The Ontology for Agents, Systems and Integration of Services: recent advancements of OASIS

Giampaolo Bella[1], Domenico Cantone[1], Marianna Nicolosi Asmundo[1] and Daniele Francesco Santamaria[1]

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

Abstract

Semantic representation is, especially through the use of Semantic Web technologies, a key enabler for many application domains, notably thanks to the representation of agents. A semantic description of agents can be practically achieved by taking a behaviouristic approach, whose essential aim is to define the operational capabilities of every agent. These are leveraged to enable agents to operate and engage with their peers. The OASIS ontology — An Ontology for Agent, Systems, and Integration of Services, presented in 2019 — pursues the behaviouristic approach to deliver a higher-level, semantic representation system as well as a communication protocol for agents and their commitments. Following the profitable adoption of OASIS as a foundational ontology for agents in the context of blockchain-oriented e-commerce, the ontology has been maintained and substantially updated over the years. This paper reports on the modelling choices and general advancements since the first publication of OASIS, especially after its profitable adoption by an European project.

Keywords

Semantic Web, Ontology, OWL, Agent, Multi-Agent Systems

1. Introduction

The gist of the *Semantic Web* [1] is to achieve the full interoperability of software applications by means of common data formats, exchange protocols, data reuse and data sharing across systems, enterprises and community boundaries. In the Semantic Web vision of the Internet, software agents are enabled to query and manipulate information on behalf of human agents by means of machine-readable data. Such data carries explicit meaning, namely expressed with formally defined semantics through suitable representation languages supporting the automatic process of information. For this purpose, the *Word Wide Web Consortium* (W3C) conceived the *Web Ontology Language* (**OWL**) [2], a family of knowledge representation languages relying on *Description Logics* [3], as the standard language for representing Semantic Web ontologies. The main advantage of the Semantic Web is to enable users and applications to access data-oriented web content: users and applications become able to discover and invoke web resources in a

WOA 2022: 23rd Workshop From Objects to Agents, September 1–2, Genova, Italy

[†] These authors contributed equally.

giampaolo.bella@unict.it (G. Bella); domenico.cantone@unict.it (D. Cantone); marianna.nicolosiasmundo@unict.it (M. Nicolosi Asmundo); daniele.santamaria@unict.it (D. F. Santamaria)
https://www.dmi.unict.it/giamp/ (G. Bella); https://www.dmi.unict.it/cantone/ (D. Cantone); https://www.dmi.unict.it/nicolosi/ (M. Nicolosi Asmundo); https://www.dmi.unict.it/santamaria/ (D. F. Santamaria)

completely automatic way. One of the most important web resources is the one that provides access to services. Generally speaking, services are intended as artifacts that do not merely provide static information but perform actions and changes over the surrounding environment. Much effort of the Semantic Web community has been oriented at the creation of Semantic Web Services [4], namely intelligent Web Services with semantically represented content, in order to enable automatic discovery, composition and invocation of services. Therefore, in those environments where people and organizations interact with autonomous systems in a dynamic way, even Semantic Web Services struggle to capture the complex interactions of the participants. This is why it is most convenient to identify the proactive stakeholders as autonomous agents [5]. Agents are defined as entities whose state is perceived as a set of mental components such as beliefs, capabilities, choices, and commitments [6], forming the foundations for the *Agent Oriented Programming* (AOP) paradigm. AOP is most often motivated by the need for open architectures that continuously evolve to adapt to external changes in a robust, autonomous, and proactive way. Coherently with this vision, defining suitable Semantic Web ontologies to represent agents still is an open challenge in the large realm of *Knowledge Representation*.

The authors' choice for the semantic description of agents focuses on the agents' mental components. Then, the behaviouristic approach inspired by the *Tropos* methodology [7] provides a useful operational semantics. Agent behaviours may be decomposed in the atomic and essential mental states, namely the tasks that each agent seeks to accomplish. The *Ontology for Agents, Systems and Integration of Services* [8], OASIS in short, applies the behaviouristic approach to deliver a representation system and a communication protocol for agents and their commitments. Agents are enabled to transmit to their peers the set of operations that they are able to perform and the type of data required to execute these. Each output is also identified in a clear, human-understandable and unambiguous way, so as to abstract from the implementation details and transparently automate the task agent discovery. Agents may then join a collaborative environment in a *plug-and-play* way, since third-party interventions are no longer required, thereby profiting from the Semantic Web features. For example, by means of Semantic Web technologies and of automated reasoners, data provides machine-understandable information that can be processed, integrated and exchanged by any type of agent. Data consistency can be easily verified, and information can be inferred and retrieved through what is already defined in the knowledge base.

This paper reports on the modelling choices and general advancements since the first publication of OASIS in 2019, mostly derived from its adoption by the ONTOCHAIN project. Focus is on the representation of agents and their commitments, and on the recent outcomes derived from taking the behaviouristic approach. Specifically, the paper illustrates the modelling approach concerning: a) *agent behaviour templates*, namely, general descriptions of agent behaviours that can be implemented and specialized by real-world agents, b) *concrete agent behaviours*, and c) *agent actions* and how they evolved since the first approach. In addition to their representations, the paper presents how behaviour templates, concrete behaviours and the resulting commitments are related to each other. OASIS also provides the definition of *plan* that is a request forwarded to agents in order to accomplish some actions associated with their behaviours. Another aspect concerning the agent realm and addressed by OASIS deals with the *entrustment*

of agents, which represents the act of engaging agents to perform requested actions.

The paper is organized as follows. Section 2 presents related works about ontologies for services and agents. Section 3 first reports on recent works on OASIS, particularly on the outcomes from its adoption to blockchain-based e-commerce following an European project, and then unfolds the behaviouristic approach of OASIS and the related aspects mentioned above. Finally, Section 4 concludes the paper with future perspectives.

2. Related Works

In the context of the semantically representing application artifacts, the Semantic Web aims at enabling users to locate, select, employ, compose, and monitor web-based services automatically, giving rise to *Semantic Web Services*. Semantic Web Services describe Web Services with semantic content thereby automatically enabling service discovery, composition, and invocation. For this reason, the DARPA Agent Markup Language (DAML) pursued the goal of developing a markup language for representing semantic relations in a machine readable way [9]. In 2003, the *World Wide Web Consortium* (W3C) proposed OWL-S [10], an ontology for describing web services and related information. OWL-S tries to enable several tasks such as automatic web services discovery, automatic web services invocation, and automatic web services composition and interoperation. OWL-S explicitly supports the description of services as classes of activities, so that agents can decide how to use, invoke, and interpret responses from them. However, since DARPA and OWL-S are conceived to describe services, they do not adequately fulfil the requirements of describing general-purpose agents operating in different types of environments [11]. Indeed, describing agents as they coincide with services, although of a limited kind, may lead to an inaccurate and ambiguous depiction of them, also because service capabilities need to be semantically described both at a high and a low level. Hence, one of the most prominent visions of the relationships between agents and services is one in which agents *exploit*, *use*, *are composed of*, *are deployed as*, and *are extended by services* [11], which remains relatively simple.

The integration of agent systems and the Semantic Web has been investigated in the last decade in several contexts [9, 12, 13] and the advantages of ontology-based applications have been recognized in the realm of MAS [14]. Those advantages are manifest a) in the context of agent matching, i.e., the capability of finding agents satisfying specific requirements and automatically engaging them, b) in the context of developing agents with standard features exploiting shared common semantic tools, or c) as decision support for supply chains.

Ontologies for MAS have been modeled by taking approaches similar to those of *Agent-Oriented Software Engineering* (AOSE) [15, 16], a software engineering paradigm for the development of complex MAS, based on the abstraction of agent roles and on their organizations. Other approaches inspired by *belief-desire-intention* agent architectures (BDI) [17] are proposed in [7] and [18]. Inspired by Bratman's theory of human practical reasoning [19], BDI systems focus on the idea that agents have certain goals (desires) and a set of plans whose realization leads to their achievement; plans are selected, thus becoming intentions, depending on the agent's perception of the circumstances (represented by a set of beliefs). Beliefs, desires, and intentions are specified at a high level, often using a powerful logic/declarative approach: this

enables the implementation of complex behaviours while still keeping code transparent and readable. The AOSE approach is intended to support all analysis and design activities in the software development process, whereas the BDI approach provides a flexible environment offering both traditional object-oriented imperative and declarative constructs, enabling the definition of robot's high-level behaviours in a simple, natural way. In [20], an ontology for agent-oriented software engineering is proposed together with a tool that uses the ontology to generate programming code for MAS, but without examining agents and their interactions in detail. Moreover, an appropriate modelling of agent communication is missing.

Some results attempt to bring uniformity and coherence to the increasing volume and diversity of information in a specific domain, but domain-legacy generally makes them not applicable to other contexts even if they provide interesting insight for a general approach. For example, in [21], the authors propose an ontology-based framework for seamlessly integrating agents and Semantic Web Services, focusing on biomedical information. In [22], an infrastructure to allow agent-oriented platforms to access and query domain-specific OWL ontologies is presented. In [23], an approach is introduced to design scalable and flexible agent-based manufacturing systems integrating automated planning with multi-agent oriented programming for the *Web of Things* (WoT). In particular, autonomous agents synthesize production plans using semantic descriptions of web-based artifacts and coordinate with one another via multi-agent organizations. Concerning the *Internet of Things* (IoT), ontological approaches mainly focus on the description of sensors, with the purpose of collecting data associated with them for generating perceptions and abstractions of the observed world [24].

A comprehensive ontology for representing IoT services is presented in [25] together with a discussion on how it can be used to support tasks such as service discovery, testing and dynamic composition, taking into account also parameters such as *Quality of Services* (QoS), *Quality of Information* (QoI), and IoT service tests. A unified semantic knowledge base for IoT is proposed in [26], capturing the complete dynamics of IoT entities, for example enabling semantic searching while hiding their heterogeneity. Agent-based IoTs are comprehensively discussed in [27].

In [28], the authors discuss the unification of the state-of-the-art architectures, as put forward by the scientific community of the *Semantic Web of Things* (SWoT), by means of an architecture based on different abstraction levels, namely *Lower*, *Middle*, and *Upper Node* (LMU-N). In [29], a lightweight and scalable model, called *IoT-lite*, is proposed to describe key IoT concepts allowing interoperability and discovery of sensory data in IoT platforms.

The W3C advances a formal model and a common representation for WoT descriptions based on a small vocabulary that makes it possible both to integrate diverse devices and to allow diverse applications to interoperate [30]. The representation system provides a way to expose the state of a thing and to invoke functions that, however, must be known in advance, and this may complicate the task of invoking agents that would like to join the environment in a plug-and-play manner. Moreover, the provided schema does not fully allow agents to interact according to the specific roles they aim to play.

It is also noteworthy to mention works devoted to business processes, to which applications of Semantic Web ontologies are well known in literature. For instance, in [31] an ontological approach to represent business processes is presented together with a prototype of a system architecture exploiting the proposed model. In [32], ontologies are used as facilities within

a framework that assists designers in the realization and analysis of complex processes. An approach to verify whether a business process is compliant with given business rules combining logic programming and ontologies is presented in [33], whereas [34] proposes an approach of using Ontology-Based Data Access (OBDA) to govern data-aware processes, in particular those executed over a relational database that issues calls to external services to acquire new information and update the data. However, the downside of limiting ontological models to business processes lies at the absence of relationships between agents and their commitments that instead represents the core of agent-oriented representations. That implies the inability of finding agents/services with specific capabilities, invoke them, and enabling their interoperability. Notably, benefits of process-oriented representations such as the facilitation mechanisms to the search and selection of process models are also offered by the agent-oriented ones as long as they are sufficiently general albeit flexible.

3. Contributions and advancements of OASIS

The first version of OASIS was published in [8] together with the architecture design and implementation of a domotic assistant based on it, called Prof-Onto.[1] The approach employed in [8] also represents a first foundational contribution to adopt Semantic Web technologies for defining a transparent communication protocol among agents. The proposed protocol is based on the exchange of RDF-based fragments of OASIS, each consisting of a description of a request that is checked, by means of ad-hoc constructed queries, against the description of the behaviour of the agent selected to satisfy it.

In [36], the OASIS ontology is extended with *Ontological Smart Contracts* (in short, OSCs) and conditionals. OSCs are intended as agreements among agents expressed through suitable ontological fragments that allow one to establish responsibilities and authorizations among agents. Conditionals have many purposes, namely they are used a) to restrict and limit agent interactions, b) to define activation mechanisms that trigger agent actions, and c) to define constraints and contract terms on OSCs. Additionally, OSCs can be secured through digital public ledgers such as the blockchain. For this purpose, [36] also sketched the architecture of a framework based on the *Ethereum* blockchain and the *Interplanetary File System*. The framework leverages a suitable Ethereum smart contract to claim and transfer non-fungible tokens (NFT) compliant with the Ethereum ERC-721 standard and representing the ownership of OWL ontological fragments stored on the IPFS. Additionally, smart contracts allow one to verify unauthorized modifications both on the hierarchy of imported ontologies and on the SPARQL queries that verify the ontology itself. The approach also constitutes a first approach to deliver a *semantic blockchain*, where operations over smart contracts are semantically represented.

In [37], the definition of digital contracts is extended over the blockchain to include smart contracts, intended as programs running on the blockchain and interpreted as digital agents in the OASIS fashion, including their operational semantics and tokens exchanged through them. It represents the first approach to formalize ERC-721 compliant smart contracts through Semantic Web ontologies by leveraging the behaviouristic approach pursued by OASIS. The main capabilities of ERC-721 compliant smart contracts are represented through OASIS and are

[1]The latest version of Prof-Ontohas been recently renamed as Clara [35]

exploited to fully enable a semantic blockchain that provides agents and humans with means for probing the Ethereum blockchain, thus discovering smart contracts and their interactions, transactions, exchanges of cryptocurrencies and tokens, on the one hand, and building oracles for distributed ledgers, on the other.

The *Next Generation Internet initiative*, launched by the European Commission in 2018, funded the NGI-ONTOCHAIN project [38], which aims at providing solutions for a secure and transparent blockchain-based knowledge management system as an interoperability service for the Internet. In its turn, ONTOCHAIN funded the third-party project POC4COMMERCE [39] intended to build the ontological framework for the entire ONTOCHAIN ecosystem, albeit with a focus on the commercial domain. POC4COMMERCE is founded on an ontological stack that adopts and extends the most prominent ontologies in the modern literature suitably selected for the blockchain-oriented commerce domain with a behaviouristic vision of its essentials: commercial actors, offers, products, and tokens emitted on the Ethereum blockchain as digital witnesses of exchanged assets. Specifically, the stack provides three ontologies, one for each underlying sub-domain of knowledge, namely a) OC-Found, which exploits OASIS for the representation of commercial participants and smart contracts, b) OC-Commerce, which adopts GoodRelations and is responsible for describing commercial offerings, assets and activities under the vision of the behaviouristic approach, and c) OC-Ethereum, which exploits BLONDiE, focused on the representation of Ethereum smart contracts and related tokens in compliance with the standards ERC20 (for fungible tokens), ERC721 (for non-fungible tokens), and ERC1155 (for semi-fungible tokens). The approach of the proposed stack is presented in [39], while its design and analysis can be found in [40]. A semantic search engine [41] harnessing the expressive power of the ontological stack is under development, currently standing at TRL3, in the context of the ONTOCHAIN project that funded the realization of POC4COMMERCE.

Therefore, the adoption of BLONDiE and GoodRelations limited the expressive power of the ONTOCHAIN ontological stack whereas a pure behaviouristic approach widens the representational goals of stakeholders. For example, in OC-Ethereum both miners and users are merely presented as static entities and their capabilities of mining blocks and submitting transactions, respectively, is neglected. Analogously, in OC-Commerce negotiations of offerings are not deeply represented and buyers are not allowed to propose custom activities regarding supply chains related with goods and services. For example, buyers cannot request for alternative payment or shipping methods, or request for a discount. Moreover, it is not clear how offered assets and sold goods are related, thus preventing the distinction between what is offered and what is acquired from buyers. For this reason, we introduced two novel ontologies, where the dependencies of OC-Ethereum from BLONDiE and of OC-Commerce from GoodRelations, respectively, give way to a pure behaviouristic approach: the first ontology, called Ether-OASIS [42], carries out the approach in [37], while the second ontology called EC-OASIS and whose development is in progress, aims at overcoming the limitations of mixing the GoodRelations and the behaviouristic approaches mentioned earlier.

Here we present the OASIS modelling approach concerning agents and their commitments as they evolved thanks to their adoption by the ONTOCHAIN project. We recall that OASIS is a foundational OWL 2 ontology that leverages the behaviouristic approach derived from the *Theory of Agents* and the related mentalistic notions to represent multi-agent systems. The approach aims at describing how agents behave with respect to the environment by representing

their essential mental states, namely goals and tasks. While goals depict the long term desires of the agents, tasks represent the atomic operations that they are able to physically carry out.

Such behaviouristic approach is an effective way for semantically describing agents by characterizing their capabilities. Agents are enabled to report the set of activities that they are able to perform, the types of data required to execute those activities as well as the expected output through the description of their behaviours. Agents' implementation details are abstracted away so as to make the discovery of agents transparent, automatic, and independent of the underlying technical infrastructure. As a consequence, agent commitments are also clearly described and the entire evolution of the environment is unambiguously represented, searchable, and accessible, so agents may join a collaborative environment in a plug-and-play fashion, as there is no more need of third-party interventions. The behaviouristic approach is exploited by OASIS 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. Finally, OASIS models the executions and assignments of tasks, restrictions on them and constraints used to establish responsibilities and authorizations among agents. Thanks to the semantic representation of their behaviours, which are publicly exposed, agents are able to manifest the set and type of operations that they are able to perform, including the type of data required to execute them together with the expected output, so that they can be suitably queried or engaged.

In OASIS, the representation of agents and their interactions is carried out along three main steps, a) an optional step that consists in modelling general abstract behaviours (called *templates*) from which concrete agent behaviours are drawn, b) modelling concrete agent behaviours, possibly, drawn by agent templates, and c) modelling actions performed by agents by associating them with the behaviours from which actions are drawn.

The first step is not mandatory and represents the first main difference with the previous release of OASIS. It consists in defining the agent's behaviour template, namely a higher-level description of behaviours of abstract agents that can be implemented to define more specific and concrete behaviours of real agents. For example, a template may be designed to describe which features a smart contract compliant with the ERC721 standard should own. Moreover, templates are useful to guide developers to define the most suitable behaviours for their agents. To describe abstract agent's capabilities to perform actions, an agent template comprises three main elements, namely a) *behaviours*, representing the mental state associated with the agent's behaviours, b) *goals*, representing the mental state associated with the objective or target that the agent would reach or achieve, which are constituted by one or more c) *tasks*. Tasks, which represent the lower-level mental state of the agent, describe atomic operations that agents may perform including, possibly, input and output parameters required to accomplish the actions. Indeed, the core of OASIS agent representation revolves around the description of atomic operations introduced by the definition of tasks, i.e., the most simple (atomic) operations that agents are able to actually perform.

The second step consists in representing the concrete agent behaviours either by specifying a shared template or by defining it from scratch. In both cases, concrete behaviours are modelled analogously to those of templates, where the models of ideal features are replaced with actual characteristics. Behaviours drawn by shared templates are clearly associated with them in order to depict the behaviour inheritance relationship.

Finally, in the third 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 introduces *plan executions*. Plan executions describe the actions performed by an agent and associated with one of its behaviours. Associations are carried on 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 also submitted in order to request agents to accomplish some tasks. Moreover, specific agents are allowed to assign a plan to selected agents through an agent *entrustment activity*.

To describe how ideal agents should perform actions, an agent template comprises three main elements, namely *behaviours*, *goals* and *tasks*. Agent tasks, in their turn, describe atomic operations that agents may perform, including possibly input and output parameters required to accomplish them. The core of agent representation, indeed, 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 with 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 that are introduced by means of entity names in the form of infinite verbs and representing the actions (e.g., *produce*, *sell*). 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 action for which a general abstract description is provided. In the latter case, the action is also defined as instance of the class *ReferenceTemplate*: such instances are used to introduce entities that represent templates for the referred element describing the characteristics that it should satisfy. By exploiting the object-property *refersAsNewTo*, the entity provides only a general description of the features needed to accomplish the task, for example, that it must be of a specific type; on the contrary, the object-property *refersExactlyTo* specifies the actual and exact entity involved in the task.
- Possibly, an instance of the class *TaskOperatorArgument*, connected by means of the object-property *hasTaskOperatorArgument* and representing additional specifications for the task operator (e.g., *on*, *off*, *left*, *right*). Instances of *TaskOperatorArgument* are referred to the operator argument by specifying either the object-property *refersAsNewTo* or *refersExactlyTo*.
- An instance of the class *TaskObjectTemplate*, connected by means of the object-property *hasTaskObjectTemplate* and representing the template of the object recipient of the action performed by the agent (e.g., *price*). Instances of *TaskObjectTemplate* are referred to the action recipient by specifying either the object-property *refersAsNewTo* or *refersExactlyTo*.
- Input parameters and output parameters, introduced in OASIS by instances of the classes *TaskInputParameterTemplate* and *TaskOutputParameterTemplate*, respectively. Instances of *TaskDescription* are related with instances of the classes *TaskInputParameterTemplate* and *TaskOutputParameterTemplate* by means of the object-properties *hasTaskInputParam-*

eterTemplate and *hasTaskOutputParameterTemplate*, respectively. Instances of *TaskInput-ParameterTemplate* and of *TaskOutputParameterTemplate* are referred to the parameter by specifying either the object-property *refersAsNewTo* or *refersExactlyTo*. Moreover, the classes *TaskInputParameterTemplate* and *TaskOutputParameterTemplate* are also subclasses of the class *TaskParameterTemplate*.

The main classes characterizing an agent template in OASIS are the following ones:

- *AgentBehaviorTemplate*. This class comprises all the individuals representing templates of agents. Instances of such a class are connected with one or more instances of the class *Behavior* by means of the OWL object-property *hasBehavior*.
- *Behavior*. Behaviours of agent templates represent containers comprising all the goals that the agent can achieve. Instances of *Behavior* 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 are able to perform. Atomic operations are the most simple actions that agents are able to execute and, hence, they represent what agents can do within the 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. Finally, tasks are linked to the individuals that describe the features of the atomic operations, i.e., the instances of the classes *TaskOperator*, *TaskOperatorArgument*, *TaskObjectTemplate TaskInputParameterTemplate*, and *TaskOutputParameterTemplate*, as described above.
- *TaskOperator*. This class characterizes the type of operation to perform. Instances of *TaskOperator* are connected with instances of the class *Action* by means of either the object-properties *refersExactlyTo* or *refersAsNewTo*. Tasks are connected with task operators by means of the object-property *hasTaskOperator*.
- *Action*. This class describes actions that can be performed by agents. Entity names of the class *Action*'s instances are introduced as infinite verbs such as *buy*, *sell*, *compute*, and so on, drawn from a common, shared and extendable vocabulary defined by the OASIS-Abox ontology[43].
- *TaskOperatorArgument*. This class defines operator arguments representing subordinate characteristics of task operators. Tasks are connected with operator arguments by means of the object-property *hasTaskOperatorArgument*.
- *TaskObjectTemplate*. Instances of this class represent the recipient of the behaviours of agent templates. Tasks are connected with object templates by means of the object-property *hasTaskObjectTemplate*.
- *TaskInputParameterTemplate*. This class represents the input parameters required to accomplish the referred operation, as for instance the type of asset on which a quality

valuation is performed. Tasks are possibly connected with the input parameters by means of the object-property *hasTaskInputParameterTemplate*.
- *TaskOutputParameterTemplate*. This class represents the output parameters obtained as a result of the referred operation. Tasks are possibly connected with the output parameters by means of the object-property *hasTaskOutputParameterTemplate*.

Figure 1 illustrates an example of agent template describing the procedure for smart contracts minting ERC721 compliant non-fungible tokens and stored on the Ethereum blockchain (see [37] for the description of the complete behaviour template).

The agent template described in the example comprises a single behaviour, constituted by a single goal that in its turn comprises a single task. The task, which represents the ability of the agent to generate an NFT, provides four elements:

- *mint_ERC721_token_task_operator*, representing the mental state of the behaviour's action (the task operator), which is associated to the OASIS-ABox individual *mint*, the latter describing the capability of generating a coin;
- *mint_ERC721_token _task_operator_argument*, introducing an additional feature (the operator argument) associated with the action and represented by the OASIS-Abox term *blockchain_digital_token*. Such feature describes the fact that the minting action is referred to a digital coin on the blockchain. Task operator and its argument describe together the capability of minting token on the blockchain;
- *mint_ERC721_token_task_object_template*, representing the recipient of the operation, which is related with an instance of the class *EthereumTokenERC721* by means of the object-property *refersAsNewTo*. Such an instance comprises all the features that the recipient of the minting action should own, that is being a ERC721 token. In fact, the concrete actions implementing the behaviour template for minting tokens are supposed to generate ERC721 compliant tokens that therefore can provide additional properties, as for example the description of specific physical assets;
- *mint_ERC721_token_task_output_template*, representing the output of the operation, namely the same token recipient of the *mint* action.

Concrete behaviours of agents have a structure analogous to the one provided by behaviour templates. The structure reflects the modelling pattern adopted for the representation of agent behaviour templates described above, but with the following differences:

- The instance of the class *AgentBehaviorTemplate* gives way to an instance of the class *Agent*, representing a concrete agent in the knowledge domain. Concrete agents are connected with templates of agent by means of the object-property *adoptsAgentBehaviorTemplate*.
- The instance of the class *TaskObjectTemplate* gives way to an instance of the class *TaskObject*, representing the real recipient of the concrete agent action.
- The instance of the class *TaskInputParameterTemplate* gives way to an instance of the class *TaskFormalInputParameter*, representing the formal input parameter of the concrete agent action.

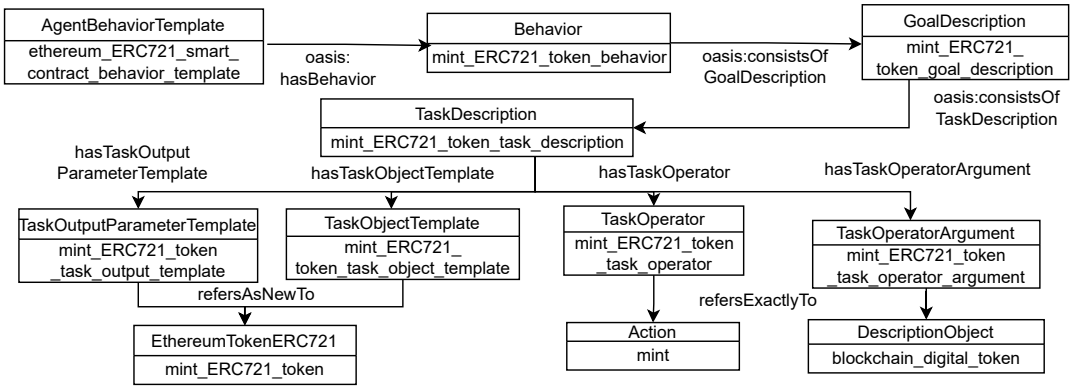

Figure 1: Example of OASIS agent template

- The instance of the class *TaskOutputParameterTemplate* gives way to an instance of the class *TaskFormalOutputParameter*, representing the formal output parameter of the concrete agent action.

With respect to the first release, it is worth noting that in order to align the representation of concrete agents with the one of agent templates, the object, the operator, and the input/output parameters of concrete agents introduce an additional layer between the agent's mental state and the effective entities associated with them. The intermediate layer is marked by the object-properties *refersExactlyTo* and *refersAsNewTo*. This choice is motivated by the fact that such entities may be involved in different states and with different purposes by many agents. Additionally, concrete agents are possibly connected with the agent template that they are drawn from. In order to describe the fact that concrete agents inherit their behaviours from a common and shared agent template, the following associations are introduced:

- The instance of the class *TaskDescription* of the concrete agent is connected by means of the object-property *overloadsTaskDescription* with the instance of the class *TaskDescription* of the agent template.
- The instance of the class *TaskObject* of the concrete agent is connected by means of the object-property *overloadsTaskObject* with the instance of the class *TaskObjectTemplate* of the agent template.
- The instance of the class *TaskOperator* of the concrete agent is connected by means of the object-property *overloadsTaskOperator* with the instance of the class *TaskOperator* of the agent template.
- The instance of the class *TaskFormalInputParameter* of the concrete agent is connected by means of the object-property *overloadsTaskInputParameter* with the instance of the class *TaskInputParameterTemplate* of the agent template.
- The instance of the class *TaskFormalOutputParameter* of the concrete agent is connected by means of the object-property *overloadsTaskOutputParameter* with the instance of the class *TaskOutputParameterTemplate* of the agent template.

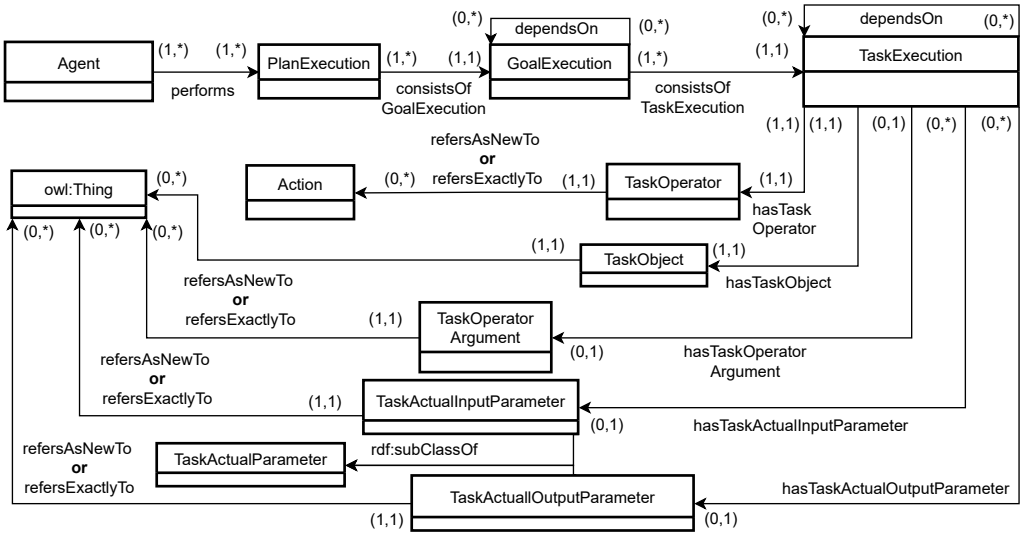

Figure 2: UML diagram of OASIS agent actions

Analogous object-properties are defined for the behaviour and goal of the agent.

Agent actions that are executed autonomously are related with the description of the concrete behaviour of the agent from which they are drawn. As depicted in Figure 2, an agent action is primarily introduced by an instance of the class *PlanExecution*, representing the agent commitment. In their turn, plan executions comprise goal executions, represented by instances of the class *GoalExecution*, while goal executions comprise task executions (instances of the class *TaskExecution*) embedding at least the following three elements:

- *TaskObject.* As in the case of agent behaviours, this class comprises elements used as recipients of performed actions.
- *TaskOperator.* As in the case of agent behaviours, this class comprises the operations performed by the agent.
- *TaskOperatorArgument.* As in the case of agent behaviours, this class comprises the specification of the operations performed by the agent. Operator arguments are introduced in agent actions only if the corresponding behaviour that generates the actions also provides operation arguments.

Unlike the tasks of agent behaviours, task executions comprise instances of the following classes, which take the place of the instances of the classes *TaskFormalInputParameter* and *TaskFormalOutputParameter*, respectively:

- *TaskActualInputParameter.* This class represents the actual input parameters exploited to accomplish the agent action. Task executions are possibly connected with the actual input parameters by means of the object-property *hasTaskActualInputParameter*.
- *TaskActualOutputParameter.* This class represents the actual output parameters obtained as result of the agent's action. Task executions are possibly connected with the output parameters by means of the object-property *hasTaskActualOutputParameter*.

Agent commitments are related with the behaviour from which they are drawn by. As a consequence, the additional layer present in the agent's behaviour between the agent mental state and the entity involved has been also introduced for commitments. These relationships are realized by:

- connecting the instance of *PlanExecution* of the agent action to the instance of the class *Behavior* of the agent behaviour through the object-property *planExecutionDrawnBy*;
- connecting the instance of *GoalExecution* of the agent action to the instance of the class *GoalDescription* of the agent behaviour through the object-property *goalExecutionDrawnBy*;
- connecting the instance of *TaskExecution* of the agent action to the instance of the class *TaskDescription* of the agent behaviour through the object-property *taskExecutionDrawnBy*;
- connecting the instance of *TaskObject* of the agent action to the instance of the class *TaskObject* of the agent behaviour through the object-property *taskObjectDrawnBy*;
- connecting each instance of *TaskActualInputParameter* of the agent action to the related instance of the class *TaskDescription* of the agent behaviour through the object-property *taskActualInputParameterDrawnBy*;
- connecting each instance of *TaskActualOutputParameter* of the agent action to the related instance of the class *TaskDescription* of the agent behaviour through the object-property *taskActualOutputParameterDrawnBy*.

Agents that desire to engage agent peers with performing actions can submit *plans*, namely descriptions of actions that they wish to be realized. Plans, already available in the first release of OASIS, are now fully fledged by way of instances of *PlanDescription*. Those instances provide a structure analogous to that of behaviours, while agents proposing plans are connected with instances of *PlanDescription* by means of the object-property *requestsPlan*. 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 with the entities describing the responsible behaviour by exploiting the following relationships:

- The instance of *PlanDescription* of the plan is connected with the instance of *Behavior* of the responsible behaviour through the object-property *planDescriptionSubmittedTo*.
- The instance of *GoalDescription* of the plan is connected with the instance of *GoalDescription* of the responsible behaviour through the object-property *goalDescriptionSubmittedTo*.
- The instance of *TaskDescription* of the plan is connected with the instance of *TaskDescription* of the responsible behaviour through the object-property *taskDescriptionSubmittedTo*.
- The instance of *TaskObject* of the plan is connected with the instance of *TaskObject* of the responsible behaviour through the object-property *taskObjectSubmittedTo*.
- The instance of *TaskOperator* of the plan is connected with the instance of *TaskOperator* of the responsible behaviour through the object-property *taskOperatorSubmittedTo*.
- The instance of *TaskFormalInputParameter* of the plan is connected with the instance of *TaskFormalInputParameter* of the responsible behaviour by means of the object-property *taskFormalInputParameterSubmittedTo*.

- The instance of *TaskFormalOutputParameter* of the plan is connected with the instance of *TaskFormalOutputParameter* of the responsible behaviour by means of the object-property *taskFormalOutputParameterSubmittedTo*.

When the agent selected for executing the plan attempts to perform the corresponding actions, a graph representing the performance is constructed and connected both with the behaviour, as described above, and with the plan. This ensures that the agent expectation and the plan realization are mapped and their alignment can be easily verified. Plan and its execution are related by:

- connecting the instance of the *PlanDescription* of the plan with the instance of the *PlanExecution* of the action by means of the object-property *hasPlanExecution*;
- connecting the instance of the *GoalDescription* of the plan with the instance of the *GoalExecution* of the action by means of the object-property *hasGoalExecution*;
- connecting the instance of the *TaskDescription* of the plan with the instance of the *TaskExecution* of the action by means of the object-property *hasTaskExecution*;
- connecting the instance of the *TaskObject* of the plan with the instance of the *TaskObject* of the action by means of the object-property *hasTaskObjectExecution*;
- connecting the instance of the *TaskFormalInputParameter* of the plan with the instance of the *TaskActualInputParameter* of the action by means of the object-property *hasTaskInputParameterExecution*;
- connecting the instance of the *TaskFormalOutputParameter* of the plan with the instance of the *TaskActualOutputParameter* of the action by means of the object-property *hasTaskOutputParameterExecution*.

Another aspect related with the multi-agent realm and addressed by the current version of OASIS concerns the *entrustment* of agents, namely the capability of some *entruster agents* to engage a *performer agent* to execute a given plan submitted by a *requester agent*. The association among the behaviours of entruster, requester, and performer are depicted in Figure 3. Entrustment is introduced by means of a graph that follows a structure analogous to the one of plan descriptions. Specifically, it is characterized by instances of the class *PlanEnstrustment*, which in their turn are connected with one or more instances of the class *GoalEntrustment* through the object-property *consistsOfGoalEntrustment*. Goals are linked with instances of the class *TaskEntrustment* by means of the object-property *consistsOfTaskEntrustment*. Moreover, in order to associate a plan to the behaviour of the agent selected to perform it, the elements of the entrustment are connected to a) the related elements of the plan by means of the subproperties of *entrustedBy* and to b) the related elements of the behaviour of the performer agent by means of the subproperty of *entrustedFrom*. A single plan entrustment may be constituted by many goal entrustments that in their turn may be constituted by one or more task entrustments, depending on how the requested plan is structured. In case that the requested plan is constituted by two or more tasks, each task can be entrusted to different agent behaviours thus ensuring the representation of cooperative systems, where the effort of solving plans is distributed among participants. Finally, when the performer agent executes the entrusted plan, a plan execution is introduced as described above. As a consequence, the plan entrustment is associated with the

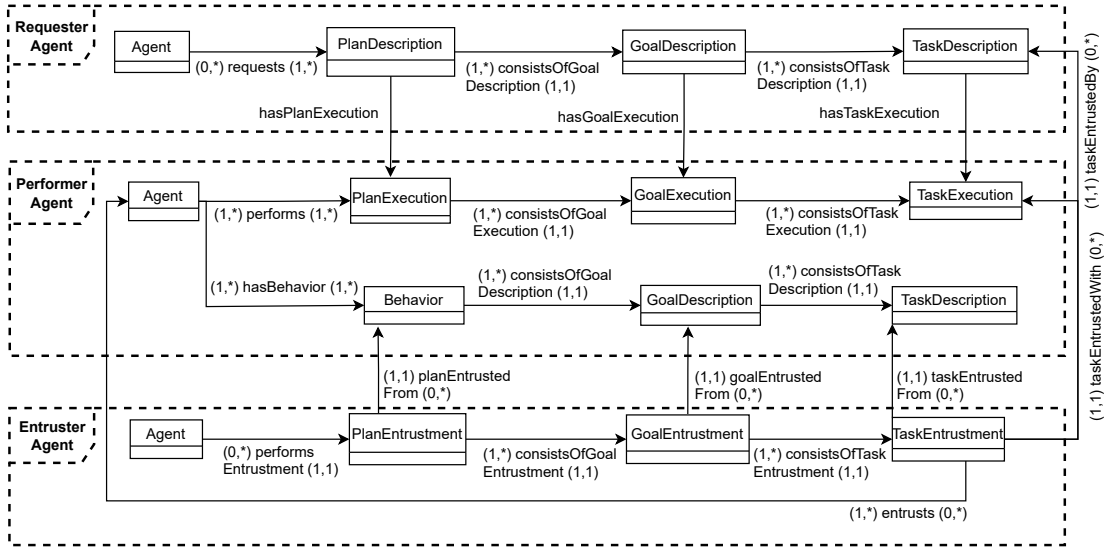

Figure 3: UML diagram of agent entrustments in OASIS

plan execution by connecting the elements of the plan entrustment to the related elements of the plan execution through the subproperties of *entrustedWith*. An analogous structure is provided for the descriptors of *TaskEntrustment*, namely, *TaskObjectEntrustment*, *TaskOperatorEntrustment* and *TasktParameterEntrustment*. Additionally, the task of the plan entrustment is associated with the performer agent by means of the object-property *entrusts*.

4. Conclusions

This paper presented the latest epistemological choices made on the *Ontology for Agents, Systems and Integration of Services* (in short, OASIS) over the years since its first publication, thereby presenting its advancements. OASIS is a foundational OWL 2 ontology that takes the behaviouristic approach derived from the *Theory of Agents* and inherits the related mentalistic notions to represent multi-agent systems. The focus in this paper was on how OASIS represents agent templates, concrete agents, agent commitments and entrustments. In particular, it was outlined how these evolved since the first release of the ontology and as a consequence of its adoption in the POC4COMMERCE project, funded by the NGI-ONTOCHAIN consortium to deliver the ontological foundation of a blockchain-oriented e-commerce. Other applications of OASIS deals with the representation of blockchain smart-contracts and related token exchange mechanisms.

Representing how agents reach consensus is one of the future challenges, together with the modelling of their behaviours implementations. We shall consider how to model in OASIS *meta-plans*, namely agent strategies designed to be achieved through plans. Moreover, we intend to apply agent conditionals to represent security constraints for cybersecurity threat contexts, in particular for the purpose of semantically representing authentication and confidentiality properties for agents. All these features will be included in the next release of OASIS, namely

OASIS 2, that will also provide many enhancements to the current model. We also consider how to integrate OASIS with the main frameworks such as JADE [44] to generate code for agents and artifacts, and how it can exploited by *CArtAgO* [45], a framework for building shared computational worlds.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (2001) 34–43. URL: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[2] P. Hitzler, M. Krötzsch, B. Parsia, P. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer, W3C Recommendation, World Wide Web Consortium, 2009. URL: http://www.w3.org/TR/owl2-primer/.

[3] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017.

[4] D. Fensel, F. M. Facca, E. P. B. Simperl, I. Toma, Semantic Web Services, Springer, 2011.

[5] N. Jennings, M. Wooldridge (Eds.), Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, Berlin, Heidelberg, 1998.

[6] Y. Shoham, Agent-oriented programming, Artificial Intelligence 60 (1993) 51–92.

[7] 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.

[8] 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.

[9] J. Hendler, Agents and the semantic web, IEEE Intelligent Systems 16 (2001) 30–37.

[10] World Wide Web Consortium, OWL-S: Semantic Markup for Web Services, 2004. URL: https://www.w3.org/Submission/OWL-S/.

[11] D. Martin, M. Burstein, S. McIlraith, M. Paolucci, K. Sycara, Owl-s and agent-based systems, in: L. Cavedon, Z. Maamar, D. Martin, B. Benatallah (Eds.), Extending Web Services Technologies: The Use of Multi-Agent Approaches, Springer US, Boston, MA, 2004, pp. 53–77.

[12] M. Hadzic, E. Chang, P. Wongthongtham, Ontology-Based Multi-Agent Systems, Springer Publishing Company, Incorporated, 2014.

[13] D. Fritzsche, M. Grüninger, K. Baclawski, M. Bennett, G. Berg-Cross, T. Schneider, R. Sriram, M. Underwood, A. Westerinen, Ontology Summit 2016 Communique: Ontologies within semantic interoperability ecosystems, Applied Ontology 12 (2017) 91–111.

[14] Q. Tran, G. Low, MOBMAS: A methodology for ontology-based multi-agent systems development, in: Inf. Softw. Technol., 50(7-8), 2008, pp. 697–722.

[15] K. H. Dam, M. Winikoff, Comparing Agent-Oriented Methodologies, in: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), Agent-Oriented Information Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 78–93.

[16] M. Cossentino, M. Gleizes, A. Molesini, A. Omicini, Processes Engineering and AOSE, in: M.-P. Gleizes, J. J. Gomez-Sanz (Eds.), Agent-Oriented Software Engineering X, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 191–212.
[17] A. Rao, M. Georgeff, BDI agents: from theory to practice, in: Proc. of the First International Conference on Multi-Agent Systems, ICMAS-95, San Francisco, CA, 1995, pp. 312–319.
[18] L. Fichera, F. Messina, G. Pappalardo, C. Santoro, A Python Framework for Programming Autonomous Robots Using a Declarative Approach, Science of Computer Programming 139 (2017) 36–55.
[19] M. Bratman, Intentions, Plans and Practical Reason, Harvard University Press, 1987.
[20] A. Freitas, R. H. Bordini, R. Vieira, Model-driven engineering of multi-agent systems based on ontologies, Applied Ontology 12 (2017) 157–188.
[21] F. García-Sánchez, J. T. Fernández-Breis, R. Valencia-García, J. M. Gómez, R. Martínez-Béjar, Combining semantic web technologies with multi-agent systems for integrated access to biological resources, Journal of Biomedical Informatics 41 (2008) 848 – 859. Semantic Mashup of Biomedical Data.
[22] A. Freitas, A. Panisson, L. Hilgert, F. Meneguzzi, R. Vieira, R. Bordini, Applying ontologies to the development and execution of multi-agent systems, Web Intelligence 15 (2017) 291–302.
[23] A. Ciortea, S. Mayer, F. Michahelles, Repurposing manufacturing lines on the fly with multi-agent systems for the web of things, in: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2018, pp. 813–822.
[24] G. Bajaj, R. Agarwal, P. Singh, N. Georgantas, V. Issarny, A study of existing Ontologies in the IoT-domain, hal-01556256 (2017).
[25] W. Wang, S. De, R. Toenjes, E. Reetz, K. Moessner, A Comprehensive Ontology for Knowledge Representation in the Internet of Things, in: 11th International Conference on Trust, Security and Privacy in Computing and Communications, IEEE, 2012, pp. 1793–1798.
[26] S. N. Akshay Uttama Nambi, C. Sarkar, R. V. Prasad, A. Rahim, A Unified Semantic Knowledge Base for IoT, in: World Forum on Internet of Things (WF-IoT), IEEE, 2014, pp. 575–580.
[27] C. Savaglio, M. Ganzha, M. Paprzycki, C. Bădică, M. Ivanović, G. Fortino, Agent-based internet of things: State-of-the-art and research challenges, Future Generation Computer Systems 102 (2020) 1038–1053.
[28] N. Seydoux, K. Drira, N. Hernandez, T. Monteil, Capturing the Contributions of the Semantic web to the IoT: a Unifying Vision, in: Semantic Web technologies for the Internet of Things, ISWC, 2017, pp. 1–8.
[29] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor, IoT-Lite: A Lightweight Semantic Model for the Internet of Things, in: IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, IEEE, 2016, pp. 475–487.
[30] World Wide Web Consortium, Web of Things (WoT) Thing Description, 2020. URL: https://www.w3.org/TR/wot-thing-description/.

[31] 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.
[32] 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.
[33] C. Corea, P. Delfmann, Detecting compliance with business rules in ontology-based process modeling, Wirtschaftsinformatik und Angewandte Informatik (2017).
[34] 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.
[35] D. Cantone, C. Longo, M. Nicolosi Asmundo, D. F. Santamaria, C. Santoro, CLARA, 2019. URL: https://github.com/dfsantamaria/CLARA.
[36] 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.
[37] 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.
[38] NGI-ONTOCHAIN, Ontochain a new software ecosystem for trusted, traceable & transparent ontological knowledge, 2020. URL: https://ontochain.ngi.eu/.
[39] 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.
[40] G. Bella, D. Cantone, M. Nicolosi-Asmundo, D. F. Santamaria, A Behaviouristic Semantic Approach to Blockchain-based E-Commerce, in: unpublished manuscript available from the authors, 2022, pp. 1–46, submitted.
[41] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo, D. F. Santamaria, POC4COMMERCE - Practical ONTOCHAIN for E-Commerce - Project Page, 2021. URL: https://github.com/dfsantamaria/POC4COMMERCE.
[42] G. Bella, D. Cantone, M. Nicolosi-Asmundo, D. F. Santamaria, Taming the Ethereum blockchain: an ontological behaviouristic perspective, in: unpublished manuscript available from the authors, 2022, pp. 1–25, submitted.
[43] D. Cantone, C. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, C. Santoro, OASIS-Abox ontology, 2019. URL: http://www.dmi.unict.it/oasis-abox.owl.
[44] 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).
[45] 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.