A Behaviouristic Approach to Representing Processes and Procedures in the OASIS 2 Ontology

Giampaolo Bella[1], Gianpietro Castiglione[1] and Daniele Francesco Santamaria[1]

[1]*Department of Mathematics and Computer Science, University of Catania, Viale Andrea Doria 6 - 95125 - Catania, Italy*

Abstract

Foundational ontologies devoted to the effective representation of processes and procedures are not widely investigated at present, thereby limiting the practical adoption of semantic approaches in real scenarios where the precise instructions to follow must be considered. Also, the representation ought to include how agents should carry out the actions associated with the process, whether or not agents are able to perform those actions, the possible roles played as well as the related events.

The OASIS 2 ontology [1, 2] provides an established model to capture agents and their interactions but lacks means for representing processes and procedures carried out by agents. This motivates the research presented in this article, which delivers an extension of the OASIS 2 ontology to combine the capabilities for representing agents and their behaviours with the full conceptualization of processes and procedures. The overarching goal is to deliver a foundational OWL ontology that deals with agent planning, reaching a balance between generality and applicability, which is known to be an open challenge.

Keywords

Semantic Web, Ontology, OWL, Agent, Process, Procedure, Event, Process, Procedure

1. Introduction

The notions of process and procedure are broadly known in the literature, even outside the computer science field, although they bear subtle differences. According to ISO 9001-2015 definitions, a process is *a set of correlated or interactive activities that transform inputs into outputs* describing *the specified way of carrying out an activity,* whereas a procedure describes *how to carry out all or part of a process.* Hence, we can argue that a process may be composed of many procedures.

Representing processes and procedures through foundational *Web Ontology Language* (OWL) ontologies is necessary both for easing the exchange of process descriptions among systems and to provide a machine-understandable interpretation of the related activities. One of the main benefits of using Semantic Web technologies in such a context is the reasoning capability that permits the inference of new facts from existing data and the verification of the knowledge base. For example, in the context of operating systems, *race conditions* [3] could lead to problems related to the lock of resources by concurrent processes. A formalization of the processes,

procedures and agents together with semantic reasoning could reduce the ambiguity that may occur when agents attempt to perform more operations at the same time.

Defining an ontological architecture for processes and procedures aims at improving so-called "automated planning and acting" [4]. One of the most remarkable attempts to standardize the planning problem is the PDDL (Planning Domain Definition Language) [5], which separates the planning problem into a) domain description and b) problem description. For instance, the PDDL in its current version introduces *derived predicates* for dependency modelling of properties between objects. Therefore, processes and procedures benefit from the adoption of ontological representations because ontologies can fully address both planning sub-problems.

Although many ontological approaches are available in the literature, they suffer from the lack of a complete and general approach to effectively represent processes and procedures, especially one that a) combines the representation of processes and procedures with agents and their commitments, b) models the events generated during the executions of processes and c) accounts for the roles that are played by the committer agents.

The current paper is motivated by the delivery of such a model. We start by considering the ontological foundations for the domain of multi-agent systems. Specifically, we take into account OASIS 2 [1, 2], a foundational OWL 2 ontology that leverages the behaviouristic approach derived from the *Theory of Agents* and the related mentalistic notions. The behaviouristic approach is an effective way of semantically describing agents by characterizing their capabilities. Agents are enabled to report the set of activities that they can perform, the types of data required to execute those activities as well as the expected outputs, through the description of their behaviours. Agents' implementation details are abstracted away to make the discovery of agents transparent, automatic, and independent of the underlying technical infrastructure. In consequence, agent commitments are clearly described, and the entire evolution of its environment is unambiguously represented, searchable, and accessible: agents may join a collaborative environment in a plug-and-play fashion, as there is no more need for third-party interventions.

Our work rests on the observation that OASIS 2 has lacked a specific characterization of processes and procedures so far, even though it models so-called *plans*, which are ways of depicting a sequence of (planned) actions to be tackled once. Therefore, plans cannot be applied again once they are performed by agents. In this paper, we extend OASIS 2 to deal with general specifications of processes and procedures that can be consumed by agents, including events and the played role, that can be practically leveraged in real scenarios such as the one concerning race conditions.

The paper is organized as follows. Section 2 presents the related work through a comparison with the OASIS 2 approach; Section 3 introduces the model of OASIS 2 devoted to the representation of agents and their behaviours. Section 4 presents the novel extension of OASIS 2 that deals with the notions of processes and procedures. Section 5 closes the paper with some hints for future outcomes.

2. Related Work

In DOLCE [6], the concepts of processes and events are presented as special types of perdurants, whereas functions and roles are formalized in some extensions of the ontology [7, 8]. Aware of

the general definitions provided by DOLCE, OASIS 2 focuses on the behaviouristic approach which is leveraged to provide agents with novel means for representing complex planning and related actions, dealing also with events. Specifically, roles are conceived by OASIS 2 as ways of enabling agents with additional behavioural capabilities, whereas events are represented to be aligned with the definition of agent behaviours. From this point of view, OASIS 2 introduces a different conceiving of events and roles since it describes agents in terms of their capabilities. A full mapping of OASIS 2 in DOLCE is feasible and one of our planned future works.

Wang et al. introduce a model for processes related to water quality monitoring [9]. The model is strictly focused on the description of observational processes concerning water pollution monitoring, thus being limited to general applicability outside of the domain.

Within CIDOC-CRM [10], there is a work-in-progress extension called CRMact [11] that defines the classes and properties for planning future activities and events. However, CRMact is mainly focused on cultural heritage and documentation records and hence is not generally applicable.

Concerning business processes, it is worth mentioning several works. Thomas et al. propose an ontological approach for representing business processes together with a system architecture prototype, exploiting the proposed model [12]. Greco et al. use ontologies as facilities within a framework for assisting designers in the realization and analysis of complex processes [13]. Corea et al. present an approach to verify whether a business process is compliant with given business rules combining logic programming and ontologies [14], while Calvanese et al. propose an approach for using Ontology-Based Data Access (OBDA) to govern data-aware processes, and in particular, those executed over a relational database that issue calls to external services to acquire new information and update data [15]. Finally, a comparison among ontologies related to business processes for task monitoring, measurements and evaluation strategies, and for modelling process information was built [16], [17], [18].

However, the downside of limiting ontological models to business processes lies in the absence of relationships between agents and their commitments that instead represent the core of agent-oriented representation. That implies the inability of finding agents/services with specific capabilities, invoking them, and enabling their interoperability. Notably, the benefits of process-oriented representations, such as the facilitation mechanisms for the search and selection of process models, are also offered by the agent-oriented ones as long as they are sufficiently general although flexible.

3. Preliminaries on OASIS 2

The first version of OASIS [19] is a foundational ontology that leverages the behaviouristic approach to characterize agents in terms of the actions they are able to perform, including purposes, goals, responsibilities, information about the world they observe and maintain, and their internal and external interactions. It models the executions and assignments of tasks, restrictions and constraints used to establish agent responsibilities and authorizations.

In the recent past, OASIS has been extended and applied to deal with so-called *Ontological Smart Contracts* [20] and with the ontological models for smart contracts on the blockchain [21]. OASIS is also part of the POC4COMMERCE project [22], funded by the NGI-ONTOCHAIN

consortium [23].

The last version of OASIS is OASIS 2 [24, 1][1], which extends OASIS with some new features, such as the entrustment of agents, and reshapes the model adopted for the representation of agents and their commitments.

Inspired by the *Tropos* methodology [25] which devises from Agent Oriented Programming (AOP), OASIS 2 represents agents through three essential and publicly shared mental states, namely (expected) *behaviours*, *goals* and *tasks*. Behaviours represent the mental state of the agent associated with its ability to modify its environment or, in general, act or do something. Goals describe mental attitudes representing preferred progressions of a particular system that the agent has chosen to put effort into bringing about [26]. Tasks depict how to carry on such progressions and describe atomic operations that agents perform.

Agents and their interactions are represented by carrying out three main steps, namely: a) an optional step that consists of modelling descriptions of general abstract behaviours, called *templates*, conceptual characterization of behaviours from which concrete agent behaviours are drawn; b) modelling concrete agent behaviours, possibly, drawn by agent templates; c) modelling actions and associating them with the corresponding behaviours. The first step, not mandatory, consists in defining the agent's behaviour template, namely a higher-level description of the behaviour of abstract agents that can be leveraged to define concrete behaviours of real agents; for example, a template is designed to describe the abstract behaviour consisting in obtaining and releasing locks on resources. Additionally, templates are useful to guide developers in the definition of the behaviours of their specific agents. To describe abstract agent's capabilities to perform actions, an agent template comprises three main elements, namely behaviour, goal and task. The latter constitutes the most simple (atomic) operations that agents are able to actually perform including, possibly, input and output parameters required to accomplish them. The second step consists of representing concrete agent behaviours either by relying on a template or by defining it from scratch. In both cases, concrete behaviours are modelled analogously to those of templates, where the models of outstanding features are replaced with actual characteristics. Behaviours drawn by shared templates are associated with them in order to depict the behaviour inheritance relationship. In the last step, actions performed by agents are described as direct consequences of some behaviours and are associated with the behaviours of the agent that performed them. To describe such an association, OASIS 2 introduces *plan executions*. Plan executions describe the actions performed by an agent, associating them with one of its behaviours. Associations are carried out by connecting the description of the performed action to the behaviour from which the action has been drawn: actions are hence described by suitable graphs that retrace the model of the agent's behaviour.

Plans can be additionally either submitted to agents as requests for performing some actions or they can be assigned by specific agents called *entruster agents*.

In OASIS 2, agent templates are defined according to the UML class diagram in Fig. 1. To consider how both abstract and concrete agents perform actions, the description of agents comprises three main elements, namely *behaviour*, *goal*, and *task*. Agent tasks, in their turn, describe atomic operations that agents perform, including possibly input and output parameters required to accomplish them. Those elements in OASIS 2 are introduced by way of the following

---

[1]The OASIS 2 ontology can be reached at [2]

Figure 1: Diagram of agent templates in OASIS 2

OWL classes:

- *Agent*: This class comprises all the individuals representing agents. Instances of such a class are connected with one or more instances of the class *Behaviour* using the OWL object-property *hasBehaviour*.
- *Behaviour*: Behaviours can be seen as collectors comprising all the goals that an agent may achieve. Instances of *Behaviour* are connected with one or more instances of the class *GoalDescription* by means of the object-property *consistsOfGoalDescription*.
- *GoalDescription*: Goals represent containers embedding all the tasks that the agent can achieve. Instances of *GoalDescription* comprised by a behaviour may also satisfy dependency relationships introduced by the object-property *dependsOn*. Goals are connected with the tasks that form them and are represented by instances of the class *TaskDescription* through the object-property *consistsOfTaskDescription*.
- *TaskDescription*: This class describes atomic operations that agents perform. Atomic operations are the most simple actions that agents are able to execute and, hence, they represent what agents can do within their environment. Atomic operations may depend on other atomic operations when the object-property *dependsOn* is specified. Atomic operations whose dependencies are not explicitly expressed are intended to be performed in any order.

The core of agent behaviour revolves around the description of atomic operations represented by instances of the class *TaskDescription* that characterizes the mental state corresponding to commitments. In their turn, instances of the class *TaskDescription* are related to the following five elements that identify the operation:

- An instance of the class *TaskOperator*, characterizing the mental state corresponding to the action to be performed. Instances of *TaskOperator* are connected either by means of the object-property *refersExactlyTo* or *refersAsNewTo* to instances of the class *Action*. The latter class describes physical actions represented by means of entity names in the form of infinite verbs (e.g., *produce*, *sell*). Specifically, the object-property *refersExactlyTo* is used to connect the task operator with a precise action having a specific IRI, whereas *refersAsNewTo* is used to connect a task operator with an entity representing an action of which only a general abstract description is given (for example, an action for which only the type is known).

  In the latter case, the entity representing the action is also defined as an instance of the *TemplateThing*: such instances are used to define entities that represent templates for the referred element and that describe the characteristics that such element should satisfy. *TemplateThing* is the class used to characterize all the individuals involved in the definition of behaviour templates and to distinguish them from the entities representing concrete behaviours, plans or actions, thus eliminating the need of having separated models for those aspects.

  In order to specify the classes of which the entity must be an instance, it is eventually possible to connect such entity by means of the object-property *refersAsInstanceOf* with the individual instances of the desired classes.

- Possibly, an instance of the class *TaskOperatorArgument*, connected using the object-property *hasTaskOperatorArgument* and representing additional specifications or subordinate characteristics of task operators (e.g., *on*, *off*, *left*, *right*). Instances of *TaskOperatorArgument* are referred to the operator argument by using either the object-property *refersAsNewTo* or *refersExactlyTo*.

- An instance of the class *TaskObject*, connected by means of the object-property *hasTaskObject* and representing the template of the object recipient of the action performed by the agent (e.g., *price*). Instances of *TaskObject* are referred to the action recipient by specifying either the object-property *refersAsNewTo* or *refersExactlyTo*.

- Input parameters and output parameters are introduced by instances of the classes *TaskInputParameter* and *TaskOutputParameter*, respectively. Instances of *TaskDescription* are related to instances of the classes *TaskInputParameter* and *TaskOutputParameter* by means of the object-properties *hasTaskInputParameter* and *hasTaskOutputParameter*, respectively. Instances of *TaskInputParameter* and of *TaskOutputParameter* are referred to the parameter by specifying either the object-property *refersAsNewTo* or *refersExactlyTo*. Moreover, the classes *TaskInputParameter* and *TaskOutputParameter* are also subclasses of *TaskParameter*.

Finally, in the case of agent behaviour templates, instances of *Agent*, *Behaviour*, *GoalDescription*, *TaskDescription*, *TaskOperator*, *TaskOperatorArgument*, *TaskObject TaskInputParameter*, and *TaskOutputParameter* are also instances of *TemplateThing*.

Fig. 2 illustrates the case study mentioned above where a lock on a resource can be obtained and released by agents. Specifically, Fig. 2 presents a template describing an abstract agent that is able to request a lock on a resource. The agent template comprises a single behaviour,
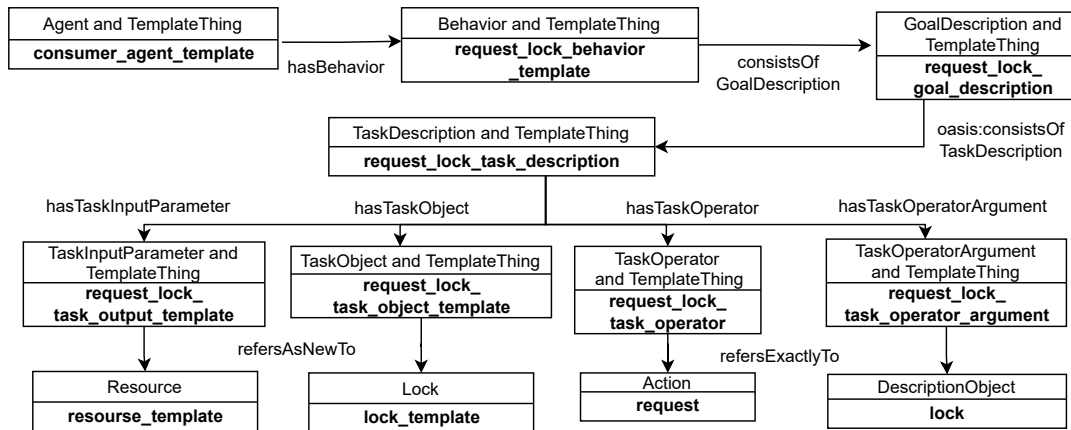
Figure 2: Example of an OASIS 2 agent template

constituted by a single goal that in its turn comprises a single task. The task, which represents the ability to request a lock, provides four elements:

- *request_lock_task_operator*, representing the mental state associated with the behaviour's action (the task operator), which in its turn is associated with the individual *request*, the latter describing the capability of requesting something.
- *request_lock_task_operator_argument*, introducing an additional feature (the operator argument) associated with the action and represented by the individual *lock*. The argument describes the fact that the request action is referred to a lock. Task operator and its argument describe together the capability of requesting a lock;
- *request_lock_task_object_template*, representing the recipient of the operation, which is related to an instance of the class *Lock* by means of the object-property *refersAsNewTo*. Such an instance comprises all the features that the recipient of the adopting action should own: the concrete actions implementing the behaviour template for requesting a lock are supposed to effectively request a lock with the desired features;
- *request_lock_task_output_template*, representing the input of the operation, namely a resource on which the lock is requested.

In the second step, concrete agent behaviours are defined either by instantiating one or more templates or from scratch. In OASIS 2, the modelling pattern of concrete behaviours has a structure analogous to one of the behaviour templates, illustrated above, with the difference that individuals used to define a concrete behaviour, instead of being instances of the class *TemplateThing*, are instances of the class *BehaviourThing*. The latter class is devoted to describe all the mental states associated with concrete behaviours of real agents that induce actions.

Concrete behaviours may be connected with the template they are drawn from. In order to describe the fact that concrete agents inherit their behaviours from a commonly shared template, the instances related to the concrete behaviours are connected with the instances of the template through the sub-properties of the object-property *overloads* as follows. The association is carried out by connecting the instances of the classes:

- *Behaviour*, by means of *overloadsBehaviour*;
- *GoalDescription*, by means of *overloadsGoalDescription*;
- *TaskDescription*, by means of *overloadsTaskDescription*;
- *TaskObject*, by means of *overloadsTaskObject*;
- *TaskOperator*, by means of *overloadsTaskOperator*;
- *TaskInputParameter*, by means of *overloadsTaskInputParameter*;
- *TaskOutputParameter*, by means of the object-property *overloadsTaskOutputParameter*.

As the last step, agent commitments devised from behaviours are introduced to describe agent actions. In OASIS 2, commitments are represented by adopting the same pattern presented for abstract behaviours with the difference that instances of the class *TemplateThing* are instead modelled as instances of the class *ExecutionThing* and the agent responsible for the execution of the action is related with the plan representing the commitment by means of the object-property *performsPlanExecution*, subproperty of *performs*. The class *ExecutionThing* is introduced to characterize all the entities involved in the definition of concrete and already performed actions and to distinguish them from the ones introduced for templates, behaviours and plans.

In order to relate agent commitments with the behaviour from which they are drawn, subproperties of the object-property *drawnBy* are introduced. Specifically, *planExecutionDrawnBy* connects the instance of *GoalDescription* of the agent action to its analogue of agent behaviour; much in the same way, *goalExecutionDrawnBy* connects the instance of the class *GoalDescription* of the commitment with its analogue, while *taskExecutionDrawnBy*, *taskObjectDrawnBy*, *taskOperatorDrawnBy*, *taskInputParameterDrawnBy*, and *taskOutputParameterDrawnBy* are introduced for *TaskDescription*, *TaskObject*, *TaskOperator*, *TaskInputParameter*, and *TaskOutputParameter*, respectively.

Usually, agents proposing plans identify the behaviours responsible for their realization beforehand, in such a way as to completely describe and trace how agent intentions are realized. In this case, the entities representing the submitted plan are related to the entities describing the responsible behaviour by means of suitable subproperties of the object-property *submittedTo*, relating instances of *PlanningThing* with instances of *BehaviourThing* as follows: a) *planDescriptionSubmittedTo*, for instances of *Behaviour*; b) *goalDescriptionSubmittedTo*, for instances of *GoalDescription*; c) *taskDescriptionSubmittedTo*, for instances of *TaskDescription*; d) *taskObjectSubmittedTo*, for instances of *TaskObject*; e) *taskOperatorSubmittedTo*, for instances of *TaskOperator*; f) *taskInputParameterSubmittedTo*, for instances of *TaskInputParameter*; g) *taskOutputParameterSubmittedTo*, for instances of *TaskOutputParameter*.

In a similar way, plans are also related to the agent's actions realizing them. For this purpose, the subproperties of the object-property *hasExecution* are introduced, namely *hasPlanExecution*, *hasGoalExecution*, *hasTaskExecution*, *hasTaskObjectExecution*, *hasTaskOperatorExecution*, *hasTaskInputParameterExecution*, and *hasTaskOutputParameterExecution*. Analogously, actions are connected with the behaviour responsible for their execution by means of suitable subproperties of the object-property *executionDrawnBy*.

4. Processes and procedures in OASIS 2

Since the first version, OASIS was already capable of describing how agent activities are carried out by suitably combining behaviours, plans and actions. Plans, in particular, can be used to describe in detail how inputs are processed to be turned into outputs, but they are not sufficient to describe general complex processes. Plans require the presence of a committer agent beforehand and can be applied only once. This is because the mental state related to the desire or wish to perform actions does not abstract from the committer agent. Plans can be leveraged to represent single applications of procedures but they should be combined properly. Then, plans can be associated with the behaviours responsible for their executions thus carrying out the actions required to accomplish the procedures. The approach is illustrated in Fig. 3. Leveraging the ISO definition, processes defined by agents are constituted by procedures. In the behaviouristic vision, the procedures states are associated with plans that are executed thus leading to actions. Actions derive from agent behaviours that in their turn can rely on specific templates. Agents performing actions are provided with behaviours either intrinsically or through a played role. Finally, actions could lead to events either incidentally or as foreseen by procedures. In the first case, events are conceived as noteworthy and extraordinary happenings that have not been previously planned, in the second case, as recurrent situations.
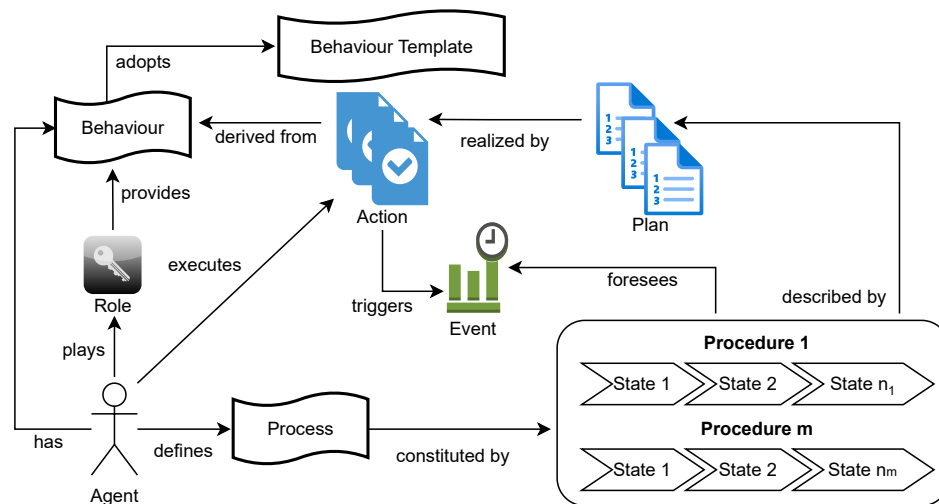
Figure 3: Process and procedure in OASIS 2

For example, agents may play the role of resource manager, hence they are able to request and release locks on resources according to such role. A process describing the steps required to request and release locks is formalized, so that they can tackle a plan for each step. Plans are executed so that the resource is released and modified. Actions are performed thanks to agent behaviours provided by the resource manager's role. During the execution of one of those actions, the system exceptionally sends a message to the committer agent. We will partially see this scenario together with the presentation of the model.

In light of the above considerations, OASIS 2 introduces the following novel OWL classes:

- *Process*, which encompasses the procedures describing how activities are carried out and;
- *Procedure*, the subclass of the class *Activity*, which introduces the set of plans required to accomplish the procedure itself and to be realized through actions. In OASIS 2 procedures are constituted by one or more *ProcedureState*, each one connected to a specific plan describing how the activity is carried out. Specifically, we identify two types of *ProcedureState*: a) *TerminatingProcedureState*, which includes *InitialProcedureState* and *FinalProcedureState*, and b) *NonTerminatingProcedureState*.

Procedures are constituted by *procedure states* that describe single steps of the procedure. *Initial procedure states* describe the beginning of the procedure, while *final procedure states* its termination. Moreover, the initial procedure state coincides with the final procedure state in the case of a single-step procedure. Finally, *non-terminating procedure states* describe the intermediate steps to be performed, including all the steps between the initial state and the final state. The schema for processes and procedures is illustrated in Fig. 4. The subproperties of the object-property *procedureConsistsOfProcedureState* are used to suitably connect a procedure with its procedure states. Specifically, the object-properties *procedureConsistsOfInitialProcedureState* and *procedureConsistsOfFinalProcedureState* (both subproperty of *procedureConsistsOfTerminatingProcedureState*) connect the procedure with its initial and final procedure state, respectively, while the object-property *procedureConsistsOfNonTerminatingProcedureState* connects the procedure with its non-terminating states. The initial procedure state is connected with the subsequent non-terminating procedure states by means of the object-property *hasNextNonTerminatingProcedureState*. The latter property is also used to connect non-terminating procedure states with their subsequent non-terminating procedure states. Whenever the next procedure state is the final state, the object-property *hasFinalProcedureState* is adopted in its place. Both *hasNextNonTerminatingProcedureState* and *hasFinalProcedureState* are defined as subproperty of the object-property *hasNext*.

Finally, since the intended meaning of processes and related procedures is to give a description of how activities should be carried out, the instances introduced so far are also instances of the class *PlanningThing*. To complete the representation of processes, it is now sufficient to connect an instance of the class *Process* with the ones of the class *Procedure* that model the process activities. In case a process is constituted by more than one procedure, the latter can be sorted by connecting them through the object-property *hasNextProcedure*.

To describe how process activities should be carried out, we introduce a behaviour as in Fig. 1, where the instances of *TemplateThing* are instead defined as instances of *PlanningThing*. The choice is motivated by the fact that templates describe abstract behaviours that are instantiated to introduce concrete behaviours, whereas plans describe actions that agents wish to perform or see accomplished: this is exactly the case of process, where a set of actions must be tackled in order to achieve the desired end. Additionally, plans allow one to describe in detail what actions are required to accomplish the activity associated with the procedure state, the input parameters and the expected output, without the need of selecting an agent responsible for those actions beforehand. To associate a procedure state with the planned behaviour, OASIS 2 introduces the object-property *isDescribedBy*.

The process concerning the case study considered at the beginning of the section is depicted in Fig. 5. The process describes the modification of a resource which requires the acquisition
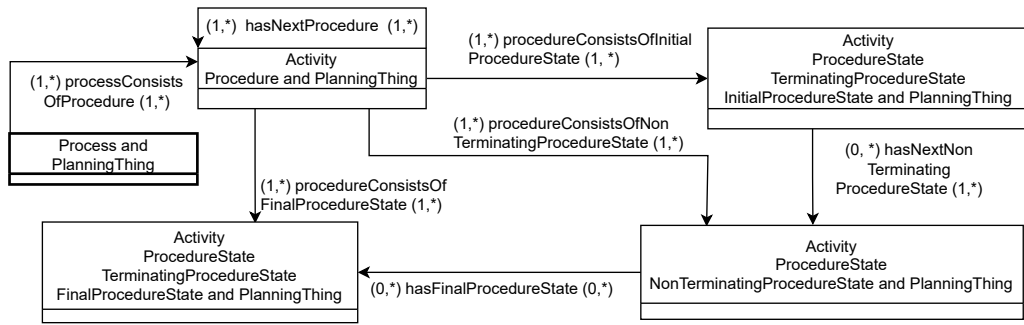
Figure 4: The OASIS 2 model for processes and procedures

and release of an exclusive lock before and after the modification, respectively.
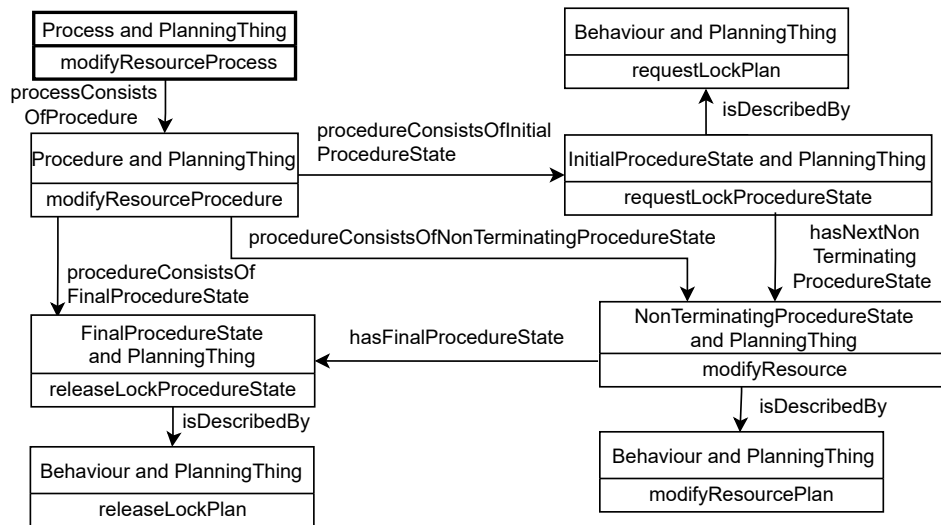

Figure 5: Example of process in OASIS 2

The process consists of a single procedure that, in its turn, consists of three distinct states, an initial state, a non-terminating state, and a final state. The initial state is associated with a behaviour describing how to request a lock on a resource. The behaviour follows the template in Fig. 2, where instances of the class *TemplateThing* are replaced with instances of the class *PlanningThing* as stated in Section 3. Analogously, the non-terminating state is associated with a behaviour describing how to modify the resource, while the final state is associated with the behaviour describing how to release the lock.

Once the process is fully described, it is possible to model its application. To do so, a model retracing the structure of a process (see Fig. 4) is introduced, where instances of the class *PlanningThing* are instead defined as instances of the class *ExecutionThing*. In a similar way to *TemplateThing* and *PlanningThing* that are introduced to represent descriptions of abstract behaviours and planned actions, respectively, *ExecutionThing* is conceived for those mental

states that represent actions already committed. In the case of processes, the class provides a means for characterizing the realization of the process, the achievement of the related activities, and the commitment of the planned actions. Moreover, to uniquely relate the elements of the process with the ones of its realization, two subproperties of the object-property *drawnBy* are introduced. Specifically, *processDrawnBy* and *procedureDrawnBy* are introduced to connect the instances of *Process* and *Procedure*, respectively. Hence, it is clear how to trace back to the agent's behaviour responsible for the execution of a process, thus unambiguously identifying actors and actions of arbitrarily complex environments.

For instance, the process of the example in Fig. 5 can be realized as partially depicted in Fig. 6 that illustrates the modification process on a resource. The schema retraces the one introduced for the process, where instances of the class *PlanningThing* are suitably replaced by instances of the class *ExecutionThing*. Specifically, the process realization introduces three behaviour executions, one for each plan of the process, namely a) *requestLockBehaviourExec*, representing the execution of the *requestLockPlan* plan; b) *modifyResourceBehaviourExec*, representing the execution of the *modifyResourcePlan* plan; c) *releaseLockBehaviourExec*, representing the execution of the *releaseLockPlan* plan.

Moreover, each behaviour of the process is connected with its analogous of the process realization by means of the object-property *hasPlanExecution*. Finally, each plan execution can be connected with the responsible behaviour by means of the object-property *planExecution-DrawnBy* (subproperty of *executionDrawnBy*, see Section 3).
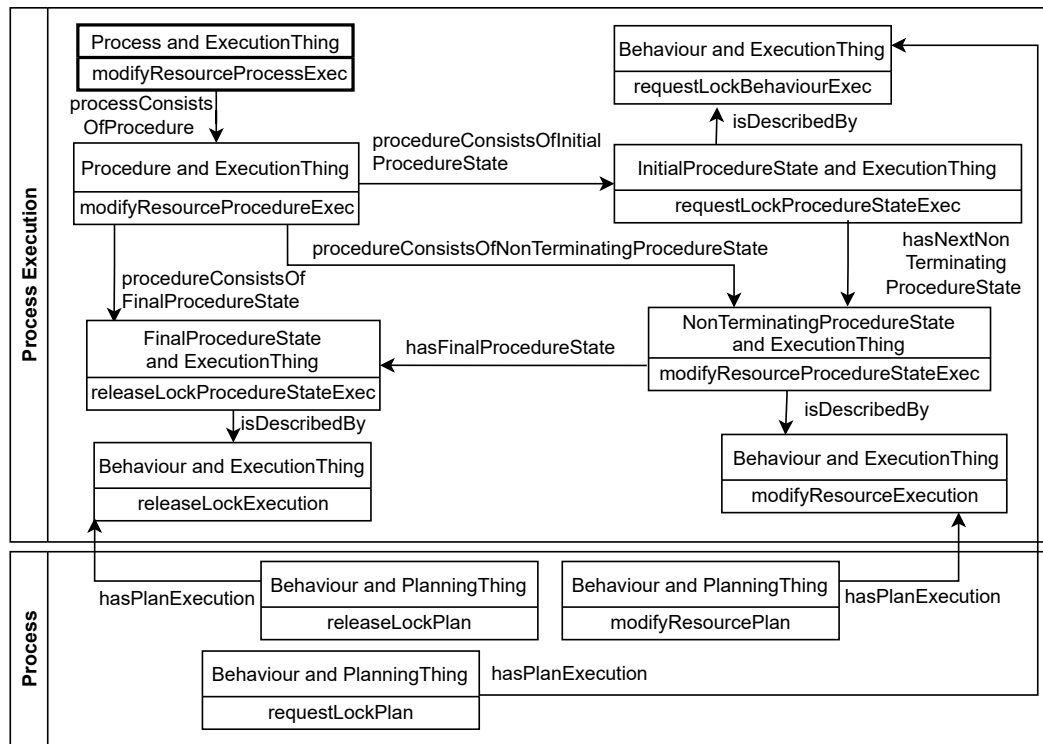

Figure 6: Example of process realization in OASIS 2

As stated above, in OASIS 2 specific events can be associated with procedure states. In OASIS 2 events are semantically represented by instances of the class *Event*, while the procedure state is related to the triggered event by means of the object-property *triggersEvent*. The phenomenon associated with the event is introduced by means of OASIS 2 actions. These are in fact sufficiently general and powerful to potentially cover up any type of phenomenon springing from an event that is fully within the vision of a behaviouristic approach.

In the case of the case study, when the system is triggered to send a message to selected users when a resource is modified, for example, because an error occurred, an action such as the one in Fig. 7 is introduced. Other information such as the type of event, its duration, or the time window in which it happens, can be additionally specified.
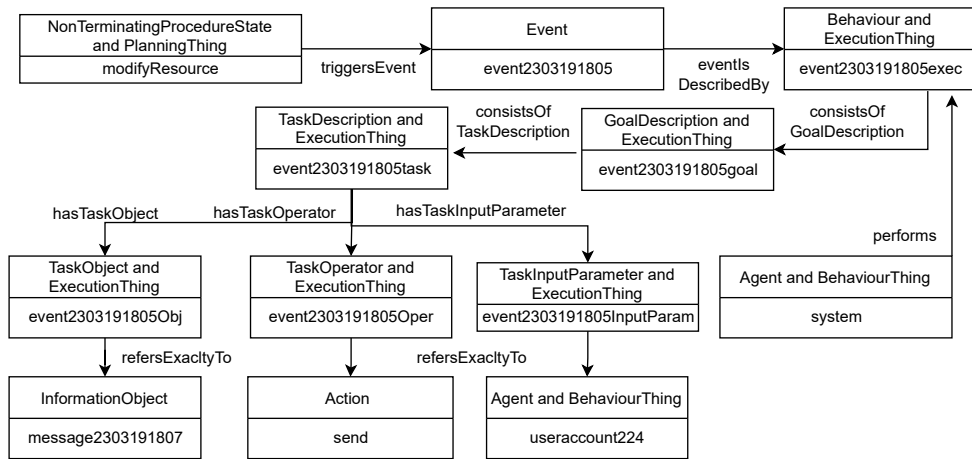

Figure 7: Example of event in OASIS 2

Finally, it is admissible that agents perform procedures exclusively according to the specific role that they play. Roles permit the separation of the concerns of behaviours intrinsically owned by agents from the ones temporarily at their disposal to execute a process. This implies that the agent is able to perform an action only when such a role is played. Hence, the behaviour responsible for performing those actions is not strictly associated with the agent but with the role. To represent the scenario described above and to model the playing of roles, OASIS 2 introduces roles as depicted in Fig. 8.
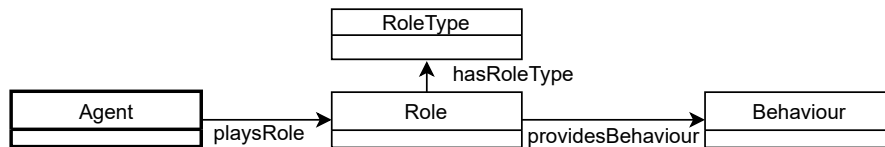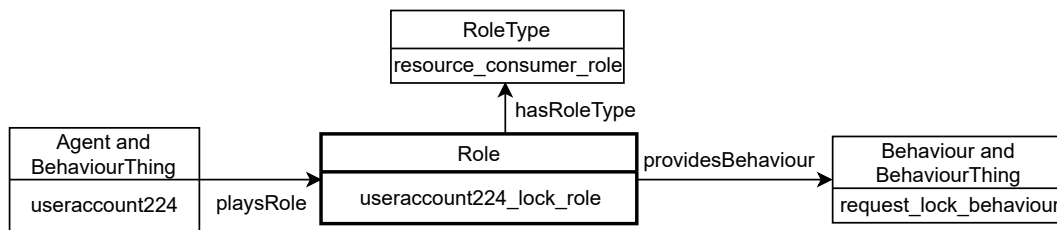

Figure 8: Roles in OASIS 2

In OASIS 2 roles, introduced as instances of the class *Role*, provide agents with behaviours by means of the object-property *providesBehaviour*. In its turn, a role is associated with the

agent playing it by means of the object-property *playRole*. Whenever agents cease to play a role, the associated behaviours are no longer available. The end of a role is specified by making the instance of *Role* deprecated, hence the instance of the class *DeprecatedThing*. This implies that actions are associated with role-oriented behaviours until the corresponding role is not deprecated. Concerning the case study, an agent playing the role consisting in requesting a lock on a resource is depicted in Fig. 9. The role is introduced as a fresh instance of the class *Role*, specifically provided for the agent's representational needs. The role types, introduced as an instance of the class *RoleType*, are defined accordingly to the domain to be described, hence they are out of the OASIS 2 scopes: in the case of the case study, a specific role type, called *resource_consumer_role*, is introduced to depict the roles dedicated for consuming resources.

Figure 9: Example of playing roles in OASIS 2

5. Conclusions

This paper introduced an extension of the OASIS 2 ontology and, therefore, inherits the behaviouristic approach to semantically represent agents and their commitments through the formalization of their mental states, namely, *behaviour*, *goal* and *task*. In particular, the proposed extension of OASIS 2 deals with the modelling of processes and procedures, thereby providing a general although practical way of representing processes and procedures to be tackled by agents through their behaviours. Now that such a foundational ontology is available, our semantic representation capabilities are substantially enhanced. In consequence, all application scenarios where specific instructions must be followed have fallen into scope.

For example, how agents reach a consensus, together with the modelling of their behaviours, is one of the future challenges. An application of OASIS 2 is foreseen for the characterisation of security directives, aiming at a structural solution for translating security documents to a mathematically-driven world. The approach targets the NIS 2 directive but other similar directives can be addressed. In addition, we intend to apply OASIS 2 to represent security constraints for cybersecurity threat contexts, in particular for the purpose of semantically representing authentication and confidentiality properties for agents.

Also, we shall consider how to integrate OASIS 2 with the PDDL and with the main frameworks such as JADE [27], all with the aim of automatically generating agents and artefacts. Similarly, an integration with *CArtAgO* [28], a framework for building shared computational worlds, appears to be valuable. The horizons of the semantic representation of agents have substantially expanded but retain the vast potential for yet more expansion in the near future.

Acknowledgments

References

[1] G. Bella, D. Cantone, C. F. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, The Ontology for Agents, Systems and Integration of Services: OASIS version 2, Intelligenza Artificiale, Vol. 17, no 1 (2023) 51–62.

[2] G. Bella, D. Cantone, M. Nicolosi-Asmundo, D. F. Santamaria, OASIS 2, 2022. URL: https://github.com/dfsantamaria/OASIS/tree/v2/OASIS-ontologies.

[3] R. H. B. Netzer, B. P. Miller, What are race conditions? some issues and formalizations, ACM Lett. Program. Lang. Syst. 1 (1992) 74–88. URL: https://doi.org/10.1145/130616.130623. doi:10.1145/130616.130623.

[4] M. Ghallab, D. Nau, P. Traverso, Automated Planning and Acting, Cambridge University Press, 2016. doi:10.1017/CBO9781139583923.

[5] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL—The Planning Domain Definition Language, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[6] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, Sweetening Ontologies with DOLCE, in: A. Gómez-Pérez, V. R. Benjamins (Eds.), Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web: 13th International Conference, EKAW 2002 Sigüenza, Spain, October 1–4, 2002 Proceedings, Springer, 2002, pp. 166–181.

[7] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, N. Guarino, Social roles and their descriptions., in: D. Dubois, C. A. Welty, M.-A. Williams (Eds.), KR, AAAI Press, 2004, pp. 267–277. URL: http://dblp.uni-trier.de/db/conf/kr/kr2004.html#MasoloVBCFGG04.

[8] S. Borgo, P. Carrara, Garbacz, P. E. Vermaas, Formalizations of functions within the dolce ontology, in: Proceedings of the Eighth International Symposium on Tools and Methods of Competitive Engineering TMCE, volume 1, 2010, pp. 113–126.

[9] X. Wang, H. Wei, N. Chen, X. He, Z. Tian, An observational process ontology-based modeling approach for water quality monitoring, Water 12 (2020). URL: https://www.mdpi.com/2073-4441/12/3/715.

[10] G. Bruseker, N. Carboni, A. Guillem, Cultural heritage data management: The role of formal ontology and cidoc crm, in: M. L. Vincent, V. M. López-Menchero Bendicho, M. Ioannides, T. E. Levy (Eds.), Heritage and Archaeology in the Digital Age: Acquisition, Curation, and Dissemination of Spatial Cultural Heritage Data, Springer International Publishing, Cham, 2017, pp. 93–131. URL: https://doi.org/10.1007/978-3-319-65370-9_6.

[11] CRMact Working Group, CRMact Model for Activity Plan, 2022. Https://cidoc-crm.org/crmact/.

[12] O. Thomas, M. Fellmann, Semantic process modeling – design and implementation of an

ontology-based representation of business processes, Business & Information Systems Engineering 1 (2009) 438–451.

[13] G. Greco, A. Guzzo, L. Pontieri, D. Saccà, An ontology-driven process modeling framework, in: F. Galindo, M. Takizawa, R. Traunmüller (Eds.), Database and Expert Systems Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 13–23.

[14] C. Corea, P. Delfmann, Detecting compliance with business rules in ontology-based process modeling, Wirtschaftsinformatik und Angewandte Informatik (2017).

[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Montali, A. Santoso, Ontology-based governance of data-aware processes, in: M. Krötzsch, U. Straccia (Eds.), Web Reasoning and Rule Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 25–41.

[16] A. Annane, M. Kamel, N. Aussenac-Gilles, Comparing business process ontologies for task monitoring, in: International Conference on Agents and Artificial Intelligence (ICAART 2020), SciTePress, Valletta, MT, 2020, pp. 634–643. doi:10.5220/0008978706340643, https://www.scitepress.org/Link.aspx?doi=10.5220/0008978706340643.

[17] P. Becker, F. Papa, L. Olsina, Process Ontology Specification for Enhancing the Process Compliance of a Measurement and Evaluation Strategy, CLEI Electronic Journal 18 (2015) 3 – 3. URL: http://www.scielo.edu.uy/scielo.php?script=sci_arttext&pid=S0717-50002015000100003&nrm=iso.

[18] S. Lee, S. Isaac, T. Bock, An ontology for process information modeling, in: M. Malaska, R. Heikkilä (Eds.), Proceedings of the 32nd International Symposium on Automation and Robotics in Construction and Mining (ISARC 2015), International Association for Automation and Robotics in Construction (IAARC), Oulu, Finland, 2015, pp. 1–7. doi:10.22260/ISARC2015/0032.

[19] D. Cantone, C. F. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, C. Santoro, Towards an Ontology-Based Framework for a Behavior-Oriented Integration of the IoT, in: Proceedings of the 20th Workshop From Objects to Agents, 26-28 June, 2019, Parma, Italy, CEUR Workshop Proceeding Vol. 2404, 2019, pp. 119–126.

[20] D. Cantone, C. F. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, C. Santoro, Ontological Smart Contracts in OASIS: Ontology for Agents, Systems, and Integration of Services, in: D. Camacho et al. (eds.), Intelligent Distributed Computing XIV, Studies in Computational Intelligence 1026, Chapter 22, 2022, pp. 237–247.

[21] G. Bella, D. Cantone, C. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, Blockchains through ontologies: the case study of the Ethereum ERC721 standard in OASIS, in: D. Camacho et al. (eds.), Intelligent Distributed Computing XIV, Studies in Computational Intelligence 1026, Chapter 23, 2022, pp. 249–259.

[22] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo, D. F. Santamaria, Semantic representation as a key enabler for blockchain-based commerce, K. Tserpes et al. (Eds.): GECON 2021, Springer Lecture Note in Computer Science 13072 (2021) 1–8.

[23] NGI-ONTOCHAIN, Ontochain a new software ecosystem for trusted, traceable & transparent ontological knowledge, 2020. URL: https://ontochain.ngi.eu/.

[24] G. Bella, D. Cantone, M. Nicolosi-Asmundo, D. F. Santamaria, The ontology for agents, systems and integration of services: recent advancements of oasis, in: Proceedings of WOA 2022- 23nd Workshop From Objects to Agents, 1–2, September 2022, Genova, Italy, CEUR Workshop Proceedings ISSN 1613-0073, Vol. 3261, 2022, pp. 176–193.

[25] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An agent-oriented software development methodology, in: Autonomous Agents Multi Agent Systems, volume 8:3, 2004, pp. 203–236.

[26] M. B. van Riemsdijk, M. Dastani, M. Winikoff, Goals in agent systems: A unifying framework, in: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), AAMAS 08, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 713–720.

[27] F. Bergenti, G. Caire, S. Monica, A. Poggi, The first twenty years of agent-based software development with jade, Autonomous Agents and Multi-Agent Systems 34 (2020).

[28] A. Ricci, M. Piunti, M. Viroli, A. Omicini, Environment Programming in CArtAgO, in: Multi-Agent Programming: Languages, Tools and Applications, Springer US, Boston, MA, 2009, pp. 259–288.