Semantic Web 15 (2024) 1863–1914
DOI 10.3233/SW-243543
IOS Press

A behaviouristic semantic approach to blockchain-based e-commerce

Giampaolo Bella, Domenico Cantone, Gianpietro Castiglione, Marianna Nicolosi Asmundo and Daniele Francesco Santamaria *

University of Catania, Viale Andrea Doria, 6 – 95125 – Catania, Italy
E-mails: giampaolo.bella@unict.it, domenico.cantone@unict.it, gianpietro.castiglione@phd.unict.it, marianna.nicolosiasmundo@unict.it, daniele.santamaria@unict.it

Editor: Sabrina Kirrane, Vienna University of Economics and Business (WU Wien), Austria
Solicited reviews: Roberto García, Universitat de Lleida, Spain; Luis-Daniel Ibáñez, University of Southampton, United Kingdom; Juan Cano-Benito, Universidad Politécnica de Madrid, Spain; Claudio Di Ciccio, Utrecht University, The Netherlands; One anonymous reviewer

Abstract. Electronic commerce and finance are progressively supporting and including decentralized, shared and public ledgers such as the blockchain. This is reshaping traditional commercial activities by advancing them towards *Decentralized Finance* (DeFi) and Commerce 3.0, thereby supporting the latter's potential to outpace the hurdles of central authority controllers and lawgivers. The quantity and entropy of the information that must be sought and managed to become active participants in such a relentlessly evolving scenario are increasing at a steady pace. For example, that information comprises asset or service description, general rules of the game, and specific technologies involved for decentralization. Moreover, the relevant information ought to be shared among innumerable and heterogeneous stakeholders, such as producers, buyers, digital identity providers, valuation services, and shipment services, to just name a few.

A clear semantic representation of such a complex and multifaceted blockchain-based e-Commerce ecosystem would contribute dramatically to make it more usable, namely more automatically accessible to virtually anyone wanting to play the role of a stakeholder, thereby reducing programmers' effort. However, we feel that reaching that goal still requires substantial effort in the tailoring of Semantic Web technologies, hence this article sets out on such a route and advances a stack of OWL 2 ontologies for the semantic description of decentralized e-commerce. The stack includes a number of relevant features, ranging from the applicable stakeholders through the supply chain of the offerings for an asset, up to the *Ethereum* blockchain, its tokens and smart contracts. Ontologies are defined by taking a behaviouristic approach to represent the various participants as agents in terms of their actions, inspired by the *Theory of Agents* and the related mentalistic notions. The stack is validated through appropriate metrics and SPARQL queries implementing suitable competency questions, then demonstrated through the representation of a real world use case, namely, the iExec marketplace.

Keywords: Ontology, OWL, Semantic Web, DeFi, agent, blockchain, Ethereum, e-commerce, supply chain, ONTOCHAIN, iExec

1. Introduction

Electronic commerce (*e-commerce*) refers to the various activities revolving around the electronically buying or selling of products through online services [61]. The term was first used in the California's Electronic Commerce

*Corresponding author. E-mail: daniele.santamaria@unict.it.

Act, enacted in 1984. From the 1990s, e-commerce experienced a huge, continuous evolution. More recently, commerce – as well as finance – embraced the *blockchain* as a platform to deploy and exchange digital decentrally managed certificates, called tokens, each associating the owner of a product or service with some predetermined rights. The blockchain is a peer-to-peer public ledger maintained by a distributed network of computational nodes without the need for trusted entities [34]. Blockchains provide several benefits such as ownership, transparency, traceability, public availability, continuity, and immutability of digital assets, all in an efficient and trustless environment where censorship is not achievable. Some blockchains such as Ethereum [96] are equipped with an execution environment allowing Turing-complete programs,[1] commonly called *smart contracts*. Smart contracts are self-executing and immutable programs, autonomously running and verified on a distributed and decentralized public network, which implement a family of blockchain-oriented applications called *Decentralized Applications* (DApps). In such type of blockchains, cryptocurrencies are used not only as payment system but mainly as fuel to consolidate transactions, hence permitting the publishing and the functioning of smart contracts for their operations, including minting, burning, modifying, and changing ownership of tokens.

DApps have grown particularly as an exchange tool for *nonfungible tokens* (NFTs), which are mainly used to provide ownership rights on unique assets, such as physical or digital products and services, in general for those whose uniqueness is hard to demonstrate (for example, digital images). NFTs are used for commercial purposes, as they provide a decentralized albeit public certificate for the underlying physical or digital good and as an incontrovertible witness that a commercial transaction occurred: at the end of 2020, the market capitalization of NFTs reached the amount of 338 million U.S. dollars [93]. Together with NFTs, blockchain systems are adopted to manage other types of tokens, such as fungible and semi-fungible tokens. While non-fungible tokens are valued on the basis of their scarcity and of their distinctive properties, fungible tokens may be exchanged with another asset of the same kind: for example, fiat currencies such as US Dollars are fungible assets since one dollar can be traded with another dollar regardless of the physical or digital coin owned.

However, some fungible tokens may become unique at a certain moment: they are known as semi-fungible tokens. For example, a guitar that is initially fungible (it is indistinguishable from any other guitar of the same type) becomes non-fungible when owned by famous musicians since it can be considered as musical heirloom.

1.1. Goals

The overarching goal of the present article is to lay out the foundations for a clear and unambiguous semantic representation of the blockchain Ethereum in the area of e-commerce, but generalizable to any Turing-complete blockchain. Therefore, we focus on the Ethereum smart contracts and transactions related to NFT trading and associated with commercial means.

Achieving this goal would have several advantages. A major one would be the simplification and automation of the tasks of querying the blockchain for identifying creation, destruction, or transfer of specific tokens and related features, as well as the smart contract exchanging specific types of tokens and the trading conditions.

It must be clear that such tasks are customarily left to (human) programmers. For example, programmers routinely have to study API documentation in order to interact with a new blockchain-based ecosystem and program new facilities. While APIs may (often) be badly documented, it is unquestionable that any developments are tightly bound to a scaling up of human resources – by contrast, a semantic representation notoriously makes the disrupting contribution of breaking up the tightness of that bound. Therefore, once the technicalities of blockchain-based e-commerce get semantically represented, much access, integration, and, in general, development effort can be offloaded to machines, which become able to interpret potentially unbound volumes of data, information, and, consequently, knowledge. Noteworthy mentioning, traditional blockchain APIs permit limited query capabilities involving transactions logs, blocks, and transactions metadata, functions signatures or, as in the case of the Etherscan search engine, keywords defined by the smart contract developer. However, they lack advanced features such as searching for NFTs, assets, or smart contracts with specific features, due to the hard-coded nature of the blockchain: APIs would be greatly enhanced and improved if DApps provided semantic meaning clearly explaining the operations they can carry out, how they process data, and the type of information they manipulate. This would, in turn,

[1] We refer to such blockchains simply as Turing-complete blockchains.

promote the widespread and widespread use of the new technologies at all levels of society. Some state of the art exists and, as we shall see below, is valid and useful, but the combination of blockchain systems and e-commerce from an epistemological standpoint is still out of reach, hence the goal for the present work.

1.2. Existing tools and chosen approaches

A few existing ontologies can be profitably leveraged. Notably, the GoodRelations [69] project has been active since 2001 and targets the semantic representation of commercial goods and services, while ontologies for blockchains were introduced much more recently, particularly with the widely-known ontologies BLONDiE [105] and Ethon [86]. However, these are pivoted on essential elements of the respective knowledge domain such as offerings and tokens, and lack on the operational use that the relevant stakeholders should make of those elements. Our approach is to frame the core elements of the commerce and blockchain domains within their effective contexts of use, including the roles that each participant may play.

As a consequence, another challenge is to take into account the relevant stakeholders, including applications, people, and assets. Because stakeholders may be active participants, it is convenient to leverage the concept of *autonomous agent* [73]. Agents are defined as entities whose state consists of mental components such as beliefs, capabilities, choices, and commitments [90]. On these premises, the *Agent Oriented Programming* (AOP) paradigm can be profitably adopted to represent open architectures that continuously evolve to adapt agents to external changes in a robust, autonomous, and proactive way. Another useful tool for our purposes is the Semantic Web, which provides a means for proof and trust models, mediation and communication, and representation mechanisms of the environment, thus supporting semantic markup annotations attached to data sources available to agents.

A practical way of semantically describing agents is through a behaviouristic approach [17], which aims at defining the tasks of each agent. Agents are enabled to publicly report the set of activities they are able to perform, the types of data required to execute those activities, as well as the expected output. Agents' implementation details are abstracted away to make the discovery of agents transparent, automatic, and independent of the underlying physical infrastructure. This implies that agents may join a collaborative environment in a plug-and-play fashion, as there is no need for third-party intervention. Additionally, thanks to the Semantic Web technology and reasoning techniques, data manipulated by agents carry machine-understandable information that can be processed, integrated, and exchanged by any type of agent at a higher level.

We contributed to the OASIS ontology [10–12,22], which applies the behaviouristic approach to deliver a general representation system and a communication protocol for agents and their interactions. Therefore, OASIS can be profitably leveraged here, so that commercial participants and their commitments are semantically described as agents by means of the actions they can perform through their commercial activities, including the applicable payment options, provision methods, and supply chains. This will allow people to freely choose products and services according to their needs and taste, as well as to increase awareness on the supply chains up to the delivery of goods.

1.3. Contributions

This article adopts the behaviouristic approach towards the representation of today's blockchain-based e-commerce, in particular leveraging token generation and exchange mechanisms as decentralized public proofs. OASIS is leveraged by three novel ontologies, namely OC-Found, OC-Commerce and OC-Ethereum, to semantically describe the blockchain-oriented e-commmerce and related stakeholders. First, the OC-Found ontology extends OASIS, thus inheriting its representation of agents and their commitments. It introduces digital identities because they enable an agent to claim and manage their online identity within a decentralized and anonymous environment such as the blockchain. Moreover, OC-Found also covers supply chains because these account for how products and services are delivered and consumed by customers, an essential feature of commerce in general. The current release of the ontology accounts for the crucial steps of the supply chain, namely *selling and delivering* [71], but the ontological model is laid out and readily extensible with additional features, such as the other supply chain steps of *planning*, *sourcing*, and *making*. Finally, quality valuation mechanisms provide commercial participants with a means of publicly witnessing their experience.

Then, the OC-Commerce ontology captures the model of commercial offerings provided by GoodRelations and extends it with the representation system of agents and their commitments derived from OC-Found and inherited from OASIS. The stack culminates with the OC-Ethereum ontology, which imports the definitions of the BLONDiE ontology for the constitutional elements of the Ethereum blockchain to include the description of Ethereum tokens and the operational semantics of smart contracts. Additionally, OC-Ethereum describes the practical uses for the Ethereum ERC20 standard for fungible tokens, for the Ethereum ERC721 standard for non-fungible tokens and for the Ethereum ERC1155 standard for semi-fungible tokens.

The contributed ontological stack is publicly released [6] under GNU General Public License, version 3. It is evaluated by means of standard structural and ontological metrics, each time by relating the root of the ontology being defined to the import from relevant ontologies. It is also sanity-checked through a number of competency questions, which are implemented by suitable SPARQL queries. Competency questions constitute questionnaires in natural language, which help to clarify the context and the scope of ontologies, aiming at verifying whether the ontologies are truly being developed towards the project objectives and are reaching the stated representational goals.

Finally, the ontological stack is demonstrated on a real-world case study, the iExec marketplace [72], to further confirm its expressiveness.

Such contributions meet the goals stated in Section 1.1 and summarize the results of the POC4COMMERCE project [9], funded by the NGI ONTOCHAIN European initiative [83]. Following NGI ONTOCHAIN's call for solutions for a secure and transparent blockchain-based knowledge management system, POC4COMMERCE delivers the *ONTOCHAIN ontological stack*, namely the core NGI ONTOCHAIN framework, with a focus on the commercial domain as outlined above, then unfolded in the sequel of this manuscript. A position paper calling for such contributions exists [7].

1.4. Structure

The paper is organized as follows. Section 2 concerns relevant related works. Section 3 outlines the existing ontologies that are leveraged through the present work. Specifically, Section 3.1 introduces the ontology OASIS, designed for representing agents and their commitments; Section 3.2 outlines the ontology GoodRelations for representing commercial offerings and selling conditions; and Section 3.3 focuses on the BLONDiE ontology for representing blockchains constitutional elements, such as block, wallet, and transaction. Then, Section 4 introduces the ontological stack and is organized into three subsections, one for each layer of the stack. Precisely, Section 4.1 presents the ontology OC-Found, which extends OASIS with supply chains, digital identities and quality valuation mechanisms; Section 4.2 introduces OC-Commerce, which imports OC-Found and extends GoodRelations with offerings, auctions, assets, and activities through the behaviouristic approach; and Section 4.3 depicts the OC-Ethereum ontology, which exploits BLONDiE by providing the representation of Ethereum smart contracts and related tokens in compliance with standards ERC20, ERC721, and ERC1155. Subsequently, Section 5 evaluates the ontological stack a) by considering both structural and ontological metrics applied to each ontology and compared with the related imported ontologies, and b) by suitable competency questions defined for each ontology and implemented as SPARQL queries. Section 6 demonstrates the stack on a real-world case study by modelling the iExec marketplace. Finally, Section 7 draws conclusions and directions for future research.

2. Related work

2.1. Services and the Semantic Web

In the context of the semantic representation of applications, the Semantic Web aims to enable 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 [67]. In 2003, the

World Wide Web Consortium (W3C) proposed OWL-S [109], 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 DAML 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 [81]. 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 at 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, or are deployed as, and are extended by services [81], which remain relatively simple. For example, *Google Maps* or *GeoSPARQL* may be conceived as agents for retrieving driving directions and described as actors able to compute the best path (in terms of time or distance) from a geographical amenity to another one that is realized through the related end-points. While *GeoSPARQL* is free to use, *Google Maps* requires an API-KEY that can be obtained free for limited use or on payment. The requirement of owning an API-KEY is a low-level mechanism related to the service, whereas the needed authorization is a high-level constraint of the agent. For these reasons, research communities have investigated several representation systems for agents and agent-based systems.

2.2. Agents and the Semantic Web

Since 2000, numerous studies have explored how agents enter and exit various interaction systems, utilizing both *commitment objects* [53] and *virtual institutions* [49]. Commitment objects are unambiguous, objective and independent of the agent mental states describing the effects that sending a message has on the social relationship between the sender and the receiver of the message (Social Semantics). By contrast, virtual institutions describe systems that regulate the behaviour of agents in a multi-agent system or in a multi-agent society (in particular their interaction) in compliance with the norms in force in that society. Therefore, Social Semantics is not able to represent agents that act autonomously, due to its declarative nature, i.e., it describes what agents do rather than how they do it [91].

The integration of agent systems and Semantic Web technologies has been investigated in the last decade in several contexts [56,66,67] and the advantages of ontology-based applications have been recognized in the analysis and development of MAS [102]. 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 using shared common semantic tools, or c) as decision support for supply chains.

Ontologies for MAS have been modelled by taking approaches similar to those of *Agent-Oriented Software Engineering* (AOSE) [39,43], 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) [87] are proposed in [17] and in [51]. Inspired by Bratman's theory of human practical reasoning [16], 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 a robot's high-level behaviour in a simple, natural way. In [54], an ontology for agent-oriented software engineering is proposed along with a tool that generates programming code for MAS using the ontology. However, it does not examine agents and their interactions in detail, and it lacks an appropriate modelling of agent communication.

2.3. Domain-legacy approaches for Semantic Web agents

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. For example, in [58], the

authors propose an ontology-based framework to seamlessly integrate agents and Semantic Web Services, focusing on biomedical information. In [55], an infrastructure to allow agent-oriented platforms to access and query domain-specific OWL ontologies is presented. In [35], the authors introduce an approach 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 to generate perceptions and abstractions of the observed world [4].

A comprehensive ontology for representing IoT services is presented in [107], 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 [1], capturing the complete dynamics of IoT entities, for example enabling semantic searching while hiding their heterogeneity.

In [89], the authors discuss about 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 [15], the authors propose a lightweight and scalable model, called *IoT-lite*, to describe key IoT concepts that allow interoperability and the discovery of sensory data on heterogeneous 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 [113]. 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 schema provided does not fully allow agents to interact according to the specific roles they aim to play.

In [22], the authors propose a first version of OASIS, an ontology to represent agents and their commitments using the behaviouristic approach adopted in this work, together with a home automation assistant based on it, called PROF-ONTO. The approach used in [22] also represents a first foundational contribution to the use of Semantic Web technologies to define a transparent communication protocol among agents. It is based on the exchange of 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 behavior of the agent selected to satisfy it. Recently, the application of OASIS in the context of cybersecurity [13,14,32], video games [45], and chatbots [78–80] has been devised.

2.4. Business processes and the Semantic Web

It is worth noting that there are well-known applications of Semantic Web ontologies in the literature devoted to business processes. For example, in [100] an ontological approach is presented to represent business processes together with a prototype of a system architecture based on the proposed model. In [62], ontologies are used as facilities within a framework that assists designers in the realization and analysis of complex processes. In [38], they present an approach that combines logic programming and ontologies to verify the compliance of a business process with given business rules. Additionally, in [18], they propose an approach using Ontology-Based Data Access (OBDA) to manage data-aware processes. These processes are addressed, particularly those executed over a relational database that make calls to external services to acquire new information and update data. However, the downside of limiting ontological models to business processes lies in the absence of relationships between agents and their commitments, which instead represents the core of agent-oriented representations. That implies the inability to find agents/services with specific capabilities, invoke them, and enable 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.

2.5. E-commerce and the Semantic Web

There are relevant works on the use of Semantic Web technologies on e-commerce. In [74], the authors identify six challenges facing the e-commerce ecosystem: inability of existing product data for automated processing;

diffuse lack of interoperability of product data; insufficient use of unique product identifiers; heterogeneity of product category taxonomies; incomplete, inconsistent or outdated product descriptions; weakness of current product recommendation systems. Then, they propose a technological stack, based on the Semantic Web, to support the creation of *intelligent* e-commerce applications. In [37], a mediation framework for e-commerce based applications leveraging a Semantic Web ontology for electronic services is presented, while the authors of [33] propose a scalable approach that automatically selects and extracts from semi-structured texts in business contexts relevant information published as RDF graphs. The authors of [63] present a comprehensive survey on ontologies for supply chains, pointing out the lack of operational semantics. In [59], this weakness is overcome by the proposed ontology, which aims at describing the supply chain processes where IoT devices are involved. However, such ontology does not account for more general features such as blockchain networks and people.

Creating shared or, at least, interoperable vocabularies for product descriptions has been recognized as a crucial task for e-commerce [94]. In [106], it is suggested that semantic vocabularies enable the implementation of *semantic search engines* [65] to find out items in a very specific range. In the last decades, several industrial *Products and Services Categorization Standards* (in short PSCS) have been proposed and adopted.[2] Among others, we recall the *United Nations Standard Products and Services Code* (UNSPSC) [99], a taxonomy of products and services specific for e-commerce; *eCl@ss* [98] was designed to create a standard classification of material and services for information exchange between suppliers and their customers; the *ECCMA Open Technical Dictionary* (eODT) [47] contains terms, definitions, and images linked to concept identifiers used to create unambiguous descriptions of individuals, organizations, locations, goods, services, processes, rules, and regulations; the *RosettaNet Technical Dictionary* (RNTD) was the reference model for products in the supply chains that use RosettaNet for their interactions, but is now off-line. Most of them converged into the *GS1* initiative [64], an organization that develops and maintains global standards for business communication such as, for example, barcodes and Electronic Product Codes [84]. Notice that the Electronic Product Codes standard provides a *Pure Identity URI* to denote a specific physical object, which can be easily integrated in Semantic Web tools. Then, *eClassOWL* [68] is the OWL counterpart of eCl@ass. In general, these classification systems consist of taxonomies for grouping similar products and, additionally, the most sophisticated ones include a dictionary of properties that can be used to describe product instances.

The proliferation of PSCS supports the observation that e-commerce stakeholders have not reached a consensus on product and service description representation systems, and this forms an obstacle for the interoperability of applications following different standards. Remarkably, as proposed in [36], Semantic Web technologies may help to overcome this issue by enabling *Ontological Mapping* between different systems. Worth of mentioning, the *schema.org* [85,112], a general purpose vocabulary largely used in tagging web page contents, allows for the enrichment of web pages with machine-readable information in the RDFa [110], JSON-LD [111] or HTML Microdata formats, and is commonly adopted by search engines for probing commercial websites.

GoodRelations is an OWL vocabulary to describe offerings of tangible goods and commodity services. Its descriptive features are broad enough to cover both product and service instance descriptions. In addition, a wide range of offerings and pricing can be modelled via GoodRelations. Offerings described using the GoodRelations vocabulary are recognized by major search engines such as Google, Yahoo, and Bing. Also, well-known content management tools, such as Joomla!, osCommerce, and Drupal, support the publication of data with the GoodRelations ontology. GoodRelations is demonstrated in [3] and integrated with *schema.org*. Of course, other trading activities beyond offerings are important and, for example, ownership information may be used for personalized recommendations, as shown in [101]. Notably, GoodRelations does not cover how negotiations are carried out and how offerings or assets are related with tokens managed on the blockchain.

2.6. Blockchain and the Semantic Web

Only recently, researchers have focused on conjoining the blockchain with ontologies [19,48]. One of the areas of investigation has concentrated in developing a characterization of blockchain concepts and technologies through ontologies. In [44], a theoretical contribution looking at the blockchain through an ontological approach has been

[2] A comparative analysis can be found in [70].

provided. In [88], the authors propose a blockchain framework for SWoT contexts settled as a *Service-Oriented Architecture* (SOA), where nodes can exploit smart contracts for registration, discovery, and selection of annotated services and resources. In [77], a preliminary work on capturing the semantics of the *Fast Healthcare Interoperability Resources* (FHIR) standard vocabulary in smart contracts is presented.

Other works aim at representing ontologies within a blockchain context. In [75], ontologies are used as a common data format for blockchain-based applications such as the proposed provenance traceability ontology, but are limited to describe implementation aspects of the blockchain.

The *Blockchain Ontology with Dynamic Extensibility* (BLONDiE) project [104] provides a comprehensive vocabulary that covers the structure of different components (wallets, transactions blocks, accounts, etc.) of blockchain platforms (Bitcoin and Ethereum) and that can be easily extended to other alternative systems. In [5], the authors propose a semantic interpretation of smart contracts as services bases on the ontology *Ethon* [86]. More debate can be found in [52] through a discussion on the blockchain as applied for tracking the provenance of knowledge, for establishing delegation schemes and for ensuring the existence of patterns in formal conceptualizations using zero-knowledge proofs. We contend that the main limitation of these approaches is the poor semantic description of smart contracts, and this precludes the discovery of unknown smart contracts and of the operations that they fulfilled during their life-span. These, however, would be extremely relevant in the area of e-commerce.

It is noteworthy to mention that blockchains are adopted as a means to achieve traceability and transparency of assets [95] also in supply chain management [46]. Moreover, Semantic Web ontologies support pragmatic traceability in supply chains [2]. Therefore, conjoining blockchains and Semantic Web enhances the methodologies and tools for traceability and auditability and would be valuable, for example, to trace both tangible and intangible assets over their supply chains (including fair market values, litigation, and insolvency proceedings and financial reporting). However, the work presented in this article focuses on the representation choices involving the representation of Ethereum smart contracts and tokens and on how they are used as a means for trading the underlying assets through public announcements.

In [21], the OASIS ontology is extended with *ontological smart contracts* (in short, OSCs) and conditionals. OSCs are intended as ontological agreements among agents to allow one to establish responsibilities and authorizations. Conditionals 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. Conditionals and OSCs are applied to add ontological capabilities to digital public ledgers such as the blockchain and the smart contracts implemented on it. The architecture of a framework based on OSCs that exploits the *Ethereum* blockchain and the *Interplanetary File System* is also sketched. In [8], 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. The same approach is part of the work presented in this paper, even though it presents structural differences on how blockchains are modelled. Finally, application of OASIS in the context of the Hyperledger Fabric blockchain has been devised [31].

3. Preliminaries

In this section we briefly illustrate the main features of the ontologies adopted by the ontological stack, namely OASIS [22], GoodRelations [69], and BLONDiE [104]. These ontologies, which better describe the basic elements of different contexts of the blockchain-based commerce, are exploited to construct a behaviouristic vision of such a domain. Specifically, the OASIS ontology is adopted to model agents and their commitments, and is extended with the definition of digital identity and supply chains; GoodRelations provides a simple characterization of the commercial offering, and is extended to include publishing mechanisms associated with supply chains and commercial agents; finally, BLONDiE is adopted to represent the constitutional elements of the Ethereum blockchain such as block, transactions, and wallet, and is extended on the one hand with the operation semantics of the Ethereum smart contracts and on the other hand with a clear specification of smart contract interactions and tokens.

3.1. *The* OASIS *ontology*

We present in detail the main features of the OASIS ontology that are adopted to build the core of the ontological stack related to the semantic representation of the agents and their commitments to the e-commerce based on Ethereum.

The *Ontology for Agents, Systems and Integration of Services* (in short, OASIS) is a foundational OWL 2 ontology that applies the behaviouristic approach to represent multi-agent systems. The behaviouristic approach is an abstraction of an operational semantics based on the definition of an agent behaviour through its decomposition in the essential mental states of the agent, namely, goals, and tasks that are sought to be accomplished. While goals depict the long term desires of the agent, tasks represent the atomic operations that it is able to physically carry out. Alongside the representation of the agent mental states, the behaviouristic approach takes care of how behaviors are invoked and put in practice by agents, thus defining agents' plans and commitments. Whereas plans represent the agent's desire of seeing a behavior materialize, commitments represent such a materialization.

Hence, OASIS characterizes 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. Additionally, OASIS models information concerning executions and assignments of tasks, restrictions on them and constraints used to establish responsibilities and authorizations among agents.

OASIS models agents by representing their behaviours, which are publicly exposed. Thanks to public behaviours, agents report the set and type of operations that they are able to perform, including the type of data required to execute them, possibly with the expected output.

Representation of agents and their interactions in OASIS is carried out along three main steps:

- modelling general abstract behaviours (called *templates*), from which concrete agent behaviours are drawn;
- modelling concrete agent behaviours drawn by agent templates;
- modelling actions performed by agents by associating them with the behaviours, from which actions are drawn.

The first step is not mandatory and consists in defining the agent's behaviour template, namely a high-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 for agents whose behaviour consists in producing and selling products to buyers, and it may be implemented by an *apple* seller that produces and sells batches of green apples. Moreover, templates are useful for helping developers define the most suitable representations for their agents.

The second step consists of representing the agent behaviors either by specifying a shared template or by defining it from scratch. To describe an abstract agent's capabilities to perform actions, an agent template comprises three main elements, namely behaviours, goals, and tasks. Agent tasks, in turn, 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 perform in practice.

Finally, in the third step, actions performed by agents are described as direct consequences of some behaviours and associated with the behaviours of the agent that generates them. To describe such an association, OASIS introduces *plan executions*. Plan executions, representing the agent commitments, describe the actions performed by agents and associated with behaviours. Plan executions are made up of goal executions, representing the execution of agent goals. Goal executions in their turn are constituted by task executions that represent the executions of agent tasks.

The association is carried on by entailing a 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.

We now describe the entities of the OASIS ontology that concern the representation of agents and their commitments that are used by the ontological stack. Figure 1 depicts the classes and properties provided by OASIS that are relevant for the goals of the ontological stack.

In OASIS, agent templates are defined according to the UML diagram in Fig. 2. To describe abstract agent capabilities of performing actions, an agent template comprises three main elements, namely *behaviours*, *goals*, and *tasks*. Agent tasks, in turn, describe atomic operations that agents may perform, including possibly input and

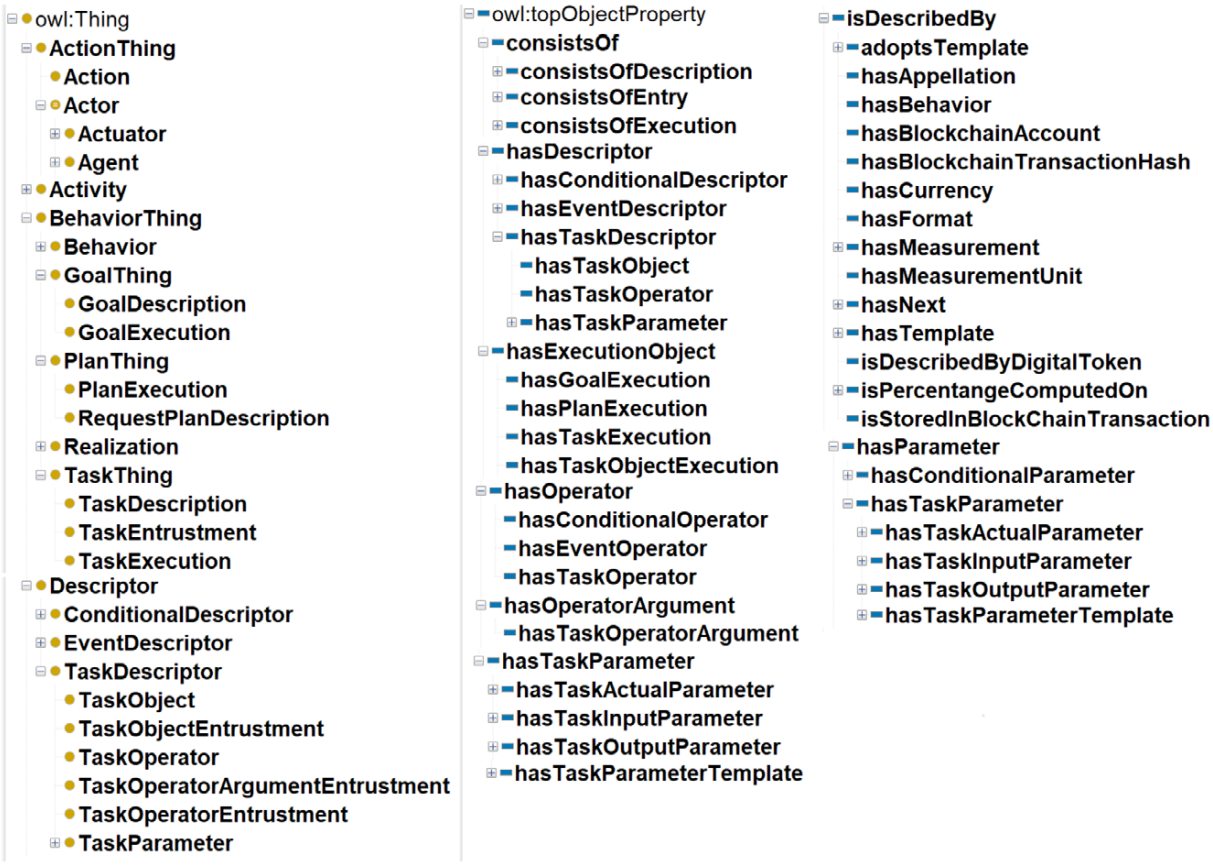

Fig. 1. Main classes and properties of OASIS.

output parameters required to accomplish the actions. The core of agent representation, indeed, revolves around the description of atomic operations introduced by the instances of the class *TaskDescription* that characterizes agent commitments. Instances of the class *TaskDescription* are related to five elements that identify the operation:

- an instance of the class *TaskOperator*, characterizing 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*, ...).[3] 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 an instance of the class *ReferenceTemplate*: instances of the class *ReferenceTemplate* are used to introduce entities that represent templates for the referred element describing the characteristics that it should satisfy. Using object property *refersAsNewTo*, the entity provides only a general description of the features needed to perform the task, for example, that it must be of a specific type; on the contrary, object property *refersExactlyTo* specifies the actual and exact entity involved in the task:
- A possible instance of the class *TaskOperatorArgument*, connected by means of the object-property *hasTaskOperatorArgument* and representing additional specifications for the task operator action (e.g, *on*, *off*, *left*, *right*, ...). Instances of *TaskOperatorArgument* are referred to the operator argument by specifying the object-property *refersAsNewTo* or *refersExactlyTo*.

[3] Instances of *Action* are introduced in the *OASIS-Abox* ontology [23].

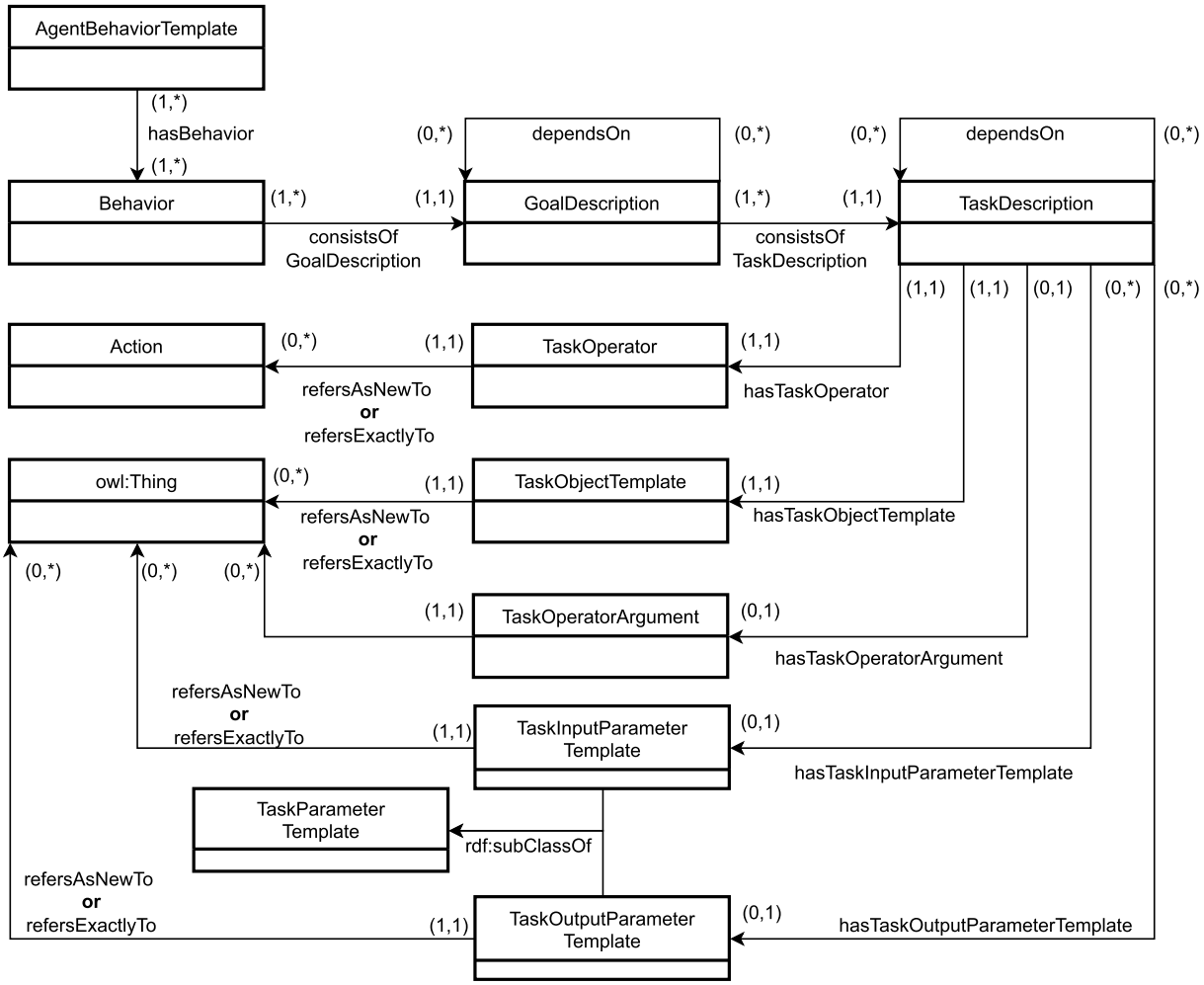

Fig. 2. UML diagram representing OASIS agent templates.

– 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., *apple*, ...). 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 *hasTaskInputParameterTemplate* and *hasTaskOutputParameterTemplate*, respectively. Instances of *TaskInputParameterTemplate* 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*.

Summarizing, the main classes characterizing an agent template are the following ones:

– *AgentBehaviorTemplate*. This class comprises all the individuals representing templates of agents. Instances of such class are connected with one or more instances of the class *Behavior* by means of the OWL object-property *hasBehavior*.

- *Behaviour*. Behaviours of agent templates represent containers embedding 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 composing them and represented by instances of the class *TaskDescription* through the object-property *consistsOfTaskDescription*.
- *TaskDescription*. This class describes the atomic operations that agents can perform. Atomic operations are the most simple actions that agents are able to practically execute and, hence, they represent what agents can do within the considered 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*. The tasks are connected with task operators using 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 extensible vocabulary defined by the OASIS-Abox ontology [23].
- *TaskOperatorArgument*. This class defines operator arguments that represent 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 example the type of asset on which a quality valuation is performed. The tasks are possibly connected to 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*.

Analogously, concrete agents are represented according to the UML diagram in Fig. 3, which reflects the modelling pattern adopted for the representation of agent behaviour templates described above, but introducing the following differences:

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

Concrete agents are possibly connected with the agent template from which they are drawn. 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.

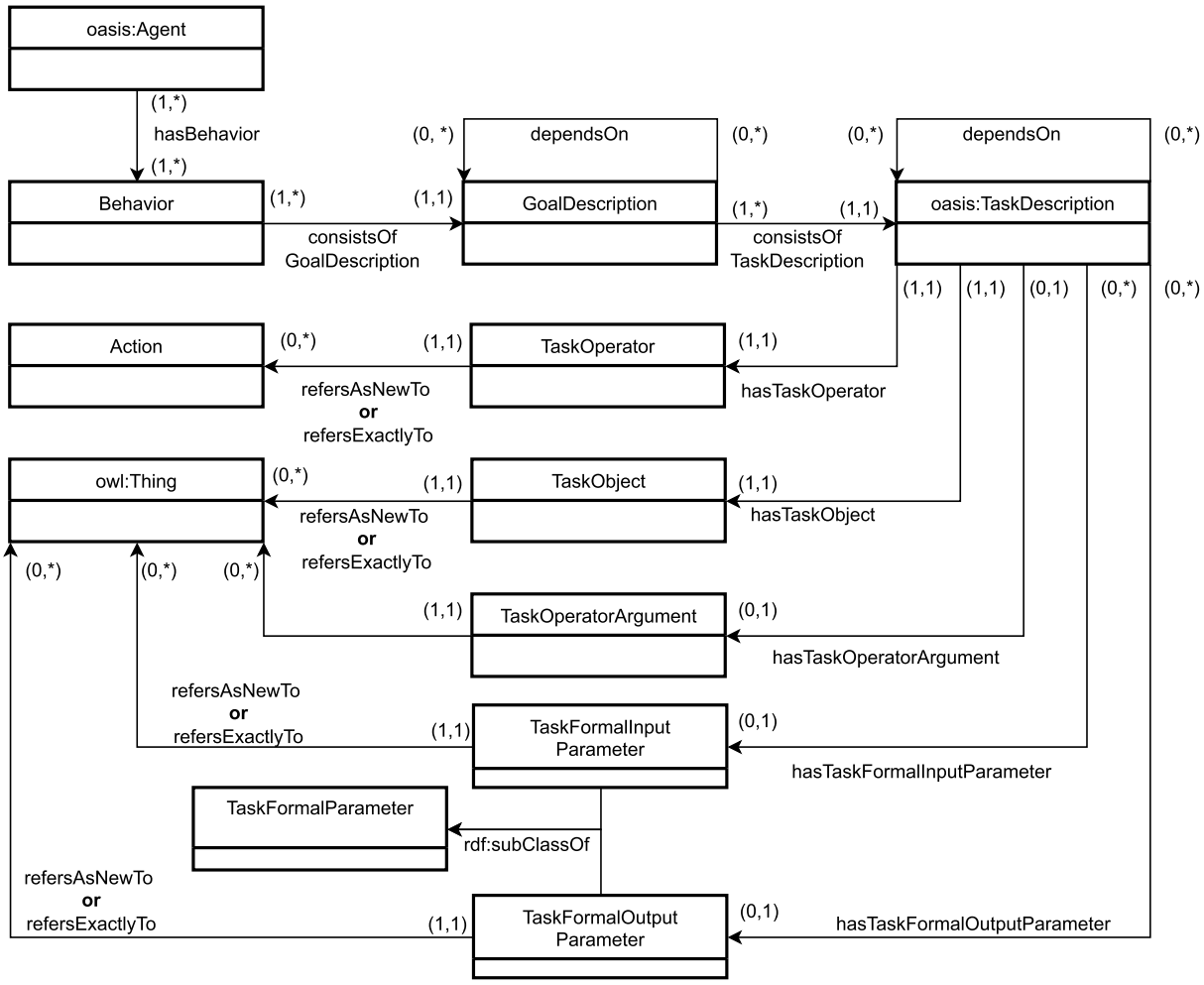

Fig. 3. UML diagram representing OASIS concrete agents.

- 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.

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

Agent actions entail the description of the concrete behaviour of the agent from which they are drawn. As depicted in Fig. 4, an agent action is primarily introduced by an instance of the class *PlanExecution*, representing the agent commitment. Plan executions comprise goal executions, represented by instances of the class *GoalExecution*, whereas, in turn, goal executions provide task executions (instances of the class *TaskExecution*) embedding the following elements:

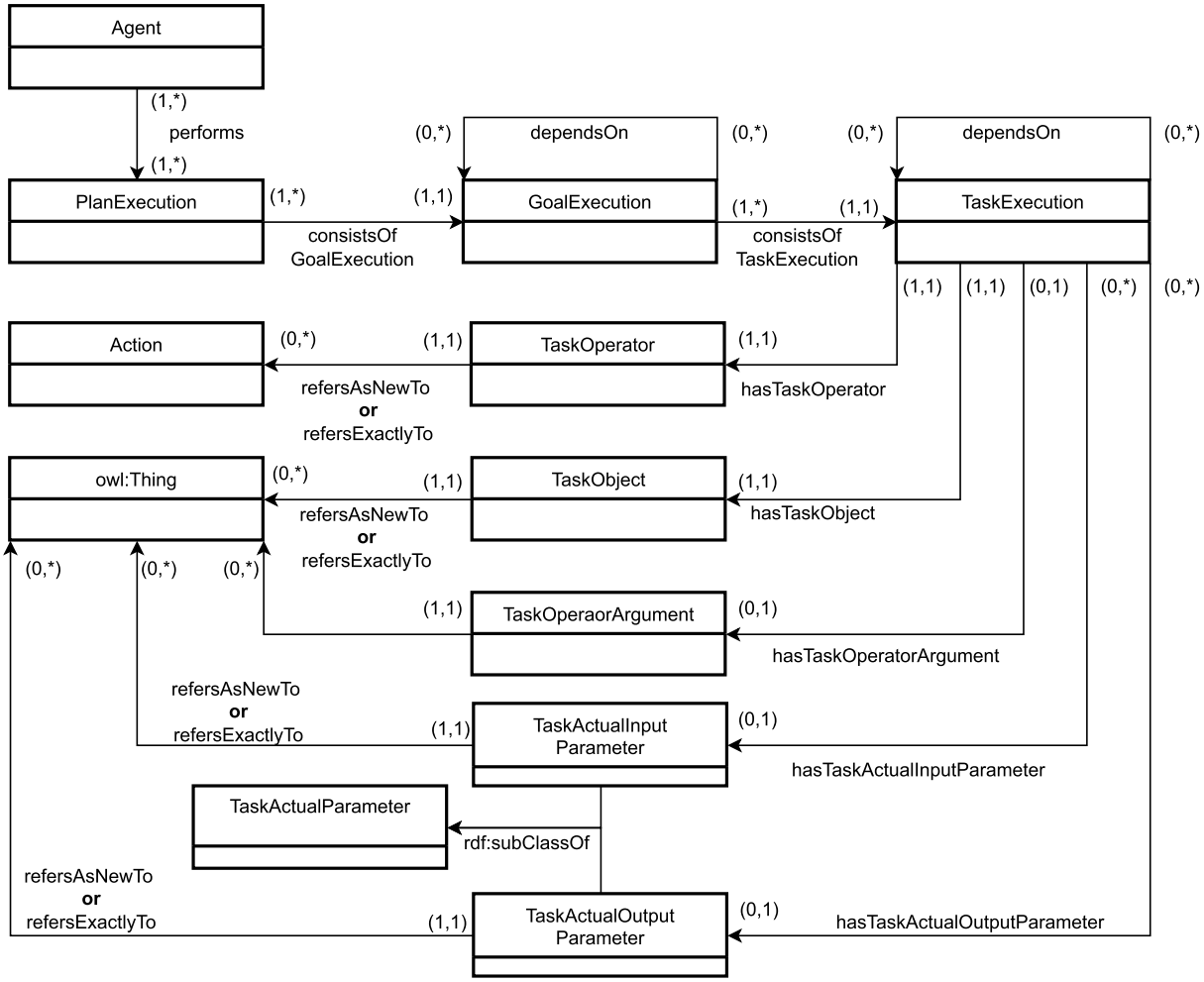

Fig. 4. UML diagram representing OASIS agent actions.

- *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 a 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.

In contrast to 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*:

- *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 an agent's action. Task executions are possibly connected with the output parameters by means of the object-property *hasTaskActualOutputParameter*.

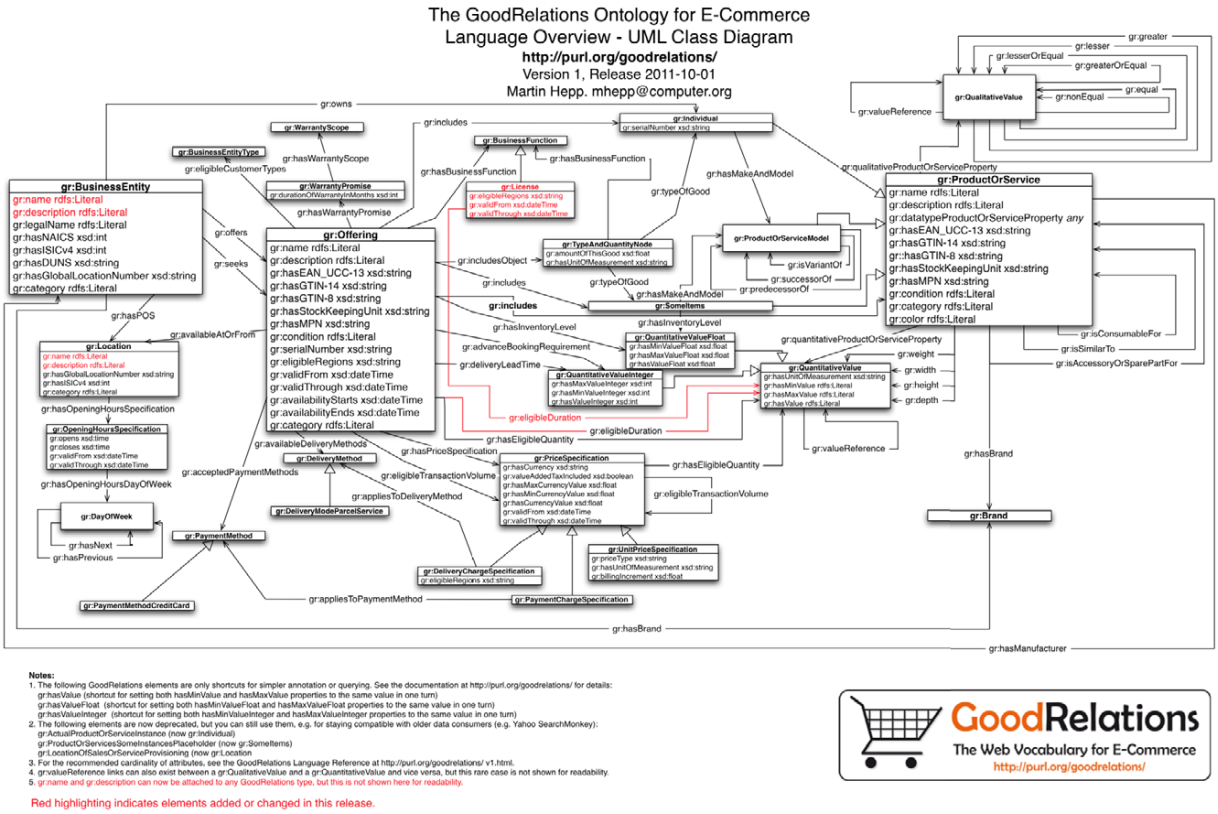

Fig. 5. UML diagram representing the GoodRelations ontology.

3.2. *The GoodRelations ontology*

GoodRelations is an OWL vocabulary describing offerings about products and services, involved legal entities, prices, selling terms, and conditions.

The core class of the GoodRelation vocabulary (see Fig. 5) allows one to represent *offerings*. An offering is an agent's announcement concerning the provision of a certain *business function* (one of "sell", "lease out", "maintain", "repair", "provide service", "dispose", and "buy") for a certain *product or service instance* to a particular target *audience* and under specific commercial conditions.

A *business entity* can create or *seek* for offerings providing goods and terms under particular conditions.

An offering can either refer to

- a clearly specified instance, called *Individual*, or
- a set of anonymous instances of a given type, called *SomeItems*, or
- a product model specification, namely *ProductOrServiceModel*.

An offering may be linked to multiple *price specifications* that specify alternative prices for non-overlapping sets of conditions that can be characterized by:

- the lower and upper limits of the eligible quantity,
- the monetary amount per unit (in combination with a currency), and
- whether this price includes local sales taxes (VAT).

It is possible to specify the availability of an offering and the related accepted *payment methods*, possibly combined with additional *payment charge specifications*, by means of several methods: in advance or after delivery, by credit card, cash or bank transfer.

The *delivery methods* associated with an offering are standardized procedures for providing the product or service, with the destination of fulfillment chosen by the customer. They can possibly be coupled with *delivery charge specifications*. Also, the product or service may be available at some physical *location* (a shop, an office, etc.) characterized by a geographical position and a set of opening hour specifications for various days of the week.

Finally, offerings may be provided with information about *warranty* on goods.

3.3. The BLONDiE ontology

The BLONDiE ontology aims to provide a simple representation of some of the most widespread blockchains, namely Bitcoin, Ethereum, and Hyperledger Fabric, and can potentially cover every blockchain available by means of suitable extensions. The ontology, depicted in the UML diagram in Fig. 6, tries to answer to at least the following main competency questions:

- Who mined a specific block?
- What is the height of a specific block?
- How many transactions are written in a block?
- Is a transaction confirmed?
- How many total coins were transferred on a block?

The domain covered by the ontology includes the structural information of Bitcoin, Ethereum, and Hyperledger Fabric blockchains, as expressed in the official documentations, and also the following elements:

- The *block-header* of a block. This is the section summarizing the block itself. It includes some metadata such as the difficulty of the block and the time when the block was mined, the Merkle root of the included transactions, nonce, and so on.
- The *block*. Blocks are containers for all transactions. In BLONDiE, a block is represented by means of all the information concerning the block itself, such as the node miner and the block height, and it is associated with the transactions contained by means of its *payload*. Blocks are specialized according to the type of blockchain (Ethereum or Bitcoin) they belong to.

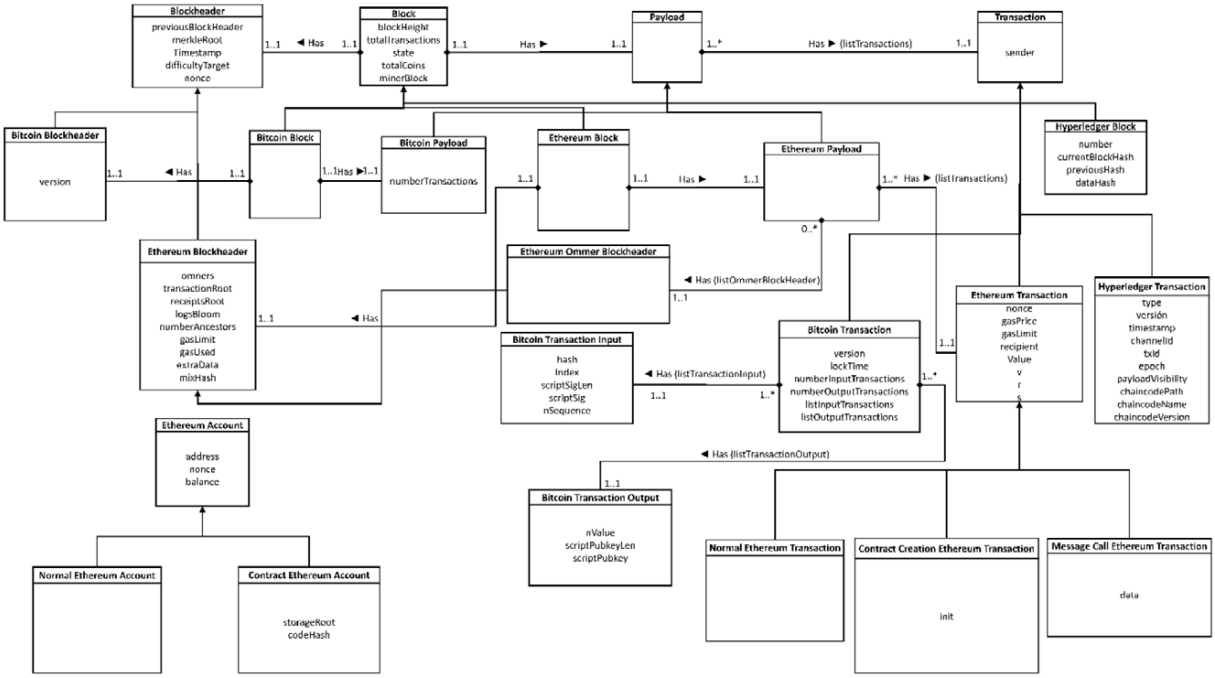

Fig. 6. UML diagram representing the BLONDiE ontology.

– The *payload*. It represents the data of the block and is specialized for each type of blockchain. Payloads in BLONDiE are associated with the related transactions.
– The *transaction*. Transactions consolidate state passages and are specialized, depending on the type of the transaction. In Ethereum, there are three types of transactions: *normal transactions* (associated with transfers of *Ether*), *contract creations* (associated with smart contract creation) and *message calls*, i.e., passages of messages from one account to another, possibly including data.[4]
– *Account*. Accounts are referred to wallets used to store cryptocurrencies, to pay for executing and to authorize transactions.

4. An ontological stack for blockchain-based e-commerce

The ontological stack adopts and extends state-of-the-art ontologies suitably selected for the blockchain-based 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 adopts and extends three ontologies, one for each underlying subdomain of knowledge, i.e., OASIS for the representation of commercial participants and smart contracts, GoodRelations for representing commercial offerings, and BLONDiE for describing Ethereum constitutional elements. By extending them, the stack constructs three novel ontologies; the first ontology is OC-Found, modelling the stakeholders of the blockchain-based commerce ecosystem and leveraging the OASIS ontology; the second ontology, called OC-Commerce and extending GoodRelations, is responsible for describing commercial offerings, assets, and activities under the vision of the behaviouristic approach; the third ontology, called OC-Ethereum and leveraging BLONDiE, is focused on the representation of Ethereum smart contracts and related tokens in compliance with the standard ERC20 for fungible tokens, ERC721 for non-fungible tokens, and ERC1155 for semi-fungible tokens.

4.1. The OC-Found ontology

The OC-Found ontology provides the semantic foundation required to represent the main actors and their actions of the blockchain-based commerce. The constructed model is also suitable for many other domains that require the characterization of agents, agent commitments, and supply chains, such as health-care, public services, and industry.

In addition to those provided by BLONDiE, the OC-Found ontology answers at least to the following competency questions (CF1-CF7), that describe classical situations about how the core of the ontology could be leveraged by final users.

CF1: Which are the participants currently available (including the associated digital identities and operations they can perform)?
The competency question CF1 is introduced to allow people and applications to probe the knowledge base for agents available to provide products and services, and the type of actions they can perform on client's request.

CF2: Which are the agents commitments and type of actions performed during their entire lifespan?
The competency question CF2 is designed to find all the actions committed by agents so as to check who executed the actions and the type of operations performed during the lifespan of the available agents. Hence, CF2 allows one to understand how the environment evolved as a consequence of the available agents' commitments.

CF3: Given the resource *resource*, which is, if any, the supply chain of *resource*?
The competency question CF3 is introduced to enable clients to discover the supply chain of a desired resource. Thanks to the supply chain, the resource can be consumed by the clients.

CF4: Given the resource *resource*, how the supply chain of *resource* is constituted?
The competency question CF4 can be seen as a specialization of CF3 and permits to look up for the specifications of the supply chain of a given resource.

[4]We recall that Ether is the cryptocurrency associated with the Ethereum blockchain.

CF5: Given the resource *resource*, which are the agents responsible for *resource*'s supply chain activities?
In line with CF3 and CF4, the competency question CF5 investigates on the agents that are responsible for the supply chain of a given resource. The identity and the behaviour of the agent is useful for the clients to establish whether they may trust the supply chain and, hence, if the resource is reliable.

CF6: Given the resource *resource*, which are the valuations (including the valuer agents) performed on *resource*?
In line with CF5, the competency question CF6 assists the client to settle his trustworthiness on a given resource by providing valuations performed on it together with the agent that performed them.

To achieve the objective, OC-Found applies the behaviouristic approach pursued by OASIS for representing agents to the commercial domain by extending the notions of agent behaviour template, agent behaviour, and agent action with the semantic representation of digital identities, supply chains, and quality valuation mechanisms.

In this section, we describe the ontological core of OC-Found and how it extends OASIS to reach the proposed goals.

The main classes and properties introduced by OC-Found to model supply chains and digital identities associated with agents are depicted in Fig. 7, which has been obtained by means of the editor Protégé [82]. Entities having the prefix *oasis* are those imported from the OASIS ontology, whereas the remaining ones are defined in OC-Found.

In OC-Found, agents are associated with digital identities, represented by instances of the class *DigitalIdentity* (subclass of the OASIS class *DescriptionObject*) through the object-property *hasDigitalIdentity* (subproperty of the OASIS property *owns*); the class hierarchy of *DigitalIdentity* can be expanded in order to describe different types of digital identities such as public keys. Additionally, agents that are also legal entities are presented by instances of the class *LegalEntities*, subclass of the OASIS class *Agent*.

In OC-Found, the life-cycle of assets is described by means of supply chains. These *encompass all of those activities associated with moving goods from the raw-materials stage through to the end user* [76]. Hence, more in general, supply chains concern all the activities describing the life-cycle of digital or physical resources from sourcing to consumption. By leveraging the above definition, OC-Found models supply chains as instances of the class *SupplyChainManagement* (subclass of the OASIS class Activity) encompassing all the activities describing the *life-cycle* of the involved resources and introduced by instances of one of the subclasses of the class *SupplyChainActivity*. Each subclass describes one of the phases of the resource's life-cycle. Each phase, in its turn, is connected with the behaviour of the agent responsible for its realization. Specifically, OC-Found currently includes the following subclasses of the class *SupplyChainActivity*:

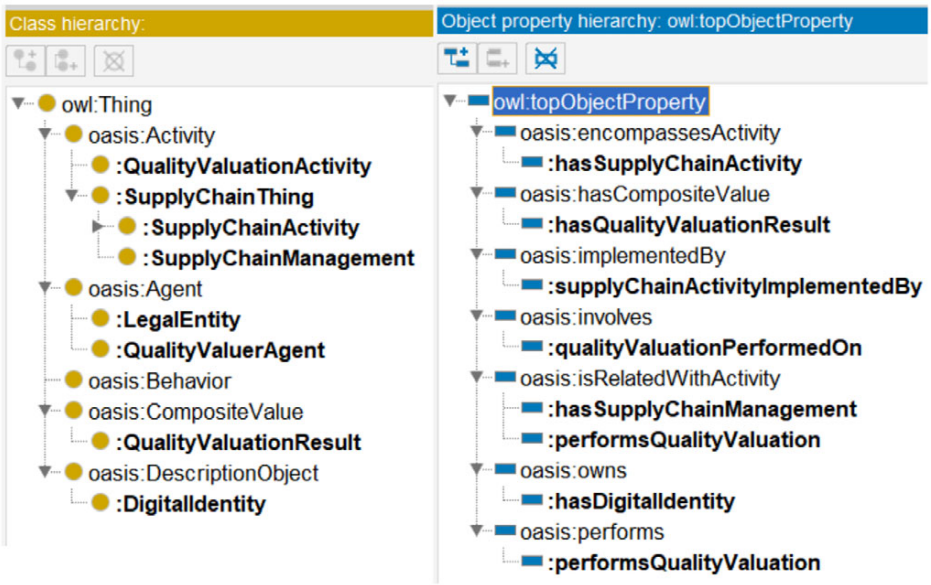

Fig. 7. Hierarchies of classes (on the left) and object-properties (on the right) in the OC-Found ontology.

- *SupplyChainOfferingProofActivity*. This class describes the activities connected with the process of releasing some proofs related with the consummation of the offering, such as digital tokens emitted to witness the transferring of ownership of the resource.
- *SupplyChainDeliveryActivity*. This class describes the process related with the delivery of the considered resource.
- *SupplyChainPaymentActivity*. This class describes the process related with the payment activity required to acquire the resource.
- *SupplyChainReleaseActivity*. This class describes the process related with the release mechanisms of the resource, such as manufacturing, production, assembling, and so on.

Resources may express many alternative supply chains that, in turn, may also specify one or more supply chain activities, depending on the specific life-cycle of the resource. Each supply chain activity, instead, must be related with the agent's behaviour responsible for its execution. The classes *SupplyChainManagement* and *SupplyChainActivity* are also defined in OC-Found as subclasses of the class *SupplyChainThing*, encompassing all the entities related with supply chains.

Usage of OC-Found classes and properties are depicted in Fig. 8, where the prefix *ocfound* is used for the namespace of the OC-Found ontology, properties are illustrated together with the related super-properties defined in OASIS, whereas OC-Found entities are reported in bold.

Resources are related with instances of the class *SupplyChainManagement* for each supply chain introduced by means of the object-property *hasSupplyChainManagement*, subproperty of the OASIS property *isRelatedWithActivity*. In turn, instances of the class *SupplyChainManagement* are connected with one or more instances of the subclasses of the class *SupplyChainActivity*, depending on the type of the activities of the resource's life-cycle to be described, through the object-property *supplyChainActivityImplementedBy* (subproperty of the OASIS property

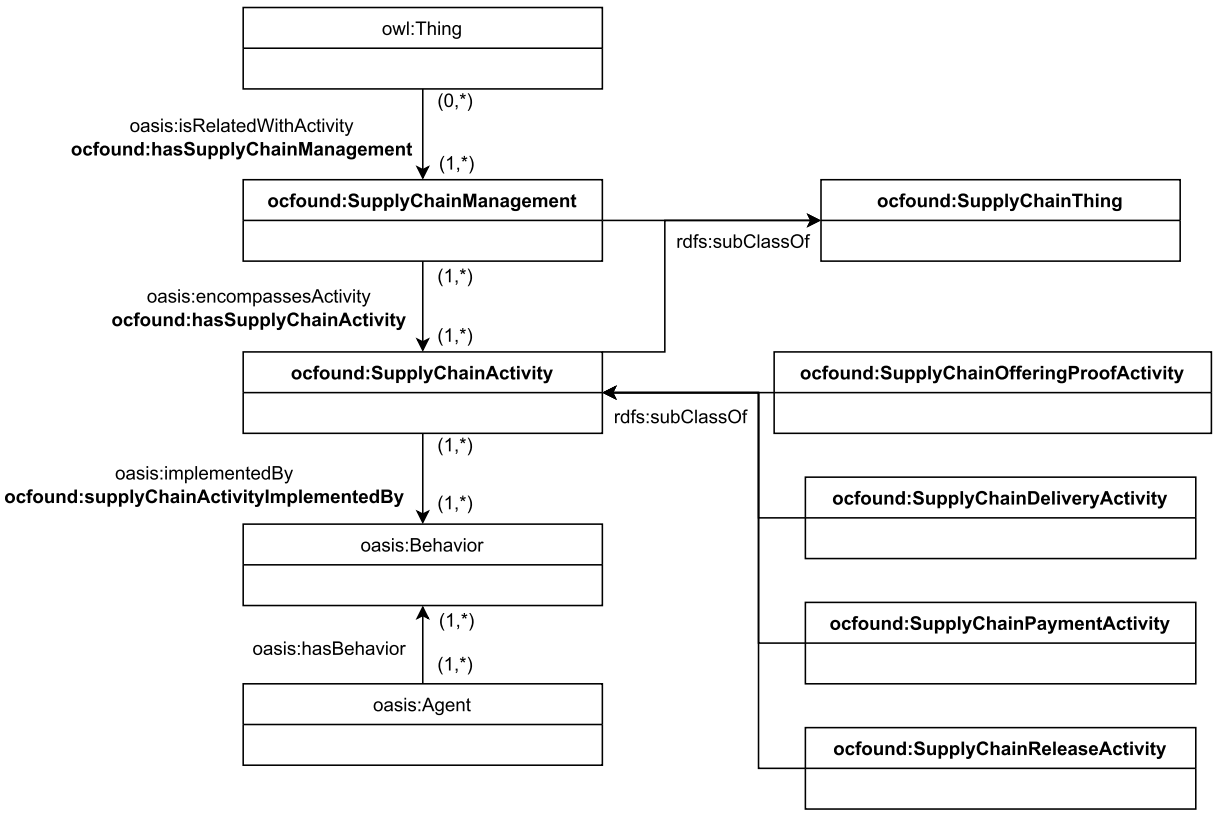

Fig. 8. UML diagram representing the OC-Found ontology.

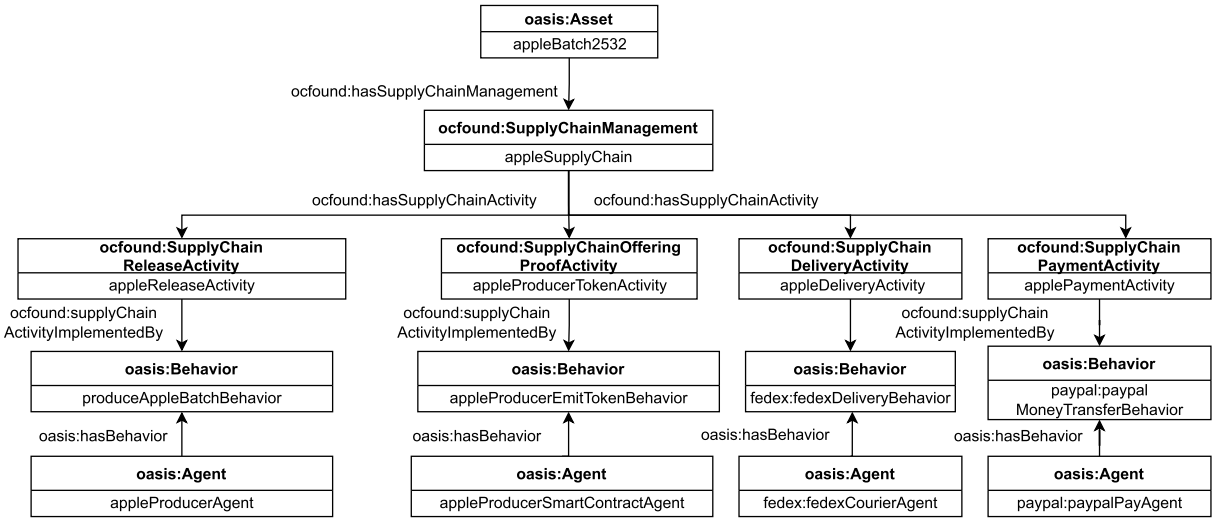

Fig. 9. UML diagram exemplifying an apple producer supply chain in OC-Found.

implementedBy). For example, the supply chain of an apple producer may be represented as depicted in Fig. 9, where classes are reported in bold and instances are reported below the related membership classes.

Specifically, the example in Fig. 9 describes an apple farmer providing for his harvest of apples collected in a batch (individual *appleBatch2532*) a supply chain (individual *appleSupplyChain*) consisting of four supply chain activities:

- the first activity (individual *appleReleaseActivity*) describes how the apple batch is produced, by connecting the activity with the behaviour responsible for producing apples, and is associated with the farmer agent (individual *appleProducerAgent*);
- the second activity introduces a token release mechanism entrusted to the smart contract of the apple producer (individual *appleProducerSmartContractAgent*), in order to provide a proof of transferring of the batch's ownership;
- the third activity (individual *appleDeliveryActivity*) describes the delivery activity of the batch, which is carried out by the agent *fedex:fedexCourierAgent*;
- the fourth activity (individual *applePaymentActivity*) is related with activities concerning the payment for the product, entrusted to the agent *paypal:paypalPayAgent*.

If dependency relationships among supply chain activities are required, the object-property of OASIS *dependsOn* can be used to connect an activity with the one it depends on. Hence, thanks to the description provided by OC-Found, the supply chain of the apple batch produced by the farmer is completely described, and the responsibilities of the agent entrusted for the activities that make up the supply chain are clearly defined.

In order to build an ontological trust management system based on participants experiences and feedbacks, OC-Found models the quality valuation processes performed on the resource either by professional quality valuer agents or by customers. With this aim, OC-Found provides the classes illustrated in Fig. 10 (entities that are newly introduced in OC-Found are reported in bold).

Agents performing professional quality valuations are defined as instances of the OC-Found class *QualityValuerAgent*. When quality valuers perform actions associated with a quality valuer behaviour, the agent is connected to the activity related with the execution of the quality valuation activity, defined as instance of the class *QualityValuationActivity*) by means of the object-property *performsQualityValuation* (subproperty of the OASIS property *isRelatedWithActivity*). In its turn, the instance of the class *QualityValuationActivity* is connected with a) the resource on which the valuation is performed, by means of the object-property *qualityValuationPerformedOn* (subproperty of the OASIS property *involves*) and with b) the result of the valuation, represented by an instance of the class

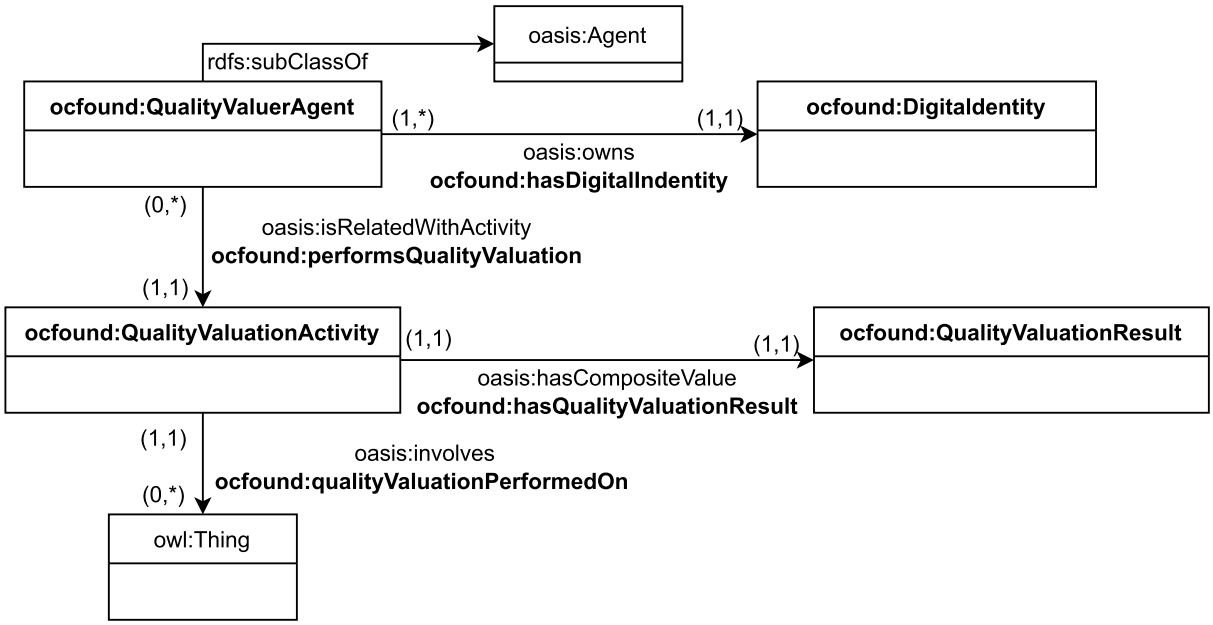

Fig. 10. UML diagram representing OC-Found quality valuation process.

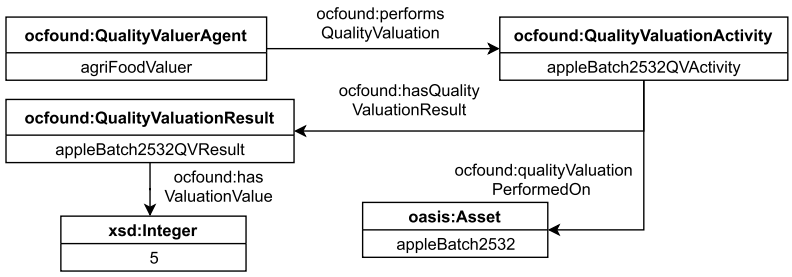

Fig. 11. UML diagram exemplifying the quality valuation on apple *batch 2563* in OC-Found.

QualityValuationResult (subclass of the OASIS class *CompositeValue*) by means of the object-property *hasQuality-ValuationResult* (subproperty of the OASIS property *hasCompositeValue*). An example of valuating the apple batch is illustrated in Fig. 11, where the valuer agent *agriFoodValuer* performs a valuation activity on the apple batch *appleBatch2532*, assigning a total result of 5.

4.2. The OC-Commerce ontology

We first show the structure of GoodRelations and how some of its concepts are extended in OC-Commerce. The OC-Commerce ontology[5] conjoins OC-Found with many of the basic features provided by the GoodRelations ontology to construct a means for representing the life-cycle of commercial assets, focused on the commerce carried through the Ethereum blockchain.

OC-Commerce extends the definition of offerings introduced in GoodRelations by means of OC-Found supply chains and agents responsible for their realization, including agent actions performed to publish, reject, accept, and close offerings. OC-Commerce is also an extension of GoodRelations for the most common commercial activities that are relevant to the e-commerce realm, which would build a blockchain-based framework to directly connect

[5]The ontology namespace is http://www.ngi.ontochain/ontologies/oc-commerce.owl.

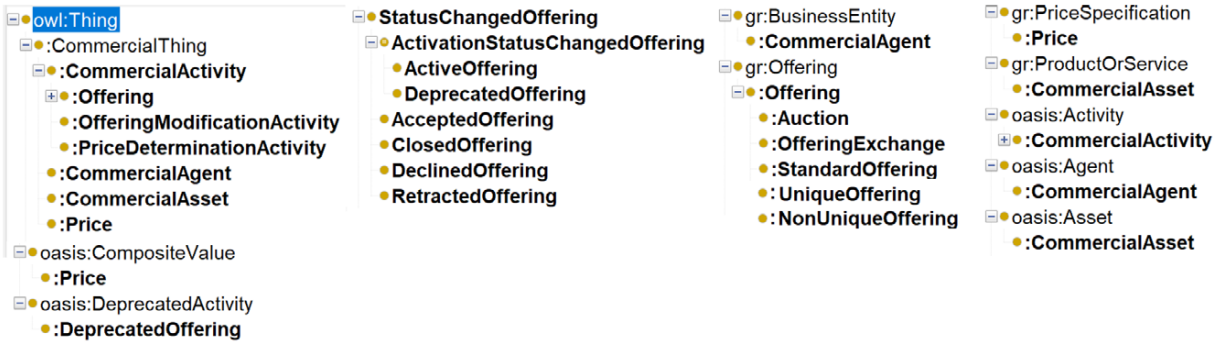

Fig. 12. Hierarchies of classes in the OC-Commerce ontology.

Fig. 13. Hierarchies of object-properties in the OC-Commerce ontology.

seller and buyer, but that are carried out by different enterprises such as eBay, Vinted, Eprice, ans so on. Specifically, OC-Commerce provides models to describe bargaining activities, auctions, modifications of offerings, valuations, and price determination mechanisms. For this reason, OC-Commerce foresees at least the following competency questions:

CC1: Which are the available offerings (including related details)?
Competency question CC1 provides a complete set of the available offering that clients can consume.

CC2: Given an offering *offering*, which is the supply chain related with *offering*?
Competency question CC2 is introduced to show the supply chain related with a given offering, explaining how clients should consume the asset traded.

CC3: Which are the accepted offerings?
Answers to competency question CC3 are intended to describe which offerings were consumed, thus providing a glue on how the market moved during its lifespan, and hence what type of resources have been traded.

The classes and object-properties introduced by OC-Commerce and the related mapping into GoodRelations are illustrated in Figs 12 and 13, respectively. The prefix *gr* is used to refer to the GoodRelations namespace.

In the same way as GoodRelations, OC-Commerce revolves around the concept of "offering", which represents the public announcement to publish or seek for a certain asset with specific supply chains and at certain conditions. In order to publish offerings, participant should expose a suitable behaviour involving the action *publish* and the task object related with a general instance of the OC-Commerce class *Offering* (subclass of the GoodRelations class *Offering*), as in the schema of Fig. 14. In an analogous way, agents enabled to request, modify, retract, close, accept, and reject offerings should manifest a suitable behaviour for each operation, where the related action is:

- *request*, if the agent is enabled to seek for an offering;
- *modify*, if the agent is enabled to change some features of a previously published offering;

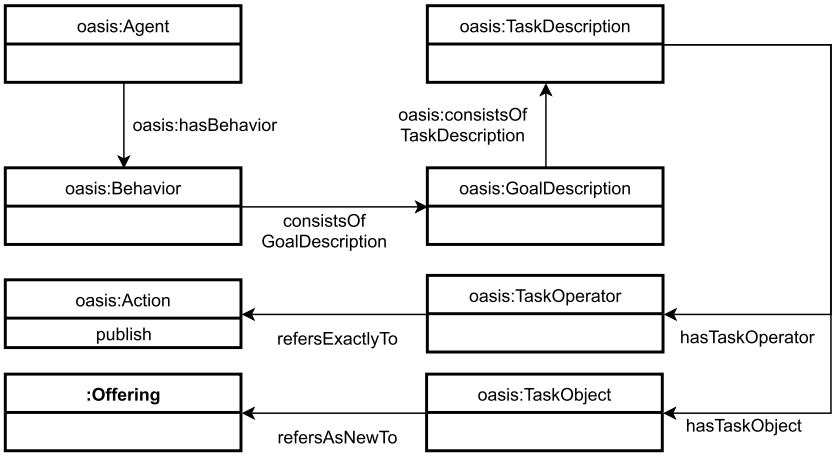

Fig. 14. UML diagram representing OC-Commerce offering publishing.

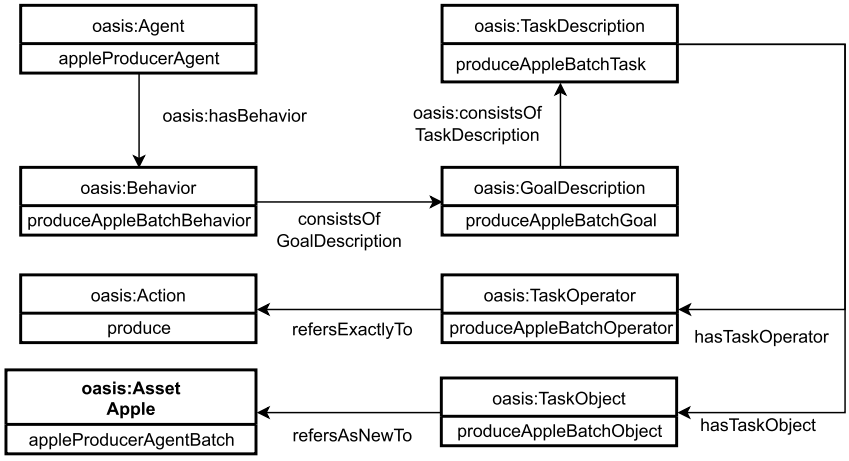

Fig. 15. UML diagram exemplifying the production of apple batch 2563 in the OC-Commerce ontology.

- *retract*, if the agent is enabled to cancel a previously published offering, meaning that it is no longer available due to some unexpected errors on the life-cycle of the asset or on the publication mechanism;
- *close*, if the agent is enabled to close a previously published offering, meaning that the offering is expired or the publisher autonomously decided to close the offering;
- *accept*, if the agent accepts the offering and the related selling condition and supply chain as it is (only counter-offerings or offerings for unique assets can be accepted);
- *decline*, if the agent rejects a proposed offering.

Before publishing an offering, the traded asset should be available beforehand through a specific agent action that releases the asset, as described in Section 4.1. For example, an apple producer, whose behaviour is depicted in Fig. 15, deploys a new asset of apples, namely *batch 2563*, and makes it available for trading, as reported in Fig. 16.

As a second step, the agent delegated to make the offering available publishes an action associated with the behaviour responsible for publishing offerings. In the case of the apple producer, a new offering concerning the batch 2563 should be published, as reported in Fig. 17.

The agent responsible for the action of publishing an offering can also be connected with the published offering by means of the object-property *publishesOffering*, subproperty of the GoodRelations property *offers*. In turn, offerings

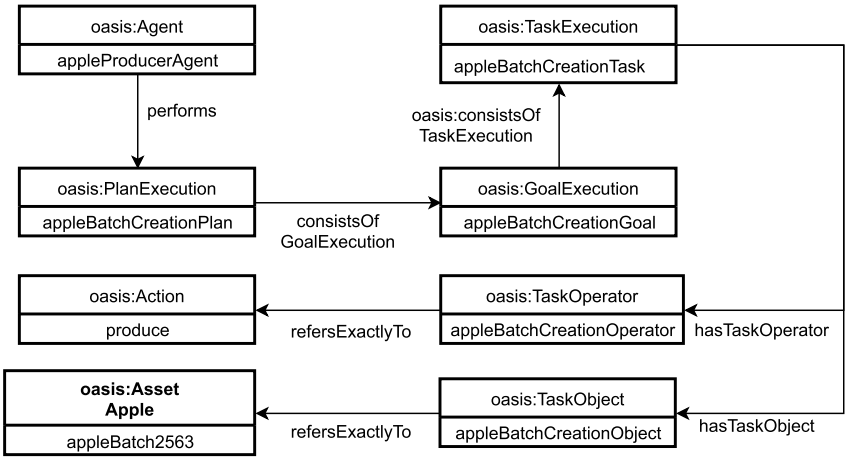

Fig. 16. UML diagram exemplifying the publication of apple batch 2563 in the OC-Commerce ontology.

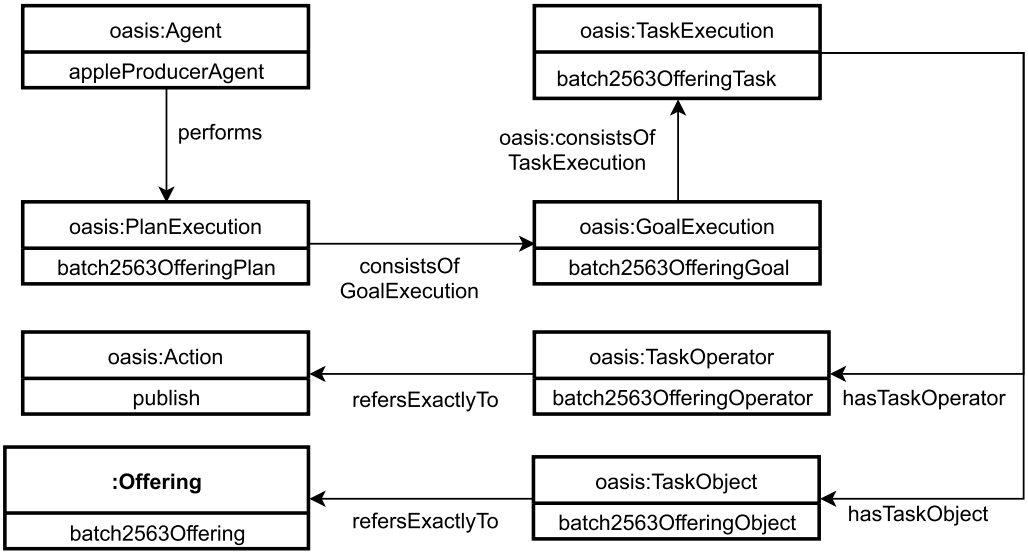

Fig. 17. UML diagram exemplifying the publication of an offering for apple batch 2563 in the OC-Commerce ontology.

must be connected both with the traded asset and with the related supply chain. In OC-Commerce, such relationships are modelled as depicted in the UML diagram in Fig. 18.

There are currently four types of offerings in OC-Commerce:

- *Standard offerings*, represented by instances of the class *StandardOffering*. In standard offerings, assets are traded by paying through crypto of FIAT currencies. Hence, supply chain activities of payments involve agent behaviours for transferring crypto of FIAT currencies, respectively. Offerings, in turn, may receive counter-offerings, represented by instances of *Offering* and connected with the initial offering by means of the object-property *isBidOnOffering*.
- *Exchange offerings*, represented by instances of the class *ExchangeOffering* and implementing bartering. In exchange offerings, assets are traded in exchange of other assets. The offering is related with the exchanged asset by means of the object-property *isOfferedFor*. Supply chain activities of payments related with exchange offerings involve the exchanged asset instead of an agent behaviour.

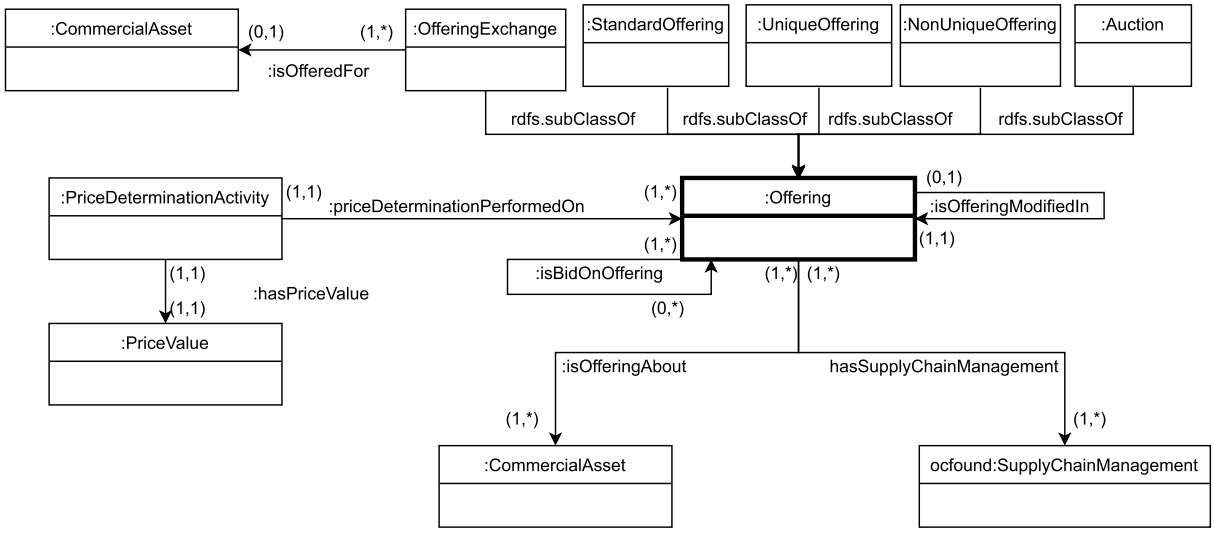

Fig. 18. UML diagram representing OC-Commerce offerings.

- *Auctions*, represented by instances of the class *Auction*. Auction bids are represented by instances of *Offering* and connected with the related auction by means of the object-property *isBidOnOffering*.
- *Counter-offerings*, represented by instances of the class *Offerings*, are offerings bid on standard offerings or exchange offerings. The object-property *isBidOnOffering* is used to connect the counter-offering with the offering it is bid on.

Moreover, offerings are also classified as

- *UniqueOffering*, when the traded asset is uniquely identifiable, such as second-hand objects, and
- *NonUniqueOffering*, when the traded asset comes from a stock of identical or indistinguishable objects.

Offerings are connected with the traded assets (instances of the class *CommercialAsset*, subclass of the OASIS class *Asset*) by means of the object-property *isOfferingAbout* (subproperty of the GoodRelations property *includes*), whereas supply chains are introduced by means of the object-property *hasSupplyChainManagement*, as described in Section 4.1.

In OC-Commerce, prices are conceived as the result of some *price determination activities* carried out either by the publisher of the offering or by suitable agents delegated to compute the value of specific assets (see Section 4.1). The activity of determining the price of an offering is represented by instances of the class *PriceDeterminationActivity*, which are connected with a) the offering by means of the object-property *priceDeterminationPerformedOn* and with b) the computed price (instance of the class *Price*) by means of the object-property *hasPriceValue*. In its turn, the instance of the class *Price* is related with the value of the price introduced as a float by means of the GoodRelations data-property *hasCurrencyValue*, where the selected currency is introduced as a string by means of the GoodRelations data-property *hasCurrency*.

Offerings change their status when they are accepted, closed, rejected or modified. An offering is also defined as an instance of the following classes:

- *AcceptedOffering*, if the offering has been accepted (only unique offerings or counter-offerings can change their status to accepted);
- *ClosedOffering*, if the offering has been closed;
- *DeclinedOffering*, if the offering has been declined (only counter-offerings can change their status to declined);
- *DeprecatedOffering*, if the offering has been replaced in favour of a new offering and hence is no longer valid;
- *RetractedOffering*, if the publisher has retracted the offering.

Our example continues with the full details of the offering for the asset batch 2563 of apples, as illustrated in Fig. 19. The offering involves the batch 2563, which is sold at the price of 1000 euros. The supply chain related with the offering is the one illustrated in Fig. 9 of Section 4.1. Then, the user Bob accepts the offering by performing the action depicted in Fig. 20 and, as a consequence, the publisher closes the offering. Subsequently, the two agents indicated in the supply chain perform the indicated actions, namely by registering the payment and shipping the product, respectively. The payment action, performed by the Paypal agent, is illustrated in Fig. 21. The action consists in transferring the established quantity of the selected currency from Bob's account to the apple producer's account, in order to pay for the apple batch in the offering. As a result of the action, a payment receipt is emitted. At this point, the Fedex agent ships the selected product, as described in Fig. 22. The user destination is represented by an instance of the GeoNames ontology [108], and a suitable receipt is generated to track the shipment. Additionally, the user Bob can assess his commercial transaction experience by valuating the quality of both the offering and the involved agents, by committing himself to an action as the one described in Section 4.1.

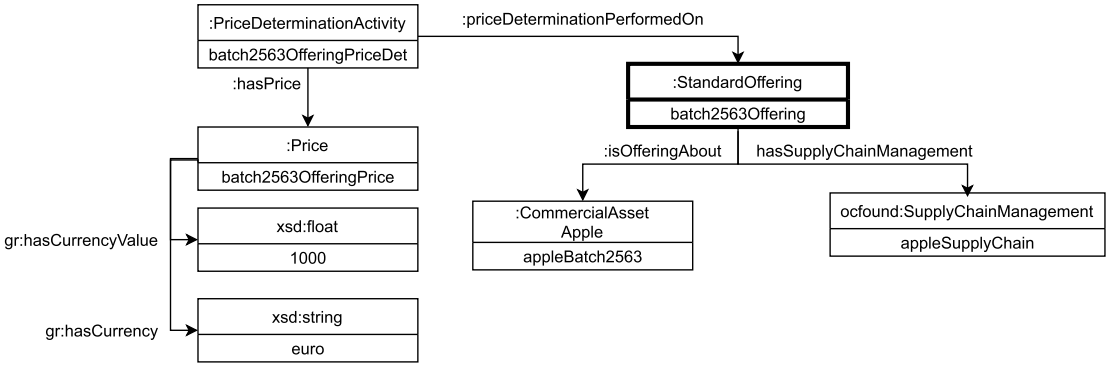

Fig. 19. UML diagram exemplifying the full details of an offering for apple batch 2563 in the OC-Commerce ontology.

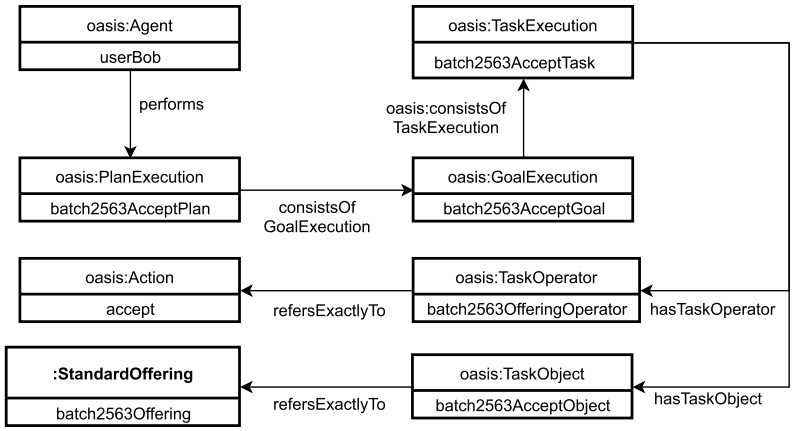

Fig. 20. UML diagram exemplifying the offering acceptance for apple batch 2563 in the OC-Commerce ontology.

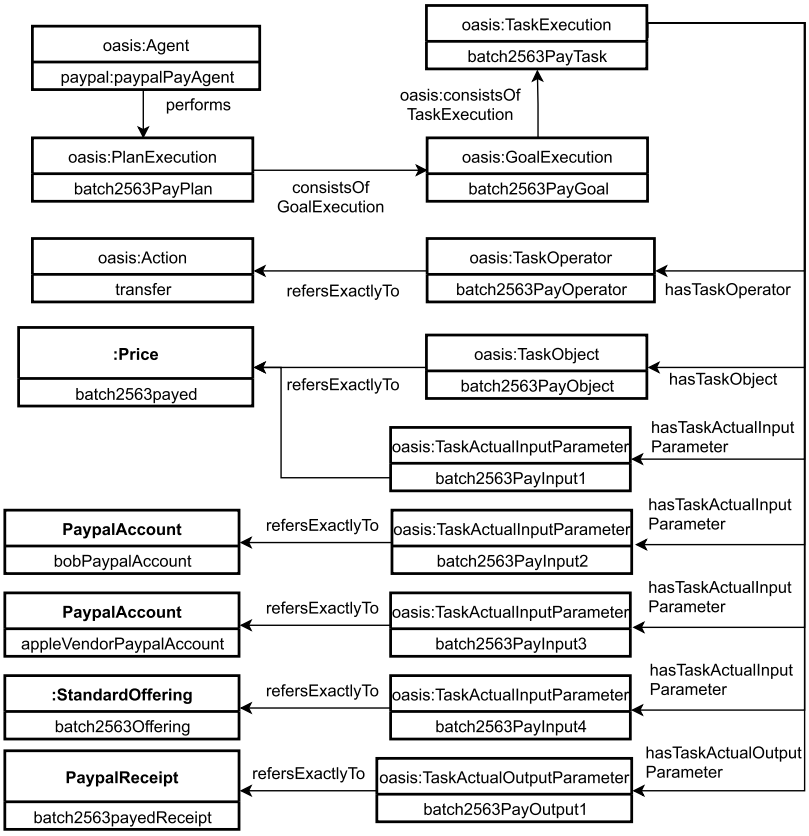

Fig. 21. UML diagram exemplifying the payment acceptance for apple batch 2563 in the OC-Commerce ontology.

Where allowed, offerings can be modified when some features, such as supply chains, have changed for any reason. A modified offering gets deprecated and replaced with a new offering endowing both all the features of the deprecated offering that are still valid and the features for which the new offering has been introduced. The UML diagram for offering modification is illustrated in Fig. 23.

When offerings are replaced by new ones, the deprecated offerings are defined as instances of the class *DeprecatedOfferings* and must not be involved in any new commercial activity. Instead, a new offering satisfying all the required features takes the place of the deprecated one. The deprecated offering is connected with the new offering by means of the object-property *isOfferingModifiedIn*. Moreover, a new modification activity needs to be introduced to possibly motivate the modification purposes, for example by specifying the agent that performed the action of modification. Modification activities are introduced as instances of the class *OfferingModificationActivity* connecting the abandoned offering by means of the object-property *hasOfferingModificationSource* and the new offering by means of the object-property *hasOfferingModificationResult*.

The OC-Commerce ontology provides representation means to describe auctions. Auctions are conceived as activities involving agents that propose bids on a particular type of offering, namely the instances of the class *Auction*. As any other type of offerings, auctions are characterized by three elements, that is, the supply chain, the traded asset, and the price, the latter conceived as the starting price of the auction. Users enabled to join the auction introduce a new *seek* action pipeline involving a new offering that represents the user bid. The bid is a general offering connected to the instance of the class *Auction* by means of the object-property *isBidOnOffering*, as illustrated in Fig. 24, in a way analogous as in counter-offerings.

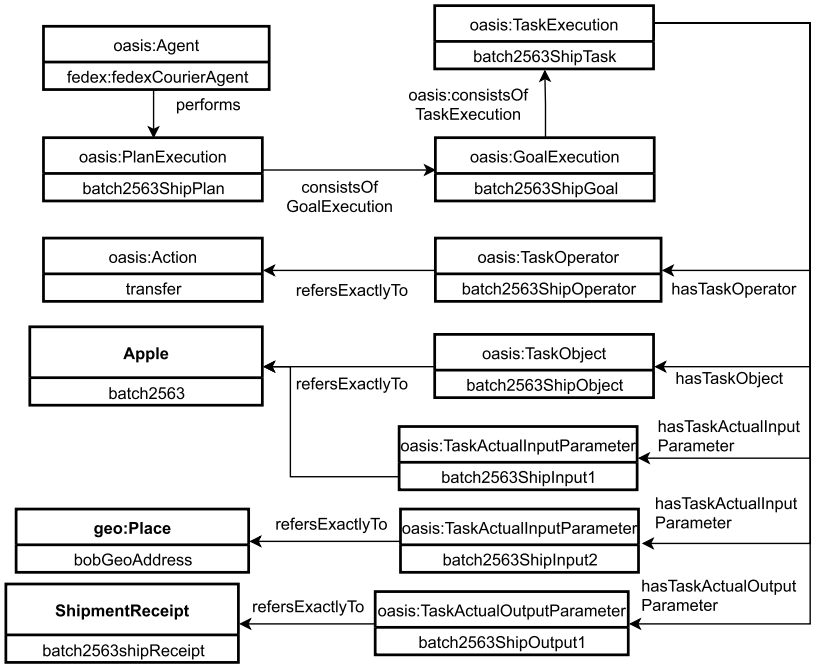

Fig. 22. UML diagram exemplifying the shipment confirmation for apple batch 2563 in the OC-Commerce ontology.

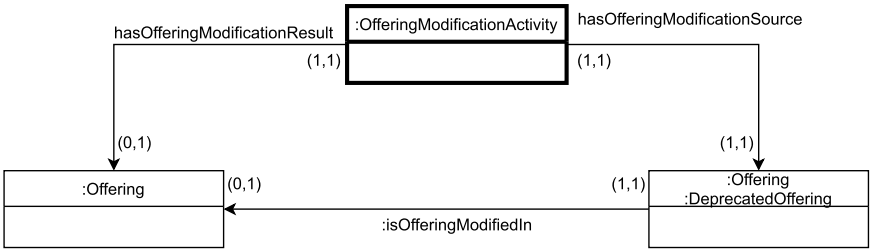

Fig. 23. UML diagram representing OC-Commerce offering modification.

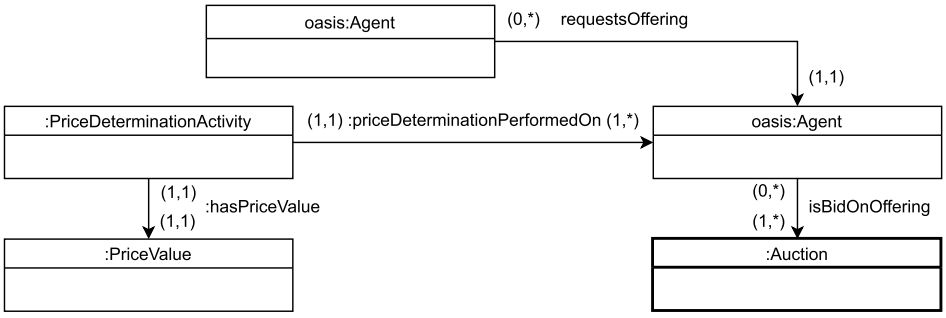

Fig. 24. UML diagram representing OC-Commerce auctions.

4.3. The OC-Ethereum ontology

The OC-Ethereum ontology extends with smart contracts and tokens the semantic model describing the essential elements of the Ethereum blockchain provided by the BLONDiE [105] ontology. OC-Ethereum conjoins the BLONDiE ontology with the OC-Commerce to provide commercial activities, in particular those based on the trading of NFTs carried out through the Ethereum blockchain, with the operational semantics as demanded by the behaviouristic approach.

Hence, OC-Ethereum answers at least to the following competency questions:

CE1: Which are the tokens minted and not destroyed, the related asset, minter, and current owner?
The competency question CE1 is defined to show the tokens available on the Ethereum blockchain and their type, providing an overview of the current arrangement of the token market.

CE2: Which are the block and the transaction hash that mint a given token?
The competency question CE2 allows users to verify that the given token has been effectively minted on the blockchain, and hence a proof for the related asset is available.

CE3: Which are the smart contracts that emit tokens related to a specific type of asset?
The competency question CE3 allows users to access the smart contracts that generate the tokens associated with the desired type of digital or physical asset.

CE4: Which is the number of tokens and the type of the related assets owned by wallets?
The competency question CE4 allows users to verify how many tokens associated with the desired asset are owned by wallets, thus permitting to check whether *whale wallets* own the desired tokens. The term whale wallet refers to individuals or entities that hold large amounts of specific cryptocurrencies or tokens, and hence have the potential to manipulate their valuations on the market.

To achieve the expected results, OC-Ethereum defines the classes and properties depicted in Figs 25 and 26, respectively.

OC-Ethereum mainly adopts the BLONDiE definitions of *EthereumBlock*, *EthereumPayload*, *EthereumTransaction*, and related subclasses, to describe the Ethereum blockchain entities securing smart contracts, tokens and operations leveraged to carry out commercial activities. Moreover, OC-Ethereum extends the BLONDiE model of

Fig. 25. Hierarchies of classes in the OC-Ethereum ontology.

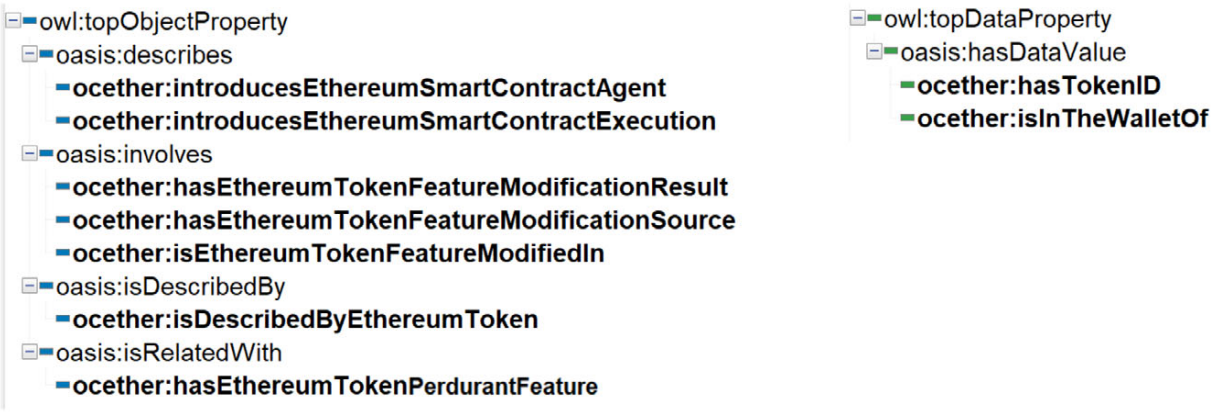

Fig. 26. Hierarchies of object-properties (on the left) and of data-properties (on the right) in the OC-Ethereum ontology.

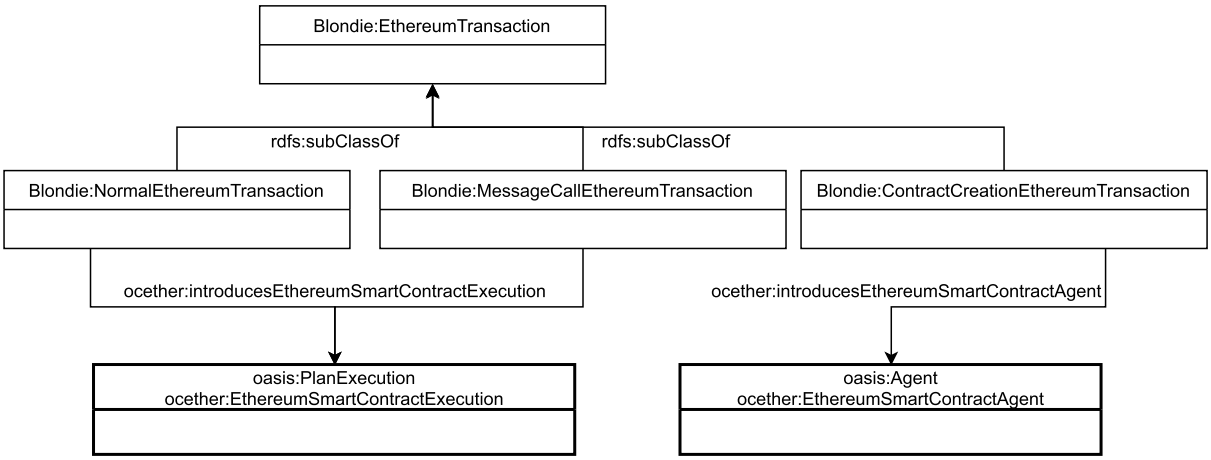

Fig. 27. UML diagram representing OC-Ethereum smart contracts.

Ethereum transactions by including an ontological representation of smart contracts and their operations published on the Ethereum blockchain, in particular of those related with the management of tokens, as depicted in Fig. 27. As a first step, OC-Ethereum connects a) transactions instantiating Ethereum smart contracts, as defined in BLONDiE with the related OASIS representation of smart contracts as agents running on the blockchain, and b) standard Ethereum transactions concerning smart contracts with the related OASIS representation of agent actions.

To provide BLONDiE with the representation of smart contracts and related operations, OC-Ethereum connects instances of the BLONDiE class *NormalEthereumTransaction* and *MessageCallEthereumTransaction* with instances of the OC-Ethereum class *EthereumSmartContractExecution* (subclass of the OASIS class *PlanExecution*) by means of the object-property *introducesEthereumSmartContractExecution*, in order to associate the transactions that secured smart contract operations with the semantic representation of the actions performed. Then, OC-Ethereum connects instances of the BLONDiE class *ContractCreationEthereumTransaction* with instances of the OC-Ethereum class *ocether:EthereumSmartContractAgent* (subclass of the OASIS class *Agent*) by means of the object-property *introducesEthereumSmartContractAgent*, thus associating the transactions that instantiate smart contracts with the ontological representation of the smart contracts in terms of OASIS agents.

Specifically, OC-Found identifies five types of Ethereum smart contract agents, suitably represented by the following classes:

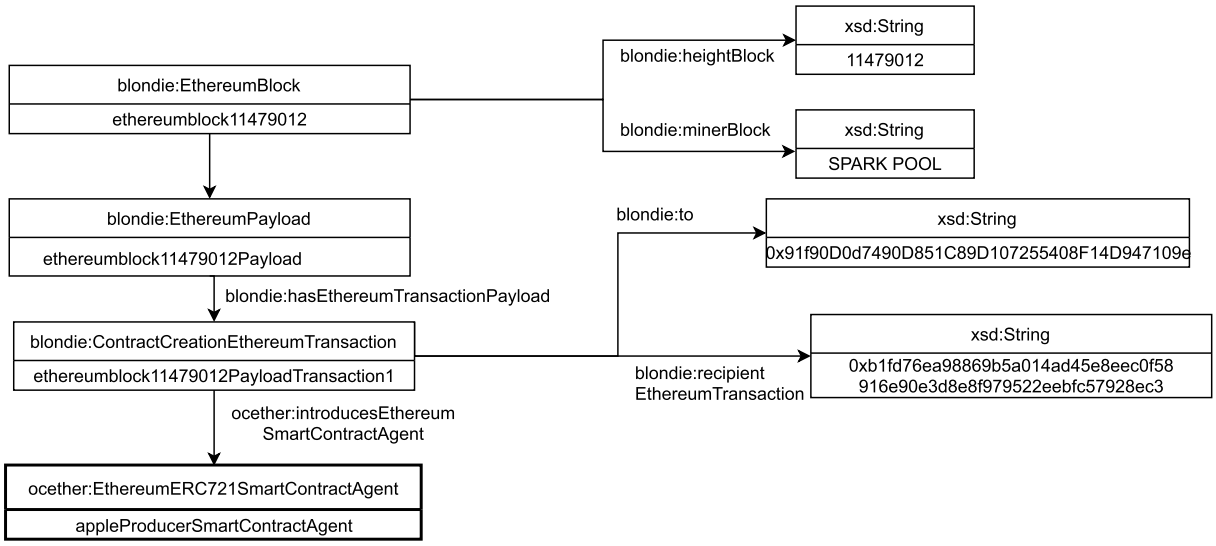

Fig. 28. UML diagram exemplifying an apple producer's smart contract in the OC-Ethereum ontology.

- *EthereumFungibleSmartContractAgent*, representing smart contracts for trading fungible tokens (henceforth called fungible smart contracts) and containing the class *EthereumERC20SmartContractAgent* that represents smart contracts compliant with the ERC20 standard protocol;
- *EthereumNonFungibleSmartContractAgent*, representing smart contracts for trading non-fungible tokens (henceforth called non-fungible smart contracts) and containing the class *EthereumERC721SmartContract-Agent* that represents smart contracts compliant with the ERC721 standard protocol.
- *EthereumSemiFungibleSmartContractAgent*, representing smart contracts for trading semi-fungible tokens (henceforth called semi-fungible smart contracts) and containing the class *EthereumERC1155SmartContract-Agent* that represents smart contracts compliant with the ERC1155 standard protocol;
- *CustomEthereumSmartContractAgent*, representing user-defined smart contracts that are not compliant with the ERC standards.

For example, the smart contract generated by our apple producer is represented by means of the fragment in Fig. 28.

The block and the transaction securing the apple producer smart contracts are introduced by means of the instances of the BLONDiE classes *EthereumBlock* and *EthereumTransaction*, respectively. The instances of the former class provide, among others, information concerning the block number (by means of the data-property *heightBlock*) and the miner of the block (by means of the data-property *minerBlock*), whereas the instances of the latter class provide information about the signed transactions, such as the transaction hash (by means of the data-property *to*) and the smart contract address (by means of the data-property *recipientEthereumTransaction*). Finally, the transaction, as modelled in BLONDiE, is provided with the ontological description of the smart contract agent by means of the OC-Ethereum object-property *introducesEthereumSmartContractAgent*. In the example considered, the entity *appleProducerSmartContractAgent* represents the smart contract of the apple producer.

In addition, the OC-Ethereum ontology extends BLONDiE by describing tokens generated and exchanged through the Ethereum blockchain, in particular for commercial purposes. As illustrated in Fig. 29, OC-Ethereum identifies four main types of tokens:

- *fungible tokens*, represented by instances of the class *EthereumFungibleToken*, containing the class *Ethereum-TokenERC20* that represents fungible tokens compliant with the ERC20 standard protocol;
- *non-fungible tokens*, represented by instances of the class *EthereumSemiFungibleToken*, containing the class *EthereumTokenERC721* that represents non-fungible tokens compliant with the ERC721 standard protocol;

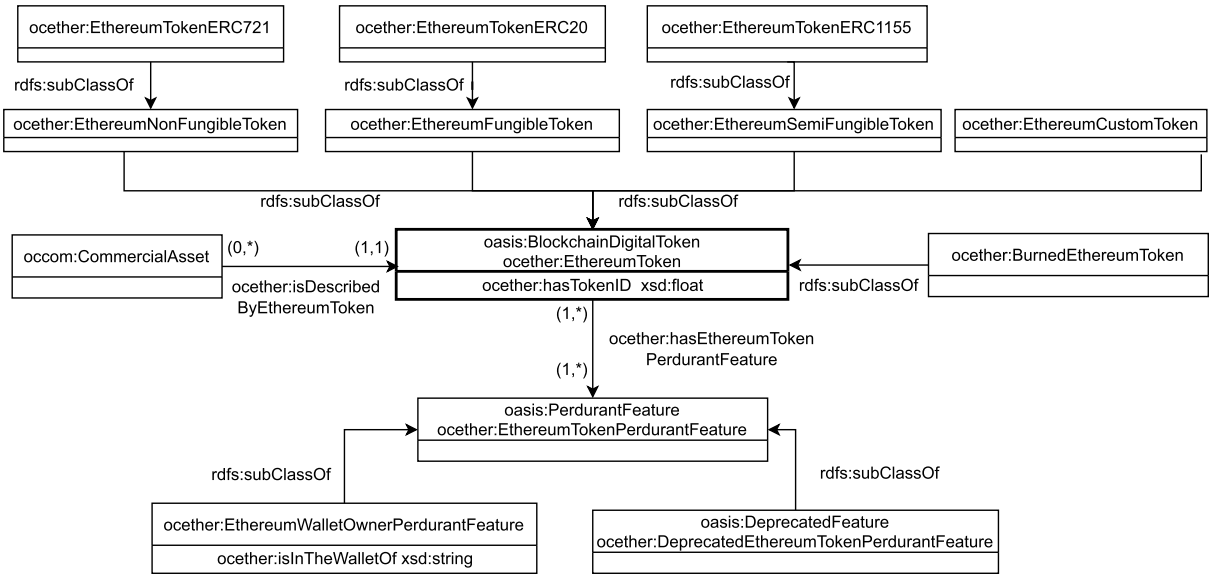

Fig. 29. UML diagram representing OC-Ethereum tokens.

- *semi-fungible tokens*, represented by instances of the class *EthereumSemiFungibleToken*, containing the class *EthereumTokenERC1155* that represents semi-fungible tokens compliant with the ERC1155 standard protocol;
- custom *user-defined tokens*, not compliant with the ERC standard protocols and represented by instances of the class *EthereumCustomToken*.

The four above-mentioned classes are defined as subclasses of the class *EthereumToken*. Additionally, tokens that have been definitively destroyed are also instances of the class *BurnedEthereumToken*.

In OC-Ethereum, commercial assets (instances of the OC-Commerce class *CommercialAsset*) that are uniquely associated with Ethereum tokens are related with instances of the class *EthereumToken* by means of the object-property *isDescribedByEthereumToken*. Tokens carry two types of features [57], namely a) *endurant features*, such as the token ID, that never change and are embedded with the entity representing the token, and b) *perdurant features* that change during the life-span of the token and are associated with an instance of the OC-Ethereum class *EthereumTokenPerdurantFeatures* (subclass of the OASIS class *PerdurantFeature*) by means of the object-property *hasEthereumTokenPerdurantFeature*. The most notable subclass of *PerdurantFeature* is the class *EthereumWalletOwnerPerdurantFeature*, which describes the wallet of the token's owner (by means of the data-properties *isInTheWalletOf*, having as range *XSD:string*). When the perdurant features of a token are modified by the smart contract managing it, they become deprecated and replaced by a new set of features by means of a modification activity. Such new features are introduced by a fresh instance of the class *PerdurantFeature*, as illustrated in Fig. 30.

In OC-Ethereum, the modification of tokens is allowed only if it involves perdurant features; hence, endurant features cannot be modified. Perdurant features may be replaced with other perdurant features by introducing an instance of the class *EthereumTokenFeatureModificationActivity*, which is connected with:

- the changed perdurant feature, which is also an instance of the class *DeprecatedEthereumTokenPerdurantFeature* by means of the object-property *hasEthereumTokenFeatureModificationSource*;
- the new perdurant feature, by means of the object-property *hasEthereumTokenFeatureModificationResult*.

Moreover, the modified perdurant feature is connected as usual with the perdurant feature that replaces it by means of the object-property *isEthereumTokenFeatureModifiedIn*, while the token embedding the features is connected with the new perdurant feature by means of the object-property *hasEthereumTokenPerdurantFeature*.

An example of representing tokens in OC-Ethereum is illustrated in Fig. 31, which shows a token emitted by the apple producer's smart contract.

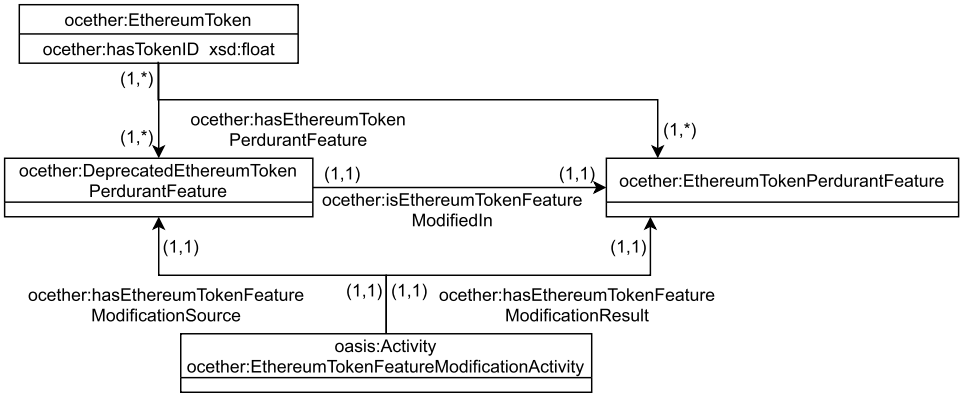

Fig. 30. UML diagram representing OC-Ethereum token modification.

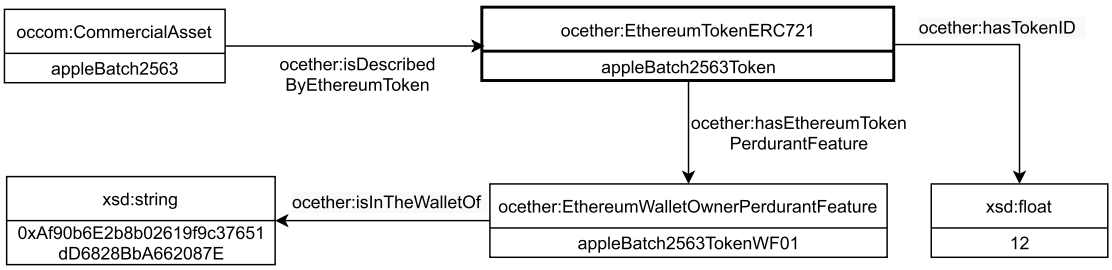

Fig. 31. UML diagram exemplifying the publication of the token for apple *batch 2563* in the OC-Ethereum ontology.

The token *appleBatch2563Token*, with identification code *12*, is associated with the apple batch 2563. The token is also associated with a perdurant feature describing the current owner's wallet. The perdurant feature is introduced by the entity *appleBatch2563TokenWF01*, instance of the class *EthereumWalletOwnerPerdurantFeature* and connected with the string representing the wallet's owner by means of the data-property *isInTheWalletOf*.

5. Evaluation of the ontological stack

The ontological stack was evaluated during the entire life-time of its design and development through four main KPIs. To begin with, the *consistency* check of the stack was carried on by means of three main OWL reasoners, thus demonstrating the soundness of the ontologies. Then, *structural metrics* of the stack that depicts the number and type of elements used to define the ontologies, such as number of classes and object-properties, were computed, providing a general evaluation of how much the ontologies are large and complex. Protégé was used to compute the structural metrics on the ontologies.

Furthermore, *ontological metrics* were computed on the *ontologies root* of the stack, i.e., the novel fragments of the ontologies excluding the imports from the external ones (namely OASIS, GoodRelations, and BLONDiE). A comparison with the imported ontologies is also provided. Ontological metrics are feature-based methods for evaluating ontologies that do not require machine learning and do not involve users. Metrics are necessary to evaluate ontologies, both during the design and implementation phase, thus allowing for fast and simple assessment of ontologies and ensuring both correct domain coverage and suitability of the ontologies. Ontological metrics were computed leveraging the OntoQA approach [97].[6] The evaluation took into account all the schema metrics defined in

[6]OntoQA is a feature-based method for evaluating ontologies by applying techniques that do not require data training and that involve users minimally.

OntoQA that address the design of the schema of an ontology, namely *relationship richness*, *inheritance richness*, *tree balance*, *attribute richness*, and *class richness*. The *relationship richness* metric reflects the diversity of the relations and the placement of the relations occurring in the ontology. An ontology that contains more property relations than class/subclass relations is richer than a taxonomy (with only class/subclass relationships). The value of relationship richness is a percentage representing how many relationships between classes are rich with respect to all the possible connections (inheritance and properties). The *inheritance richness* describes the distribution of information across the different levels of the inheritance tree of the classes. It represents an indicator of how well knowledge in the ontology is grouped into distinct categories and subcategories. Such a measure can help to distinguish a horizontal ontology either from a vertical ontology, or from an ontology with different levels of specialization. A horizontal (or flat) ontology has a few inheritance levels, and each class has a relatively large number of subclasses. On the contrary, a vertical ontology contains many inheritance levels, where classes have some subclasses. An ontology with a low inheritance richness would be of a vertical nature, which might reflect the fact that the ontology represents a very specific and well detailed knowledge. An ontology with a high value of inheritance richness has a horizontal nature, which means that the ontology represents a wide range of general knowledge. The *tree balance* metric is referred to how many class hierarchies differ in deepness. This may be related to the fact that some hierarchies are very deep, whereas others are not. The *class richness* is related to how instances are distributed across classes. An ontology having a very low class richness does not have data exemplifying the knowledge represented in the model. On the other hand, a high value of such metric (close to 100%) indicates that the data represents most of the knowledge described in the considered ontology. Finally, the *attribute richness* calculates the average number of attributes per class, which gives insight into how much knowledge about classes is represented in the model. An ontology with a high value for attribute richness indicates that each class has averagely a high number of attributes, namely that it is specified in detail, whereas a low value might indicate that little information is provided about each class.

The proposed competency questions are implemented by suitable SPARQL queries in order to be performed against the developed ontologies. SPARQL queries also constitute *regression* and *integration* tests for the ontological stack.

In what follows, we report the results of the evaluation methodology, including the SPARQL queries implementing the competency questions, adopted and applied to the three ontologies, OC-Found, OC-Commerce, and OC-Ethreum, together with a discussion of the results obtained.

5.1. Evaluation of the OC-Found ontology

In this section, we introduce the evaluation methods and the results obtained for the OC-Found ontology. We first present the ontological metrics and then the SPARQL queries implementing the identified competency questions.

5.1.1. Metrics

To begin with, we discuss the results of the evaluation of the OC-Found ontology. Consistency of OC-Found has been checked by the reasoners Pellet [92], HermiT [60], and FaCT++ [103]. The main structural metrics computed on the OC-Found ontology are reported in Table 1. In the first column, we report the metrics computed on the imported ontology OASIS, whereas the metrics on the root of OC-Found, i.e., the new fragments introduced in OC-Found, are reported in the second column. As discussed above, the root of OC-Found refers to supply chain and digital identities that are introduced by specializing some entities provided by OASIS. Consequently, there is a noticeable difference in size between the two ontologies.

The ontological metrics computed on OC-Found are reported in Table 2 and compared with those computed on OASIS.

OC-Found exhibits a relationship richness of 57.14%, indicating a well-balanced mix of generic relationships and class hierarchies. The value obtained from OC-Found is higher than the one computed on OASIS, since it considers a small domain that is modelled in detail. The inheritance richness stands at 2%, confirming the vertical nature of the ontology, which is mainly focused on supply chains. The sharp difference with the value obtained from OASIS is determined by the fact that the core of OC-Found consisting in representing agents and their commitments is inherited from OASIS and thus it does not affect the resulting value. The foundational nature of the OC-Found

Table 1
Structural metrics on OC-Found

Metric	OASIS import	Root of OC-Found
Axiom count	1260	68
Logical axiom count	621	32
Declaration axiom count	404	19
Class count	203	16
Object-property count	177	14
Subclass axiom count	209	12
Sub-object-property axiom count	169	8
Object property domain axiom count	92	6
Object property range axiom count	121	6
Annotation assertion	227	17

Table 2
Ontological metrics on OC-Found

Evaluation criteria	OASIS import	Root of OC-Found	Delta %
Relationship richness	48.52%	57.14%	+15.08%
Inheritance richness	2.67%	2%	−33.5%
Tree balance	1.74%	0.90%	−93.33%
Attribute richness	0.77%	0.61%	−26.22%
Class richness	11.55%	3.96%	−11.11%

ontology concentrated on a relatively small domain is also confirmed by its low class richness (11.11%) and attribute richness (that is 0.61%), whose values are close to those for OASIS. Finally, OC-Found provides a tree balance of 0.90% (e.g., *SupplyChainActivity*), meaning that some hierarchies have been well described, others remain very general (e.g., *DigitalIdentity*), whereas others that are imported from OASIS (e.g., *Agent* and *Behavior*) do not provide any contribution to the final value of the metric.

5.1.2. SPARQL queries

We are ready to introduce the SPARQL queries that implement the competency questions **CF1-CF7** (see Section 4.1). Specifically,

CF1: Which are the participants currently available (including the associated digital identities and operations they can perform)?

The answer to **CF1** is entailed by the SPARQL query QF1.

Query QF1

```
SELECT DISTINCT ?agent ?identity ?operation ?operationOn
WHERE {
        ?agent ocfound:hasDigitalIdentity ?identity.
        ?agent ocfound:hasDigitalIdentity ?identity.
        ?agent oasis:hasBehavior ?behaviour.
        ?behaviour oasis:consistsOfGoalDescription ?goal.
        ?goal oasis:consistsOfTaskDescription ?task.
        ?task oasis:hasTaskOperator ?operator.
        ?operator oasis:refersExactlyTo ?operation.
         ?task oasis:hasTaskObject ?object.
         ?object oasis:refersAsNewTo ?ob.
         ?ob a ?operationOn
         FILTER( ?operationOn != owl:NamedIndividual)
         }
```

CF2: Which are the agents commitments and type of actions performed during their entire lifespan?

The answer to **CF2** is entailed by the SPARQL query QF2.

Query QF2

```
1:  SELECT DISTINCT ?agent ?operation ?operationOn ?typeOf
2:  WHERE {
3:          ?agent oasis:performs ?agentExe.
4:          ?agentExe oasis:hasTaskObject ?taskExe.
5:          ?agentExe oasis:hasTaskOperator ?operator.
6:          ?operator oasis:refersExactlyTo ?operation.
7:          ?taskExe oasis:refersExactlyTo ?operationOn.
8:          ?operationOn a ?typeOf.
9:          FILTER( ?typeOf != owl:NamedIndividual)
10:         }
```

CF3: Given the resource *resource*, which is, if any, the supply chain of *resource*?

The answer to **CF3** is entailed by the SPARQL query QF3.

Query QF3

```
1: Let resource be the resource of which the supply chain should be discovered
2:  SELECT ?supplychain
3:  WHERE {
4:          resource ocfound:hasSupplyChainManagement ?supplychain .
5:          }
```

CF4: Given the resource *resource*, how the supply chain of *resource* is constituted?

The answer to **CF4** is entailed by the SPARQL query QF4.

Query QF4

```
1: Let resource be the resource of which the supply chain activities should be discovered
2:  SELECT ?supplychainActivity
3:  WHERE {
4:          resource ocfound:hasSupplyChainManagement ?supplychain.
5:          ?supplychain ocfound:hasSupplyChainActivity ?supplychainActivity.
6:          }
```

CF5: Given the resource *resource*, which are the agents responsible for *resource*'s supply chain activities?

The answer to **CF5** is entailed by the SPARQL query QF5.

Query QF5

```
1: Let resource be the resource of which the supply chain activity agents should be discovered
2:  SELECT ?agent ?supplychainActivity
3:  WHERE {
4:          resource ocfound:hasSupplyChainManagement ?supplychain.
5:          ?supplychain ocfound:hasSupplyChainActivity ?supplychainActivity.
6:          ?supplychainActivity ocfound:supplyChainActivityImplementedBy ?behaviour.
7:          ?behaviour a oasis:Behavior.
8:          ?agent oasis:hasBehavior ?behaviour.
9:          }
```

CF6: Given the resource *resource*, which are the valuations (including the valuer agents) performed on *resource*?

The answer to **CF6** is entailed by the SPARQL query QF6.

CF7: Given the resource *resource*, which is the average score of valuation of *resource* and how many valuations are there?

The answer to **CF7** is entailed by SPARQL query QF7.

Query QF6

```
Let resource be the resource of which valuations should be discovered
   SELECT DISTINCT ?agent ?score
   WHERE {
            ?agent oasis:performs ?agentExe.
            ?agentExe oasis:hasTaskObject ?taskExe.
            ?agentExe oasis:hasTaskOperator ?operator.
            ?operator oasis:refersExactlyTo oabox:perform.
            ?taskExe oasis:refersExactlyTo ?qualityValuation.
            ?qualityValuation a ocfound:QualityValuationActivity.
             ?qualityValuation ocfound:hasQualityValuationResult ?result.
             ?qualityValuation ocfound:qualityValuationPerformedOn resource.
             ?result ocfound:hasValuationValue ?score.
             }
```

Query QF7

```
Let resource be the resource of which the average score of valuations should be discovered
   SELECT (AVG(?score) AS ?AverageScore) (COUNT(?agent) AS ?numberOfValuation)
   WHERE {
            ?agent oasis:performs ?agentExe.
            ?agentExe oasis:hasTaskObject ?taskExe.
            ?agentExe oasis:hasTaskOperator ?operator.
            ?operator oasis:refersExactlyTo oabox:perform.
            ?taskExe oasis:refersExactlyTo ?qualityValuation.
            ?qualityValuation a ocfound:QualityValuationActivity.
             ?qualityValuation ocfound:hasQualityValuationResult ?result.
             ?qualityValuation ocfound:qualityValuationPerformedOn resource.
             ?result ocfound:hasValuationValue ?score.
             }
```

5.2. Evaluation of the OC-Commerce ontology

We begin by introducing the evaluation methods and the related results for the OC-Commerce ontology. We first present the structural and ontological metrics and then the SPARQL queries that implement the provided competency questions.

5.2.1. Metrics

In this section we discuss the ontological metrics computed on the OC-Commerce ontology. Consistency of the OC-Commerce ontology was checked by the reasoners Pellet, HermiT and FaCT++ as usual. The main structural metrics computed on OC-Commerce are reported in Table 3. The metrics on the imported ontology GoodRelations are reported in the first column, whereas the root of the OC-Commerce ontology are reported in the second one.

The difference of dimension between OC-Commerce and GoodRelations is the result of their inheritance relationship, since OC-Commerce requires many entities provided by GoodRelations and by OC-Found. The ontological metrics computed by OntoQA on OC-Commerce are reported in Table 4 and compared with GoodRelations.

Table 3
Structural metrics on OC-Commerce

Metric	GoodRelations import	Root of OC-Commerce
Axiom count	1188	113
Logical axiom count	450	61
Declaration axiom count	196	29
Class count	38	30
Object property count	53	18
Subclass axiom count	19	25
Sub-object-property axiom count	4	15
Object property domain axiom count	50	10
Object property range axiom count	50	10
Annotation assertion	0	23

Table 4
Ontological metrics on OC-Commerce

Evaluation criteria	GoodRelations import	Root of OC-Commerce	Delta %
Relationship richness	84.67%	40.81%	−107.47%
Inheritance richness	2.11%	2.07%	−2.85%
Tree balance	0.98%	1.35%	+27.40%
Attribute richness	0.5%	0.40%	−19.99%
Class richness	83.76%	6.25%	−1240%

The imported ontology GoodRelations is reported in the first column, whereas the root of the OC-Commerce ontology is reported in the second one, and the difference in percentage among the two ontologies is reported in the last column.

Relationship richness of OC-Commerce reports a lower value than GoodRelations because many entities are inherited and extended, and therefore they cannot be considered in the evaluation of the metric. Thus, the value still indicates a good balancing between description of properties and class hierarchies, with a propensity for the former. The difference of inheritance richness between OC-Commerce and GoodRelations is notable, as OC-Commerce's class hierarchies such as *Offerings* and *PriceDeterminationActivity* have very different levels of depth. Tree balance of OC-Commerce is higher than the one computed on GoodRelations, since classes such as *Offering* are deeply specialized in OC-Commerce in a vertical way. GoodRelations provides a higher value of attribute richness due to the thorough characterization of some classes, such as *Offering*. These classes are not considered in the evaluation of OC-Commerce, as they are imported, and therefore they do not belong to the root of the ontology. Class richness of OC-Commerce remains low, since data are not present within the ontology, whereas GoodRelations introduces many individuals used to apply the *punning* technique.

OC-Commerce may also be usefully compared with OC-Found. Relationship richness of OC-Commerce reports a lower value than OC-Found (the latter is 57.14%) due to the inheritance relationship between the two ontologies. The value for the inheritance richness is close to the one of OC-Found, as they are vertical ontologies. The tree balance of OC-Commerce, as expected, is also higher than the one computed on OC-Found, because some classes introduced in OC-Found are specialized in the context of digital commerce (e.g., *Asset*). Attribute richness and class richness of OC-Commerce are close to the values computed on OC-Found, confirming the verticality of the ontological stack developed at this step.

5.2.2. SPARQL queries

We now discuss the SPARQL queries that implement the competency questions **CC1-CC3** (see Section 4.2) defined for the OC-Commerce ontology. Specifically, the competency questions are implemented as follows.

CC1: Which are the available offerings (including related details)?

The answer to **CC1** is entailed by the SPARQL query QC1.

Query QC1

```
SELECT DISTINCT ?offering ?type ?value ?currency
WHERE {
        ?taskExec a oasis:TaskExecution.
        ?taskExec oasis:hasTaskObject ?taskob.
        ?taskob oasis:refersExactlyTo ?offering.
        ?offering a ?offer.
        FILTER(?offer = occom:Offering)
        FILTER NOT EXISTS { ?offering a occom:DeprecatedOffering.}
        FILTER NOT EXISTS { ?offering a occom:ClosedOffering.}
        FILTER NOT EXISTS { ?offering a occom:RetractedOffering.}
        ?product a ?type.
        FILTER( ?type != owl:NamedIndividual)
        ?priceDetActivity occom:priceDeterminationPerformedOn ?offering.
        ?priceDetActivity occom:hasPriceValue ?price.
        ?price gr:hasCurrencyValue ?value.
        ?price gr:hasCurrency ?currency.
        }
```

CC2: Given an offering *offering*, which is the supply chain related with *offering*?

The answer to **CC2** is entailed by the SPARQL query QC2.

Query QC2

```
Let offering be the offering of which the supply chain should be discovered.
  SELECT DISTINCT ?chainActivity ?type ?agent
  WHERE {
        offering ocfound:hasSupplyChainManagement ?chainManagement.
        ?chainManagement ocfound:hasSupplyChainActivity ?chainActivity.
        ?chainActivity a ?type.
        FILTER( ?type != owl:NamedIndividual)
        ?chainActivity ocfound:supplyChainActivityImplementedBy ?behaviour.
        ?agent oasis:hasBehavior ?behaviour.
         }
```

CC3: Which are the accepted offerings?

Answers to competency question **CC3** are entailed by the SPARQL query QC3.

Query QC3

```
  SELECT ?agent ?offering ?accepted
  WHERE {
        ?agent oasis:performs ?agentExe.
        ?agentExe oasis:hasTaskObject ?taskExe.
        ?agentExe oasis:hasTaskOperator ?operator.
        ?operator oasis:refersExactlyTo oabox:accept.
        ?taskExe oasis:refersExactlyTo ?offering.
        ?offering a ?accepted.
        FILTER( ?accepted = occom:AcceptedOffering)
        }
```

5.3. Evaluation of the OC-Ethereum ontology

Next, we introduce the evaluation methods and related results obtained for the OC-Ethereum ontology. We first present the ontological metrics and then the implementation of the competency questions in the SPARQL query language.

5.3.1. Metrics

As in the case of OC-Found and OC-Commerce, consistency check has been carried out by the reasoners Pellet, HermiT, and FaCT++. Table 5 reports the structural metrics computed on the root of OC-Ethereum, hence excluding the values inherited from OC-Commerce, OC-Found, and BLONDiE, in the first column. In the second column, Table 5 reports the values obtained from the imported ontology BLONDiE.

We recall that BLONDiE provides the description of three blockchains, namely Ethereum, Hyperledger Fabric, and Bitcoin, whereas OC-Ethereum is focused on the former. Thus, there is a notable difference in size between the two ontologies, even though OC-Ethereum provides more classes because of the hierarchies introduced to describe smart contracts and tokens.

The ontological metrics computed by OntoQA on OC-Ethereum are reported in Table 6 and compared with those obtained from BLONDiE. A comparison between the results obtained on OC-Found and on OC-Commerce is in order.

With respect to BLONDiE, the difference of relationship richness in OC-Ethereum is notable because BLONDiE makes large use of data-properties to describe the constitutional elements of the three blockchains introduced. BLONDiE reports a higher inheritance richness with respect to OC-Ethereum, because BLONDiE covers a larger domain. Moreover, the tree balance of BLONDiE is also higher than the one of OC-Ethereum because OC-Ethereum inherits many classes and properties that are not included in the computation of the metric. Class richness of OC-Ethereum remains low (9.09%), as it contains just few individuals, whereas BLONDie records a 0%, since it does

Table 5
Structural metrics on OC-Ethereum

Metric	BLONDiE import	Root of OC-Ethereum
Axiom count	323	149
Logical axiom count	210	65
Declaration axiom count	98	58
Class count	23	45
Object property count	11	11
Data property count	64	3
Subclass axiom count	16	28
Sub-object-property axiom count	0	7
Object property domain axiom count	11	7
Object property range axiom count	11	8
Data property domain axiom count	64	2
Data property range axiom count	64	2
Annotation assertion	15	26

Table 6
Ontological metrics on OC-Ethereum

Evaluation criteria	BLONDiE import	Root of OC-Ethereum	Delta %
Relationship richness	82.41%	29.16%	−182.61%
Inheritance richness	2.28%	1.36%	−67.64%
Tree balance	0.88%	0.83%	−6.02%
Attribute richness	0.30%	0.59%	+49.15%
Class richness	0	9.09%	+INF

not include any individual. Finally, we can appreciate a higher attribute richness in OC-Ethereum with respect to BLONDiE, since smart contracts and tokens are described in more details.

A comparison between OC-Ethereum and the ontologies on the upper layers of the stack, namely OC-Found and OC-Commerce, follows. Relationship richness of OC-Ethereum is close to the value computed on OC-Commerce and OC-Found (that are 40.81% and 57.14%, respectively) confirming that the ontology provides a sufficient balancing between descriptions of properties and class hierarchies. Inheritance and attribute richness are close to those calculated from OC-Commerce and OC-Found, confirming the vertical nature of OC-Ethereum, as also proved by the tree balance value (0.83%). The tree balance of OC-Ethereum, in particular, is also lower than the one computed on OC-Commerce and OC-Found, due to the intrinsic difference of deepness level between some class hierarchies, as for example between the class hierarchy of *EthereumToken* and the class hierarchy of *EthereumTokenPerdurantFeature*. Finally, we notice similar values of class richness among the three ontologies, since only few individuals were introduced coherently with the verticality of their nature.

5.3.2. SPARQL queries

We are now ready to present the implementation of the provided competency questions **CE1-CE4** (see Section 4.3) for the OC-Ethereum ontology in the SPARQL query language.

CE1: Which are the tokens minted and not destroyed, the related asset, minter, and current owner?

The answer to **CE1** is entailed by the SPARQL query QE1.

CE2: Which are the block and the transaction hash that mint a given token?

The answer to **CE2** is entailed by the SPARQL query QE2.

CE3: Which are the smart contracts that emit tokens related to a specific type of asset?

The answer to **CE3** is entailed by the SPARQL query QE3.

Query QE1

```
SELECT ?agent ?token ?tokentype ?asset ?owner
WHERE {
        ?agent oasis:performs ?agentExe.
        ?agentExe oasis:hasTaskObject ?taskExe.
        ?agentExe oasis:hasTaskOperator ?operator.
        ?operator oasis:refersExactlyTo oabox:mint.
        ?taskExe oasis:refersExactlyTo ?token.
        ?operationOn a ?tokentype.
        ?tokenType rdfs:subClassOf ocether:EthereumTokenERC721.
        FILTER( ?tokentype != owl:NamedIndividual)
        FILTER NOT EXISTS { ?operationOn a ocether:BurnedEthereumToken }
        ?asset ocether:isDescribedByEthereumToken ?operationOn.
        ?token ocether:hasEthereumTokenPerdurantFeature ?feature.
        ?feature a ?ownerFeature.
        FILTER(?ownerFeature = ocether:EthereumWalletOwnerPerdurantFeature)
        FILTER NOT EXISTS {?feature a ocether:DeprecatedEthereumTokenPerdurantFeature.}
        ?feature ocether:isInTheWalletOf ?owner.
        }
```

Query QE2

Let *token* be the token of which block number and transaction hash should be discovered

```
SELECT ?blockNumber ?hash
WHERE {
        ?block blon:heightBlock ?blockNumber.
        ?block blon:hasEthereumPayloadBlock ?payload.
        ?payload blon:hasEthereumTransactionPayload ?transaction.
        ?transaction blon:recipientEthereumTransaction ?hash.
        ?transaction ocether:introducesEthereumSmartContractExecution ?action.
        ?action oasis:consistsOfGoalExecution ?goal.
        ?goal oasis:consistsOfTaskExecution ?agentExe.
        ?agent oasis:performs ?agentExe.
        ?agentExe oasis:hasTaskObject ?taskExe.
        ?agentExe oasis:hasTaskOperator ?operator.
        ?operator oasis:refersExactlyTo oabox:mint.
        ?taskExe oasis:refersExactlyTo token.
        }
```

Query QE3

Let *assetType* the type of asset related with the smart contract to be discovered

```
SELECT DISTINCT ?agent ?hash ?address
WHERE {
        ?agent oasis:performs ?agentExe.
        ?agentExe oasis:hasTaskObject ?taskExe.
        ?agentExe oasis:hasTaskOperator ?operator.
        ?operator oasis:refersExactlyTo oabox:mint.
        ?taskExe oasis:refersExactlyTo ?token.
        ?asset ocether:isDescribedByEthereumToken ?token.
         ?asset a assetType.
         ?block blon:heightBlock ?blockNumber.
         ?block blon:hasEthereumPayloadBlock ?payload.
         ?payload blon:hasEthereumTransactionPayload ?transaction.
         ?transaction blon:recipientEthereumTransaction ?hash.
         ?transaction ocether:introducesEthereumSmartContractAgent ?agent.
         ?transaction blon:to ?address.
         }
```

CE4: Which is the number of tokens and the type of the related assets owned by wallets?

The answer to **CE4** is entailed by the SPARQL query QE4.

Query QE4

```
SELECT ?owner (COUNT(?operationOn) as ?tokenCounter) ?assetType
WHERE {
        ?asset a ?assetType.
        FILTER(?assetType != owl:NamedIndividual).
        ?asset ocether:isDescribedByEthereumToken ?operationOn.
        ?token ocether:hasEthereumTokenPerdurantFeature ?feature.
        ?feature a ?ownerFeature.
        FILTER(?ownerFeature = ocether:EthereumWalletOwnerPerdurantFeature)
        FILTER NOT EXISTS {?feature a ocether:DeprecatedEthereumTokenPerdurantFeature.}
         ?feature ocether:isInTheWalletOf ?owner.
         }
         GROUP BY ?assetType ?owner
```

6. A real-world case study: The iExec marketplace

In this section, we show how to apply our ontological stack to model a real world use case, namely the iExec marketplace [72]. iExec is one of the main founders of the ONTOCHAIN project and is present in the blockchain context since 2007 thanks to its first decentralized marketplace for computing assets, the iExec marketplace. The *iExec marketplace* connects buyers with sellers of cloud resources, building an ecosystem of decentralized and autonomous, privacy-preserving applications that are also scalable, secure, with a simplified mechanism to access services, datasets, and computing resources. The intangibility of the assets traded in the iExec marketplace and their particular supply chains are challenging for the ontological stack, since it represents a proving ground for the generality of the ontological model that have been applied to physical resources traded through common supply chains so far.

We first provide an overview of the basic concepts and functioning of the iExec marketplace, then we illustrate in detail how its main features are represented through the ontological stack.

6.1. Outlining the iExec marketplace

In the iExec marketplace, cloud resources are of three main types, namely *applications*, *datasets*, and *computing resources*.

Applications are standalone computer programs that can be downloaded and executed in a decentralized way by a remote machine as *tasks*. Tasks admit execution parameters and input files, accessing data available on the iExec marketplace as *datasets*, and producing files containing the results of the computation. The computational resources required to carry out application executions are provided by *workers*, i.e., machines on the iExec marketplace that download and execute applications (according to iExec).

Application providers, namely actors providing applications via the iExec marketplace, can define commercial conditions (in particular, usage fees) for the execution of their applications. Such commercial conditions are encoded into offerings, called *app orders*, available through the iExec marketplace.

The structure of app orders is described in Fig. 32, where:

- `app` is a unique identifier for the application, namely the address of the smart contract managing the decentralized application;
- `appprice` is the price for a single execution of the app;
- `volume` is the maximum number of executions of the app referred in the order;
- `tags` are application-specific additional computational requirements (for example, execution in a trusted environment);
- `datasetrestrict` and `workerpoolrestrict` are optional conditions that restrict executions to specific datasets and/or to the specified worker pool;
- `requesterrestrict` provides additional restrictions of execution requests, to be described later on;
- `salt` is a random value to ensure order uniqueness;
- `sign` is the EIP712 cryptographic signature [50] of the order.

```
struct AppOrder
{
  address app;
  uint256 appprice;
  uint256 volume;
  uint256 tag;
  address datasetrestrict;
  address workerpoolrestrict;
  address requesterrestrict;
  bytes32 salt;
  bytes   sign;
}
```

Fig. 32. Structure of the app order in the iExec marketplace.

```
struct DatasetOrder
{
  address dataset;
  uint256 datasetprice;
  uint256 volume;
  uint256 tag;
  address apprestrict;
  address workerpoolrestrict;
  address requesterrestrict;
  bytes32 salt;
  bytes   sign;
}
```

Fig. 33. Dataset order's structure in the iExec marketplace.

Dataset providers, namely actors providing datasets via the iExec marketplace, publish *dataset orders* describing the commercial conditions regarding the datasets to be used in task executions. The mechanism of publishing dataset orders is analogous to that of app orders. Even the structure of dataset orders (see Fig. 33) is similar to the structure of app order, and hence their semantics fields can be easily deduced from the analogous app order fields.

Workers are grouped into *worker pools*, each one associated with a *worker pool manager*. Any application execution is performed by a worker pool by following the so-called *Proof of Contribution Protocol* (*PoCo*) [40,41]. In this phase, workers can possibly retrieve the dataset required for the execution. Each execution is started by the worker pool manager, which acts as *scheduler* during the corresponding *PoCo* protocol run. Further details about how worker pools perform application executions can be found in [72].

Commercial conditions about the usage of computational resources are defined and published by the worker pool manager on the iExec market place as *worker pool orders* (see Fig. 34). The structure of worker pool orders is similar to that of app orders, except for the fields *category* and *trust* that are not available in app orders. Specifically,

- `category` describes the *size* of the computation in terms of maximum task execution time, ranging from XS for 4 minutes through to XL for 10 hours;
- `trust` is a confidence level for accepting contributions of workers in the PoCo execution.

Users who need to perform computation are called *requester*. Requesters retrieve app orders, worker pool orders and dataset orders from the iExec marketplace or from other sources. However, such orders are signed by their providers, so that they can be used in disputes. Once the requester has acquired an app order, a suitable worker pool order and, possibly, a suitable dataset order, he creates a *request order* (see Fig. 35) satisfying all the restrictions specified. The request order, together with the app and dataset order, is signed by the requester and submitted to the *iExec clerk* smart contract. The iExec clerk verifies the signatures and the satisfiability of the orders, and writes the agreement on the blockchain. The PoCo protocol is then started in order to perform the execution of the requested application (see [42] for details).

```
struct WorkerpoolOrder
{
  address workerpool;
  uint256 workerpoolprice;
  uint256 volume;
  uint256 tag;
  uint256 category;
  uint256 trust;
  address apprestrict;
  address datasetrestrict;
  address requesterrestrict;
  bytes32 salt;
  bytes   sign;
}
```

Fig. 34. Worker pool order's structure in the iExec marketplace.

```
struct RequestOrder
{
  address app;
  uint256 appmaxprice;
  address dataset;
  uint256 datasetmaxprice;
  address workerpool;
  uint256 workerpoolmaxprice;
  address requester;
  uint256 volume;
  uint256 tag;
  uint256 category;
  uint256 trust;
  address beneficiary;
  address callback;
  string  params;
  bytes32 salt;
  bytes   sign;
}
```

Fig. 35. Request order's structure in the iExec marketplace.

6.2. *Representing the iExec marketplace*

We now describe how the iExec marketplace can be modelled through the proposed ontological stack. The proposed encoding focuses on offerings of assets provided by the iExec marketplace to ease the discovery of services and commercial available conditions.

We first introduce novel classes, properties, and individuals that represent the structures of the iExec marketplace. For those entities, the namespace ixec: http://ontology.iex.ec is defined.

In order to represent items traded through the iExec marketplace, namely *executions* of iExec applications, we recall that they are characterized by:

- the application that will be executed,
- the worker pool, with the corresponding worker pool manager that has in charge the execution and, possibly,
- a dataset used by the application.

As a first step, we notice that usages of applications and datasets in program execution are *assets* that can be traded: hence, they can be represented as instances of two novel subclasses of *oasis:DigitalServiceAsset*, namely *iexec:Application* and *iexec:Dataset*, respectively. Asset identifiers are encoded in individual IRIs of suitable naming schemes defined by the asset providers, the latter modelled as *oasis:Agent* instances.

As described above, executions are actually performed by *worker pools*. Worker pools are organizations of agents, coordinated by worker pool managers. For worker pools and worker pool managers, we introduce the classes

iexec:WorkerPool and *iexec:WorkerPoolManager*. Each worker pool is connected to its manager via the property *iexec:hasWorkerPoolManager*.

As illustrated in Fig. 35, executions have also some optional fields represented by means of suitable properties, grouped as *optional execution properties*. Specifically, the model is designed as follows:

- `tag` is represented by the property *iexec:hasTag*. Application providers and worker pools can provide ad-hoc taxonomies of tags for their purposes.
- `category`, which indicates the maximum elapsed time for the execution, is represented by the object-property *iexec:hasCategory*.
- `trust`, which is an integer value indicating a confidence level for the computation result, is represented by the *iexec:hasTrust* data-type property.
- `params`, which indicates documents publicly available on the web used in the execution as input parameters for the application, is represented by the object-property *iexec:hasParam*.

We are ready to describe how to publish *offerings* to sell iExec assets through the ontological stack. For *AppOrder*, *DatasetOrder*, and *WorkerPoolOrder*, respectively illustrated in Figs 32, 33, and 34, we provide three subclasses of the class *Offering* of OC-Commerce, namely *iexec:AppOffering*, *iexec:DatasetOffering*, and *iexec:WorkerPoolOffering*, respectively. Converting iexec orders into corresponding and compliant individuals of the ontological stack is straightforward:

- the order identifier must be included in the offering individual IRI, given a IRI scheme provided for this purpose;
- prices are provided as instances of the class *UnitPriceSpecification* in OC-Commerce;
- contents of the `volume` fields correspond to the notion of *eligible quantity* in OC-Commerce;
- `apprestrict`, `datasetrestrict`, and `requestrestrict` are modelled by the properties *hasAppRestrict*, *hasDatasetRestrict*, and *hasRequestRestrict*, respectively;
- optional properties are modelled with the *optional execution properties* introduced before;
- `salt` and `sign` can be retrieved, if needed, by directly accessing the original order from which the offering was created.

Applications and datasets are particular assets that cannot be *released* or *delivered*, but can be used in the context of an execution and consumed by means of the related worker pool offerings. Thus, supply chains are required only for worker pool offerings that embed applications and datasets offerings. As a consequence, classes representing the app and dataset offerings, i.e., *iexec:AppOffering* and *iexec:DatasetOffering*, respectively, are not associated with supply chains. Instead, a supply chain is specified for the class representing the worker pool offering, namely *iexec:WorkerPoolOffering*, related with the former offerings, as described in Fig. 36. The supply chain of the worker pool offering will be explained later on. The worker pool offering is accepted by clients through a client's *accept* behaviour that produces as output an instance of the class *iexec:RequestOrder*, representing a request order generated by requester from app, dataset, and worker pool offerings. The representation of a request order object through the corresponding *iexec:RequestOrder* individual is carried on similarly to order objects, except for the fields `beneficiary` and `callback`.

The iExec clerk smart contract verifies the request order and signs a corresponding *deal*, then stores it on the blockchain. For this reason, instances of the class *iexec:RequestOrder* are used by the iExec clerk agent as described in Fig. 37. For the iExec clerk agent, we define a suitable behaviour, namely *iexec:establishDeal*, admitting the following parameters:

- an *iexec:AppOffering*, corresponding to an app order,
- optionally, an *iexec:DatasetOffering*, corresponding to a dataset order,
- an *iexec:WorkerPoolOffering*, corresponding to a worker pool order,
- an *iexec:RequestOrder*, corresponding to the execution request.

The iExec clerk behaviour provides the action *iexec:validate*, so as to check that the set of orders passed as parameters is valid and to produce the iExec deal, represented by the class *iexec:iExecDeal*, as output.

The execution of tasks is actually performed by the worker pool agent by means of the iExec deal. The worker pool agent, whose behaviour is described in Fig. 38, accepts as input the iExec deal and computes the corresponding iExec task.

The iExec clerk agent and the worker pool agent are exploited to implement the supply chain of worker pool orders. Specifically, the *OfferingProof* supply chain of the worker pool order is related with the behaviour of the iExec clerk agent (see Fig. 37), whereas the release supply chain consists of an activity carried on by the behaviour of the worker pool agent (Fig. 38). Finally, the payment supply chain consists of an activity implementing the payment using the iExec digital currency *RLC* manager by the *iEx.ec_Network_Token* smart contract.

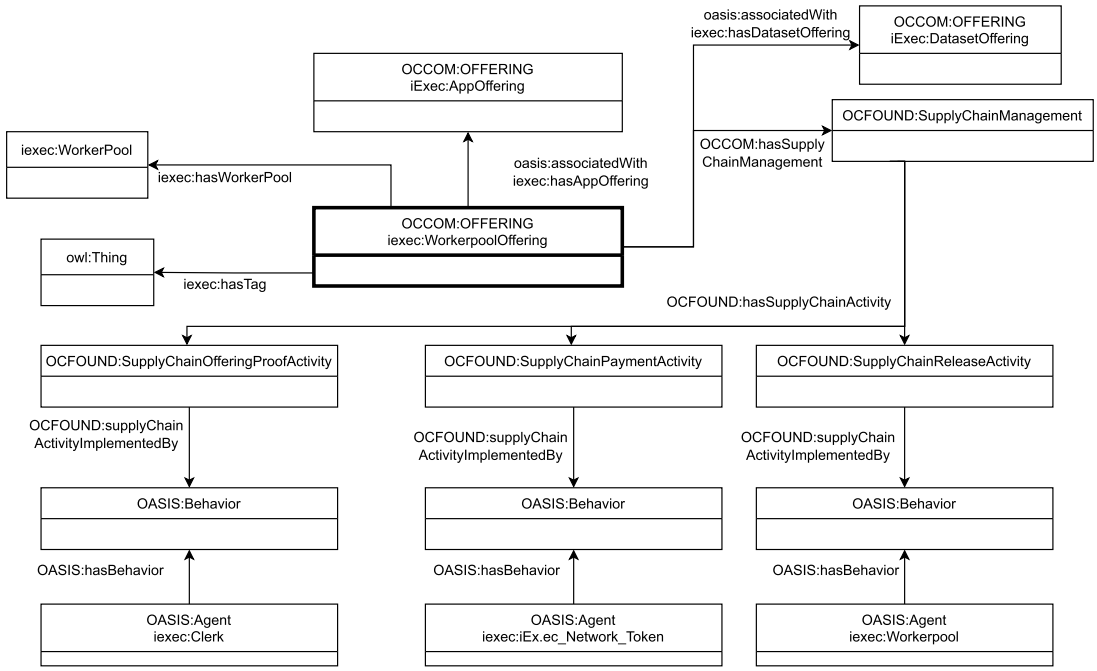

Fig. 36. UML diagram representing the worker pool offering.

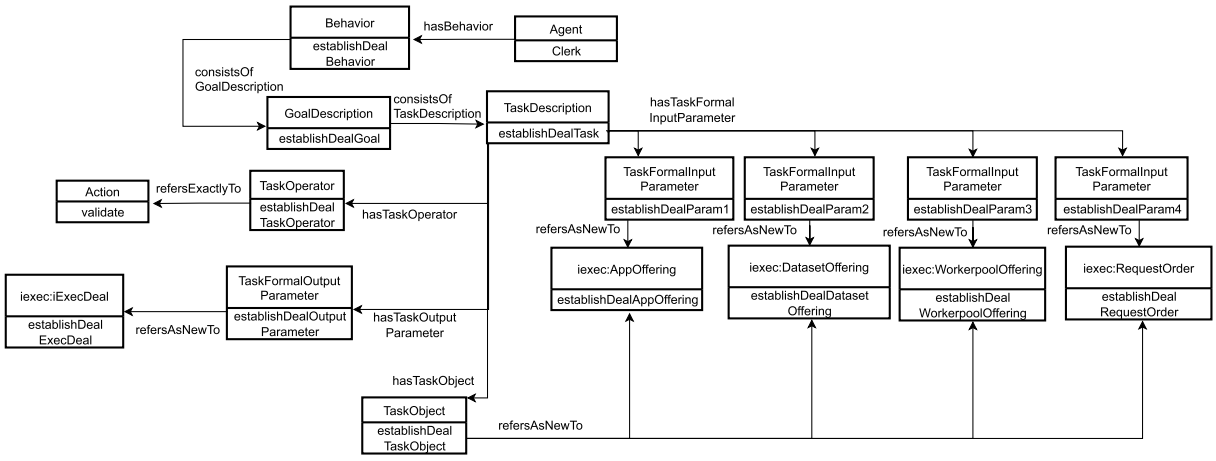

Fig. 37. UML diagram representing the iExec clerk agent.

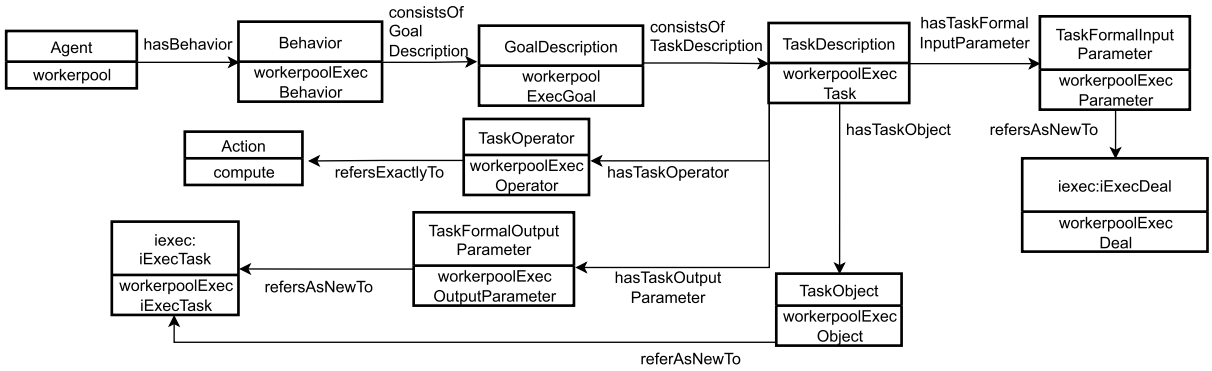

Fig. 38. UML diagram representing the iExec worker pool agent.

7. Conclusions

Coherently with the goals stated in Section 1.1, this article sought a formal semantic representation that captures commercial activities carried out through smart contracts on the Ethereum blockchain. Not only is such a representation machine readable, but it also supports the interlinking with other out-of-chain information and, at a different level, with formal reasoning. A semantic behaviouristic conception of blockchains enables the automatic discovery of smart contracts, the interconnection of services running on different blockchains (i.e., cross-chain integration), and the integration between on-chain and off-chain services. Such features turn out to be more interesting when smart contracts are implemented as mechanisms for generating and exchanging tokens. A desirable means for token exchange systems is a precise and intelligent query mechanism capable of determining what, when, and how certain assets are generated, exchanged or destroyed. For example, intelligent systems may be aware of the activation of smart contracts for generating tokens with specific characteristics, e.g., the type of exchanged asset, the exchange of particular tokens at certain conditions, or their destruction. More in general, intelligent systems may be aware of the activation of smart contracts and of all the related activities over the blockchain.

This paper made the contributions outlined in Section 1.3. Specifically, it unfolded a behaviouristic approach for semantically representing blockchain-based e-commerce that uses the token generation and exchange mechanisms as decentralized and public proofs. Due to the need to account for both the distinctive blockchain features and for the typical commercial activities, the epistemological process results in an ontological stack of height three. The bottom ontology, called OC-Found, extends OASIS to describe digital identities and supply chains. The second ontology, that is OC-Commerce, captures the model of commercial offerings provided by GoodRelations and extends it with the representation system of agents and their commitments, also by including auctions. The stack culminates with the ontology called OC-Ethereum, which adopts the BLONDiE ontology for representing the constitutional elements of the Ethereum blockchain, but extending it with the modelling of Ethereum tokens and with the operational semantics of smart contracts.

However, the results presented in this article are not conclusive. A semantic search engine is under development, currently standing at TRL3. Other future work is clearly defined. OC-Found will include mechanisms for realizing digital identities for people and applications selected by the consortium. The OC-Commerce ontology will be extended to be independent of GoodRelations with new modelling approaches for bargaining offerings. OC-Ethereum will be extended with new blockchains and behaviours for managing non-standard and user-defined tokens. As in the same line of OC-Ethereum, novel ontologies for many other Turing-complete blockchains will be introduced in the stack together with the most widespread cryptocurrencies. Additionally, novel ontologies for other domains, such as health, industry, smart cities and government, will be part of the ontological stack. Another challenge is to enhance iExec offerings and transactions, as soon as they become available, by their semantic representations, for example in the style of the model presented in this paper. The experience we have gained thus far as well as the overall flexibility of our approach makes us optimistic on the successful pursuance of these research and development

directions. Finally, representing the ontological stack through the decidable fragments of set-theory as in [20,24–30] is one of the future works.

Acknowledgements

This work was supported by the project "FuSeCar" funded by the MUR Progetti di Ricerca di Rilevante Interesse Nazionale (PRIN) Bando 2022 – grant 2022W3EPEP – CUP: E53D23008210006.

The work of POC4Commerce has been supported by the ONTOCHAIN NGI European project grant agreement no. 957338. We are thankful to the ONTOCHAIN Consortium who mentored and assisted the research.

Domenico Cantone acknowledges partial support from project "STORAGE–Università degli Studi di Catania, Piano della Ricerca 2020/2022, Linea di intervento 2" and from the "Naples Dante Project" funded by the MUR Progetti di Ricerca di Rilevante Interesse Nazionale (PRIN) Bando 2022, grant 2022ZJ4N9E. Domenico Cantone is member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

Gianpietro Castiglione acknowledges a studentship by Intrapresa S.r.l. and Italian "Ministero dell'Università e della Ricerca" (D.M. n. 352/2022).

Daniele Francesco Santamaria acknowledges the Research Program PIAno di inCEntivi per la Ricerca di Ateneo 2020/2022 – Linea di Intervento 3 "Starting Grant" – University of Catania.

References

[1] S.N. Akshay Uttama Nambi, C. Sarkar, R.V. Prasad and A. Rahim, A unified semantic knowledge base for IoT, in: *World Forum on Internet of Things (WF-IoT)*, IEEE, 2014.

[2] B. Alves, M.I. Schumacher, F. Cretton, A.L. Calvé, G. Cherix, D. Werlen, C. Gapany, B. Baeryswil, D. Gerber and P. Cloux, Fairtrace – a Semantic-Web oriented traceability solution applied to the textile traceability, in: *ICEIS*, 2013.

[3] J. Ashraf, O.K. Hussain and F.K. Hussain, Empirical analysis of domain ontology usage on the web: ECommerce domain in focus, *Concurrency and Computation Practice and Experience* **26**(5) (2014), 1157–1184. doi:10.1002/cpe.3089.

[4] G. Bajaj, R. Agarwal, P. Singh, N. Georgantas and V. Issarny, A study of existing ontologies in the IoT-domain, hal-01556256, 2017.

[5] H. Baqa, N. Truong, N. Crespi, G.M. Lee and F. Le Gall, Semantic smart contracts for blockchain-based services in the Internet of things, in: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019, pp. 1–5. doi:10.1109/NCA.2019.8935016.

[6] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo and D.F. Santamaria, 2021, POC4COMMERCE – practical ONTOCHAIN for e-commerce – project page, https://github.com/dfsantamaria/POC4COMMERCE.

[7] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo and D.F. Santamaria, Semantic representation as a key enabler for blockchain-based commerce, in: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* LNCS, Vol. 13072, 2021, pp. 191–198. doi:10.1007/978-3-030-92916-9_17.

[8] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo and D.F. Santamaria, Blockchains through ontologies: The case study of the Ethereum ERC721 standard in oasis, *Studies in Computational Intelligence* **1026** (2022), 249–259. doi:10.1007/978-3-030-96627-0_23.

[9] G. Bella, D. Cantone, C. Longo, M. Nicolosi-Asmundo and D.F. Santamaria, POC4COMMERCE – making ONTOCHAIN practical for e-commerce, 2020, https://ontochain.ngi.eu/content/poc4commerce-practical-ontochain-commerce.

[10] G. Bella, D. Cantone, C.F. Longo, M. Nicolosi-Asmundo and D.F. Santamaria, The ontology for agents, systems and integration of services: OASIS version 2, *Intelligenza Artificiale* **17**(1) (2023), 51–62. doi:10.3233/IA-230002.

[11] G. Bella, D. Cantone, M. Nicolosi Asmundo and D.F. Santamaria, The ontology for agents, systems and integration of services: Recent advancements of OASIS, in: *23rd Workshop from Objects to Agents, WOA 2022*, Genova, 1–3 September 2022, Vol. 3261, CEUR-WS, 2022, pp. 176–193, ISSN 16130073.

[12] G. Bella, G. Castiglione and D.F. Santamaria, A behaviouristic approach to representing processes and procedures in the OASIS 2 ontology, in: *Proceedings of the Joint Ontology Workshops 2023, Episode IX: The Quebec Summer of Ontology, Co-Located with the 13th International Conference on Formal Ontology in Information Systems (FOIS 2023)*, Sherbrooke, Québec, Canada, July 19–20, 2023, CEUR Workshop Proceedings, Vol. 3637, 2023, pp. 1–17.

[13] G. Bella, G. Castiglione and D.F. Santamaria, An ontological approach to compliance verification of the NIS 2 directive, in: *Proceedings of the Joint Ontology Workshops 2023, Episode IX: The Quebec Summer of Ontology, Co-Located with the 13th International Conference on Formal Ontology in Information Systems (FOIS 2023)*, Sherbrooke, Québec, Canada, July 19–20, 2023, CEUR Workshop Proceedings, Vol. 3637, 2023, pp. 1–12.

[14] G. Bella, G. Castiglione and D.F. Santamaria, An automated method for the ontological representation of security directives, in: *Proceedings of the Joint Ontology Workshops 2023, Episode IX: The Quebec Summer of Ontology, Co-Located with the 13th International Conference on Formal Ontology in Information Systems (FOIS 2023)*, Sherbrooke, Québec, Canada, July 19–20, 2023, CEUR Workshop Proceedings, Vol. 3637, 2023, pp. 1–17.

[15] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi and 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.

[16] M.E. Bratman, *Intentions, Plans and Practical Reason*, Harvard University Press, 1987.

[17] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, Tropos: An agent-oriented software development methodology, *Autonomous Agents Multi Agent Systems* **8**(3) (2004), 203–236. doi:10.1023/B:AGNT.0000018806.20944.ef.

[18] D. Calvanese, G. De Giacomo, D. Lembo, M. Montali and A. Santoso, Ontology-based governance of data-aware processes, in: *Web Reasoning and Rule Systems*, M. Krötzsch and U. Straccia, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 25–41. doi:10.1007/978-3-642-33203-6_4.

[19] J. Cano-Benito, A. Cimmino and R. García-Castro, Towards blockchain and Semantic Web, in: *Business Information Systems Workshops*, W. Abramowicz and R. Corchuelo, eds, Springer International Publishing, Cham, 2019, pp. 220–231. doi:10.1007/978-3-030-36691-9_19.

[20] D. Cantone, C. Longo, M. Nicolosi-Asmundo and D.F. Santamaria, Web ontology representation and reasoning via fragments of set theory, in: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9209 2015, pp. 61–76. ISBN 978-331922001-7. doi:10.1007/978-3-319-22002-4_6.

[21] D. Cantone, C.F. Longo, M. Nicolosi Asmundo, D.F. Santamaria and C. Santoro, Ontological smart contracts in OASIS: Ontology for agents, systems, and integration of services, *Studies in Computational Intelligence* **1026** (2022), 237–247. doi:10.1007/978-3-030-96627-0_22.

[22] D. Cantone, C.F. Longo, M. Nicolosi-Asmundo, D.F. Santamaria and C. Santoro, Towards an ontology-based framework for a behavior-oriented integration of the IoT, in: *20th Workshop from Objects to Agents, WOA 2019*, Parma, 26–28 June 2019, Vol. 2404, CEUR-WS, 2019, pp. 119–126, ISSN 16130073.

[23] D. Cantone, C.F. Longo, M. Nicolosi-Asmundo, D.F. Santamaria and C. Santoro, OASIS-Abox ontology, https://github.com/dfsantamaria/OASIS.

[24] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, Conjunctive query answering via a fragment of set theory, in: *17th Italian Conference on Theoretical Computer Science, ICTCS 2016*, Lecce, 7–9 September 2016, Vol. 1720, CEUR-WS, 2016, pp. 23–35, ISSN 16130073.

[25] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, A C++ reasoner for the description logic DLD4,×, in: *Joint 18th Italian Conference on Theoretical Computer Science and the 32nd Italian Conference on Computational Logic, ICTCS 2017 and CILC 2017*, Naples, 26–28 September 2017, Vol. 1949, CEUR-WS, 2017, pp. 276–280, ISSN 16130073.

[26] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, A set-theoretic approach to ABox reasoning services, in: *EKAW*, S. Costantini, E. Franconi, W. Van Woensel, R. Kontchakov, F. Sadri, and D. Roman, eds, Lecture Notes in Computer Science, Vol. 10364, Springer, 2017, pp. 87–102. doi:10.1007/978-3-319-61252-2_7.

[27] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, A set-based reasoner for the description logic DL4,xD, in: *3rd International Workshop on Sets and Tools (SETS18)*, Southamptom, UK, June 5, 2018, CEUR Workshop Proceedings, Vol. 2199, 2018, pp. 52–66.

[28] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, An optimized KE-tableau-based system for reasoning in the description logic DL4,×D, in: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* LNCS, Vol. 11092, 2018, pp. 239–247. doi:10.1007/978-3-319-99906-7_16.

[29] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, A set-theoretic approach to reasoning services for the description logic DL4xD, *Fundamenta Informaticae* **176**(3–4) (2020), 349–384. doi:10.3233/FI-2020-1977.

[30] D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, An improved set-based reasoner for the description logic DL4xD, *Fundamenta Informaticae* **178**(4) (2021), 315–346. doi:10.3233/FI-2021-2009.

[31] D. Cantone, D.F. Santamaria and V. Spinello, A tool to easing the configuration and deploying process of hyperledger fabric, in: *Proceedings of the 3rd International Workshop on Blockchain for Trusted Data Sharing – B4TDS*, Ascoli Piceno, Italy, September 15, 2023, Vol. 3514, CEUR-WS, 2023, pp. 224–234, ISSN 16130073.

[32] G. Castiglione, G. Bella and D.F. Santamaria, Towards grammatical tagging for the legal language of cybersecurity, in: *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES '23*, Association for Computing Machinery, New York, NY, USA, 2023. ISBN 9798400707728. doi:10.1145/3600160.3605069.

[33] B. Charron, Y. Hirate, D. Purcell and M. Rezk, Extracting semantic information for e-commerce, in: *The Semantic Web – ISWC 2016*, P. Groth, E. Simperl, A. Gray, M. Sabou Martaand Krötzsch, F. Lecue, F. Flöck and Y. Gil, eds, Springer International Publishing, Cham, 2016, pp. 273–290. doi:10.1007/978-3-319-46547-0_27.

[34] K. Christidis and M. Devetsikiotis, Blockchains and smart contracts for the Internet of things, *IEEE Access* **4** (2016), 2292–2303. doi:10.1109/ACCESS.2016.2566339.

[35] A. Ciortea, S. Mayer and 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.

[36] O. Corcho, A. Gomez-Perez and F.D. Informatica, Solving integration problems of e-commerce standards and initiatives through ontological mappings, *International Journal of Intelligent Systems* **16**(16) (2001), 2001.

[37] O. Corcho, A. Gomez-Perez, A. Leger, C. Rey and F. Toumani, An ontology-based mediation architecture for e-commerce applications, in: *Intelligent Information Processing and Web Mining*, M.A. Kłopotek, S.T. Wierzchoń and K. Trojanowski, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 477–486. doi:10.1007/978-3-540-36562-4_51.
[38] C. Corea and P. Delfmann, *Detecting Compliance with Business Rules in Ontology-Based Process Modeling, Wirtschaftsinformatik und Angewandte Informatik*, 2017.
[39] M. Cossentino, M. Gleizes, A. Molesini and A. Omicini, Processes engineering and AOSE, in: *Agent-Oriented Software Engineering X*, M.-P. Gleizes and J.J. Gomez-Sanz, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 191–212.
[40] H. Croubois, PoCo series #1 – about trust and agents incentives, 2017, https://medium.com/iex-ec/about-trust-and-agents-incentives-4651c138974c.
[41] H. Croubois, PoCo series #3 – protocol update, 2018, https://medium.com/iex-ec/poco-series-3-poco-protocole-update-a2c8f8f30126.
[42] H. Croubois, PoCo series #5 – open decentralized brokering on the iExec platform, 2018, https://medium.com/iex-ec/poco-series-5-open-decentralized-brokering-on-the-iexec-platform-67b266e330d8.
[43] K.H. Dam and M. Winikoff, Comparing agent-oriented methodologies, in: *Agent-Oriented Information Systems*, P. Giorgini, B. Henderson-Sellers and M. Winikoff, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 78–93. doi:10.1007/978-3-540-25943-5_6.
[44] J. de Kruijff and H. Weigand, Understanding the blockchain using enterprise ontology, in: *CAiSE*, 2017.
[45] S. De Martino, M. Nicolosi-Asmundo, S.A. Rizzo and D.F. Santamaria, Modeling the video game environment: The VideOWL ontology, in: *24th Workshop from Objects to Agents, WOA 2023*, Roma, 6–8 November 2023, CEUR Workshop Proceedings, Vol. 3579, 2023, pp. 191–205.
[46] C. Di Ciccio, A. Cecconi, J. Mendling, D. Felix, D. Haas, D. Lilek, F. Riel, A. Rumpl and P. Uhlig, Blockchain-based traceability of inter-organisational business processes, in: *Business Modeling and Software Design*, B. Shishkov, ed., Springer International Publishing, Cham, 2018, pp. 56–68. ISBN 978-3-319-94214-8.
[47] Electronic Commerce Code Management Association (ECCMA), ECCMA Data Requirements Registry (eDRR), 2000, https://eccma.org/resources/ (last visit 01/12/21).
[48] M.D. English, S. Auer and J. Domingue, *Block Chain Technologies & the Semantic Web: A Framework for Symbiotic Development*, 2015.
[49] M. Esteva, Electronic institutions: From specification to development, IIIA monograph series, PhD thesis, 19, 2003.
[50] Ethereum, EIP-712 protocol 2018, https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md (last visit 01/12/21).
[51] L. Fichera, F. Messina, G. Pappalardo and C. Santoro, A python framework for programming autonomous robots using a declarative approach, *Science of Computer Programming* **139** (2017), 36–55. doi:10.1016/j.scico.2017.01.003.
[52] H.G. Fill, Applying the concept of knowledge blockchains to ontologies, in: *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*, 2019.
[53] N. Fornara and M. Colombetti, A commitment-based approach to agent communication, *Applied Artificial Intelligence* (2004), 853–866. doi:10.1080/08839510490509054.
[54] A. Freitas, R.H. Bordini and R. Vieira, Model-driven engineering of multi-agent systems based on ontologies, *Applied Ontology* **12** (2017), 157–188. doi:10.3233/AO-170182.
[55] A. Freitas, A. Panisson, L. Hilgert, F. Meneguzzi, R. Vieira and R. Bordini, Applying ontologies to the development and execution of multi-agent systems, *Web Intelligence* **15** (2017), 291–302. doi:10.3233/WEB-170366.
[56] D.M. Fritzsche, M. Grüninger, K. Baclawski, M. Bennett, G. Berg-Cross, T. Schneider, R.D. Sriram, M. Underwood and A. Westerinen, Ontology summit 2016 communique: Ontologies within semantic interoperability ecosystems, *Applied Ontology* **12**(2) (2017), 91–111. doi:10.3233/AO-170181.
[57] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider, Sweetening ontologies with DOLCE, in: *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. 1–4, http://link.springer.de/link/service/series/0558/bibs/2473/24730166.htm. ISBN 978-3-540-45810-4. doi:10.1007/3-540-45810-7_18.
[58] F. García-Sánchez, J.T. Fernández-Breis, R. Valencia-García, J.M. Gómez and R. Martínez-Béjar, Combining Semantic Web technologies with multi-agent systems for integrated access to biological resources, *Journal of Biomedical Informatics* **41**(5) (2008), 848–859.
[59] G.L. Geerts and D.E. O'Leary, A supply chain of things: The EAGLET ontology for highly visible supply chains, *Decision Support Systems* **63** (2014), 3–22. doi:10.1016/j.dss.2013.09.007.
[60] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang, HermiT: An OWL 2 reasoner, *Journal of Automated Reasoning* **53**(3) (2014), 245–269. doi:10.1007/s10817-014-9305-1.
[61] N. Goldmann, E-commerce, *Journal of Internet Banking and Commerce* **26**(3) (2021), 286–295.
[62] G. Greco, A. Guzzo, L. Pontieri and D. Saccà, An ontology-driven process modeling framework, in: *Database and Expert Systems Applications*, F. Galindo, M. Takizawa and R. Traunmüller, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 13–23. doi:10.1007/978-3-540-30075-5_2.
[63] T. Grubic and I.-S. Fan, Supply chain ontology: Review, analysis and synthesis, *Computers in Industry* **61**(8) (2010), 776–786. doi:10.1016/j.compind.2010.05.006.
[64] GS1, GS1 initiative, 1971, https://www.gs1.org (last visit 01/12/21).
[65] R. Guha, R. McCool and E. Miller, Semantic search, in: *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, ACM, New York, NY, USA, 2003, pp. 700–709, http://portal.acm.org/citation.cfm?doid=775152.775250. ISBN 1-58113-680-3. doi:10.1145/775152.775250.

[66] M. Hadzic, E. Chang and P. Wongthongtham, *Ontology-Based Multi-Agent Systems*, Springer Publishing Company, Incorporated, 2014, 9783642425493. ISBN 3642425496.
[67] J. Hendler, Agents and the Semantic Web, *IEEE Intelligent Systems* **16**(2) (2001), 30–37. doi:10.1109/5254.920597.
[68] M. Hepp, Products and services ontologies: A methodology for deriving OWL ontologies from industrial categorization standards, *Int. J. Semantic Web Inf. Syst.* **2**(1) (2006), 72–99, http://dblp.uni-trier.de/db/journals/ijswis/ijswis2.html#Hepp06. doi:10.4018/jswis.2006010103.
[69] M. Hepp, GoodRelations: An ontology for describing products and services offers on the web, in: *EKAW*, A. Gangemi and J. Euzenat, eds, Lecture Notes in Computer Science, Vol. 5268, Springer, 2008, pp. 329–346, http://dblp.uni-trier.de/db/conf/ekaw/ekaw2008.html#Hepp08. ISBN 978-3-540-87695-3.
[70] M. Hepp, J. Leukel and V. Schmitz, A quantitative analysis of eClass, UNSPSC, eOTD, and RNTD: Content, coverage, and maintenance, in: *2005 IEEE International Conference on e-Business Engineering (ICEBE 2005)*, 18–21 October 2005, F.C.M. Lau, H. Lei, X. Meng and M. Wang, eds, IEEE Computer Society, Beijing, China, 2005, pp. 572–581. doi:10.1109/ICEBE.2005.15.
[71] S. Huan, S. Sheoran and G. Wang, A review and analysis of Supply Chain Operations Reference (SCOR) model, *Supply Chain Management: An International Journal* **9** (2004), 23–29. doi:10.1108/13598540410517557.
[72] iExec Team, iExec, 2017, https://iex.ec/.
[73] N.R. Jennings and M.J. Wooldridge (eds), *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag, Berlin, Heidelberg, 1998. ISBN 3540635912.
[74] J. Jovanovic and E. Bagheri, Electronic commerce meets the Semantic Web, *IT Prof.* **18** (2016), 56–65, http://dblp.uni-trier.de/db/journals/itpro/itpro18.html#JovanovicB16.
[75] H.M. Kim and M. Laskowski, Toward an ontology-driven blockchain design for supply-chain provenance, *Int. Syst. in Accounting, Finance and Management* **25**(1) (2018), 18–27.
[76] P.D. Larson and D.S. Rogers, Supply chain management: Definition, growth and approaches, *Journal of Marketing Theory and Practice* **6**(4) (1998), 1–5. doi:10.1080/10696679.1998.11501805.
[77] M. Li, L. Xia and O. Seneviratne, Leveraging standards based ontological concepts in distributed ledgers: A healthcare smart contract example, in: *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, 2019, pp. 152–157. doi:10.1109/DAPPCON.2019.00029.
[78] C.F. Longo, P.M. Riela, D.F. Santamaria, C. Santoro and A. Lieto, A framework for cognitive chatbots based on abductive–deductive inference, *Cognitive Systems Research* **81** (2023), 64–79. doi:10.1016/j.cogsys.2023.05.002.
[79] C.F. Longo, C. Santoro, D. Cantone, M. Nicolosi-Asmundo and D.F. Santamaria, SW-CASPAR: Reactive-cognitive architecture based on natural language processing for the task of decision-making in the open-world assumption, in: *22nd Workshop from Objects to Agents, WOA 2021*, Bologna, 1–3 September 2021, Vol. 2963, CEUR-WS, 2021, pp. 178–193.
[80] C.F. Longo, C. Santoro, M. Nicolosi-Asmundo, D. Cantone and D.F. Santamaria, Towards ontological interoperability of cognitive IoT agents based on natural language processing, *Intelligenza Artificiale* **16**(1) (2022), 93–112. doi:10.3233/IA-210125.
[81] D. Martin, M. Burstein, S. McIlraith, M. Paolucci and K. Sycara, OWL-s and agent-based systems, in: *Extending Web Services Technologies: The Use of Multi-Agent Approaches*, L. Cavedon, Z. Maamar, D. Martin and B. Benatallah, eds, Springer US, Boston, MA, 2004, pp. 53–77. ISBN 978-0-387-23344-4. doi:10.1007/0-387-23344-X_3.
[82] M.A. Musen, The protégé project: A look back and a look forward, *AI Matters* **1**(4) (2015), 4–12. doi:10.1145/2757001.2757003.
[83] NGI-ONTOCHAIN, ONTOCHAIN a new software ecosystem for trusted, traceable & transparent ontological knowledge, 2020, https://ontochain.ngi.eu/.
[84] G.M. Organisations, Tag data standard 1.13, GS1, 2019, https://www.gs1.org/standards/epcrfid-epcis-id-keys/epc-rfid-tds/1-13.
[85] P.F. Patel-Schneider, Analyzing schema.org, in: *International Semantic Web Conference (1)*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C.A. Knoblock, D. Vrandecic, P. Groth, N.F. Noy, K. Janowicz and C.A. Goble, eds, Lecture Notes in Computer Science, Vol. 8796, Springer, 2014, pp. 261–276, http://dblp.uni-trier.de/db/conf/semweb/iswc2014-1.html#Patel-Schneider14. ISBN 978-3-319-11963-2.
[86] J. Pfeffer, A. Beregszazi and S. Li, Ethon – an Ethereum ontology, 2016, available on-line, https://ethon.consensys.net/index.html.
[87] A. Rao and M. Georgeff, BDI agents: From theory to practice, in: *Proceedings of the First International Conference on Multi-Agent Systems, ICMAS-95*, San Francisco, CA, 1995, pp. 312–319.
[88] M. Ruta, F. Scioscia, S. Ieva, G. Capurso, A. Pinto and E. Di Sciascio, A blockchain infrastructure for the Semantic Web of Things, in: *26th Italian Symposium on Advanced Database Systems (SEBD 2018)*, 2018.
[89] N. Seydoux, K. Drira, N. Hernandez and 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.
[90] Y. Shoham, Agent-oriented programming, *Artificial Intelligence* **60**(1) (1993), 51–92. doi:10.1016/0004-3702(93)90034-9.
[91] M.P. Singh, *A Social Semantics for Agent Communication Languages*, North Carolina State University at Raleigh, USA, 1999.
[92] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur and Y. Katz, Pellet: A practical OWL-DL reasoner, *Web Semantics* **5**(2) (2007), 51–53. doi:10.1016/j.websem.2007.03.004.
[93] Statista, Market capitalization of transactions globally involving a non-fungible token, (NFT) from 2018 to 2020, 2021, last visit on 01/12/2021, https://www.statista.com/statistics/1221742/nft-market-capitalization-worldwide/.
[94] M. Stonebraker and J.M. Hellerstein, Content integration for e-business, in: *SIGMOD Conference*, S. Mehrotra and T.K. Sellis, eds, ACM, 2001, pp. 552–560, http://dblp.uni-trier.de/db/conf/sigmod/sigmod2001.html#StonebrakerH01. ISBN 1-58113-332-4.
[95] J. Sunny, N. Undralla and V. Madhusudanan Pillai, Supply chain transparency through blockchain-based traceability: An overview with demonstration, *Computers & Industrial Engineering* **150** (2020), 106895, https://www.sciencedirect.com/science/article/pii/S0360835220305829. doi:10.1016/j.cie.2020.106895.

[96] N. Szabo, Formalizing and securing relationships on public networks, *First Monday* **2**(9) (1997). doi:10.5210/fm.v2i9.548.
[97] S. Tartir, I.B. Arpinar, M. Moore, A.P. Sheth and B. Aleman-Meza, OntoQA: Metric-based ontology quality analysis, in: *Proceedings of IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005.
[98] The ECLASS organization, Eclass – standard for master data and semantics for digitalization, 2000, https://www.eclass.eu/en/index.html (last visit 01/12/21).
[99] The United Nations Development Program (UNDP) and Dun & Bradstreet Corporation (D& B), United Nations Standard Products and Services Code (UNSPSC), 2003, https://www.unspsc.org/ (last visit 01/12/21).
[100] O. Thomas and M. Fellmann, Semantic process modeling – design and implementation of an ontology-based representation of business processes, *Business & Information Systems Engineering* **1** (2009), 438–451. doi:10.1007/s12599-009-0078-8.
[101] L. Török and M. Hepp, Towards portable shopping histories: Using GoodRelations to expose ownership information to E-commerce sites, in: *The Semantic Web: Trends and Challenges – 11th International Conference, ESWC 2014, Proceedings*, Anissaras, Crete, Greece, May 25–29, 2014, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab and A. Tordai, eds, Lecture Notes in Computer Science, Vol. 8465, Springer, 2014, pp. 691–705. doi:10.1007/978-3-319-07443-6_46.
[102] Q. Tran and G. Low, MOBMAS: A methodology for ontology-based multi-agent systems development, *Inf. Softw. Technol.* **50** (2008), 697–722.
[103] D. Tsarkov and I. Horrocks, *FaCT++ Description Logic Reasoner: System Description*, A. Reasoning, U. Furbach and N. Shankar, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 292–297. ISBN 978-3-540-37188-5.
[104] H.E. Ugarte Rojas, A more pragmatic web 3.0: Linked blockchain data, in: *Google Scholar*, 2017.
[105] H. Ugarte-Rojas and B. Chullo-Llave, BLONDiE: Blockchain ontology with dynamic extensibility, CoRR, 2020, arXiv:2008.09518.
[106] D. Vandic, J. van Dam and F. Frasincar, Faceted product search powered by the Semantic Web, *Decis. Support Syst.* **53**(3) (2012), 425–437. doi:10.1016/j.dss.2012.02.010.
[107] W. Wang, S. De, R. Toenjes, E. Reetz and 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.
[108] M. Wick, *Geonames Ontology*, 2015, http://www.geonames.org/about.html.
[109] World Wide Web Consortium, OWL-S: Semantic markup for web services, 2004, https://www.w3.org/Submission/OWL-S/.
[110] World Wide Web Consortium, RDFa primer: Bridging the human and data webs, 2008, http://www.w3.org/TR/xhtml-rdfa-primer/.
[111] World Wide Web Consortium, JSON-LD, 2010, https://json-ld.org/.
[112] World Wide Web Consortium, Schema.org, 2015, https://www.w3.org/community/schemaorg/.
[113] World Wide Web Consortium, Web of Things (WoT) Thing Description, 2020, https://www.w3.org/TR/wot-thing-description/.