) → ∃e. Entity(e) ∧ Setting(e,x) ∧ ∃t1,t2. PresentAt(e,t1) ∧ PresentAt(y,t2) ∧ t2<t1
P-SAT(x,y) → SD(x,y)

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

P-SAT(x,y) → ∃t1,t2. PresentAt(x,t1) ∧ PresentAt(y,t2) ∧ t2<t1

R-SAT also assumes *redundant satisfaction* and *qualified satisfaction*, but it works out that semantics with entities in the situation that entirely exist prior to the description. This seems paradoxical, since a description hardly motivates what happens if it is not present to any agent involved in things happening. For this reason, we postulate a so-called **specific retroactive 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 it could not be present before the situation.
R-SAT 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:

R-SAT(x,y) → ∀e,t1. (Entity(e) ∧ Setting(e,x) ∧ PresentAt(y,t1)) → ∃t2. PresentAt(e,t2) ∧ t2<t1

SRD(x,y) $=_{df}$ D(x,y) ∧ ∃t1,t2. PresentAt(x,t1) ∧ PresentAt(y,t2) ∧ t1<t2
R-SAT(x,y) → SRD(x,y)

C-SAT - like R-SAT - concerns entities that exist in a situation entirely prior to the description. Moreover, it assumes redundant satisfaction. But, differently from P-SAT and R-SAT, no qualified satisfaction is assumed. In fact, C-SAT implies no dependency of a situation on its description.
C-SAT 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.:

C-SAT(x,y) → ∀e,t1. (Entity(e) ∧ Setting(e,x) ∧ PresentAt(y,t1)) → ∃t2. PresentAt(e,t2) ∧ t2<t1

C-SAT(x,y) → ¬D(x,y)

While each SAT 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 can P-SAT it.

SAT 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 D&S as being the abstract specification of a *system* 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 in the components of a system will take into account that those components should allow for the *enactment* of a P-SAT 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.
The grounding for a regulation should allow for *checking* a C-SAT 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.
The grounding for an explanation should allow for either *retrieving* or *simulating* (previewing) an R-SAT 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 SAT subrelations concerns the production of optimal descriptions according to available resources, a task addressed by either classical planning and 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-SAT from given elements of potential situations, which aims to discover a description equivalent to the best plan that can be P-SATed by any new situation that includes those elements.

On the other hand, these considerations will be made more practical during the implementation of the Metokis case studies.

3.4 The Plan Ontology

The plan ontology depends on the D&S ontology, and specializes it with tasks, goals, P-SAT rules, etc.

3.4.1 Plans and goals

A **plan** is a description that defines or uses at least one task (see below) and one agentive role or figure, and that has at least one goal as a part:

Plan(x) → Description(x)
Plan(x) → ∃t. Task(t) ∧ Uses(x,t)
Plan(x) → ∃c. ((AgentiveRole(c) ∨ Figure(c)) ∧ Uses(x,c)
Plan(x) → ∃g. Goal(g) ∧ 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 varied proper parts (regulations, goals, laws), including other plans:

Subplan(x) $=_{df}$ Plan(x) ∧ ∃y. Plan(y) ∧ ProperPart(y,x)

If a plan uses a figure, that figure is defined by a constitutive description. If a plan defines a figure, the related constitutive description is a proper part of the plan:

ConstitutiveDescription(x) → Description(x)
∀x,f. (Plan(x) ∧ Figure(f) ∧ Uses(x,f) ∧ ¬Defines(x,f)) → ∃y. ConstitutiveDescription(y) ∧ Defines(y,f)

$\forall x,f.\ (Plan(x) \wedge Figure(f) \wedge Defines(x,f)) \rightarrow \exists y.\ ConstitutiveDescription(y) \wedge ProperPart(x,y) \wedge Defines(y,f)$

For example, 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 a desire (another kind of description) that is a part of a plan.
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:

$DesireTowards(x,y) \rightarrow AttitudeTowards(x,y)$
$Desire(x) \rightarrow Description(x)$
$Desire(x) \rightarrow \exists yz.\ ((IntentionalAgentiveRole(y) \vee IntentionalFigure(y)) \wedge Course(z) \wedge Uses(x,y) \wedge Uses(x,z) \wedge DesireTowards(y,z)$

For example, a *desire to start a relationship* can become a *goal of starting a relationship* if someone *takes action* (or lets someone else take it for her sake) to obtain it.
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:

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

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

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

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 it is a subgoal of:

$InfluenceOn(x,y) =_{df} Goal(x) \wedge Goal(y) \wedge \exists z.\ Plan(z) \wedge Subgoal(z,x) \wedge MainGoal(z,y)$

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

$DispositionTo(x,y) =_{df} AgentiveRole(x) \wedge Goal(y) \wedge \exists p,g.\ Plan(p) \wedge Goal(g) \wedge ProperPart(p,g) \wedge Uses(p,x) \wedge Goal(g) \wedge InfluenceOn(g,y)$

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 having that task 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 specialise the adoption relation. Goals and plans can be in fact adopted with different constraints:

$AdoptsGoal(x,y) =_{df} Agent(x) \wedge Goal(y) \wedge \forall z.\ (Course(z) \wedge Uses(y,z)) \rightarrow DesireTowards(x,z)$
$AdoptsPlan(x,y) =_{df} Agent(x) \wedge Plan(y)$

In those interesting cases, given a plan and its *main* goal (see below), 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:

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

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:

? AdoptsPlan(x,y) → ∃z. MainGoal(y,z) ∧ AdoptsGoal(x,z)
? AdoptsPlan(x,y) → ∀z. SubGoal(y,z) → AdoptsGoal(x,z)

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

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

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 selecting 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 of restrictions over the *classes* of entities that can be selected in the plan execution: this is what we call an *abstract plan* (see below).
While a *tailored plan* in DDPO is equivalent to a so-called *circumstantial plan*, namely a plan that specifies each entity that can be selected 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 select any *named entity*. This condition cannot be formalized in FOL, since the following axiom:

AbstractPlan(x) → Plan(x) ∧ ∀yz. ((Role(y) ∧ Uses(x,y)) → (Endurant(z) ∧ Selects(y,z))) ∧ ∀wk. ((Task(w) ∧ Uses(x,w)) → (Perdurant(k) ∧ Selects(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* selects *some (if any) instance of person*.

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.):

CircumstantialPlan(x) $=_{df}$ Plan(x) ∧ ∀y. (Concept(y) ∧ Uses(x,y)) → ∃z. Entity(z) ∧ Selects(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 tasks selects an event that is valued by a definite temporal value in a definite space region:

SaturatedPlan(x) $=_{df}$ Plan(x) ∧ ∃y,z. Parameter(y) ∧ Parameter(z) ∧ Uses(x,y) ∧ Uses(x,z) ∧ ∃t,s. ValuedBy(y,t) ∧ TimeInterval(t) ∧ ¬∃t1. TimeInterval(t1) ∧ ValuedBy(y,t1) ∧ t≠t1 ∧ ValuedBy(z,s) ∧ SpaceRegion(s) ∧ ¬∃s1. SpaceRegion(s1) ∧ ValuedBy(y,s1) ∧ s≠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.[6]

Plan executions are situations that proactively satisfy a plan (cf. definition of P-SAT above):

PlanExecution(x) $=_{df}$ Situation(x) ∧ ∃y. Plan(y) ∧ P-SAT(x,y)

Subplan executions are parts of the whole plan execution:

∀p1,p2,s1,s2. (Plan(p1) ∧ Plan(p2) ∧ ProperPart(p1,p2) ∧ P-SAT(p1,s1) ∧ P-SAT(p2,s2)) → ProperPart(s1,s2)

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

GoalSituation(x) $=_{df}$ Situation(x) ∧ ∃y. Goal(y) ∧ SAT(x,y)

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

GoalSituation(x) → ∀y,p,s. (Goal(y) ∧ SAT(x,y) ∧ Plan(p) ∧ ProperPart(p,y) ∧ P-SAT(s,p)) → ¬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:

∀p1,p2,s1¬∀s2. (Plan(p1) ∧ Plan(p2) ∧ ProperPart(p1,p2) ∧ SAT(p1,s1) ∧ SAT(p2,s2)) → ProperPart(s1,s2)

This helps to account for the following cases:

- Execution of plans containing *abort* or *suspension* conditions (in those cases, the plan is 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:

Precondition(p,s) → Plan(p) ∧ Situation(s)
Precondition(p,s) → ∀s1. (PlanExecution(s1) ∧ P-SAT(s1,p)) → (∃d. SAT(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:

Postcondition(p,s) → Plan(x) ∧ Situation(s)
Postcondition(p,s) → ∀s1. (PlanExecution(s1) ∧ P-SAT(s1,p)) → (∃d. SAT(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, implying that, for all plan executions of that

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

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:

AccompanyingCondition(p,s,t) → Plan(p) ∧ Situation(s) ∧ Task(t)
AccompanyingCondition(p,s,t) → Defines(p,t)
AccompanyingCondition(p,s,t) → ∀s1. (PlanExecution(s1) ∧ P-SAT(s1,p) ∧ s≠s1) → (∃d,e. SAT(s,d) ∧ Perdurant(e) ∧ Sequences(t,e) ∧ Setting(e,s1) ∧ Precedes(s,e))

3.4.2 Tasks

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

Task(x) $=_{df}$ Course(x) ∧ ∃y,z. Plan(y) ∧ Defines(y,x) ∧ ((IntentionalAgentiveRole(z) ∨ IntentionalFigure(z)) ∧ Uses(y,z) ∧ DesireTowards(z,x)

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 **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:

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

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

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

ComplexTask(x) $=_{df}$ Task(x) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z∧ Component(x,y) ∧ 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. BTW, even if two tasks have a direct successor relation holding for them, the actions sequenced by them could overlap temporally:

DirectSuccessor(x,y) → Entity(x) ∧ Entity(y)
Successor(x,y) → Entity(x) ∧ Entity(y)

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

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

SequentialTask(x) $=_{df}$ ComplexTask(x) ∧ (∀y,z. (Component(x,y) ∧ Component(x,z) ∧ y≠z) → (Successor(y,z) ∨ Successor(z,y))) ∧ (¬∃w. Component(x,w) ∧ 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 as components.

HybridTask(x) $=_{df}$ ComplexTask(x) ∧ ∃y,z. Component(x,y) ∧ Component(x,z) ∧ y≠z ∧ ControlTask(y) ∧ 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:

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

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.

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

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

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:

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

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):

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

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 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* (duty+right) towards a control task that has some complex task as a direct successor.
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**):

$LoopTask(x) =_{df} ControlTask(x) \wedge \exists y,z,w,\mathcal{P}.\ Task(y) \wedge Action(z) \wedge Action(w) \wedge z \neq w \wedge DirectSuccessor(x,y) \wedge Sequences(y,z) \wedge Sequences(y,w) \wedge MinimalCommonType(z,w,\mathcal{P})$

For example, "*repeat the poem on and on*" is a complex task controlled by a loop task.
Notive 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:

$ExitCondition(x,y) \rightarrow LoopTask(x) \wedge DeliberationTask(y)$
$IterationInterval(x,y) \rightarrow LoopTask(x) \wedge TimeInterval(y)$
$IterationInterval(x,y) \rightarrow \forall z.\ (Action(z) \wedge Successor(x,z)) \rightarrow TemporalLocation(z,y)$
$IterationCardinality(x,y) \rightarrow {}^{*}LoopTask(x) \wedge Integer(y)$

A **cyclical task** is a complex task that is controlled by a loop task, and has a case task as a component. The case task specifies the exit condition(s) of the cyclical task indirectly (only the

decisions that haven't the cyclical task as successor are exit conditions), while a loop task specifies which is the exit condition:

CyclicalTask(x) $=_{df}$ ComplexTask(x) ∧ ∃y,z,w. LoopTask(y) ∧ DirectSuccessor(y,x) ∧ CaseTask(z) ∧ DirectSuccessor(z,y) ∧ Component(x,z) ∧ DeliberationTask(w) ∧ DirectSuccessor(z,w) ∧ Component(x,w) ∧ ExitCondition(y,w)

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

A **branching task** is a control task that articulates a complex task into an ordered set of tasks:

BranchingTask(x) $=_{df}$ ControlTask(x) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(x,y) ∧ DirectSuccessor(x,z)

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 decision state as outcome (sequenced by a **deliberation task**):

CaseTask(x) $=_{df}$ BranchingTask(x) ∧ (∀y. Sequences(x,y) → DecisionActivity(y)) ∧ ∀z. DirectSuccessor(x,z) → DeliberationTask(z)

CaseTask(x) → ∃y,z,w,k. DeliberationTask(y) ∧ DeliberationTask(z) ∧ y≠z ∧ DirectSuccessor(x,y) ∧ DirectSuccessor(x,z) ∧ ActionTask(w) ∧ ActionTask(k) ∧ w≠k ∧ DirectSuccessor(y,w) ∧ DirectSuccessor(z,k) ∧ ∀e1,e2. Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ Sequences(w,e1) ∧ Sequences(k,e2) ∧ ¬Overlaps(e1,e2)

DeliberationTask(x) $=_{df}$ ControlTask(x) ∧ (∀y. (Sequences(x,y) → DecisionState(y)) ∧ ∃z. DirectSuccessor(z,x) ∧ CaseTask(z)

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.

An **alternate task** is a case task branching to exactly two deliberation tasks:

AlternateTask(x) $=_{df}$ CaseTask(x) ∧ ∃y,z. DeliberationTask(y) ∧ DeliberationTask(z) ∧ y≠z ∧ DirectSuccessor(x,y) ∧ DirectSuccessor(x,z) ∧ ¬∃w. w≠y ∧ w≠z ∧ DeliberationTask(w) ∧ DirectSuccessor(x,w)

A **concurrency 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 concurrency 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:

ConcurrencyTask(x) $=_{df}$ BranchingTask(x) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(x,y) ∧ DirectSuccessor(x,z) ∧ ∀e1,e2. (Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ Sequences(y,e1) ∧ Sequences(z,e2)) → Overlaps(e1,e2)

ConcurrencyTask(x) → ∃y. SynchroTask(y) ∧ Successor(x,y)

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

SynchroTask(x) $=_{df}$ ControlTask(x) ∧ ∃t1,t2,t3. (ConcurrencyTask(t1) ∨ AnyOrderTask(t1)) ∧ Successor(t1,x) ∧ (ComplexTask(t2) ∨ ActionTask(t2)) ∧ (ComplexTask(t3) ∨ ActionTask(t3)) ∧ DirectSuccessor(t2,x) ∧ DirectSuccessor(t3,x)

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.

A **parallel task** is a concurrent task branching to at least two tasks that sequence temporally coinciding perdurants:

ParallelTask(x) $=_{df}$ ConcurrencyTask(x) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(x,y) ∧ DirectSuccessor(x,z) ∧ ∀e1,e2. (Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ Sequences(y,e1) ∧ Sequences(z,e2)) → Coincides(e1,e2)

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 an any order task:

AnyOrderTask(x) $=_{df}$ BranchingTask(x) ∧ ∃y,z. Task(y) ∧ Task(z) ∧ y≠z ∧ DirectSuccessor(x,y) ∧ DirectSuccessor(x,z) ∧ ∀e1,e2. (Perdurant(e1) ∧ Perdurant(e2) ∧ e1≠e2 ∧ Sequences(y,e1) ∧ Sequences(z,e2)) → TemporalRelation(e1,e2)

AnyOrderTask(x) → ∃y. SynchroTask(y) ∧ Successor(x,y)

A **beginning task** is a control task that is the predecessor of all tasks defined in the plan:

BeginningTask(x) $=_{df}$ ControlTask(x) ∧ ∀y,p. (Task(y) ∧ Plan(p) ∧ Component(p,x) ∧ Component(p,y) ∧ x≠y) → Successor(x,y)

An **ending task** is a control task that has no successor tasks defined in the plan:

EndingTask(x) $=_{df}$ ControlTask(x) ∧ ∀p¬∃y. (Task(y) ∧ Plan(y) ∧ Component(p,x) ∧ Component(p,y) ∧ x≠y) → Successor(y,x)

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

MaximalTask(x) $=_{df}$ ComplexTask(x) ∧ ∀y. (Task(y) ∧ Component(p,y)) → Part(x,y)

3.4.3 Satisfaction in DDPO

The SAT relation (as well as its subrelations) is usually constrained for special classes of descriptions, 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-SAT is provided here:

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

Intuitively, we are suggesting that for a plan to be satisfied, we require that the following components select some entitiy in the situation setting: *i)* all parameter for tasks, *ii)* at least one role, *iii)* no optional or discarded tasks, *iv)* all control tasks, *v)* all action tasks that are not bound by a control task, *vi)* all action tasks bound by a synchronization task, *vii)* at least one action task from any set bound by the same branching task, *viii)* at least one ending task.

Notice that we are not including the satisfaction of the plan's goal among the P-SAT 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 one ending task selects a perdurant (e.g. abortion, suspension). Additional axioms can catch these notions:

AbortionTask(x) → EndingTask(x)
SuspensionTask(x) → EndingTask(x)
CompletionTask(x) → EndingTask(x)

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

∀x,p,e. (CompletionTask(x) ∧ Plan(p) ∧ Defines(p,x) ∧ PlanExecution(e) ∧ P-SAT(e,p)) → ∃g,s. GoalSituation(s) ∧ Goal(g) ∧ SAT(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 no ending task could 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:

OptionallyUsedBy(x,y) → UsedBy(x,y) ∧ Concept(x) ∧ Description(y)

Of course, within plans an optional task should be placed in a way that preserves the topology (the connectedness) of the maximal task: in fact, an optional task can appear only as a direct successor to a concurrent task or an any order task:

OptionallyUsedBy(x,y) → ∃z. (ConcurrencyTask(z) ∨ AnyOrderTask(z)) ∧ Component(y,z) ∧ DirectSuccessor(z,x)

For example, "*eat your watermelon slice anyway you like, possibly without using your hands at all*" are action tasks controlled by a 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:

DiscardedWithin(x,y) → UsedBy(x,y) ∧ Task(x) ∧ Plan(y)
DiscardedWithin(x,y) → ∃z. DeliberationTask(z) ∧ Component(y,z) ∧ DirectSuccessor(z,x)

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

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

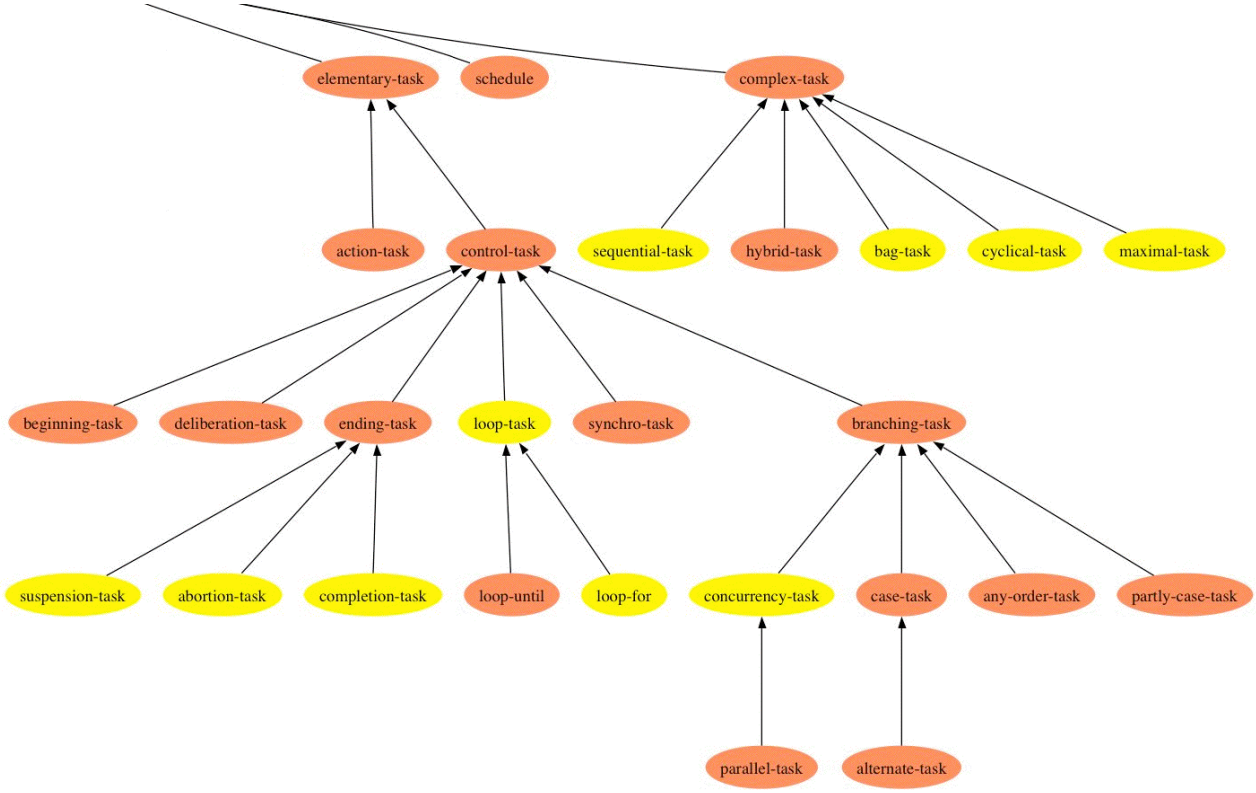


Fig. 9 Taxonomy of tasks in DDPO. Orange nodes represent *completely defined* OWL classes (having necessary and sufficient conditions). Yellow nodes represent *partially defined* OWL classes (having only necessary conditions). Arrows represent subclass-of (IS_A) relations.

3.4.4 Plan composition

It is possible to represent **plan composition**, when two plans are defined separately, and 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:

$TaskJuxtaposition(x,y,z) =_{df} MaximalTask(x) \wedge MaximalTask(y) \wedge x \neq y \wedge DirectSuccessor(x,y) \wedge ProperPart(z,x) \wedge ProperPart(z,y)$

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

$TaskSum(x,y,z) =_{df} \forall w,k.\ (MaximalTask(x) \wedge MaximalTask(y) \wedge x \neq y \wedge Task(w) \wedge Task(k) \wedge w \neq k \wedge Component(x,w) \wedge Component(y,k)) \rightarrow Component(z,w) \wedge Component(z,k)$

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:

$TaskProduct(x,y,z) =_{df} \forall w,k.\ (MaximalTask(x) \wedge MaximalTask(y) \wedge x \neq y \wedge ActionTask(w) \wedge ActionTask(k) \wedge w \neq k \wedge Component(x,w) \wedge Component(y,k)) \rightarrow (Component(z,w) \wedge Component(z,k) \wedge ((Successor(w,k) \vee Successor(k,w)) \vee (\exists t.\ ConcurrencyTask(t) \wedge DirectSuccessor(t,w) \wedge DirectSuccessor(t,k)) \vee (\exists t.\ AnyOrderTask(t) \wedge DirectSuccessor(t,w) \wedge DirectSuccessor(t,k)))$

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.:

Refines(x,y) → ProperPart(y,x) ∧ (Concept(x) ∧ Concept(y)) ∨ (Figure(x) ∧ Figure(y))
TaskJuxtaposition(x,y,z) → (Refines(z,x) ∧ Refines(z,y))
TaskSum(x,y,z) → (Refines(z,x) ∧ Refines(z,y))
TaskProduct(x,y,z) → (Refines(z,x) ∧ Refines(z,y))

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

Expands(x,y) → ProperPart(y,x) ∧ Description(x) ∧ Description(y)
Expands(x,y) → (∃z. Description(z) ∧ PropertPart(x,z) ∧ ¬PropertPart(y,z)) ∨ ∃w,k. Concept(w) ∧ Concept(k) ∧ Refines(k,w) ∧ UsedBy(k,x) ∧ UsedBy(w,y) ∧ ¬UsedBy(k,y) ∧ ¬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[7]:

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).

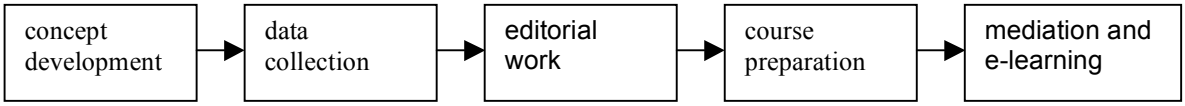
[7]From a DDPO point of view, 1 and 2 are better characterized as *descriptions* while 3 are *situations*.

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:

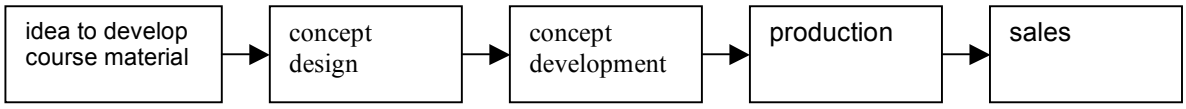


Each *SITUATION* is described in terms of its key actor(s), their roles, duties, rights, obligations and tasks. It should be noticed that, as put in Doc-1, there are some overlaps and/or unclear distribution of information between these entries and that some of the entries are "nested". For instance,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



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.

4.1.1 Schemas for Concept Design and Concept Development

CONCEPT DESIGN: 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.	**CONCEPT DEVELOPMENT:** If the business plan is accepted, a pilot version of the new learning material is developed which can be directly produced.
ROLES: • PROJECT MANAGER (PM): Duties/Responsibility: o compile the development plan Rights: o request information (from various other roles) o staff the project (involving other roles) o taking decisions on content and strategy Obligations: o compile a directly applicable development plan o organize and co-ordinate all other involved roles Goals/Motivation: 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): Duties/Responsibility:	**ROLES:** • PROJECT MANAGER (PM): Duties/Responsibility: o co-ordinate and plan steps of compilation of the new material Rights: 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 Obligations: 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 Goals/Motivation: 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): Duties/Responsibility:

- evaluate the technical perspective of the PM concept
- represent the technical strategy of KLETT

Rights:
- be involved in development of new technical concepts, i.e. make suggestions on, and enjoin in, themes concerning technical details

Obligations:
- support the PM with technical decisions, i.e. provide info so that the right standards can be met and the right decisions can be made

Goals/Motivation:
- guarantee corporate identity of technical software products and their performance

• ASSISTANT(S):

Duties/Responsibility:
Etc….

• EDITOR(S):

Duties/Responsibility:
Etc….

• AUTHOR(S):

Duties/Responsibility:
- provide his ideas, experience and knowledge of administrative issues as regards content of new material to editor(s), PM and/or assistant(s)

Rights:
- request info concerning his topic in his part of material
- be mentioned namely in the end product

Obligations:
- provide overview of content in his assigned sections of material
- provide an estimation of how long it will take to provide content
- commit himself to provide described content once the material will be developed

Goals/Motivation:
- be mentioned namely in the end product

- supervise the production of new material from a technical point of view

Rights:
- be involved in all technical decisions concerning new material
- declare technical standards for KLETT
- share decisions with the PM on kind of technology used

Obligations:
- support the PM in technical decisions, i.e. provide info so that the right standards can be met and the right decisions can be made

Goals/Motivation:
- guarantee corporate identity of technical software products and their performance

• ASSISTANT(S):

Duties/Responsibility:
Etc….

• EDITOR(S):

Duties/Responsibility:
Etc….

• AUTHOR(S):

Duties/Responsibility:
- provide elaborated content which will be integrated in the new material

Rights:
- request info concerning his task and part of material
- be mentioned namely in the end product
- be informed about timetable of concept development
- obtain clear guidance for compilation of his part

Obligations:
- provide content in the form he has committed to
- keep the deadlines he has commited to in previous phase (Concept Design)

Goals/Motivation:
- be mentioned namely in the end product

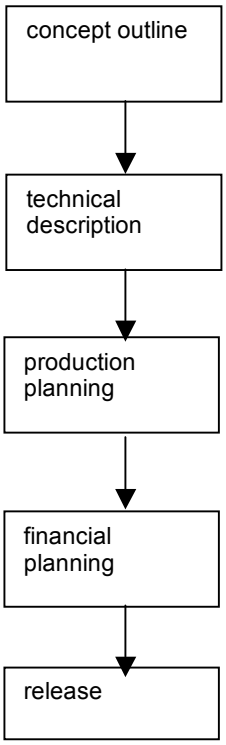
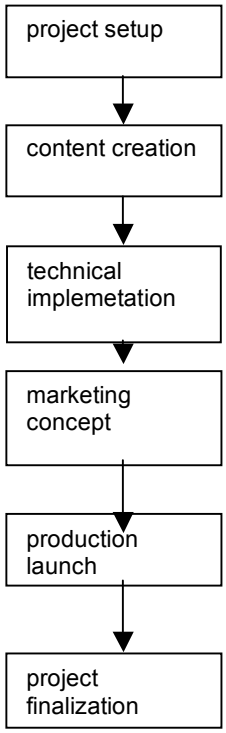
• CONSULTANT(S):

• CONSULTANT(S): Duties/Responsibility: Etc….	 Duties/Responsibility: Etc….

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 • (EXECUTIVE) TEAM[1] Roles: o PM, Assistant(s) Functions: o organize Concept Design o elaborate business plan • (EDITORIAL) TEAM[2] Roles: o PM, 2 to 10 Editors Functions: o make all decisions regarding content of new material (implies responsibility for content) . . . . • SUBTEAM[1] Roles: o TEAM[1], Organizational, Consultants(Controllers) [?]	**TEAM TYPES** • PROJECT MANAGEMENT TEAM[1] Roles: o PM + Assistant(s) Functions: o organize Concept Development o elaborate new material • 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:

Functions: ○ [control tasks?] • SUBTEAM^n Roles: ○ PM, each Editor, whole of TEAM^{2,} Author(s), Consultant(s) Functions: ○ collaborate on decisions regarding content of new material	○ [control tasks?] • SUBTEAM^n Roles: ○ PM, each Editor, whole of TEAM^{2,} Author(s), Consultant(s) Functions: ○ collaborate on decisions regarding content of new material

INTERACTION PHASES CONCEPT DESIGN	**INTERACTION PHASES CONCEPT DEVELOPMENT**
	

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)

AgentiveRole(project_manager)
AgentiveRole(technical_project_manager)
AgentiveRole(author)
AgentiveRole(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)

Right(project_manager, decide_on_content)
Obligation(project_manager, set_deadlines_to_authors)
Obligation(project_manager, coordinate_compilation_of_new_learning_material)
Obligation(technical_project_manager, provide_info_on_technical_standards)
Obligation(author, provide_info_on_administrative_issues_regarding_content)
Obligation(author, provide_content)
Duty(assistant, disburden_project_manager)
Duty(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.

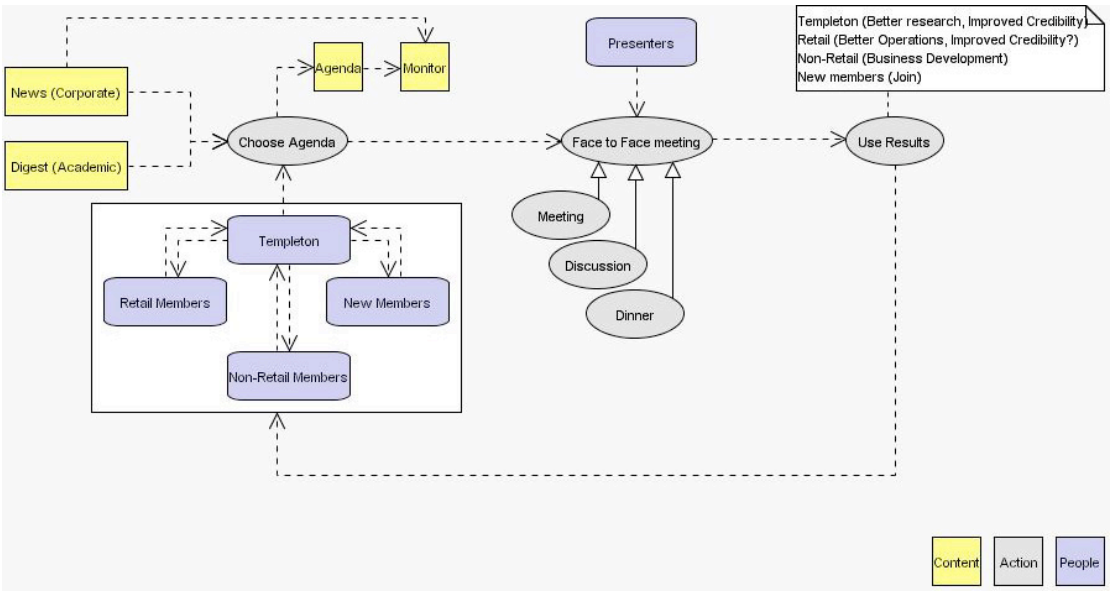


Fig. 10 Outline of ORFG system

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)

AgentiveRole(retail_executive)
AgentiveRole(non_retail_executive)
AgentiveRole(speaker)
AgentiveRole(supporting_staff)
AgentiveRole(potential_new_member)
AgentiveRole(representative)
AgentiveRole(oxirm_representative)
Specializes(representative, oxirm_representative)
Institution(oxirm)
Deputes(oxirm, oxirm_representative)
∀x,y. (RetailAssociation(x) ∧ Deputes(x,y) → y = retail_association_representative)
AgentiveRole(retail_association_representative)
Specializes(representative, retail_association_representative)
RetailAssociation(x) → 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 an advanced version of 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

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** IO, one or more *combinatorial structure(s)* (or **code**), according to which IO is **ordered**, a *meaning* (or *conceptualization*) that IO **expresses,** a *reference* that IO **is about**, and one or more agents that **interpret** IO (see [20] for a review of the relations between ontology and semiotics, and a similar account of semiotic relations):

Code(x) → Description(x)
InformationObject(x) → SocialObject(x)
InformationObject(x) → ∃y. Code(y) ∧ OrderedBy(x,y)
InformationObject(x) → ∃y. Entity(y) ∧ RealizedBy(x,y,t)
InformationObject(x) → ∀y. Expresses(x,y,t) → Description(y)
InformationObject(x) → ∀y. About(x,y,t) → Entity(y)
InformationObject(x) → ∀y. Interprets(y,x,t) → Agent(y)

For example, Dante's Comedy Italian text is an IO, *ordered* by Middle Age Italian (the code), *realized* by e.g. a paper copy of the 1861 edition with Doré's illustrations, *expresses* a certain plot and related meanings (literal or metaphorical), as *interpreted* (conceived) by an agent with an average knowledge of MA Italian and literary criticism, and is *about* facts like Dante's travel to the hereafter (Fig. 13).

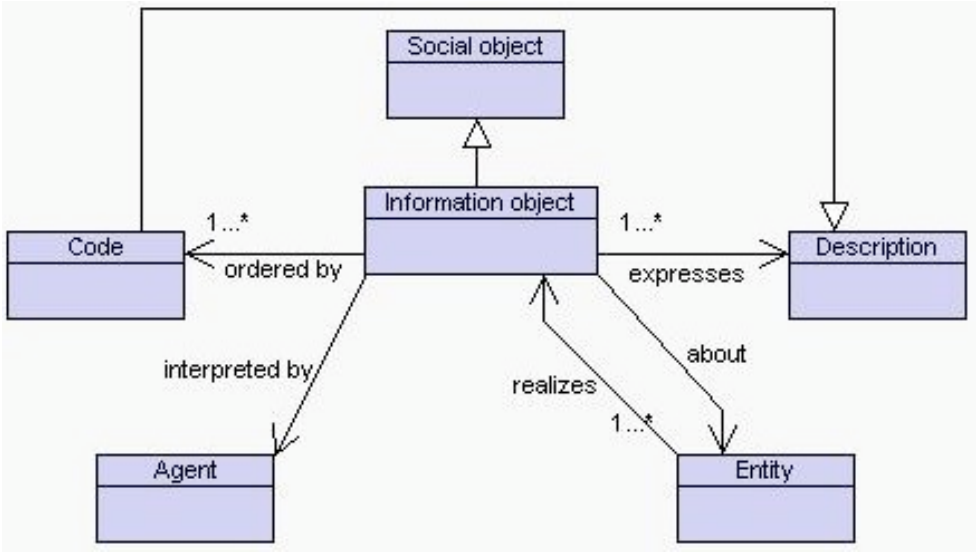


Fig. 11 The basic IO design pattern

Therefore, we are assuming the following functional reductions: *meanings* (anyhow a meaning is theorized: as a mental content, intertextual reference, propositional content, etc.) can be partly formalized as *descriptions*; *supports* are *entities* of some kind (e.g. a paper sheet, a sound, a sequence of bits or pulses, etc.); *referenced entities* are entities whatsoever (usually being *set* in *situations* that satisfy an expressed description); interpretations of IOs are confined to endurants that can have (directly or indirectly) *intentionality*: *agents*.

IOs are necessarily *encoded* by some code, and must be *realized* by some entity. 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 conceive of a description expressed by them.
This theory is compliant with so-called *communication elements* (roles) defined by Jakobson [21]: an information object works as *message*, a code as *code*, a supporting entity acts as *channel*, an interpreting agent works as *encoder* or *decoder*, and an expressed description as *context*.
Among the semiotic relations used above to create the IO pattern:

Orders(x,y) → Code(x)
Realizes(x,y,t) → Entity(x)
Expresses(x,y,t) → Description(y)
About(x,y,t) → Entity(x)
Interprets(x,y,t) → Agent(x)

realizes, *expresses*, *about*, and *interprets* must be taken as *temporally indexed*.
'Interprets' can be either *encoding* or *decoding*. Decoding follows encoding (if any):

Encodes(x,y,t) → Interprets(x,y)
Decodes(x,y,t) → Interprets(x,y)
Decodes(x,y,t) → $\forall z,t_1$. Encodes(z,y,t_1) → >(t,t_1)

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:

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 of a description expressed by the IO; of course two agents can conceive of different descriptions, then resulting in different interpretations):

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

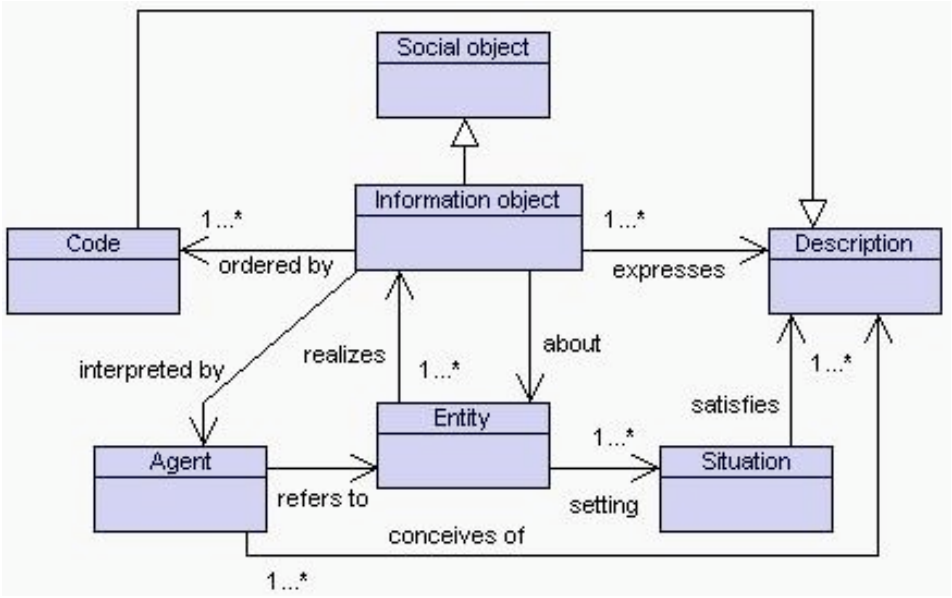


Fig. 12 The IO pattern with some implied paths: situations exist for the setting of *realization*, as well as *aboutness*; agents *refer* to entities that interpreted IOs are about, etc.

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 requires 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 entity the IO is about:

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** entities by conceiving of a description appropriate to context:

RefersTo(x,y,t) → Agent(x) ∧ Entity(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 whatever entities, provided that the structure of an entity is such that the ordering of an IO is properly mapped. On the other hand, an IO can be *about* whatever entities, provided that the entity can be situated into a situation that satisfies a conceivable description.[8]
Possibly, this twofold nature of entities wrt to semiotic properties accounts for the enormous expressive power of e.g. human codes: 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. Some structure is referred by using entities that realize IOs (*reconstruction,* or *representation*), while other structure is created in order to refer to other entities, including IOs themselves (*construction*).
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, then there is a sense in which any entity that realizes an IO also realizes an IO about itself.

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.
Of course, the converse of the previous axiom does not hold in general:

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

For example, the information about a woman can be realized by entities different from that woman (as when referring to an absent woman).
In other words, an entity (in a semiotic perspective) always realizes two information objects: one about itself, and another about something else. In the non-semiotic cases, the information objects are identical (an entity only realizes information about itself).
Therefore 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 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 the "intrinsic" aboutness for 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):

- the *1861 Edition copy* realizes the *Comedy*, but also itself, showing its attributes (paper, graphics, conservation state, etc.);
- interpreters include at least *Dante* as encoder and a *reader from 1861* as decoder;
- the decoder and the edition fit into a *communicative situation* from *1861*;
- while the encoder conceived of *Comedy's plot*, the reader in 1861 could have conceived an interpretation of the *Comedy* as an *initiatic travel* (as documented for instance in Rosicrucian circles)

[8] 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).

- the situation depicted of *Dante's visiting the hereafter* satisfies *Comedy's plot*, but not necessarily the *initiatic travel* reading, whose consistency depends on the conclusiveness of the literary bundle of descriptions that have enabled that reading

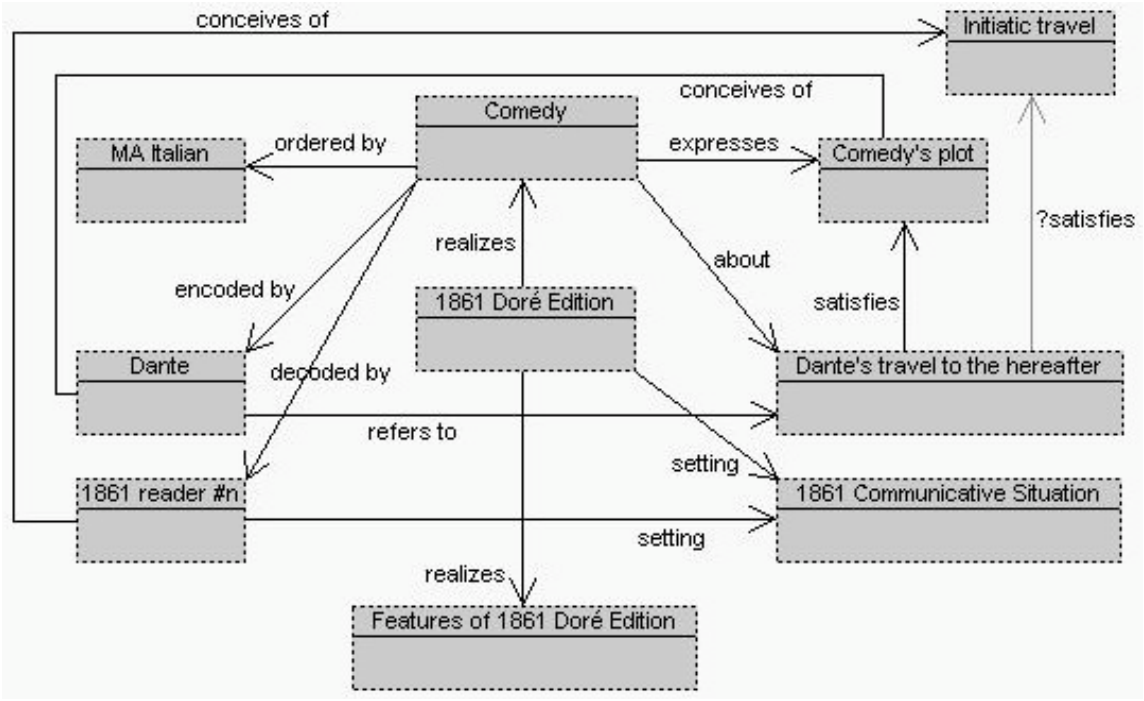


Fig. 13 A model about *Comedy* using the extended IO design pattern.

5.3 Using the IO design pattern

As a summary, we can conceive of a maximal semiotic relation S, and its minimal binary projection:

S^6(e,io,c,a,d,n,t) =$_{df}$ Entity(e) ∧ InformationObject(io) ∧ Code(c) ∧ Agent(a) ∧ Description(d) ∧ Entity(n) ∧ Realizes(e,io,t) ∧ OrderedBy(io,c) ∧ Interprets(a,io,t) ∧ Expresses(io,d,t) ∧ About(io,n,t)
S^2(e,n,t) =$_{df}$ Entity(e) ∧ Entity(n) ∧ ∃io,c,a,d. S(e,io,c,a,d,n,t)

The maximal projection of S can be used to query content bases whose schemas have been aligned to the IO design pattern.
The minimal projections can be used to assert (or query) simple facts, e.g. S^2(e,n,t) tells that a certain entity 'represents' another entity: this is a typical *loose* way of talking in common sense (but also in many databases).
In a *semantic data mining* perspective, S^2(e,n,t) can be enriched with data from the same or other databases that are expected on the basis of the maximal version S^6(e,io,c,a,d,n,t). For instance, knowing that an mp3 file has an S^2 relation to a performance from a John Coltrane's live concert, semantic data mining can use matching techniques to find the musical code (jazz, instruments, eventually style), the recording code (mp3), the encoder of the musical IO (John Coltrane), the encoder of the recording IO (a producer), the available decoders (a jazz critic), the tune played (e.g. *Impressions*), data about the musical setting (e.g. space and time of concert, duration of performance, etc.), data about the recording (duration, quality, etc.).

An interesting case of semiotic entrenchment has appeared in (computational) *knowledge representation* (KR). Computational (software) domain has a reality on 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:

Duck(x) ∧ Numerosity(x,30)
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[9]).
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. In next versions of the deliverable, we will take other examples, including multimedia content.

[9] 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 (version 374) in OWL-DL abstract syntax form. The concrete syntax version has been validated for OWL-DL, and checked for consistency and classified with FaCT++.

OWL Species Validation Report
URI: http://212.34.219.175/DLP374.owl

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://ontology.ip.rm.cnr.it/DOLCE-Lite-Plus#>)

Ontology( <http://212.34.219.175/DLP374.owl>

 ObjectProperty(a:abstract-location
  inverseOf(a:abstract-location-of)
  domain(a:non-physical-endurant)
  range(a:abstract-region))
 ObjectProperty(a:abstract-location-of
  inverseOf(a:abstract-location)
  domain(a:abstract-region)
  range(a:non-physical-endurant))
 ObjectProperty(a:admits
  inverseOf(a:admitted-by)
  domain(a:description)
  range(a:region))
 ObjectProperty(a:admitted-by
  inverseOf(a:admits)
  domain(a:region)
  range(a:description))
 ObjectProperty(a:approximate-location
  inverseOf(a:approximate-location-of)
  domain(intersectionOf(complementOf(a:region) a:entity))
  range(intersectionOf(complementOf(a:region) a:entity)))
 ObjectProperty(a:approximate-location-of
  inverseOf(a:approximate-location))
 ObjectProperty(a:attitude-target-of
  inverseOf(a:attitude-towards)
  domain(a:course))
 ObjectProperty(a:attitude-towards
  inverseOf(a:attitude-target-of)
  domain(unionOf(a:role a:figure))
  range(a:course))
 ObjectProperty(a:bdi
  inverseOf(a:bdi-target-of)
  domain(a:agentive-role)
  range(a:task))
 ObjectProperty(a:bdi-target-of
  inverseOf(a:bdi)
  domain(a:course)
  range(a:agent-case-role))
 ObjectProperty(a:boundary
  inverseOf(a:boundary-of))
 ObjectProperty(a:boundary-of
```

```
 inverseOf(a:boundary))
ObjectProperty(a:c-sat
 inverseOf(a:c-sat-by))
ObjectProperty(a:c-sat-by)
ObjectProperty(a:carried-by)
ObjectProperty(a:carries
 inverseOf(a:carried-by))
ObjectProperty(a:co-participates-with
 inverseOf(a:co-participates-with)
 domain(a:endurant)
 range(a:endurant))
ObjectProperty(a:coincides
 inverseOf(a:coincides)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:complete-participant
 inverseOf(a:complete-participant-in)
 range(a:endurant))
ObjectProperty(a:complete-participant-in
 inverseOf(a:complete-participant)
 domain(a:endurant))
ObjectProperty(a:component
 inverseOf(a:component-of))
ObjectProperty(a:component-of
 inverseOf(a:component))
ObjectProperty(a:concluded-by
 inverseOf(a:concludes))
ObjectProperty(a:concludes
 inverseOf(a:concluded-by))
ObjectProperty(a:consequence
 inverseOf(a:consequence-of)
 domain(a:perdurant))
ObjectProperty(a:consequence-of
 inverseOf(a:consequence)
 range(a:perdurant))
ObjectProperty(a:constant-participant
 inverseOf(a:constant-participant-in))
ObjectProperty(a:constant-participant-in
 inverseOf(a:constant-participant))
ObjectProperty(a:constituent
 inverseOf(a:constituent-of))
ObjectProperty(a:constituent-of
 inverseOf(a:constituent))
ObjectProperty(a:constrained-by
 inverseOf(a:constrains)
 domain(a:perdurant)
 range(a:regulation))
ObjectProperty(a:constrains
 inverseOf(a:constrained-by)
 domain(a:regulation)
 range(a:perdurant))
ObjectProperty(a:d-constituent
 inverseOf(a:d-constituent-of)
 domain(a:endurant)
 range(a:role))
ObjectProperty(a:d-constituent-of
 inverseOf(a:d-constituent)
 domain(a:role)
 range(a:endurant))
ObjectProperty(a:d-used-by
 inverseOf(a:d-uses))
ObjectProperty(a:d-uses
 inverseOf(a:d-used-by)
 domain(a:description)
 range(unionOf(a:figure a:concept)))
ObjectProperty(a:defined-by
 inverseOf(a:defines))
ObjectProperty(a:defines
 inverseOf(a:defined-by)
```

```
 domain(a:description)
 range(unionOf(a:figure a:concept)))
ObjectProperty(a:depend-on-spatial-location
 inverseOf(a:depend-on-spatial-location-of)
 domain(a:non-physical-endurant)
 range(a:space-region))
ObjectProperty(a:depend-on-spatial-location-of
 inverseOf(a:depend-on-spatial-location)
 domain(a:space-region)
 range(a:non-physical-endurant))
ObjectProperty(a:descriptive-depend-on-of
 inverseOf(a:descriptively-depends-on)
 domain(a:non-physical-endurant)
 range(a:endurant))
ObjectProperty(a:descriptive-origin
 inverseOf(a:descriptive-origin-of))
ObjectProperty(a:descriptive-origin-of
 inverseOf(a:descriptive-origin))
ObjectProperty(a:descriptively-depends-on
 inverseOf(a:descriptive-depend-on-of)
 domain(a:endurant)
 range(a:non-physical-endurant))
ObjectProperty(a:desire-target-of
 inverseOf(a:desire-towards))
ObjectProperty(a:desire-towards
 inverseOf(a:desire-target-of)
 domain(unionOf(a:agentive-role a:intentional-figure))
 range(a:course))
ObjectProperty(a:direct-predecessor
 inverseOf(a:direct-successor))
ObjectProperty(a:direct-successor
 inverseOf(a:direct-predecessor))
ObjectProperty(a:disability-target-of
 inverseOf(a:disability-towards))
ObjectProperty(a:disability-towards
 inverseOf(a:disability-target-of))
ObjectProperty(a:discarded-within
 inverseOf(a:discards)
 domain(a:task)
 range(a:plan))
ObjectProperty(a:discards)
ObjectProperty(a:duration
 inverseOf(a:duration-of)
 domain(a:perdurant))
ObjectProperty(a:duration-of
 inverseOf(a:duration)
 range(a:perdurant))
ObjectProperty(a:duty-target-of
 inverseOf(a:duty-towards))
ObjectProperty(a:duty-towards
 inverseOf(a:duty-target-of))
ObjectProperty(a:e-depend-on-of
 inverseOf(a:e-depends-on)
 domain(a:endurant)
 range(a:endurant))
ObjectProperty(a:e-depends-on
 inverseOf(a:e-depend-on-of)
 domain(a:endurant)
 range(a:endurant))
ObjectProperty(a:e-temporal-location
 inverseOf(a:e-temporal-location-of)
 domain(a:endurant)
 range(a:temporal-region))
ObjectProperty(a:e-temporal-location-of
 inverseOf(a:e-temporal-location)
 domain(a:temporal-region)
 range(a:endurant))
ObjectProperty(a:enforced-by)
ObjectProperty(a:enforces
```

```
 inverseOf(a:enforced-by)
 domain(a:institution)
 range(a:regulation))
ObjectProperty(a:exact-location
 inverseOf(a:exact-location-of)
 domain(a:entity)
 range(a:region))
ObjectProperty(a:exact-location-of
 inverseOf(a:exact-location)
 domain(a:region)
 range(a:entity))
ObjectProperty(a:exit-condition
 inverseOf(a:exit-condition-of)
 domain(a:loop-task)
 range(a:deliberation-task))
ObjectProperty(a:exit-condition-of
 inverseOf(a:exit-condition))
ObjectProperty(a:expanded-by
 inverseOf(a:expands))
ObjectProperty(a:expands
 inverseOf(a:expanded-by)
 domain(a:description)
 range(a:description))
ObjectProperty(a:expected-by
 inverseOf(a:expects)
 domain(a:perdurant)
 range(a:description))
ObjectProperty(a:expected-setting
 inverseOf(a:expected-setting-for)
 domain(unionOf(a:role a:parameter a:course))
 range(a:situation))
ObjectProperty(a:expected-setting-for
 inverseOf(a:expected-setting)
 domain(a:situation)
 range(unionOf(a:role a:parameter a:course)))
ObjectProperty(a:expects
 inverseOf(a:expected-by)
 domain(a:description)
 range(a:perdurant))
ObjectProperty(a:exploited-by
 inverseOf(a:exploits)
 domain(a:endurant)
 range(a:method))
ObjectProperty(a:exploits
 inverseOf(a:exploited-by)
 domain(a:method)
 range(a:endurant))
ObjectProperty(a:expressed-according-to
 inverseOf(a:expression-means-for)
 domain(a:information-object)
 range(a:combinatorial-system))
ObjectProperty(a:expression-means-for
 inverseOf(a:expressed-according-to)
 domain(a:combinatorial-system)
 range(a:information-object))
ObjectProperty(a:fiat-place
 inverseOf(a:fiat-place-of)
 domain(a:endurant)
 range(a:non-physical-endurant))
ObjectProperty(a:fiat-place-of
 inverseOf(a:fiat-place)
 domain(a:non-physical-endurant)
 range(a:endurant))
ObjectProperty(a:follows Transitive
 inverseOf(a:precedes)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:functional-depend-on-of
 inverseOf(a:functionally-depends-on)
```

```
 domain(a:role)
 range(a:role))
ObjectProperty(a:functional-participant
 inverseOf(a:functional-participant-in)
 domain(a:perdurant)
 range(a:endurant))
ObjectProperty(a:functional-participant-in
 inverseOf(a:functional-participant)
 domain(a:endurant)
 range(a:perdurant))
ObjectProperty(a:functionally-depends-on
 inverseOf(a:functional-depend-on-of)
 domain(a:role)
 range(a:role))
ObjectProperty(a:generic-depend-on-of
 inverseOf(a:generically-depends-on))
ObjectProperty(a:generic-location
 inverseOf(a:generic-location-of))
ObjectProperty(a:generic-location-of
 inverseOf(a:generic-location))
ObjectProperty(a:generic-target
 inverseOf(a:generic-target-of)
 domain(a:activity)
 range(a:endurant))
ObjectProperty(a:generic-target-of
 inverseOf(a:generic-target)
 domain(a:endurant)
 range(a:activity))
ObjectProperty(a:generically-depends-on
 inverseOf(a:generic-depend-on-of))
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(a:perdurant)
 range(a:time-interval))
ObjectProperty(a:has-member
 inverseOf(a:member-of))
ObjectProperty(a:has-method
 inverseOf(a:method-of)
 domain(a:activity)
 range(a:method))
ObjectProperty(a:has-quale
 inverseOf(a:quale-of)
 domain(a:quality)
 range(a:quale))
ObjectProperty(a:has-quality
 inverseOf(a:inherent-in)
 domain(a:entity)
 range(a:quality))
ObjectProperty(a:has-state
 inverseOf(a:state-of)
 range(a:state))
ObjectProperty(a:has-substrate
 inverseOf(a:substrate-of))
ObjectProperty(a:has-t-quality
 inverseOf(a:t-inherent-in)
 domain(a:entity)
 range(a:quality))
ObjectProperty(a:has-target
 inverseOf(a:target-of)
 domain(a:perdurant))
ObjectProperty(a:host
```

```
 inverseOf(a:host-of)
 domain(a:feature)
 range(a:entity))
ObjectProperty(a:host-of
 inverseOf(a:host)
 domain(a:entity)
 range(a:feature))
ObjectProperty(a:identity-c Transitive
 inverseOf(a:identity-c))
ObjectProperty(a:identity-n Transitive
 inverseOf(a:identity-n))
ObjectProperty(a:immediate-relation
 inverseOf(a:immediate-relation)
 domain(a:entity)
 range(a:entity))
ObjectProperty(a:immunity-target-of
 inverseOf(a:immunity-towards))
ObjectProperty(a:immunity-towards
 inverseOf(a:immunity-target-of))
ObjectProperty(a:indirectly-played-by
 inverseOf(a:indirectly-plays)
 domain(a:endurant)
 range(a:endurant))
ObjectProperty(a:indirectly-plays
 inverseOf(a:indirectly-played-by)
 domain(a:endurant)
 range(a:endurant))
ObjectProperty(a:inherent-in
 inverseOf(a:has-quality)
 domain(a:quality)
 range(a:entity))
ObjectProperty(a:instrument
 inverseOf(a:instrument-of)
 domain(a:activity)
 range(a:physical-object))
ObjectProperty(a:instrument-of
 inverseOf(a:instrument)
 domain(a:physical-object)
 range(a:activity))
ObjectProperty(a:interpretant
 inverseOf(a:interpretant-of)
 domain(a:expression)
 range(a:meaning))
ObjectProperty(a:interpretant-of
 inverseOf(a:interpretant)
 domain(a:meaning)
 range(a:expression))
ObjectProperty(a:involved-in
 inverseOf(a:involves)
 domain(a:endurant)
 range(a:description))
ObjectProperty(a:involves
 inverseOf(a:involved-in)
 domain(a:description)
 range(a:endurant))
ObjectProperty(a:iteration-interval
 inverseOf(a:iteration-interval-of)
 domain(a:loop-task)
 range(a:time-interval))
ObjectProperty(a:iteration-interval-of
 inverseOf(a:iteration-interval)
 domain(a:time-interval))
ObjectProperty(a:legal-attitude-target-of
 inverseOf(a:legal-attitude-towards))
ObjectProperty(a:legal-attitude-towards
 inverseOf(a:legal-attitude-target-of))
ObjectProperty(a:liability-target-of
 inverseOf(a:liability-towards))
ObjectProperty(a:liability-towards
```

```
 inverseOf(a:liability-target-of))
ObjectProperty(a:made-by
 inverseOf(a:makes)
 domain(a:endurant))
ObjectProperty(a:makes
 inverseOf(a:made-by)
 range(a:endurant))
ObjectProperty(a:material-place
 inverseOf(a:material-place-of)
 domain(a:physical-endurant)
 range(a:physical-endurant))
ObjectProperty(a:material-place-of
 inverseOf(a:material-place)
 domain(a:physical-endurant)
 range(a:physical-endurant))
ObjectProperty(a:mediated-relation
 inverseOf(a:mediated-relation)
 domain(a:entity)
 range(a:entity))
ObjectProperty(a:meets
 inverseOf(a:met-by))
ObjectProperty(a:member-of
 inverseOf(a:has-member))
ObjectProperty(a:mereotopological-association
 inverseOf(a:mereotopological-association))
ObjectProperty(a:met-by
 inverseOf(a:meets))
ObjectProperty(a:metaphorically-played-by
 inverseOf(a:metaphorically-plays))
ObjectProperty(a:metaphorically-plays
 inverseOf(a:metaphorically-played-by))
ObjectProperty(a:method-of
 inverseOf(a:has-method)
 domain(a:method)
 range(a:activity))
ObjectProperty(a:non-right-target-of
 inverseOf(a:non-right-towards))
ObjectProperty(a:non-right-towards
 inverseOf(a:non-right-target-of))
ObjectProperty(a:optionally-used-by
 inverseOf(a:optionally-uses)
 domain(a:concept)
 range(a:description))
ObjectProperty(a:optionally-uses)
ObjectProperty(a:origin
 inverseOf(a:origin-of))
ObjectProperty(a:origin-of
 inverseOf(a:origin))
ObjectProperty(a:overlaps
 inverseOf(a:overlaps))
ObjectProperty(a:p-depend-on-of
 inverseOf(a:p-depends-on)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:p-depends-on
 inverseOf(a:p-depend-on-of)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:p-sat
 inverseOf(a:p-sat-by))
ObjectProperty(a:p-sat-by)
ObjectProperty(a:p-spatial-location
 inverseOf(a:p-spatial-location-of)
 domain(a:perdurant)
 range(a:space-region))
ObjectProperty(a:p-spatial-location-of
 inverseOf(a:p-spatial-location)
 domain(a:space-region)
 range(a:perdurant))
```

```
ObjectProperty(a:parametrical-depend-on-of
 inverseOf(a:parametrically-depends-on))
ObjectProperty(a:parametrically-depends-on
 domain(a:parameter)
 range(a:parameter))
ObjectProperty(a:parametrized-by
 inverseOf(a:parametrizes)
 range(a:parameter))
ObjectProperty(a:parametrizes
 inverseOf(a:parametrized-by)
 domain(a:parameter))
ObjectProperty(a:part
 inverseOf(a:part-of))
ObjectProperty(a:part-of
 inverseOf(a:part))
ObjectProperty(a:participant
 inverseOf(a:participant-in)
 domain(a:perdurant)
 range(a:endurant))
ObjectProperty(a:participant-in
 inverseOf(a:participant)
 domain(a:endurant)
 range(a:perdurant))
ObjectProperty(a:participant-place
 inverseOf(a:participant-place-of)
 domain(a:perdurant)
 range(a:endurant))
ObjectProperty(a:participant-place-of
 inverseOf(a:participant-place)
 domain(a:endurant)
 range(a:perdurant))
ObjectProperty(a:partly-compresent-with
 inverseOf(a:partly-compresent-with)
 domain(a:endurant)
 range(a:endurant))
ObjectProperty(a:patient
 inverseOf(a:patient-of)
 domain(a:perdurant))
ObjectProperty(a:patient-of
 inverseOf(a:patient)
 range(a:perdurant))
ObjectProperty(a:performed-by
 inverseOf(a:performs)
 domain(a:perdurant)
 range(unionOf(a:agentive-physical-object a:agentive-role)))
ObjectProperty(a:performs
 inverseOf(a:performed-by)
 domain(unionOf(a:agentive-physical-object a:agentive-role))
 range(a:perdurant))
ObjectProperty(a:physical-depend-on-of
 inverseOf(a:physically-depends-on)
 domain(a:physical-endurant)
 range(a:non-physical-endurant))
ObjectProperty(a:physical-location
 inverseOf(a:physical-location-of)
 domain(a:physical-endurant)
 range(a:physical-region))
ObjectProperty(a:physical-location-of
 inverseOf(a:physical-location)
 domain(a:physical-region)
 range(a:physical-endurant))
ObjectProperty(a:physically-depends-on
 inverseOf(a:physical-depend-on-of)
 domain(a:non-physical-endurant)
 range(a:physical-endurant))
ObjectProperty(a:place
 inverseOf(a:place-of)
 domain(a:endurant)
 range(a:physical-endurant))
```

```
ObjectProperty(a:place-of
 inverseOf(a:place)
 domain(a:physical-endurant)
 range(a:endurant))
ObjectProperty(a:played-by
 inverseOf(a:plays)
 domain(a:role)
 range(a:endurant))
ObjectProperty(a:plays
 inverseOf(a:played-by)
 domain(a:endurant)
 range(a:role))
ObjectProperty(a:postcondition
 inverseOf(a:postcondition-of)
 domain(a:description)
 range(a:situation))
ObjectProperty(a:postcondition-of
 inverseOf(a:postcondition)
 domain(a:situation)
 range(a:description))
ObjectProperty(a:power-target-of
 inverseOf(a:power-towards))
ObjectProperty(a:power-towards
 inverseOf(a:power-target-of))
ObjectProperty(a:precedes Transitive
 inverseOf(a:follows)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:precondition
 inverseOf(a:precondition-of)
 domain(a:description)
 range(a:situation))
ObjectProperty(a:precondition-of
 inverseOf(a:precondition)
 domain(a:situation)
 range(a:description))
ObjectProperty(a:predecessor
 inverseOf(a:successor))
ObjectProperty(a:prescribed-by
 inverseOf(a:prescribes))
ObjectProperty(a:prescribes
 inverseOf(a:prescribed-by))
ObjectProperty(a:present-at
 inverseOf(a:time-of-presence-of)
 domain(a:endurant)
 range(a:time-interval))
ObjectProperty(a:privilege-target-of
 inverseOf(a:privilege-towards))
ObjectProperty(a:privilege-towards
 inverseOf(a:privilege-target-of))
ObjectProperty(a:product
 inverseOf(a:product-of)
 domain(a:activity))
ObjectProperty(a:product-of
 inverseOf(a:product)
 range(a:activity))
ObjectProperty(a:proper-part
 inverseOf(a:proper-part-of))
ObjectProperty(a:proper-part-of
 inverseOf(a:proper-part))
ObjectProperty(a:q-location
 inverseOf(a:q-location-of)
 domain(a:quality)
 range(a:region))
ObjectProperty(a:q-location-of
 inverseOf(a:q-location)
 domain(a:region)
 range(a:quality))
ObjectProperty(a:q-present-at
```

```
 inverseOf(a:time-of-q-presence-of)
 domain(a:physical-quality)
 range(a:time-interval))
ObjectProperty(a:q-realized-by
 inverseOf(a:q-realizes)
 domain(a:region)
 range(a:entity))
ObjectProperty(a:q-realizes
 inverseOf(a:q-realized-by)
 domain(a:entity)
 range(a:region))
ObjectProperty(a:q-represented-by
 inverseOf(a:q-represents)
 domain(a:region)
 range(a:information-object))
ObjectProperty(a:q-represents
 inverseOf(a:q-represented-by)
 domain(a:information-object)
 range(a:region))
ObjectProperty(a:quale-of
 inverseOf(a:has-quale)
 domain(a:quale)
 range(a:quality))
ObjectProperty(a:r-location
 inverseOf(a:r-location-of)
 domain(a:region)
 range(a:region))
ObjectProperty(a:r-location-of
 inverseOf(a:r-location))
ObjectProperty(a:r-sat
 inverseOf(a:r-sat-by))
ObjectProperty(a:r-sat-by)
ObjectProperty(a:realized-by
 inverseOf(a:realizes)
 domain(a:non-physical-object)
 range(a:entity))
ObjectProperty(a:realizes
 inverseOf(a:realized-by)
 domain(a:entity)
 range(a:non-physical-object))
ObjectProperty(a:reference-theme
 inverseOf(a:reference-theme-of))
ObjectProperty(a:reference-theme-of
 inverseOf(a:reference-theme))
ObjectProperty(a:referenced-by
 inverseOf(a:references)
 domain(a:entity)
 range(a:non-physical-object))
ObjectProperty(a:references
 inverseOf(a:referenced-by)
 domain(a:non-physical-object)
 range(a:entity))
ObjectProperty(a:refined-by)
ObjectProperty(a:refines
 inverseOf(a:refined-by)
 domain(unionOf(a:figure a:concept))
 range(unionOf(a:figure a:concept)))
ObjectProperty(a:regulated-by
 inverseOf(a:regulates)
 range(a:regulation))
ObjectProperty(a:regulates
 inverseOf(a:regulated-by)
 domain(a:regulation))
ObjectProperty(a:represented-by
 inverseOf(a:represents)
 domain(a:non-physical-object)
 range(a:information-object))
ObjectProperty(a:represents
 inverseOf(a:represented-by)
```

```
 domain(a:information-object)
 range(a:non-physical-object))
ObjectProperty(a:requisite
 inverseOf(a:requisite-for)
 domain(unionOf(a:role a:course))
 range(a:parameter))
ObjectProperty(a:requisite-for
 inverseOf(a:requisite)
 domain(a:parameter)
 range(unionOf(a:role a:figure a:course)))
ObjectProperty(a:resource
 inverseOf(a:resource-for)
 domain(a:activity)
 range(a:amount-of-matter))
ObjectProperty(a:resource-for
 inverseOf(a:resource)
 domain(a:amount-of-matter)
 range(a:activity))
ObjectProperty(a:result
 inverseOf(a:result-of)
 domain(a:activity)
 range(a:perdurant))
ObjectProperty(a:result-of
 inverseOf(a:result)
 domain(a:perdurant)
 range(a:activity))
ObjectProperty(a:right-target-of
 inverseOf(a:right-towards))
ObjectProperty(a:right-towards
 inverseOf(a:right-target-of))
ObjectProperty(a:ruled-by
 inverseOf(a:rules)
 domain(a:role)
 range(a:socially-constructed-person))
ObjectProperty(a:rules
 inverseOf(a:ruled-by)
 domain(a:socially-constructed-person)
 range(a:role))
ObjectProperty(a:satisfied-by)
ObjectProperty(a:satisfies
 inverseOf(a:satisfied-by)
 domain(a:situation)
 range(a:description))
ObjectProperty(a:selected-by
 inverseOf(a:selects))
ObjectProperty(a:selects
 inverseOf(a:selected-by)
 domain(a:concept)
 range(a:entity))
ObjectProperty(a:sequenced-by
 inverseOf(a:sequences)
 domain(a:perdurant)
 range(a:course))
ObjectProperty(a:sequences
 inverseOf(a:sequenced-by)
 domain(a:course)
 range(a:perdurant))
ObjectProperty(a:setting
 inverseOf(a:setting-for)
 domain(a:entity)
 range(a:situation))
ObjectProperty(a:setting-for
 inverseOf(a:setting)
 domain(a:situation)
 range(a:entity))
ObjectProperty(a:sibling-part
 inverseOf(a:sibling-part))
ObjectProperty(a:sibling-task
 inverseOf(a:sibling-task)
```

```
 domain(a:task)
 range(a:task))
ObjectProperty(a:situation-place
 inverseOf(a:situation-place-of)
 domain(a:situation)
 range(a:endurant))
ObjectProperty(a:situation-place-of
 inverseOf(a:situation-place)
 domain(a:endurant)
 range(a:situation))
ObjectProperty(a:spatial-location
 inverseOf(a:spatial-location-of)
 domain(a:physical-endurant)
 range(a:space-region))
ObjectProperty(a:spatial-location-of
 inverseOf(a:spatial-location)
 domain(a:space-region)
 range(a:physical-endurant))
ObjectProperty(a:specialized-by
 inverseOf(a:specializes)
 domain(a:non-physical-object)
 range(a:non-physical-object))
ObjectProperty(a:specializes
 inverseOf(a:specialized-by)
 domain(a:concept)
 range(a:concept))
ObjectProperty(a:specific-constant-depend-on-of
 inverseOf(a:specifically-constantly-dependent-on))
ObjectProperty(a:specifically-constantly-dependent-on
 inverseOf(a:specific-constant-depend-on-of))
ObjectProperty(a:started-by
 inverseOf(a:starts))
ObjectProperty(a:starts
 inverseOf(a:started-by))
ObjectProperty(a:state-of
 inverseOf(a:has-state)
 domain(a:state))
ObjectProperty(a:strong-connection
 inverseOf(a:strong-connection))
ObjectProperty(a:subject-target-of
 inverseOf(a:subjected-to)
 domain(a:course)
 range(a:patient-role))
ObjectProperty(a:subjected-to
 inverseOf(a:subject-target-of)
 domain(a:patient-role)
 range(a:course))
ObjectProperty(a:substrate-of
 inverseOf(a:has-substrate))
ObjectProperty(a:successor
 inverseOf(a:predecessor)
 domain(a:entity)
 range(a:entity))
ObjectProperty(a:t-inherent-in
 inverseOf(a:has-t-quality)
 domain(a:quality)
 range(a:entity))
ObjectProperty(a:target-of
 inverseOf(a:has-target)
 range(a:perdurant))
ObjectProperty(a:task-postcondition
 inverseOf(a:task-postcondition-of)
 domain(a:task)
 range(a:situation))
ObjectProperty(a:task-postcondition-of
 inverseOf(a:task-postcondition)
 domain(a:situation)
 range(a:task))
ObjectProperty(a:task-precondition
```

```
 inverseOf(a:task-precondition-of)
 domain(a:task)
 range(a:situation))
ObjectProperty(a:task-precondition-of
 inverseOf(a:task-precondition)
 domain(a:situation)
 range(a:task))
ObjectProperty(a:temporal-connection
 inverseOf(a:temporal-connection)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:temporal-intersection
 inverseOf(a:temporal-intersection)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:temporal-location
 inverseOf(a:temporal-location-of)
 domain(a:perdurant)
 range(a:temporal-region))
ObjectProperty(a:temporal-location-of
 inverseOf(a:temporal-location)
 domain(a:temporal-region)
 range(a:perdurant))
ObjectProperty(a:temporal-relation
 inverseOf(a:temporal-relation)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:temporally-contained-in
 inverseOf(a:temporally-contains)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:temporally-contains
 inverseOf(a:temporally-contained-in)
 domain(a:perdurant)
 range(a:perdurant))
ObjectProperty(a:temporarily-depends-on
 inverseOf(a:temporary-depend-on-of))
ObjectProperty(a:temporary-component
 inverseOf(a:temporary-component-of))
ObjectProperty(a:temporary-component-of
 inverseOf(a:temporary-component))
ObjectProperty(a:temporary-depend-on-of
 inverseOf(a:temporarily-depends-on))
ObjectProperty(a:temporary-part
 inverseOf(a:temporary-part-of))
ObjectProperty(a:temporary-part-of
 inverseOf(a:temporary-part))
ObjectProperty(a:temporary-participant
 inverseOf(a:temporary-participant-in)
 domain(a:perdurant)
 range(a:endurant))
ObjectProperty(a:temporary-participant-in
 inverseOf(a:temporary-participant)
 domain(a:endurant)
 range(a:perdurant))
ObjectProperty(a:temporary-proper-part
 inverseOf(a:temporary-proper-part-of))
ObjectProperty(a:temporary-proper-part-of
 inverseOf(a:temporary-proper-part))
ObjectProperty(a:theme
 inverseOf(a:theme-of)
 range(a:information-object))
ObjectProperty(a:theme-of
 inverseOf(a:theme)
 domain(a:information-object))
ObjectProperty(a:time-of-happening-of
 inverseOf(a:happens-at)
 domain(a:time-interval)
 range(a:perdurant))
```

```
ObjectProperty(a:time-of-presence-of
 inverseOf(a:present-at)
 domain(a:time-interval)
 range(a:endurant))
ObjectProperty(a:time-of-q-presence-of
 inverseOf(a:q-present-at)
 domain(a:time-interval)
 range(a:physical-quality))
ObjectProperty(a:total-participant
 inverseOf(a:total-participant-in)
 domain(a:perdurant)
 range(a:endurant))
ObjectProperty(a:total-participant-in
 inverseOf(a:total-participant)
 domain(a:endurant)
 range(a:perdurant))
ObjectProperty(a:unit
 inverseOf(a:unit-of)
 domain(a:region)
 range(a:measurement-unit))
ObjectProperty(a:unit-of
 inverseOf(a:unit)
 domain(a:measurement-unit))
ObjectProperty(a:use-context-of
 inverseOf(a:used-in)
 domain(a:perdurant)
 range(a:endurant))
ObjectProperty(a:use-target-of
 inverseOf(a:use-within)
 domain(a:course)
 range(a:instrumentality-role))
ObjectProperty(a:use-within
 inverseOf(a:use-target-of)
 domain(a:instrumentality-role)
 range(a:course))
ObjectProperty(a:used-by
 inverseOf(a:uses))
ObjectProperty(a:used-in
 inverseOf(a:use-context-of)
 domain(a:endurant)
 range(a:perdurant))
ObjectProperty(a:uses
 inverseOf(a:used-by))
ObjectProperty(a:value-for
 inverseOf(a:valued-by)
 domain(a:region)
 range(a:parameter))
ObjectProperty(a:valued-by
 inverseOf(a:value-for)
 domain(a:parameter)
 range(a:region))
ObjectProperty(a:weak-connection
 inverseOf(a:weak-connection))

DatatypeProperty(a:counted-by
 domain(a:region)
 range(xsd:integer))
DatatypeProperty(a:has-informal-description
 domain(a:entity)
 range(xsd:string))
DatatypeProperty(a:iteration-cardinality
 domain(a:loop-task)
 range(xsd:integer))
DatatypeProperty(a:quantitatively-admits
 domain(a:description)
 range(xsd:integer))
DatatypeProperty(a:title
 domain(a:information-object)
 range(xsd:string))
```

```
Class(a:abortion-task partial
 a:ending-task)

Class(a:abstract partial
 restriction(a:has-quality cardinality(0))
 restriction(a:has-quality cardinality(0))
 a:entity)

Class(a: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(a:abstract-plan partial
 a:plan)

Class(a: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(a:abstract-quality partial
 a:quality
 restriction(a:inherent-in allValuesFrom(a:non-physical-endurant))
 restriction(a:inherent-in someValuesFrom(a:non-physical-endurant))
 restriction(a:has-quality allValuesFrom(a:abstract-quality))
 restriction(a:q-location allValuesFrom(a:abstract-region)))

Class(a:abstract-region partial
 restriction(a:q-location-of allValuesFrom(a:abstract-quality))
 restriction(a:part allValuesFrom(a:abstract-region))
 a:region)

Class(a: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(a:accomplishment partial
 a:event)

Class(a: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(a:achievement partial
 a:event)

Class(a: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(a:action partial
restriction(a:generically-depends-on someValuesFrom(a:cognitive-state))
restriction(a:participant someValuesFrom(a:social-agent))
a:accomplishment)

Class(a: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(a:action-task complete
intersectionOf(restriction(a:sequences cardinality(0)) a:elementary-task))

Class(a: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(a:activity partial
restriction(a:sequenced-by someValuesFrom(a:course))
a:action
restriction(a:generically-depends-on someValuesFrom(a:course)))

Class(a: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(a:agent complete
intersectionOf(restriction(a:participant-in someValuesFrom(intersectionOf(a:activity restriction(a:sequenced-by someValuesFrom(a:task))))) unionOf(a:agentive-physical-object a:agentive-role) a:endurant))

Class(a:agent partial
annotation(rdfs:comment "This is a catch-all class of agentive entities. It includes both agentive functional roles (figures), and agentive physical objects. The only constraint is their participation in some (rational) activity. Further distinctions must be added for autonomy, intentionality, etc. Currently, agent means intentional agent.")
)

Class(a:agent-case-role partial
a:case-role)

Class(a:agent-case-role partial
annotation(rdfs:comment "Agent-role is here a placeholder within the case system (cf. Fillmore, Minsky). It is used to define so-called 'functional' participant relations, but in DAML+OIL version there is no trace of that use (due to lack of expressivity).
We expect to build a linkage between the case system and the agentive/non-agentive functional roles currently defined in the theory. This is currently under investigation.
The main issue is that the agentive/non-agentive distinction, which is 'attached' to roles, can be overruled by a role in the case system. In other words, an agentive-functional-role' can play roles other than 'agent-role' in the case system.")
)

Class(a:agentive-group partial
a:agentive-physical-object)

```
Class(a:agentive-physical-object partial
 a:physical-object)

Class(a:agentive-physical-object partial
 annotation(rdfs:comment "A strictly agentive object carrying out some function.
Within Physical objects, a special place have 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. 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.
Agentivity here means that agentive physical object *can* play an Agentive-Functional-Role, not that they *must* do it, then there is no existential axiom.")
)

Class(a:agentive-role partial
 a:role)

Class(a:agentive-role partial
 annotation(rdfs:comment "A role played by some object that intentionally carries out a process or event, or bears a state.
By intentional agent we mean here any object oriented to achieve a given
state of the world. Intentionality can be either external or internal.
A cognitive agent has an explicit representation for goals, intentions, and beliefs.
Intentionality and representation-explicitness are addressed by the theory
of 'Modalities' in D&S, which is still under development and will be enhanced
by ontologies of agents currently being examined.
The perdurant carried out can be partly present even in absence of it or of
its whole (other agents can realize it).
Examples of Agentive Functional Roles are social agents like
'the president of United States': we may think that the latter, besides depending generically on a community of US citizens, depends also generically on 'George Bush qua legal person' (since the president can be substituted), which in turn depends specifically on 'George Bush qua human being'.
Social agents are not constituted by agentive physical objects (although they depend on them), while they can constitute societies or organizations, like the Italian Government, Mercedes-Benz, etc.
Agentive-functional-role is a low-level role for agentivity, meaning that it is played by physical agents or by other agentive functional roles.
In this theory there is a related functional role called 'Agent-Role' that is a generalized 'case' role for attributing intentionality.")
)

Class(a:agentive-temporary-role partial
 a:agentive-role)

Class(a:agentive-temporary-role partial
 annotation(rdfs:comment "A role for talking of someone at certain phases of his/her own life. It can be used also to map temporal parts of agentive objects from a 4D ontology.")
)

Class(a:alternate-task complete
 intersectionOf(restriction(a:direct-successor cardinality(2)) a:case-task))

Class(a:alternate-task partial
 annotation(rdfs:comment "A case task branched to exactly 2 tasks, not executable in parallel.")
)

Class(a:amount-of-matter partial
 a:physical-endurant)

Class(a: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(a:any-order-task complete
```

```
 intersectionOf(restriction(a:successor someValuesFrom(a:synchro-task)) restriction(a:direct-successor
someValuesFrom(intersectionOf(a:task restriction(a:sequences
allValuesFrom(intersectionOf(restriction(a:temporal-relation
someValuesFrom(intersectionOf(restriction(a:sequenced-by someValuesFrom(a:task)) a:perdurant)))
a:perdurant)))))) a:branching-task))

 Class(a:any-order-task partial
  annotation(rdfs:comment "An any order task is a branching task that defines no order in the successor tasks. Its
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.")
)

 Class(a:arbitrary-sum partial
  restriction(a:part minCardinality(2))
  a:endurant)

 Class(a: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(a:artifact-role partial
  a:consequence-role)

 Class(a:artifact-role partial
  annotation(rdfs:comment "An artifact role is a kind of consequence  role motivated by an intentional activity.")
)

 Class(a:assessment-quality partial
  a:abstract-quality)

 Class(a:axiom partial
  a:formal-expression)

 Class(a:axiomatic-system partial
  a:information-encoding-system)

 Class(a:axiomatic-system partial
  annotation(rdfs:comment "An information encoding system that provides roles and operations to define formal
descriptions (e.g. theories).")
)

 Class(a:axiomatization partial
  restriction(a:part someValuesFrom(a:axiom))
  a:formal-system)

 Class(a:bag-task partial
  a:complex-task
  restriction(a:component minCardinality(2))
  restriction(a:component cardinality(0)))

 Class(a: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(a:beginning-task complete
  intersectionOf(restriction(a:successor someValuesFrom(a:task)) restriction(a:predecessor cardinality(0))
a:control-task))

 Class(a:beginning-task partial
  annotation(rdfs:comment "A beginning task is a control task that is the predecessor of all tasks defined in the
plan.")
)

 Class(a:biological-object partial
  a:physical-body)
```

```
 Class(a:branching-task complete
  intersectionOf(restriction(a:direct-successor minCardinality(2)) a:control-task))

 Class(a:branching-task partial
  restriction(a:sequences allValuesFrom(a:planning-activity))
  restriction(a:represented-by allValuesFrom(a:fork-node)))

 Class(a:branching-task partial
  annotation(rdfs:comment "A task that articulates the plan into an ordered set of tasks.")
)

 Class(a:c-context complete
  intersectionOf(restriction(a:played-by allValuesFrom(a:description)) a:communication-role restriction(a:defined-
by someValuesFrom(a:communication-method)) restriction(a:specializes someValuesFrom(a:s-context))))

 Class(a:case-role partial
  a:non-agentive-functional-role
  restriction(a:defined-by someValuesFrom(a:case-system)))

 Class(a:case-role partial
  annotation(rdfs:comment "Case roles are functional roles that are constitutent of the case system of
descriptions. The case system goes back at least to Aristotle's 'aitiai', and has been proposed in various forms by
Port Royal's grammarians and recently by Charles Fillmore, Roger Shank, Ray Jackendoff, John Sowa, etc.
The case system can be used on top of functional descriptions to distinguish
forms of behaviour.
They can also be used to specialize the 'participation' relation.
Case roles constitute a partition. This is untenable without the notion of
description, since participants can change through time: for example, an
object can be an agent for part of an activity, and then become a patient.
By using descriptions, we can simply state that for one part of an activity,
the object *plays* the role of agent, and for another part, it plays the
role of patient.
The case system will be connected to rest of D&S as soon as possible. The main
issue is that the agentive/non-agentive distinction, which is 'attached'
to roles, can be overruled by a role in the case system. In other words, an
'agentive-role' can play roles other than 'agent-role' in the case
system.")
)

 Class(a:case-system complete
  intersectionOf(a:description restriction(a:defines someValuesFrom(a:case-role))))

 Class(a:case-task complete
  intersectionOf(restriction(a:direct-successor minCardinality(2)) restriction(a:sequences
allValuesFrom(a:decision-activity)) restriction(a:direct-successor allValuesFrom(a:deliberation-task)) a:branching-
task))

 Class(a:case-task partial
  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).

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

 Class(a:causal-role partial
  a:non-agentive-functional-role)

 Class(a:channel-role partial
  restriction(a:played-by allValuesFrom(unionOf(a:physical-quality a:physical-endurant
intersectionOf(restriction(a:participant someValuesFrom(a:physical-endurant)) a:perdurant))))
  restriction(a:defined-by someValuesFrom(a:communication-method))
  a:communication-role)

 Class(a:chemical-object partial
  a:physical-body)
```

```
 Class(a:circumstantial-plan complete
  intersectionOf(a:plan restriction(a:d-uses allValuesFrom(intersectionOf(restriction(a:selects
someValuesFrom(a:entity)) a:concept)))))

 Class(a: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(a:classification-system partial
  a:information-encoding-system)

 Class(a: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(a:code-role partial
  restriction(a:played-by allValuesFrom(a:semiotic-code))
  a:communication-role
  restriction(a:defined-by someValuesFrom(a:communication-method)))

 Class(a:cognitive-event partial
  restriction(a:has-substrate someValuesFrom(a:natural-person))
  a:event)

 Class(a:cognitive-event partial
  annotation(rdfs:comment "An event occurring in the (embodied) mind.")
)

 Class(a:cognitive-state partial
  a:state
  restriction(a:has-substrate someValuesFrom(a:natural-person)))

 Class(a:cognitive-state partial
  annotation(rdfs:comment "A state of the (embodied) mind")
)

 Class(a:combinatorial-system partial
  a:information-encoding-system)

 Class(a: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(a:commerce-role partial
  a: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:commitment partial
  a:obligation)

 Class(a:communication partial
  restriction(a:sequenced-by allValuesFrom(a:communication-turns))
  a:accomplishment)

 Class(a:communication 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(a:communication-method partial
restriction(a:defines someValuesFrom(a:code-role))
restriction(a:defines someValuesFrom(a:channel-role))
restriction(a:defines someValuesFrom(a:c-context))
a:description
restriction(a:defines someValuesFrom(a:interpreter-role))
restriction(a:defines someValuesFrom(a:message-role)))

Class(a:communication-method partial
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 method, we also need other components that are not specified in Jakobson's theory: 'Communication-Turns' governing the sequence of a communication process, and 'Communication-Parameters', governing the attributes 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).")
)

Class(a:communication-parameter partial
restriction(a:defined-by someValuesFrom(a:communication-method))
a:parameter)

Class(a:communication-role partial
restriction(a:defined-by someValuesFrom(a:communication-method))
a:non-agentive-functional-role)

Class(a:communication-role partial
annotation(rdfs:comment "The non-agentive roles from the theory of communication of Jakobson's. The agentive ones (encoder and decoder) are under 'interpreter role' among agentive functional roles.")
)

Class(a:communication-situation complete
intersectionOf(restriction(a:setting-for someValuesFrom(a:social-agent)) a:situation restriction(a:setting-for someValuesFrom(a:communication)) restriction(a:satisfies someValuesFrom(a:communication-method)) restriction(a:setting-for someValuesFrom(a:information-object))))

Class(a:communication-turns partial
restriction(a:defined-by someValuesFrom(a:communication-method))
a:course)

Class(a:completion-task partial
a:ending-task)

Class(a:complex-task complete
intersectionOf(restriction(a:component minCardinality(2)) a:task))

Class(a:complex-task partial
annotation(rdfs:comment "A task that has at least two other tasks as components.")
)

Class(a:concept partial
restriction(a:specializes allValuesFrom(restriction(a:selected-by someValuesFrom(a:concept))))
a:non-physical-object
restriction(a:selects someValuesFrom(a:entity))
restriction(a:defined-by someValuesFrom(a:description)))

Class(a:concept partial
annotation(rdfs:comment "AKA C-Description.
A non-physical object that is defined by a description s, and whose function is selecting entities from a ground ontology in order to build situations that can satisfy s.")
)

Class(a:concurrency-task partial
restriction(a:direct-successor someValuesFrom(intersectionOf(restriction(a:sequences allValuesFrom(intersectionOf(restriction(a:overlaps someValuesFrom(intersectionOf(restriction(a:sequenced-by someValuesFrom(a:task)) a:perdurant))) a:perdurant))) a:task)))

```
 restriction(a:successor someValuesFrom(a:synchro-task))
 a:branching-task)

Class(a:concurrency-task partial
 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).")
)

Class(a:consequence-role partial
 a:case-role
 restriction(a:functionally-depends-on someValuesFrom(unionOf(a:substrate-role a:agent-case-role))))

Class(a:consequence-role partial
 annotation(rdfs:comment "Consequence is a role played by some endurant that participates in a perdurant. The role-player does not carry out the perdurant, and comes into being only when the perdurant or a functional part of it (its 'prerequisite') has been completed.")
)

Class(a:constitutive-description partial
 a:description
 restriction(a:defines someValuesFrom(a:figure)))

Class(a:constitutive-description partial
 annotation(rdfs:comment "A description whose main purpose is defining a figure.")
)

Class(a:contract partial
 a:regulation
 restriction(a:part someValuesFrom(a:promise)))

Class(a:control-task complete
 intersectionOf(a:elementary-task restriction(a:sequences allValuesFrom(unionOf(a:decision-state a:planning-activity)))))

Class(a: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.")
)

Class(a:country partial
 a:political-geographic-object)

Class(a:country partial
 annotation(rdfs:comment "This needs dependency axioms with Physical-Place.")
)

Class(a:course partial
 restriction(a:part allValuesFrom(a:course))
 restriction(a:defined-by someValuesFrom(a:description))
 restriction(a:sequences allValuesFrom(a:perdurant))
 a:concept
 restriction(a:attitude-target-of allValuesFrom(unionOf(a:role a:figure))))

Class(a:course partial
 annotation(rdfs:comment "A concept that selects (in particular, it 'sequences') perdurants (processes, events, or states), as a component of some s-description. Courses are the descriptive counterpart of perdurants, and, as perdurants have endurants as participatants, they are usually the target of attitudes of some functional role. This relation is named 'modality target of', because it actually reifies at first order a typology of modal relations.")
)

Class(a:creative-object partial
 a:information-object)

Class(a:cyclical-task partial
```

```
 a:complex-task
 restriction(a:component someValuesFrom(a:case-task))
 restriction(a:component someValuesFrom(a:deliberation-task))
 restriction(a:direct-predecessor someValuesFrom(a:loop-task))
 restriction(a:represented-by someValuesFrom(a:loop-node)))

Class(a:cyclical-task partial
 annotation(rdfs:comment "A cyclical task is a complex task that is controlled by a loop task, and has a case task as a component. The case task specifies the exit condition(s) of the cyclical task indirectly (only the decisions that havent the cyclical task as successor are exit conditions), while a loop-until task specifies which the exit condition is.

The full axiom cannot be expressed in OWL-DL.")
)

Class(a:decision-activity partial
 a:planning-activity)

Class(a:decision-state partial
 a:state)

Class(a:decoder partial
 a:interpreter-role)

Class(a:deliberation-task complete
 intersectionOf(restriction(a:direct-predecessor someValuesFrom(a:case-task)) restriction(a:sequences allValuesFrom(a:decision-state)) a:control-task))

Class(a:deliberation-task partial
 annotation(rdfs:comment "A deliberation task is a control task that sequences deliberation states (decisions taken after a case task execution).")
)

Class(a:dependent-place partial
 a:feature)

Class(a:description partial
 restriction(a:satisfied-by allValuesFrom(a:situation))
 a:non-physical-object
 restriction(a:d-uses someValuesFrom(unionOf(a:figure a:concept)))
 restriction(a:defines allValuesFrom(unionOf(a:figure a:concept))))

Class(a: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.")
)

Class(a:desire partial
 restriction(a:d-uses someValuesFrom(intersectionOf(restriction(a:desire-towards someValuesFrom(a:course)) unionOf(a:agentive-role a:intentional-figure))))
 a:modal-description)

Class(a: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(a:diagram partial
 a:diagrammatic-object)

Class(a:diagram-component complete
 intersectionOf(restriction(a:temporary-component-of someValuesFrom(a:diagram)) a:diagrammatic-object))

Class(a:diagrammatic-object partial
 a:information-object)

Class(a:disability complete
```

```
 intersectionOf(restriction(a:d-uses someValuesFrom(intersectionOf(a:role restriction(a:disability-towards
someValuesFrom(a:course))))) a:modal-description))

 Class(a:document partial
  a:text)

 Class(a:document partial
  annotation(rdfs:comment "A formatted text, still independent from a *physical* document. Besides a language,
its encoding may be made according to a document template.")
)

 Class(a:document-template partial
  restriction(a:expression-means-for allValuesFrom(a:document))
  a:combinatorial-system)

 Class(a:duty complete
  intersectionOf(restriction(a:defines someValuesFrom(intersectionOf(a:role restriction(a:duty-towards
someValuesFrom(a:course))))) a:modal-description))

 Class(a:elementary-task complete
  intersectionOf(restriction(a:component cardinality(0)) a:task))

 Class(a:elementary-task partial
  annotation(rdfs:comment "An atomic task.")
)

 Class(a:encoder partial
  a:interpreter-role)

 Class(a:ending-task complete
  intersectionOf(restriction(a:successor cardinality(0)) restriction(a:predecessor someValuesFrom(a:task))
a:control-task))

 Class(a:ending-task partial
  annotation(rdfs:comment "An ending task is a control task that has no successor tasks defined in the plan.")
)

 Class(a:endurant partial
  restriction(a:constituent allValuesFrom(a:endurant))
  restriction(a:participant-in allValuesFrom(a:perdurant))
  restriction(a:part allValuesFrom(a:endurant))
  a:entity
  restriction(a:participant-in someValuesFrom(a:perdurant)))

 Class(a: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(a:entity partial
  annotation(rdfs:comment "AKA 'particular'.
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(a:event partial
  a:perdurant)

 Class(a: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.

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 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(a:expression partial
  restriction(a:played-by allValuesFrom(a:information-object))
  a:semiotic-role)

 Class(a:expression partial
  annotation(rdfs:comment "Expressions are played by information objects and are semiotic roles. They are used to fill the first domain of the  so-called 'interpretation function'. It may be 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.")
)

 Class(a:fact partial
  a:abstract)

 Class(a:feature partial
  restriction(a:host someValuesFrom(a:endurant))
  a:physical-endurant)

 Class(a: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.")
)

 Class(a:feature-role partial
  a:non-agentive-functional-role)

 Class(a:feature-role partial
  annotation(rdfs:comment "A role played by some feature of a physical object.")
)

 Class(a:figure partial
  restriction(a:defined-by someValuesFrom(a:constitutive-description))
  a:non-physical-object
  restriction(a:attitude-towards allValuesFrom(a:course))
  restriction(a:requisite allValuesFrom(a:parameter)))

 Class(a:figure partial
  annotation(rdfs:comment "Figures are non-physical objects defined or used by descriptions, but differently from concepts, they do not select entities.
Examples of figures are organisations, political geographic objects, sacred symbols, etc.")
)

 Class(a:flow-chart partial
  a:diagram)

 Class(a:flow-chart-component complete
  intersectionOf(restriction(a:temporary-component-of someValuesFrom(a:flow-chart)) a:diagram-component))

 Class(a:flow-chart-node partial
  a:flow-chart-component)
```

```
Class(a:flux complete
 intersectionOf(restriction(a:constituent someValuesFrom(a:accomplishment)) a:process))

Class(a:flux partial
 annotation(rdfs:comment "Fluxes are processes that (also) contain accomplishments as constituents. In other
words, fluxes emerge out of accomplishments.")
)

Class(a:fork-node complete
 intersectionOf(a:flow-chart-node restriction(a:direct-successor minCardinality(2))))

Class(a:formal-expression partial
 a:linguistic-object)

Class(a:formal-system partial
 a:formal-expression)

Class(a:functional-matter complete
 intersectionOf(restriction(a:plays someValuesFrom(a:role)) a:amount-of-matter))

Class(a:functional-matter partial
 restriction(a:used-in someValuesFrom(a:activity)))

Class(a:geographical-feature partial
 a:feature)

Class(a:geographical-object partial
 a:physical-place)

Class(a:geographical-role partial
 a:non-physical-place)

Class(a:goal complete
 intersectionOf(restriction(a:proper-part-of someValuesFrom(a:plan)) a:desire))

Class(a: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(a:goal-situation complete
 intersectionOf(restriction(a:satisfies someValuesFrom(a:goal)) a:situation))

Class(a: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(a:grammar partial
 a:combinatorial-system)

Class(a:hybrid-task complete
```

```
 intersectionOf(a:complex-task restriction(a:component someValuesFrom(a:action-task)) restriction(a:component
someValuesFrom(a:control-task))))

 Class(a: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(a:iconic-object partial
  a:information-object)

 Class(a:immunity complete
  intersectionOf(a:modal-description restriction(a:d-uses someValuesFrom(intersectionOf(a:role
restriction(a:immunity-towards someValuesFrom(a:course)))))))

 Class(a:indicator partial
  a:parameter)

 Class(a:informal-encoding-system partial
  a:information-encoding-system)

 Class(a:informal-encoding-system partial
  annotation(rdfs:comment "An information encoding system that provides roles and operations to define informal
descriptions (e.g.  narratives).")
)

 Class(a:information-collection partial
  restriction(a:has-member minCardinality(2))
  a:information-object)

 Class(a:information-encoding-system partial
  restriction(a:involves someValuesFrom(a:information-object))
  a:description)

 Class(a: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(a:information-gathering partial
  a:activity)

 Class(a:information-object partial
  restriction(a:expressed-according-to someValuesFrom(a:combinatorial-system))
  restriction(a:plays someValuesFrom(a:expression))
  a:non-agentive-functional-role)

 Class(a:information-object partial
  annotation(rdfs:comment "Information objects are special roles played by physical representation entities. They
are generated according to some description system. Consequently, they are dependent from an encoding as
well as from a concrete realization. From a communication perspective, an information object can play the role of
\"message\". From a semiotic perspective, it playes the role of \"expression\".")
)

 Class(a:institution partial
  a:organization)

 Class(a:instrumentality-role partial
  restriction(a:functionally-depends-on someValuesFrom(unionOf(a:substrate-role a:agent-case-role)))
  a:case-role)

 Class(a:instrumentality-role partial
  annotation(rdfs:comment "Instrumentality is a role played by some endurant that participates in a perdurant. It
can carry out parts of or even the whole perdurant, but only if there is something playing agent- or substrate-role
that bootstraps the perdurant. It can bear only external intentionality, although there can be a compresent internal
intentionality. This deals with the complexity of 'delegation'.")
```

```
)

 Class(a:intentional-figure partial
  a:feature)

 Class(a:interpretation-function complete
  intersectionOf(a:description restriction(a:defines someValuesFrom(a:semiotic-role))))

 Class(a:interpretation-function partial
  annotation(rdfs:comment "A description that can includes roles either for semiotics or for formal semantics.
Here we only characterize semiotic roles: 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).")
)

 Class(a:interpreter-role partial
  restriction(a:defined-by someValuesFrom(a:communication-method))
  restriction(a:played-by allValuesFrom(a:social-agent))
  a:agentive-role)

 Class(a:join-node complete
  intersectionOf(restriction(a:direct-predecessor minCardinality(2)) a:flow-chart-node restriction(a:predecessor
someValuesFrom(a:fork-node))))

 Class(a:language partial
  a:semiotic-code)

 Class(a:legal-possession-entity partial
  a:social-role)

 Class(a:liability complete
  intersectionOf(restriction(a:d-uses someValuesFrom(intersectionOf(a:role restriction(a:liability-towards
someValuesFrom(a:course))))) a:modal-description))

 Class(a:life-cycle partial
  a:course)

 Class(a:linguistic-object partial
  a:information-object)

 Class(a:literature partial
  a:information-collection)

 Class(a:logical-operator partial
  a:formal-expression)

 Class(a:logical-role partial
  a:non-agentive-functional-role)

 Class(a:logical-role partial
  annotation(rdfs:comment "A functional role used to express logical levels within some layering description. A
typical example is the Linnean taxonomic ordering, where Phylum or Species are hierarchical roles.")
)

 Class(a:loop-for partial
  a:loop-task)

 Class(a:loop-for partial
  annotation(rdfs:comment "A loop task with a defined number (and possibly frequency) of iterations.")
)

 Class(a:loop-node partial
  a:flow-chart-node)

 Class(a:loop-task partial
  a:control-task)
```

```
Class(a:loop-task partial
 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.")
)

Class(a:loop-until complete
 intersectionOf(a:loop-task restriction(a:exit-condition someValuesFrom(a:deliberation-task))))

Class(a:loop-until partial
 annotation(rdfs:comment "A loop task, which specifies when a certain condition becomes true for a cyclical task to exit.")
)

Class(a:main-goal partial
 a:goal
 restriction(a:proper-part-of cardinality(0)))

Class(a:main-goal 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(a:material-artifact partial
 restriction(a:involved-in someValuesFrom(a:project))
 a:system-as-artifact
 a:non-agentive-physical-object)

Class(a:material-representation-artifact complete
 intersectionOf(a:material-artifact restriction(a:realizes someValuesFrom(a:information-object))))

Class(a:maximal-task partial
 a:complex-task)

Class(a: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(a:meaning partial
 restriction(a:played-by allValuesFrom(a:non-physical-object))
 a:semiotic-role)

Class(a:meaning partial
 annotation(rdfs:comment "Meanings are played by descriptions whatsoever and are semiotic roles. They are used to fill the range of the  so-called 'interpretation function'. It is not equivalent to any communication function. Descriptions playing meaning have different natures according to the situation referenced by S-Contexts. In other words, meanings are just what ontology is supposed to explicit, thus they cannot be thematized within the same ontology that describes them (both used and mentioned).")
)

Class(a:measurement-unit partial
 a:abstract-region)

Class(a:mental-object complete
 intersectionOf(restriction(a:physically-depends-on minCardinality(1)) restriction(a:physically-depends-on maxCardinality(1)) a:description))

Class(a: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(a:message-role partial
  restriction(a:specializes someValuesFrom(a:expression))
  a:communication-role
  restriction(a:defined-by someValuesFrom(a:communication-method))
  restriction(a:played-by allValuesFrom(a:information-object)))

 Class(a:method partial
  a:description)

 Class(a:modal-description complete
  intersectionOf(a:description restriction(a:d-uses someValuesFrom(a:course)) restriction(a:temporary-part-of
someValuesFrom(a:description)) restriction(a:d-uses someValuesFrom(intersectionOf(a:role restriction(a:attitude-
towards someValuesFrom(a:course)))))))

 Class(a: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 an attitude towards some course (right, power, duty, etc).
Notice that modal descriptions can appear in conventionalized descriptions as well as in idiosyncratic
assessements, narratives, promises, etc.")
)

 Class(a:narrative partial
  a:informal-encoding-system
  a:description
  restriction(a:represented-by someValuesFrom(a:text)))

 Class(a:natural-person partial
  a:agentive-physical-object)

 Class(a:natural-person partial
  annotation(rdfs:comment "A person ontologically dependent on  an organism")
)

 Class(a:non-agentive-functional-object complete
  intersectionOf(a:non-agentive-physical-object restriction(a:plays someValuesFrom(a:role))))

 Class(a:non-agentive-role partial
  a:role)

 Class(a:non-agentive-functional-role partial
  annotation(rdfs:comment " A non-agentive role is the specification of a function without an (internal or external)
intention (e.g. 'container', 'burnt area', etc).
More precisely, the intention could be provided by someone playing an agentive role in the same description, but
the non-agentive role is not agentive per se (for example, it cannot represent a desire towards some course).”
)

 Class(a:non-agentive-physical-object partial
  a:physical-object)

 Class(a: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(a:non-agentive-temporal-role partial
  a:non-agentive-functional-role)

 Class(a:non-agentive-temporal-role partial
```

```
 annotation(rdfs:comment "A functional role for talking of something at certain phases of its own life. It can be
used also to map temporal parts of non-agentive objects from a 4D ontology.")
)

 Class(a:non-physical-collection partial
  restriction(a:has-member minCardinality(2))
  a:non-physical-object)

 Class(a: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.")
)

 Class(a:non-physical-endurant partial
  restriction(a:has-quality allValuesFrom(a:abstract-quality))
  a:endurant
  restriction(a:constituent allValuesFrom(a:non-physical-endurant))
  restriction(a:part allValuesFrom(a:non-physical-endurant)))

 Class(a:non-physical-endurant partial
  annotation(rdfs:comment "An endurant with no mass, generically constantly depending on some intentional
agent.")
)

 Class(a:non-physical-object partial
  restriction(a:part allValuesFrom(a:non-physical-object))
  restriction(a:generically-depends-on someValuesFrom(a:communication))
  restriction(a:generically-depends-on someValuesFrom(unionOf(a:agentive-physical-object a:agentive-role)))
  a:non-physical-endurant)

 Class(a: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(a:non-physical-place partial
  restriction(a:physically-depends-on someValuesFrom(a:physical-object))
  a:non-agentive-functional-role)

 Class(a:non-physical-place partial
  annotation(rdfs:comment "AKA locative role. This is a role (e.g. closed area) or a figure (e.g. Italy) for places.
Non-physical places physically depend on physical objects (in locational cases, physical places).")
)

 Class(a:non-right complete
  intersectionOf(a:modal-description restriction(a:d-uses someValuesFrom(intersectionOf(a:role restriction(a:non-
right-towards someValuesFrom(a:course)))))))

 Class(a:norm partial
  a:regulation)

 Class(a:obligation partial
  a:modal-description)

 Class(a:organization partial
  a:socially-constructed-person)

 Class(a:parallel-task complete
  intersectionOf(restriction(a:direct-successor minCardinality(2)) a:concurrency-task))

 Class(a:parameter partial
  restriction(a:defined-by someValuesFrom(a:description))
  restriction(a:requisite-for allValuesFrom(unionOf(a:role a:figure a:course)))
  a:concept
  restriction(a:valued-by someValuesFrom(a:region))
  restriction(a:valued-by allValuesFrom(a:region)))
```

```
Class(a:parameter partial
 annotation(rdfs:comment "A c-description that selects (in particular, it is 'valued by') regions, as a component of some s-description. Parameters are the descriptive counterpart of regions, and, as regions represent the qualities of perdurants or endurants, they can be requisites for some functional role or course.
A parameter has at least one region that is value for it.")
)

Class(a:partly-case-task complete
 intersectionOf(restriction(a:direct-successor someValuesFrom(a:case-task)) restriction(a:direct-successor someValuesFrom(intersectionOf(complementOf(a:case-task) a:task))) a:branching-task))

Class(a:path complete
 intersectionOf(restriction(a:sequences allValuesFrom(a:phenomenon)) a:course))

Class(a:path partial
 annotation(rdfs:comment "A course used to sequence phenomena (non-intentional processes).")
)

Class(a:patient-role partial
 restriction(a:functionally-depends-on someValuesFrom(unionOf(a:substrate-role a:agent-case-role)))
 a:case-role)

Class(a:patient-role partial
 annotation(rdfs:comment "Patient is a role played by some endurant that participates in a perdurant without carrying it out, either without doing it intentionally but being affected by it, or by having a 'passive' intentionality.")
)

Class(a:perdurant partial
 restriction(a:participant someValuesFrom(a:endurant))
 restriction(a:participant allValuesFrom(a:endurant))
 restriction(a:part allValuesFrom(a:perdurant))
 restriction(a:has-quality allValuesFrom(a:temporal-quality))
 a:entity
 restriction(a:has-quality someValuesFrom(a:temporal-location_q))
 restriction(a:constituent allValuesFrom(a:perdurant)))

Class(a: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(a:phenomenon partial
 a: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
 a:non-agentive-physical-object)

Class(a:physical-endurant partial
 restriction(a:has-quality allValuesFrom(a:physical-quality))
 a:endurant
 restriction(a:part allValuesFrom(a:physical-endurant))
```

```
 restriction(a:has-quality someValuesFrom(a:physical-quality))
 restriction(a:constituent allValuesFrom(a:physical-endurant)))

Class(a:physical-object partial
 a:physical-endurant)

Class(a: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(a:physical-phenomenon partial
 a:phenomenon)

Class(a:physical-place partial
 a:non-agentive-physical-object)

Class(a:physical-quality partial
 a:quality
 restriction(a:inherent-in someValuesFrom(a:physical-endurant))
 restriction(a:q-location allValuesFrom(a:physical-region))
 restriction(a:inherent-in allValuesFrom(a:physical-endurant))
 restriction(a:has-quality allValuesFrom(a:physical-quality)))

Class(a:physical-region partial
 restriction(a:part allValuesFrom(a:physical-region))
 restriction(a:q-location-of allValuesFrom(a:physical-quality))
 a:region)

Class(a: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(a:plan partial
 restriction(a:d-uses someValuesFrom(unionOf(a:figure a:agentive-role)))
 a:method
 restriction(a:proper-part someValuesFrom(a:goal))
 restriction(a:represented-by allValuesFrom(a:information-object))
 restriction(a:d-uses someValuesFrom(a:task)))

Class(a: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 also use either an agentive role or figure.
Finally, a plan has a goal as proper part, and can also have regulations and other descriptions as proper parts.")
)

Class(a:plan-assessment-quality complete
 intersectionOf(a:assessment-quality restriction(a:inherent-in someValuesFrom(a:plan))))

Class(a:plan-execution complete
 intersectionOf(restriction(a:p-sat someValuesFrom(a:plan)) a:situation))

Class(a: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(a:plan-information complete
 intersectionOf(restriction(a:represents someValuesFrom(a:plan)) restriction(a:represents
someValuesFrom(a:plan)) a:information-object restriction(a:present-at someValuesFrom(a:time-interval))))

Class(a:plan-information partial
```

```
 annotation(rdfs:comment "Documents, models, or diagrams that present the information about a plan.")
)

 Class(a:planning-activity partial
  a:activity)

 Class(a:political-geographic-object partial
  a:geographical-role
  restriction(a:physically-depends-on someValuesFrom(a:geographical-object)))

 Class(a:power complete
  intersectionOf(restriction(a:defines someValuesFrom(intersectionOf(a:role restriction(a:power-towards
someValuesFrom(a:course))))) a:modal-description))

 Class(a:predicate-name partial
  a:formal-expression)

 Class(a:privilege complete
  intersectionOf(restriction(a:d-uses someValuesFrom(intersectionOf(a:role restriction(a:privilege-towards
someValuesFrom(a:course))))) a:modal-description))

 Class(a:procedural-quality complete
  intersectionOf(a:temporal-quality restriction(a:t-inherent-in someValuesFrom(a:activity))))

 Class(a:process partial
  a:stative)

 Class(a: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.")
)

 Class(a:project partial
  a:method)

 Class(a: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(a:promise partial
  a:commitment)

 Class(a:proper-noun partial
  a:linguistic-object)

 Class(a:quale complete
  intersectionOf(restriction(a:proper-part cardinality(0)) a:region))

 Class(a:qualitative-role partial
  a:non-agentive-functional-role)

 Class(a:quality partial
  restriction(a:q-location allValuesFrom(a:region))
  restriction(a:inherent-in someValuesFrom(a:entity))
  a:entity
  restriction(a:has-quality allValuesFrom(a:quality)))

 Class(a: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(a:quality-space complete
```

```
 intersectionOf(restriction(a:overlaps allValuesFrom(complementOf(a:quality-space))) a:region))

 Class(a: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(a:reconstructed-flux complete
  intersectionOf(a:flux restriction(a:has-member allValuesFrom(a:accomplishment))))

 Class(a:reconstructed-flux partial
  annotation(rdfs:comment "Reconstructed fluxes are fluxes that only contain  accomplishments as members.")
)

 Class(a:region partial
  a:abstract
  restriction(a:part allValuesFrom(a:region))
  restriction(a:q-location-of allValuesFrom(a:quality)))

 Class(a: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(a:regulation partial
  a:description)

 Class(a:regulation partial
  annotation(rdfs:comment "A description usually requiring a C-SAT satisfaction for a situation. Norms, codes of
practice, etc. are examples.")
)

 Class(a:relevant-part partial
  a:feature)

 Class(a:responsibility partial
  restriction(a:d-uses someValuesFrom(a:status))
  restriction(a:d-uses someValuesFrom(a:task))
  a:commitment)

 Class(a: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(a:right complete
  intersectionOf(a:modal-description restriction(a:d-uses someValuesFrom(intersectionOf(a:role restriction(a:right-
towards someValuesFrom(a:course)))))))

 Class(a:role partial
  restriction(a:requisite allValuesFrom(a:parameter))
  restriction(a:played-by allValuesFrom(a:endurant))
  a:concept
  restriction(a:attitude-towards allValuesFrom(a:course))
  restriction(a:defined-by someValuesFrom(a:description)))

 Class(a:role partial
  annotation(rdfs:comment "Also known as 'functional role'.
A concepts that selects (in particular, it is  'played by') endurants, as a component of some description.  Roles are
the descriptive counterpart of endurants, and, as endurants participate in perdurants, they usually have attitudes
towards descriptions of perdurants. This relation is named 'modality target', because it actually reifies at first order
a typology of modal relations.")
)

 Class(a:s-context partial
```

```
 restriction(a:played-by allValuesFrom(a:description))
 a:semiotic-role)

Class(a:s-context partial
 annotation(rdfs:comment "S-Context (semiotic context) is played by descriptions and are semiotic roles. They
are used to fill the second domain of the  so-called 'interpretation function'. It may be equivalent to the 'context'
communication role, but since communication theory and semiotic theories are different, it is more correct to say
that a c-context (communication context) *plays* an s-context.")
)

Class(a:saturated-plan complete
 intersectionOf(restriction(a:d-uses cardinality(1)) restriction(a:d-uses cardinality(1)) a:plan))

Class(a: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(a:schedule complete
 intersectionOf(a:task restriction(a:requisite cardinality(1))))

Class(a: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(a:semiotic-code partial
 a:combinatorial-system)

Class(a:semiotic-role partial
 a:non-agentive-functional-role
 restriction(a:defined-by someValuesFrom(a:interpretation-function)))

Class(a:semiotic-role partial
 annotation(rdfs:comment "A semiotic role is a non-agentive role defined by an interpretation function.
It should be played within a communication setting by a description that participates in a communication (act).
Semiotic roles are used to fill the universe of the so-called 'interpretation function'.
Two of them are specialized by two communication functions (message and context).")
)

Class(a:sequential-task partial
 a:complex-task
 restriction(a:component cardinality(0))
 restriction(a:component minCardinality(2)))

Class(a: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(a:set partial
 a:abstract)

Class(a:simple-node partial
 a:flow-chart-node)

Class(a:situation complete
 intersectionOf(restriction(a:satisfies someValuesFrom(a:description)) restriction(a:setting-for
someValuesFrom(a:entity)) a:entity))

Class(a:situation partial
 restriction(a:part allValuesFrom(a:situation)))
```

```
 Class(a:situation partial
  annotation(rdfs:comment "A situation is an entity 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 of settings (situations fron SC, contexts, episodes, states of affairs,
structures, configurations, cases, etc.).
This results to be a new category in DOLCE, but it could be equivalently modelled as a special complex perdurant
defined through its relations to qualities, regions, and endurants. In fact, 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 entities in the
setting.")
)

 Class(a:social-agent partial
  a:social-object
  a:agentive-role)

 Class(a:social-agent partial
  annotation(rdfs:comment "An agentive functional role created and maintained by a society")
)

 Class(a:social-description complete
  intersectionOf(restriction(a:descriptively-depends-on someValuesFrom(a:role)) a:social-object
restriction(a:physically-depends-on minCardinality(2)) a:description))

 Class(a: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(a:social-figure partial
  a:social-object
  a:figure
  restriction(a:generically-depends-on someValuesFrom(a:social-unit)))

 Class(a:social-figure partial
  annotation(rdfs:comment "A figure whose constitutive description is shared by a community.")
)

 Class(a:social-object complete
  intersectionOf(unionOf(a:social-role a:social-description a:social-agent a:social-figure) a:non-physical-object))

 Class(a:social-object partial
  annotation(rdfs:comment "A catch-all class for entities from the social world. It includes agentive roles, non-
agentive roles created by a community, social descriptions, and social figures.")
)

 Class(a:social-role partial
  a:social-object
  a:non-agentive-functional-role)

 Class(a:social-role partial
  annotation(rdfs:comment "A role created and maintained by a society.")
)

 Class(a:social-unit partial
  a:social-agent)

 Class(a:socially-constructed-person partial
  a:social-figure)

 Class(a:socially-constructed-person partial
  annotation(rdfs:comment "A definite social figure that is constructed 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(a:space-region partial
 restriction(a:part allValuesFrom(a:space-region))
 a:physical-region
 restriction(a:q-location-of allValuesFrom(a:spatial-location_q)))

Class(a:spatial-location_q partial
 a:physical-quality)

Class(a:spatio-temporal-region partial
 a:space-region)

Class(a:state partial
 a:stative)

Class(a: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.")
)

Class(a:statement partial
 a:linguistic-object)

Class(a:stative partial
 a:perdurant)

Class(a: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(a:status partial
 a:role)

Class(a:stylesheet partial
 a:combinatorial-system
 restriction(a:expression-means-for allValuesFrom(a:document)))

Class(a:subject partial
 restriction(a:plays someValuesFrom(a:s-context))
 a:classification-system)

Class(a:subject partial
 annotation(rdfs:comment "\"Any reified knowledge domain, informally referred. Intuitively, a formal description is
the formal counterpart of a subject, while an informal description is its informal counterpart.
Subjects are often 'opaque', meaning that no related list of information objects is provided (e.g. in flat catalogues).
On the other hand, any subject, together with the contents derivable from a referred information collection,
constitutes such a list.\"")
)

Class(a:subplan complete
 intersectionOf(restriction(a:proper-part-of someValuesFrom(a:plan)) a:plan))

Class(a:substance-role partial
 a:non-agentive-functional-role)

Class(a:substance-role partial
 annotation(rdfs:comment "A role played by some substance.")
)

Class(a:substrate-role partial
 a:case-role)

Class(a:substrate-role partial
 annotation(rdfs:comment "Substrate is a role played by some endurant that carries out a process or event, or
bears a state, without doing it intentionally. Another condition is that no part of the perdurant can exist if the
endurant (or its whole) playing the substrate-role does not exist. On the contrary, an agent-role provides
```

intentionality, and the perdurant carried out can be partly present even in absence of it or of its whole (other agent-roles can realize it.")
)

Class(a:suspension-task partial
a:ending-task)

Class(a:synchro-task complete
intersectionOf(restriction(a:direct-predecessor minCardinality(2)) a:control-task restriction(a:predecessor someValuesFrom(unionOf(a:any-order-task a:concurrency-task)))))

Class(a:synchro-task partial
restriction(a:represented-by allValuesFrom(a:join-node)))

Class(a:synchro-task partial
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.")
)

Class(a:system-as-artifact complete
intersectionOf(a:non-agentive-physical-object restriction(a:involved-in someValuesFrom(unionOf(a:project a:plan)))))

Class(a:system-as-description partial
a:description
restriction(a:satisfied-by allValuesFrom(a:system-as-situation)))

Class(a:system-as-situation partial
a:situation
restriction(a:satisfies someValuesFrom(a:system-as-description)))

Class(a:target-role partial
a:patient-role
restriction(a:functionally-depends-on someValuesFrom(a:agent-case-role)))

Class(a:task complete
intersectionOf(restriction(a:desire-target-of someValuesFrom(unionOf(a:agentive-role a:intentional-figure))) restriction(a:defined-by someValuesFrom(a:method)) a:course))

Class(a: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.
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 an agentive role or figure has a desire attitude towards them (possibly different from the one that put the task into action). In principle, tasks could be transformed into explicit plans.")
)

Class(a:technique partial
a:method)

Class(a:temporal-location_q partial
a:temporal-quality)

Class(a:temporal-quality partial
restriction(a:inherent-in allValuesFrom(a:perdurant))
restriction(a:has-quality allValuesFrom(a:temporal-quality))
a:quality
restriction(a:q-location allValuesFrom(a:temporal-region))
restriction(a:inherent-in someValuesFrom(a:perdurant)))

Class(a:temporal-region partial
restriction(a:part allValuesFrom(a:temporal-region))
a:region

```
 restriction(a:q-location-of allValuesFrom(a:temporal-quality)))

 Class(a:temporal-region partial
  annotation(rdfs:comment "A region at which only temporal qualities can be  directly located. It assumes a metrics
for time.")
)

 Class(a:term partial
  a:linguistic-object)

 Class(a:terminology partial
  a:classification-system)

 Class(a:text partial
  restriction(a:expressed-according-to someValuesFrom(a:language))
  a:linguistic-object)

 Class(a:text partial
  annotation(rdfs:comment "A complex linguistic object,  expressed according to a language and still independent
from  a particular physical support.")
)

 Class(a:theory partial
  restriction(a:d-uses someValuesFrom(a:formal-expression))
  a:axiomatic-system)

 Class(a:time-interval partial
  a:temporal-region)

 Class(a:unitary-collection partial
  a:non-agentive-physical-object
  restriction(a:has-member minCardinality(2)))

 Class(a:unitary-collection partial
  annotation(rdfs:comment "A collection of physical objects that is characterized by a conventional or emergent
property.")
)

 AnnotationProperty(rdfs:comment)

 Individual(a:abandoned
  type(a:procedural-quality))
 Individual(a:aborted
  type(a:procedural-quality))
 Individual(a:accepted
  type(a:plan-assessment-quality))
 Individual(a:activated
  type(a:procedural-quality))
 Individual(a:completed
  type(a:procedural-quality))
 Individual(a:considered
  type(a:plan-assessment-quality))
 Individual(a:decided
  type(a:procedural-quality))
 Individual(a:possible
  type(a:plan-assessment-quality))
 Individual(a:reactivated
  type(a:procedural-quality))
 Individual(a:ready
  type(a:procedural-quality))
 Individual(a:recorded
  type(a:procedural-quality))
 Individual(a:rejected
  type(a:plan-assessment-quality))
 Individual(a:reserved
  type(a:procedural-quality))
 Individual(a:suspended
  type(a:procedural-quality))
```

```
DisjointClasses(a:situation a:perdurant)
DisjointClasses(a:quality a:situation)
DisjointClasses(a:parameter a:course)
DisjointClasses(a:quality a:perdurant)
DisjointClasses(a:abstract a:endurant)
DisjointClasses(a:feature a:physical-object)
DisjointClasses(a:amount-of-matter a:physical-object)
DisjointClasses(a:abstract a:perdurant)
DisjointClasses(a:non-agentive-physical-object a:agentive-physical-object)
DisjointClasses(a:abstract a:situation)
DisjointClasses(a:arbitrary-sum a:physical-endurant)
DisjointClasses(a:physical-quality a:temporal-quality)
DisjointClasses(a:physical-region a:abstract-region)
DisjointClasses(a:physical-region a:temporal-region)
DisjointClasses(a:description a:concept)
DisjointClasses(a:endurant a:situation)
DisjointClasses(a:abstract-region a:temporal-region)
DisjointClasses(a:physical-quality a:abstract-quality)
DisjointClasses(a:abstract a:quality)
DisjointClasses(a:endurant a:quality)
DisjointClasses(a:role a:parameter)
DisjointClasses(a:role a:course)
DisjointClasses(a:physical-endurant a:non-physical-endurant)
DisjointClasses(a:abstract-quality a:temporal-quality)
DisjointClasses(a:arbitrary-sum a:non-physical-endurant)
DisjointClasses(a:endurant a:perdurant)
DisjointClasses(a:amount-of-matter a:feature)

EquivalentClasses(

intersectionOf(
 restriction(a:d-uses someValuesFrom(intersectionOf(restriction(a:sequences
someValuesFrom(intersectionOf(restriction(a:setting someValuesFrom(a:situation)) a:perdurant))) a:ending-
task)))
 restriction(a:d-uses someValuesFrom(intersectionOf(restriction(a:optionally-used-by cardinality(0))
restriction(a:direct-predecessor someValuesFrom(a:branching-task)) restriction(a:sequences
someValuesFrom(intersectionOf(restriction(a:setting someValuesFrom(a:situation)) a:perdurant)))
restriction(a:discarded-within cardinality(0)) a:action-task)))
 restriction(a:d-uses allValuesFrom(intersectionOf(restriction(a:requisite-for someValuesFrom(a:task))
restriction(a:valued-by someValuesFrom(intersectionOf(restriction(a:setting someValuesFrom(a:situation))
a:region))) a:parameter)))
 restriction(a:d-uses allValuesFrom(intersectionOf(restriction(a:sequences
someValuesFrom(intersectionOf(restriction(a:setting someValuesFrom(a:situation)) a:perdurant))) a:control-
task)))
 restriction(a:d-uses allValuesFrom(intersectionOf(restriction(a:discarded-within cardinality(0))
restriction(a:sequences someValuesFrom(intersectionOf(restriction(a:setting someValuesFrom(a:situation))
a:perdurant))) a:action-task restriction(a:optionally-used-by cardinality(0)) restriction(a:direct-successor
someValuesFrom(a:synchro-task)))))
 restriction(a:d-uses someValuesFrom(intersectionOf(unionOf(a:role a:figure) restriction(a:played-by
someValuesFrom(intersectionOf(a:endurant restriction(a:setting someValuesFrom(a:situation)))))))
a:plan
 restriction(a:d-uses allValuesFrom(intersectionOf(restriction(a:sequences
someValuesFrom(intersectionOf(restriction(a:setting someValuesFrom(a:situation)) a:perdurant)))
restriction(a:direct-predecessor cardinality(0)) a:action-task))))

intersectionOf(
 restriction(a:p-sat-by someValuesFrom(a:plan-execution))
 a:plan))

SubClassOf(
intersectionOf(restriction(a:d-used-by someValuesFrom(a:responsibility)) a:status)
intersectionOf(restriction(a:duty-towards someValuesFrom(intersectionOf(restriction(a:d-used-by
someValuesFrom(a:responsibility)) a:task))) restriction(a:right-towards
someValuesFrom(intersectionOf(restriction(a:d-used-by someValuesFrom(a:responsibility)) a:task)))))

SubClassOf(
intersectionOf(restriction(a:d-used-by someValuesFrom(a:plan)) a:task)
restriction(a:component-of someValuesFrom(intersectionOf(restriction(a:defined-by someValuesFrom(a:plan))
a:maximal-task))))
```

SubClassOf(
intersectionOf(restriction(a:generic-location-of someValuesFrom(restriction(a:setting someValuesFrom(a:situation)))) a:region)
intersectionOf(restriction(a:generic-location-of someValuesFrom(a:situation)) a:region))

SubPropertyOf(a:carries a:mediated-relation)
SubPropertyOf(a:makes a:co-participates-with)
SubPropertyOf(a:q-present-at a:mediated-relation)
SubPropertyOf(a:q-realized-by a:mediated-relation)
SubPropertyOf(a:concludes a:temporally-contained-in)
SubPropertyOf(a:selects a:references)
SubPropertyOf(a:immunity-towards a:legal-attitude-towards)
SubPropertyOf(a:satisfied-by a:references)
SubPropertyOf(a:fiat-place a:approximate-location)
SubPropertyOf(a:interpretant a:functional-depend-on-of)
SubPropertyOf(a:coincides a:temporal-relation)
SubPropertyOf(a:partly-compresent-with a:mediated-relation)
SubPropertyOf(a:host a:immediate-relation)
SubPropertyOf(a:represents a:references)
SubPropertyOf(a:participant a:immediate-relation)
SubPropertyOf(a:result-of a:mediated-relation)
SubPropertyOf(a:mereotopological-association a:mediated-relation)
SubPropertyOf(a:has-member a:constituent)
SubPropertyOf(a:temporary-part a:part)
SubPropertyOf(a:target-of a:patient-of)
SubPropertyOf(a:participant-place a:generic-location)
SubPropertyOf(a:sibling-task a:mediated-relation)
SubPropertyOf(a:plays a:selected-by)
SubPropertyOf(a:privilege-towards a:legal-attitude-towards)
SubPropertyOf(a:identity-c a:immediate-relation)
SubPropertyOf(a:power-towards a:legal-attitude-towards)
SubPropertyOf(a:expects a:mediated-relation)
SubPropertyOf(a:performs a:functional-participant-in)
SubPropertyOf(a:admits a:mediated-relation)
SubPropertyOf(a:use-within a:attitude-towards)
SubPropertyOf(a:rules a:mediated-relation)
SubPropertyOf(a:p-depends-on a:immediate-relation)
SubPropertyOf(a:non-right-towards a:legal-attitude-towards)
SubPropertyOf(a:disability-towards a:legal-attitude-towards)
SubPropertyOf(a:overlaps a:mediated-relation)
SubPropertyOf(a:references a:immediate-relation)
SubPropertyOf(a:liability-towards a:legal-attitude-towards)
SubPropertyOf(a:refines a:proper-part)
SubPropertyOf(a:subjected-to a:attitude-towards)
SubPropertyOf(a:has-state a:substrate-of)
SubPropertyOf(a:total-participant-in a:participant-in)
SubPropertyOf(a:q-represents a:mediated-relation)
SubPropertyOf(a:legal-attitude-towards a:attitude-towards)
SubPropertyOf(a:sequences a:selects)
SubPropertyOf(a:reference-theme a:co-participates-with)
SubPropertyOf(a:has-quale a:q-location)
SubPropertyOf(a:strong-connection a:mediated-relation)
SubPropertyOf(a:setting-for a:constituent)
SubPropertyOf(a:postcondition a:mediated-relation)
SubPropertyOf(a:uses a:co-participates-with)
SubPropertyOf(a:proper-part a:part)
SubPropertyOf(a:material-place a:place)
SubPropertyOf(a:instrument-of a:used-in)
SubPropertyOf(a:constituent a:immediate-relation)
SubPropertyOf(a:co-participates-with a:mediated-relation)
SubPropertyOf(a:specifically-constantly-dependent-on a:immediate-relation)
SubPropertyOf(a:t-inherent-in a:inherent-in)
SubPropertyOf(a:meets a:temporal-connection)
SubPropertyOf(a:place a:approximate-location)
SubPropertyOf(a:used-in a:functional-participant-in)
SubPropertyOf(a:attitude-towards a:references)
SubPropertyOf(a:exact-location a:generic-location)
SubPropertyOf(a:expected-setting-for a:mediated-relation)

SubPropertyOf(a:generically-depends-on a:immediate-relation)
SubPropertyOf(a:parametrized-by a:mediated-relation)
SubPropertyOf(a:temporary-part a:partly-compresent-with)
SubPropertyOf(a:prescribes a:performs)
SubPropertyOf(a:duty-towards a:legal-attitude-towards)
SubPropertyOf(a:value-for a:selected-by)
SubPropertyOf(a:c-sat a:satisfies)
SubPropertyOf(a:starts a:temporally-contained-in)
SubPropertyOf(a:regulates a:satisfied-by)
SubPropertyOf(a:d-constituent a:mediated-relation)
SubPropertyOf(a:enforces a:involved-in)
SubPropertyOf(a:boundary-of a:proper-part-of)
SubPropertyOf(a:duration a:temporal-location)
SubPropertyOf(a:e-depends-on a:specifically-constantly-dependent-on)
SubPropertyOf(a:sibling-part a:mediated-relation)
SubPropertyOf(a:p-spatial-location a:exact-location)
SubPropertyOf(a:weak-connection a:immediate-relation)
SubPropertyOf(a:exploits a:mediated-relation)
SubPropertyOf(a:consequence-of a:functional-participant-in)
SubPropertyOf(a:temporal-intersection a:temporal-relation)
SubPropertyOf(a:right-towards a:legal-attitude-towards)
SubPropertyOf(a:method-of a:expects)
SubPropertyOf(a:involves a:mediated-relation)
SubPropertyOf(a:identity-n a:immediate-relation)
SubPropertyOf(a:task-precondition a:mediated-relation)
SubPropertyOf(a:successor a:immediate-relation)
SubPropertyOf(a:d-uses a:temporary-component)
SubPropertyOf(a:patient-of a:functional-participant-in)
SubPropertyOf(a:descriptively-depends-on a:e-depends-on)
SubPropertyOf(a:discarded-within a:d-used-by)
SubPropertyOf(a:desire-towards a:attitude-towards)
SubPropertyOf(a:part a:immediate-relation)
SubPropertyOf(a:constant-participant a:participant)
SubPropertyOf(a:temporal-connection a:temporal-relation)
SubPropertyOf(a:resource-for a:used-in)
SubPropertyOf(a:generic-target-of a:functional-participant-in)
SubPropertyOf(a:bdi a:attitude-towards)
SubPropertyOf(a:precedes a:temporal-relation)
SubPropertyOf(a:r-sat a:satisfies)
SubPropertyOf(a:inherent-in a:immediate-relation)
SubPropertyOf(a:spatial-location a:physical-location)
SubPropertyOf(a:abstract-location a:exact-location)
SubPropertyOf(a:p-sat a:satisfies)
SubPropertyOf(a:present-at a:mediated-relation)
SubPropertyOf(a:temporary-proper-part a:proper-part)
SubPropertyOf(a:temporally-contains a:temporal-relation)
SubPropertyOf(a:depend-on-spatial-location a:exact-location)
SubPropertyOf(a:product-of a:functional-participant-in)
SubPropertyOf(a:approximate-location a:generic-location)
SubPropertyOf(a:physically-depends-on a:e-depends-on)
SubPropertyOf(a:complete-participant a:participant)
SubPropertyOf(a:specializes a:immediate-relation)
SubPropertyOf(a:indirectly-plays a:mediated-relation)
SubPropertyOf(a:theme a:patient)
SubPropertyOf(a:q-location a:immediate-relation)
SubPropertyOf(a:substrate-of a:total-participant-in)
SubPropertyOf(a:situation-place a:approximate-location)
SubPropertyOf(a:temporarily-depends-on a:e-depends-on)
SubPropertyOf(a:temporary-proper-part a:partly-compresent-with)
SubPropertyOf(a:component a:proper-part)
SubPropertyOf(a:task-postcondition a:mediated-relation)
SubPropertyOf(a:expands a:proper-part)
SubPropertyOf(a:direct-successor a:successor)
SubPropertyOf(a:iteration-interval a:mediated-relation)
SubPropertyOf(a:exit-condition a:successor)
SubPropertyOf(a:metaphorically-plays a:plays)
SubPropertyOf(a:happens-at a:mediated-relation)
SubPropertyOf(a:expressed-according-to a:mediated-relation)
SubPropertyOf(a:descriptive-origin a:fiat-place)

```
 SubPropertyOf(a:temporal-relation a:mediated-relation)
 SubPropertyOf(a:defines a:d-uses)
 SubPropertyOf(a:geographic-part-of a:fiat-place)
 SubPropertyOf(a:functional-participant a:participant)
 SubPropertyOf(a:functionally-depends-on a:e-depends-on)
 SubPropertyOf(a:requisite-for a:references)
 SubPropertyOf(a:realized-by a:references)
 SubPropertyOf(a:temporary-component a:partly-compresent-with)
 SubPropertyOf(a:temporary-participant a:participant)
 SubPropertyOf(a:temporary-component a:component)
 SubPropertyOf(a:temporal-location a:exact-location)
 SubPropertyOf(a:constrains a:expects)
 SubPropertyOf(a:parametrically-depends-on a:e-depends-on)
 SubPropertyOf(a:origin a:material-place)
 SubPropertyOf(a:precondition a:mediated-relation)
 SubPropertyOf(a:generic-location a:mediated-relation)
 SubPropertyOf(a:e-temporal-location a:exact-location)
 SubPropertyOf(a:realizes a:referenced-by)
 SubPropertyOf(a:optionally-used-by a:d-used-by)
 SubPropertyOf(a:physical-location a:exact-location)
 SubPropertyOf(a:r-location a:immediate-relation)

)
```

6.2 OWL-RDF abstract syntax of the sample Klett model

```
 Individual(Klett:acquire_a_development_plan
  type(a:goal))
 Individual(Klett:acquire_idea
  type(a:plan))
 Individual(a:assistant
  type(a:agentive-role)
  value(a:duty-towards Klett:disburden_project_manager))
 Individual(a:author
  type(a:agentive-role)
  value(a:obligation-towards Klett:provide_content)
  value(a:obligation-towards Klett:providing_info_on_administrative_issues_regarding_content))
 Individual(Klett:bring_high_profits_to_Klett
  type(a:goal))
 Individual(Klett:compilation_of_new_learning_material
  type(a:complex-task))
 Individual(Klett:compile_development_plan
  type(a:action-task))
 Individual(Klett:concept_design
  type(a:plan)
  value(a:d-uses Klett:technical_project_manager)
  value(a:d-uses Klett:project_manager)
  value(a:d-uses Klett:assistant)
  value(a:d-uses Klett:author)
  value(a:d-uses Klett:standard)
  value(a:proper-part Klett:acquire_a_development_plan)
  value(a:defines Klett:compile_development_plan)
  value(a:defines Klett:providing_info_on_administrative_issues_regarding_content)
  value(a:defines Klett:disburden_project_manager)
  value(a:defines Klett:provide_info_on_technical_standards)
  value(a:defines Klett:decide_on_content))
 Individual(Klett:concept_development
  type(a:plan)
  value(a:d-uses Klett:technical_project_manager)
  value(a:d-uses Klett:project_manager)
  value(a:d-uses Klett:assistant)
  value(a:d-uses Klett:disburden_project_manager)
  value(a:d-uses Klett:author)
  value(a:d-uses Klett:provide_info_on_technical_standards)
  value(a:d-uses Klett:standard)
  value(a:proper-part Klett:develop_a_pilot_version_of_new_learning_material)
  value(a:defines Klett:provide_content)
  value(a:defines Klett:coordinate_compilation_of_new_learning_material)
```

```
 value(a:defines Klett:set_deadlines_to_authors))
Individual(Klett:coordinate_compilation_of_new_learning_material
 type(a:any-order-task)
 value(a:direct-successor Klett:compilation_of_new_learning_material))
Individual(Klett:decide_on_content
 type(a:action-task))
Individual(Klett:develop_a_pilot_version_of_new_learning_material
 type(a:goal))
Individual(Klett:disburden_project_manager
 type(a:action-task))
Individual(Klett:producing_1_piece_of_new_learning_material
 type(a:plan)
 value(a:proper-part a:sales)
 value(a:proper-part a:production)
 value(a:proper-part Klett:bring_high_profits_to_Klett)
 value(a:proper-part Klett:concept_design)
 value(a:proper-part Klett:concept_development)
 value(a:proper-part Klett:acquire_idea))
Individual(a:production
 type(a:plan))
Individual(Klett:project_manager
 type(a:agentive-role)
 value(a:obligation-towards Klett:coordinate_compilation_of_new_learning_material)
 value(a:obligation-towards Klett:set_deadlines_to_authors)
 value(a:right-towards Klett:decide_on_content)
 value(a:duty-towards Klett:compile_development_plan))
Individual(Klett:provide_content
 type(a:action-task))
Individual(Klett:provide_info_on_technical_standards
 type(a:action-task))
Individual(Klett:providing_info_on_administrative_issues_regarding_content
 type(a:action-task))
Individual(Klett:right_as_value
 type(a:parameter)
 value(a:requisite-for a:standard))
Individual(a:sales
 type(a:plan))
Individual(Klett:set_deadlines_to_authors
 type(a:action-task))
Individual(a:standard
 type(a:non-agentive-role))
Individual(Klett:technical_project_manager
 type(a:agentive-role)
 value(a:obligation-towards Klett:provide_info_on_technical_standards))
```