Semantic Web 1 (0) 1–4
IOS Press

A Behavioristic Semantic Approach to Blockchain-based E-Commerce

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

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

E-mails: giamp@dmi.unict.it, domenico.cantone@unict.it, marianna.nicolosi@unict.it, santamaria@dmi.unict.it

Abstract. Electronic commerce and finance are progressively supporting and including decentralized, shared and public ledgers such as the blockchain. This is reshaping traditional business by advancing it 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 the 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 ecosystem would contribute dramatically to pan it out and make it more usable, namely more readily accessible to virtually anyone wanting to play the role of a stakeholder. 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 applicable business features, ranging from the relevant 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 behavioristic approach to representing business participants as agents in terms of their actions, taking inspiration from the *Theory of Agents* and its mentalistic notions. The stack is validated through appropriate metrics and competency questions, then demonstrated through the mapping of a real world use case, the iExec marketplace.

Keywords: Ontology, Semantic Web, DeFi, Agent, Ethereum

1. Introduction

Electronic commerce (*e-commerce*) refers to the various activities revolving around the electronically buying or selling of products through online services [1]. The term was first used in the California's Electronic Commerce Act, enacted in 1984. From the 1990s, e-commerce experienced a huge, continuos evolution. Much more recently, commerce — as well as finance — embraced the *blockchain* as a platform to deploy and exchange digital 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 [2]. Blockchains provide several benefits such as ownership, transparency, traceability, public availability, continuity and immutability of digital assets, all in an efficient and trust-less environment where censorship is not

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

1570-0844/$35.00 © 0 – IOS Press and the authors. All rights reserved

achievable. One of the most popular applications of Turing-complete blockchains such as Ethereum [3] to business activities is the *smart contract*. Smart contracts are self-executing and immutable programs, autonomously running and verified on a distributed and decentralized public network, which implement decentralized applications on blockchain systems called *Dapps*.

1.1. Goal

Dapps have grown particularly as an exchange tool for *non-fungible tokens* (NFTs). NFTs 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). At the end of 2020, the market capitalization of NFTs reached the amount of 338 millions of U.S. dollars [4].

The overarching goal of the present article is to lay out the foundations for a clear and unambiguous semantic representation of the blockchain in the area of e-commerce. Achieving this goal would have several advantages. A major one would be the simplification and automation for the tasks of probing the blockchain for identifying the desired activity or token and related features, as well as the smart contract exchanging specific types of tokens and the trading conditions. This would, in turn, promote the widespread and pervasive uses of the new technologies through all levels of society. Some state of the art exists and, as we shall see below, is valid and useful, but what still lies at the core of work in progress is the conjunction of business and blockchain systems from an epistemological point of view, 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 ontology [5] project has been active since 2001 and targets the semantic representation of business-related goods and services, while ontologies for blockchains were introduced much more recently, notably with the widely-known BLONDiE [6] and Ethon [7] ontologies. 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.

In consequence, another challenge is to account for the relevant stakeholders, including applications, people and assets. Because the stakeholders may be active participants, it is convenient to leverage the concept of *autonomous agent* [8]. Agents are defined as entities whose state consists of mental components such as beliefs, capabilities, choices and commitments [9]. 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, 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 behavioristic approach [10], 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 so as to make the discovery of agents transparent, automatic and independent from 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 [11], which applies the behavioristic 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 would be semantically described as agents by means of the actions they can perform, including supply chains, payment and provision methods. This will allow people to freely choose products and services according to need and liking, as well as to gain in awareness on the full supply chains up to the delivery of their choices.

1.3. Contributions

The *Next Generation Internet initiative*, launched by the European Commission in 2018, founded the ONTOCHAIN project [12], which is aimed at solutions for a secure and transparent blockchain-based knowledge management system as an interoperability service for the Internet. ONTOCHAIN funded the third-party project POC4COMMERCE [13] to build the ontological framework for the entire ONTOCHAIN ecosystem, albeit with a focus on the commercial domain. The present article reports on the foundational results of POC4COMMERCE.

We adopt the behavioristic approach towards the representation of today's blockchain-oriented e-commerce, in particular exploiting token generation and exchange mechanisms as decentralized public proofs. OASIS is leveraged to define three novel ontologies. First, the OC-Found ontology extends OASIS to describe digital identities and supply chains. 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 imported from OC-Found. The stack culminates with the OC-Ethereum ontology, which imports the definitions of the BLONDiE ontology for the constitutional elements of the Ethereum blockchain so as to include the description of Ethereum tokens and the operational semantics of smart contracts. Additionally, OC-Ethereum describes the practical uses for standard ERC20 fungible tokens, ERC721 non-fungible tokens and ERC1155 semi-fungible tokens.

The contributed ontological stack is publicly released [14]. 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 also expressed as SPARQL queries to facilitate the reader's self check. Finally, the ontological stack is demonstrated on a real-world case study, the iExec marketplace [15], to further confirm its expressiveness. We shall see that the iExec marketplace can be easily mapped out.

1.4. Structure

The paper is organized as follows. Section 2 concerns the relevant related works. Section 3 outlines the existing ontologies that are leveraged through the present work. Specifically, Section 3.1 introduces the OASIS ontology, designed for representing agents and their commitments; Section 3.2 outlines the GoodRelations ontology for representing commercial offerings and selling conditions; Section 3.3 focuses on the BLONDiE ontology for representing blockchains constitutional elements, such as block, wallet and transaction. Section 4 introduces our contributed ONTOCHAIN ontological stack and is organized in three subsections, one for each layer of the stack. Precisely, Section 4.1 presents the OC-Found ontology, 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 behavioristic approach; 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. Section 5 evaluates the ontological stack by a) considering both structural and ontological metrics applied on 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 mapping the iExec marketplace. Finally, Section 7 draws conclusions and directions for future research.

2. Related Works

In the context of the semantic representation of applications, the Semantic Web aims at enabling users to locate, select, employ, compose and monitor web-based services automatically, giving rise to *Semantic Web Services*. Semantic Web Services describe Web Services with semantic content thereby automatically enabling service discovery, composition and invocation. For this reason, The DARPA Agent Markup Language (DAML) pursued the goal of developing a markup language for representing semantic relations in a machine readable way [16]. In 2003, the *World Wide Web Consortium* (W3C) proposed OWL-S [17], an ontology for describing web services and related information. OWL-S tries to enable several tasks such as automatic web services discovery, automatic web services

invocation and automatic web services composition and interoperation. OWL-S explicitly supports the description of services as classes of activities, so that agents can decide how to use, invoke and interpret responses from them. However, Since DARPA and OWL-S are conceived to describe services, they do not adequately fulfil the requirements of describing general-purpose agents operating in different types of environments [18]. Indeed, describing agents as they coincide with services, although of a limited kind, may lead to an inaccurate and ambiguous depiction of them, also because service capabilities need to be semantically described both at a high and a low level. Hence, one of the most prominent visions of the relationships between agents and services is one in which agents exploit, use, are composed of, or are deployed as, and are extended by services [18], which remain relatively simple. For instance, *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 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.

As from 2000, many results concerning how agents enter and leave different interaction systems have been provided both by exploiting *commitment objects* [19] and *virtual institutions* [20]. 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 behavior 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[21].

The integration of agent systems and Semantic Web technologies has been investigated in the last decade in several contexts [16, 22, 23] and the advantages of ontology-based applications have been recognized in the analysis and development of MAS [24]. Those advantages are manifest a) in the context of agent matching, i.e., the capability of finding agents satisfying specific requirements and automatically engaging them, b) in the context of developing agents with standard features exploiting shared common semantic tools, or c) as decision support for supply chains.

Ontologies for MAS have been modeled by taking approaches similar to those of *Agent-Oriented Software Engineering* (AOSE) [25, 26], 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) [27] are proposed in [10] and [28]. Inspired by Bratman's theory of human practical reasoning [29], 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 behaviors 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 behavior in a simple, natural way. In [30], an ontology for agent-oriented software engineering is proposed together with a tool that uses the ontology to generate programming code for MAS, but without examining agents and their interactions in detail. Moreover, an appropriate modelling of agent communication is missing.

Some results attempt to bring uniformity and coherence to the increasing volume and diversity of information in a specific domain, but domain-legacy generally generally makes them not applicable to other contexts. For example, in [31], the authors propose an ontology-based framework for seamlessly integrating agents and Semantic Web Services, focusing on biomedical information. In [32], an infrastructure to allow agent-oriented platforms to access and query domain-specific OWL ontologies is presented. In [33], 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 for generating perceptions and abstractions of the observed world [34].

A comprehensive ontology for representing IoT services is presented in [35], 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 [36], capturing the complete dynamics of IoT entities, for example enabling semantic searching while hiding their heterogeneity.

In [37], 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 [38], the authors propose a lightweight and scalable model, called *IoT-lite*, to describe key IoT concepts allowing interoperability and discovery of sensory data in 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 [39]. The representation system provides a way to expose the state of a thing and to invoke functions that, however, must be known in advance, and this may complicate the task of invoking agents that would like to join the environment in a plug-and-play manner. Moreover, the provided schema does not fully allow agents to interact according to the specific roles they aim to play.

In [11] the authors propose a first version of OASIS, an ontology for representing agents and their commitments by exploiting the behavioristic approach adopted in this work, together with a domotic assistant based on it, called PROF-ONTO. The approach employed in [11] also represents a first foundational contribution for using Semantic Web technologies for defining 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.

There exists relevant work on the use of Semantic Web technologies over e-commerce. In [40], the authors identify six challenges facing the e-commerce ecosystem: inaptness 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 recommender systems. Then, they propose a technological stack, based on the Semantic Web, to support the creation of *intelligent* e-commerce applications.

Creating shared or, at least, interoperable vocabularies for product descriptions have been recognized as a crucial task for e-commerce [41]. In [42], it is suggested that semantic vocabularies enable the implementation of *semantic search engines* [43] 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 (a comparative analysis can be found in [44]). Among others, we recall the *United Nations Standard Products and Services Code* (UNSPSC) [45], a taxonomy of products and services specific for e-commerce; *eCl@ss* [46] 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 [48], an organization that develops and maintains global standards for business communication such as, for example, barcodes and Electronic Product Codes [49]. 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* [50] 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 [51], Semantic Web technologies may help to overcome this issue by enabling *Ontological Mapping* between different systems.

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 [52] and integrated with the general purpose vocabulary largely used in tagging web page contents *schema.org*, as reported in [53]. Of course, other trading activities beyond offerings are important and, for example, ownership information may be used for personalized recommendations, as shown in [54].

Notably, GoodRelations does not cover how negotiations are carried out and how offerings or assets are related with tokens managed on the blockchain. Only recently have researchers focused on conjoining the blockchain with ontologies [55, 56]. One of the areas of investigation has concentrated in developing a characterization of blockchain concepts and technologies through ontologies. In [57], a theoretical contribution looking at the blockchain through an ontological approach has been provided. In [58], 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. Other works aim at representing ontologies within a blockchain context. In [59], 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 [60] 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 [61], the authors propose a semantic interpretation of smart contracts as services bases on the *Ethon ontology* [7]. More debate can be found in [62] 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.

In [63], the ontology OASIS 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 [64], 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 approach is part of the work presented in this paper even though it presents structural differences on how blockchains are modelled.

3. Preliminaries

In this section we briefly illustrate the main features of the ontologies adopted by the ontological stack, namely OASIS [11], GoodRelations [5] and BLONDiE [60]. These ontologies, which better describe the basic elements of different contexts of the blockchain-oriented commerce, are exploited to construct a behavioristic 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 so as 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, transaction and wallet, and is extended with the operation semantics of the Ethereum smart contracts and with a clear specification of smart contract interactions and tokens.

3.1. The ontology OASIS

The *Ontology for Agents, Systems and Integration of Services* (in short, OASIS) is a foundational OWL 2 ontology that applies the behavioristic approach to represent multi-agent systems in order to characterize agents in terms of the actions they are able to perform, including purposes, goals, responsibilities, information about the world they observe and maintain, and their internal and external interactions. 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 behaviors, which are publicly exposed. Thanks to public behaviors, agents report the set and type of operations that they are able to perform including, possibly, 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 behaviors (called *templates*), from which concrete agent behaviors are drawn;
- modelling concrete agent behaviors drawn by agent templates;
- modelling actions performed by agents by associating them with the behaviors, from which actions are drawn.

The first step is not mandatory and consists in defining the agent's behavior template, namely a high-level description of behaviors of abstract agents that can be implemented to define more specific and concrete behaviors of real agents. For example, a template may be designed for agents whose behavior 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 to guide developers to define the most suitable representations for their agents.

The second step consists in representing the agent behaviors either by specifying a shared template or by defining it from scratch. To describe abstract agent's capabilities to perform actions, an agent template comprises three main elements, namely behaviors, goals and tasks. Agent tasks, in their 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 behaviors and associated with the behaviors of the agent that generate them. To describe such an association, OASIS introduces *plan executions*. Plan executions describe the actions performed by an agent and associated with one of its behaviors. The associations is carried on by entailing a description of the performed action to the behavior from which the action has been drawn: actions are hence described by suitable graphs that retrace the model of the agent's behavior.

3.2. The ontology GoodRelations

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

The core class of the GoodRelation vocabulary 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 which 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 available to provide the product or service with the destination of fulfillment chosen by the customer. They can be possibly 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 ontology BLONDiE

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 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 including the following elements:

– The *block-header* of a block. This is the section summarizing the block itself. It includes several 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.
– 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 creation* (associated with smart contract creation) and *message calls*, i.e., passages of messages from one account to another, possibly including data. We recall that Ether is the cryptocurrency adopted by the Ethereum blockchain to pay for executing transactions.
– *Account*. Accounts are referred to the wallets used to store cryptocurrencies, to pay for executing and to authorize transactions.

4. The ONTOCHAIN ontological stack

The ontological stack adopts and extends state-of-the-art ontologies suitably selected for the blockchain-oriented commerce domain with a behavioristic 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 sub-domain 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-oriented commerce ecosystem and exploiting 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 behavioristic approach; the third ontology, called OC-Ethereum and exploiting 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-oriented commerce but that 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. To achieve the objective, OC-Found applies the behavioristic approach pursued by OASIS for representing agents to the commercial domain by extending the notions of agent behavior template, agent behavior and agent action, with the semantic representation of digital identities, supply chains and quality valuation mechanisms.

In this section, we first describe the entities of the OASIS ontology that are used by OC-Found and then how OC-FOUND extends them to reach the proposed goals.

In OASIS, agent templates are defined according to the UML diagram depicted in Figure 1. To describe abstract agent capabilities of performing actions, an agent template comprises three main elements, namely *behaviors*, *goals* and *tasks*. Agent tasks, in their turn, describe atomic operations that agents may perform, including possibly input and 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 characterises agent commitments. Instances of the class *TaskDescription* are related with 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*, ...).[1] The object-property *refersExactlyTo* is used to connect the task operator with a precise action having a specific IRI, whereas *refersAsNewTo* is used to connect a task operator with an action for which a general abstract description is provided. In the latter case, the action is also defined as instance of the class *ReferenceTemplate*: instances of the class *ReferenceTemplate* are used to introduce entities that represent templates for the referred element describing the characteristics that it should satisfy. By exploiting the object-property *refersAsNewTo*, the entity provides only a general description of the features needed to accomplish the task, for example, that it must be of a specific type; on the contrary, the object-property *refersExactlyTo* specifies the actual and exact entity that is 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 either the object-property *refersAsNewTo* or *refersExactlyTo*.
- An instance of the class *TaskObjectTemplate*, connected by means of the object-property *hasTaskObjectTemplate* and representing the template of the object recipient of the action performed by the agent (e.g., *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

[1] Instances of *Action* are introduced in the *OASIS-Abox* ontology [65]

either the object-property *refersAsNewTo* or *refersExactlyTo*. Moreover, the classes *TaskInputParameterTemplate* and *TaskOutputParameterTemplate* are also subclasses of the class *TaskParameterTemplate*.

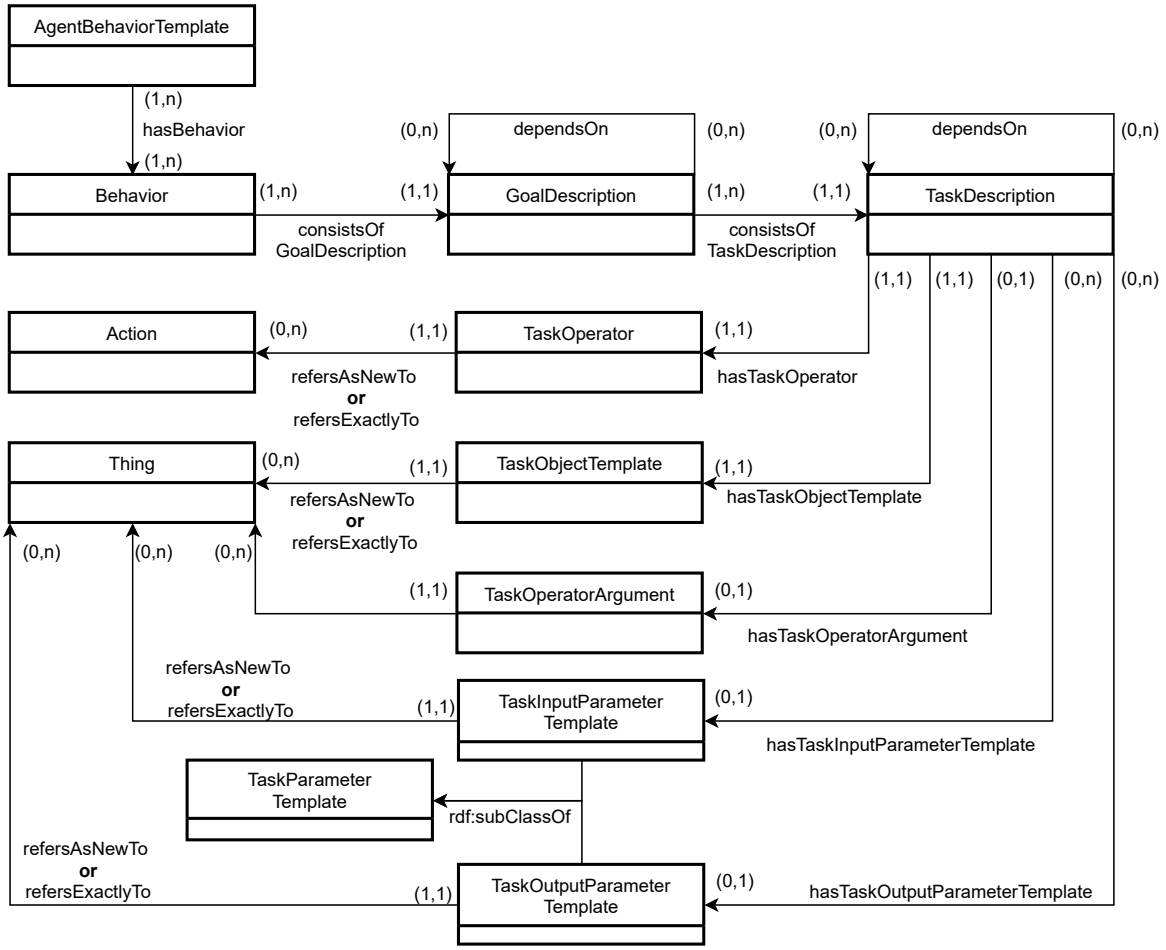

Fig. 1. UML diagram representing OASIS agent templates in OC-Found

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*.
- *Behavior*. Behaviors 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 behavior 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 atomic operations that agents are able to 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 as to be performed in any order. Finally, tasks are linked to the individuals that describe the features of the atomic operations, i.e., the instances of the classes *TaskOperator*, *TaskOperatorArgument*, *TaskObjectTemplate TaskInputParameterTemplate* and *TaskOutputParameterTemplate*, as described above.

- *TaskOperator*. This class characterizes the type of operation to perform. Instances of *TaskOperator* are connected with instances of the class *Action* by means of either the object-properties *refersExactlyTo* or *refersAsNewTo*. Tasks are connected with task operators by means of the object-property *hasTaskOperator*.
- *Action*. This class describes actions that can be performed by agents. Entity names of the class *Action*'s instances are introduced as infinite verbs such as *buy*, *sell*, *compute*, and so on, drawn from a common, shared and extendable vocabulary defined by the OASIS-Abox ontology [65].
- *TaskOperatorArgument*. This class defines operator arguments representing subordinate characteristics of task operators. Tasks are connected with operator arguments by means of the object-property *hasTaskOperatorArgument*.
- *TaskObjectTemplate*. Instances of this class represent the recipient of the behaviors of agent templates. Tasks are connected with object templates by means of the object-property *hasTaskObjectTemplate*.
- *TaskInputParameterTemplate*. This class represents the input parameters required to accomplish the referred operation, as for instance the type of asset on which a quality valuation is performed. Tasks are possibly connected with the input parameters by means of the object-property *hasTaskInputParameterTemplate*.
- *TaskOutputParameterTemplate*. This class represents the output parameters obtained as a result of the referred operation. Tasks are possibly connected with the output parameters by means of the object-property *hasTaskOutputParameterTemplate*.

Analogously, concrete agents are represented according to the UML diagram in Figure 2, which reflects the modelling pattern adopted for the representation of agent behavior templates described above, but introducing the following differences:

- The instance of the class *AgentBehaviorTemplate* gives way to an instance of the class *Agent*, representing a concrete agent in the knowledge domain.
- 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 that they are drawn from. In order to describe the fact that concrete agents inherit their behaviors 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 *overloads* with the instance of the class *TaskDescription* of the agent template.
- The instance of the class *TaskObject* of the concrete agent is connected by means of the object-property *overloads* with the instance of the class *TaskObjectTemplate* of the agent template.
- The instance of the class *TaskFormalInputParameter* of the concrete agent is connected by means of the object-property *overloads* 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 *overloads* with the instance of the class *TaskOutputParameterTemplate* of the agent template.

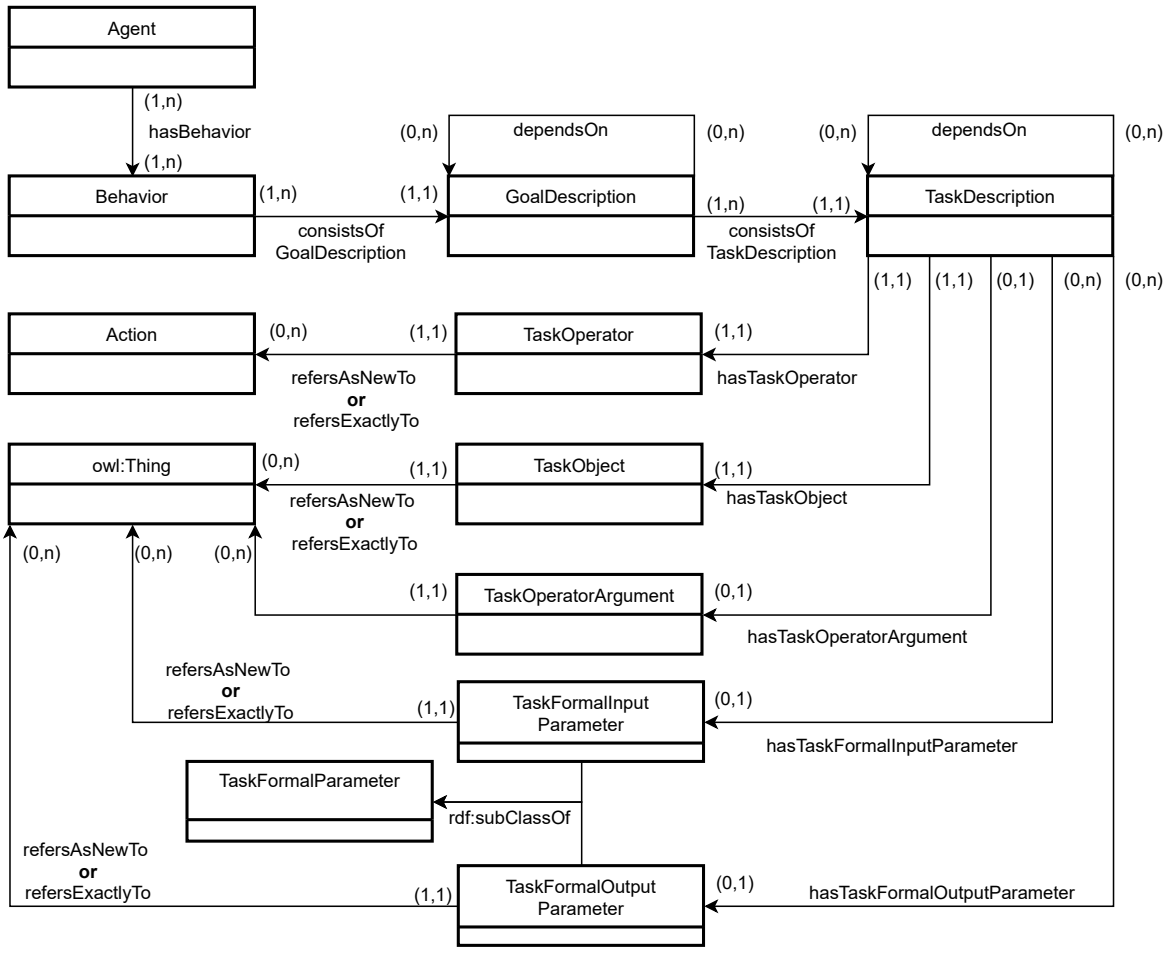

Fig. 2. UML diagram representing OASIS concrete agents in OC-Found

Agent actions entail the description of the concrete behavior of the agent from which they are drawn. As depicted in Figure 3, 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 their turns, goal executions provide task executions (instances of the class *TaskExecution*) embedding the following elements:

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

In contrast to the tasks of agent behaviors, 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*.

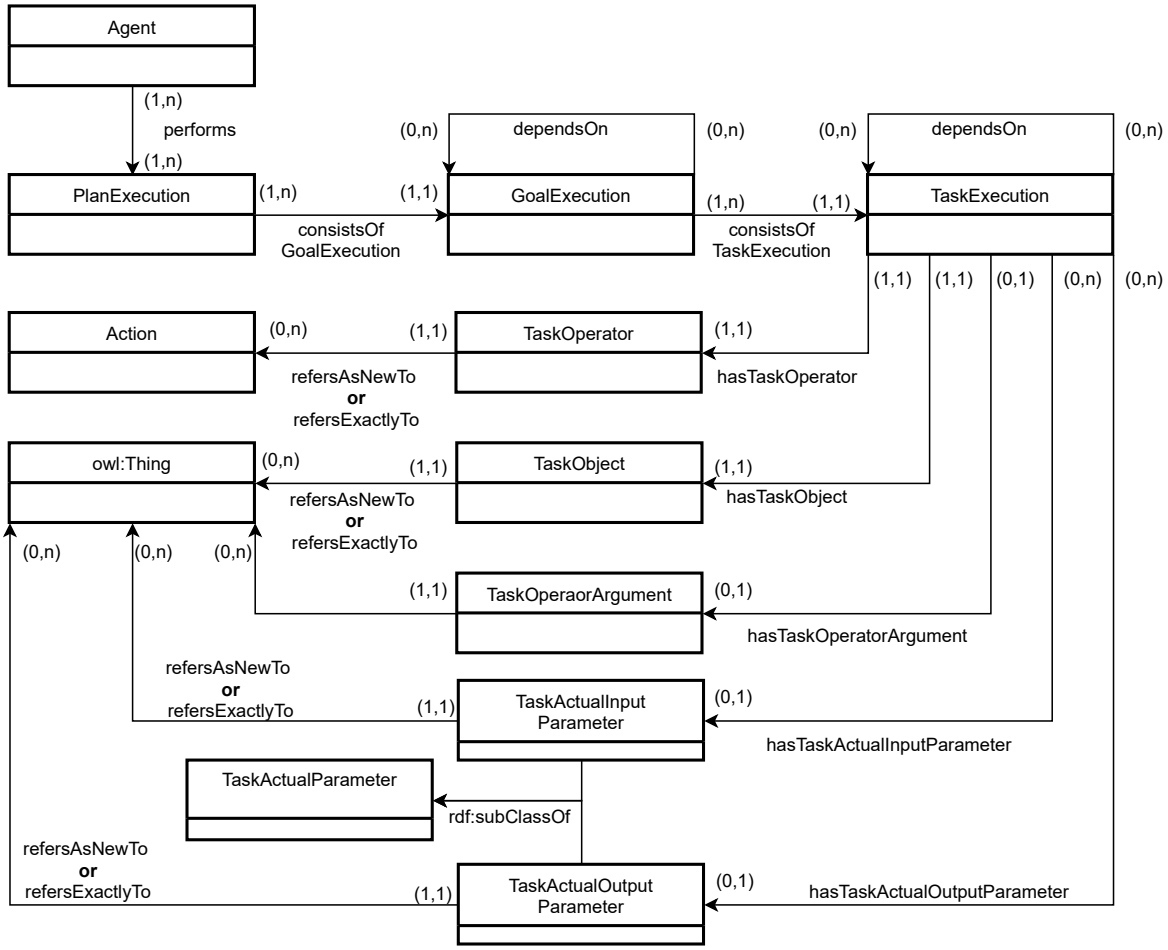

Fig. 3. UML diagram representing OASIS agent actions in OC-Found

We now discuss the ontological core of OC-Found. The main classes and properties introduced by OC-Found to model supply chains and digital identities associated with agents are depicted in Figure 4, which has been obtained by the editor Protégé [66]. Entities having the prefix *oasis* are those imported from the OASIS ontology, whereas the remaining entities 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*.

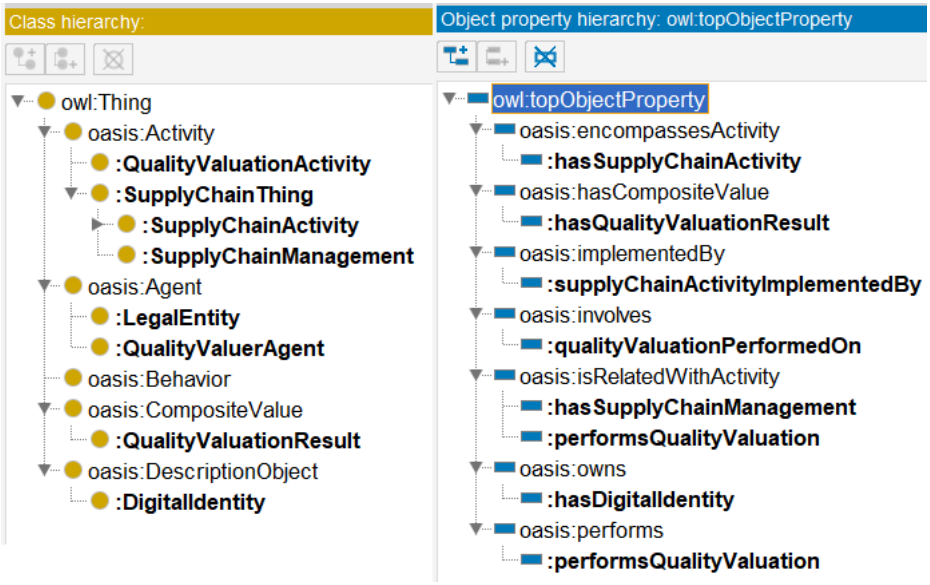

Fig. 4. Hierarchies of classes (on the left) and object-properties (on the right) in the OC-Found ontology

In OC-Found the life-cycle of assets is described by means of supply chains. Supply chains *encompass all of those activities associated with moving goods from the raw-materials stage through to the end user* [67]. Hence, more generally speaking, 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 behavior of the agent responsible for its realization. Specifically, OC-Found currently includes the following subclasses of the class *SupplyChainActivity*:

- *SupplyChainProofOfWorkActivity*. This class describes the activities related with the process of releasing some proofs of work, such as digital tokens emitted to witness the transferring of ownership of the resource.
- *SupplyChainDeliveryActivity*. This class describes the process related with the delivering 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 supply chains that have to be considered as interchangeable supply chains. Moreover, supply chains 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 behavior responsible for its execution. The classes *SupplyChainManagement* and *SupplyChainActivity* are also defined in OC-Found as subclass of the class *SupplyChainThing*, encompassing all the entities related with supply chains. Usage of OC-Found classes and properties are depicted in Figure 5, 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 their 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 *implementedBy*). For instance, the supply chain of an apple producer may be represented as depicted in Figure 6, where classes are reported in bold, whereas instances are reported below the related membership classes.

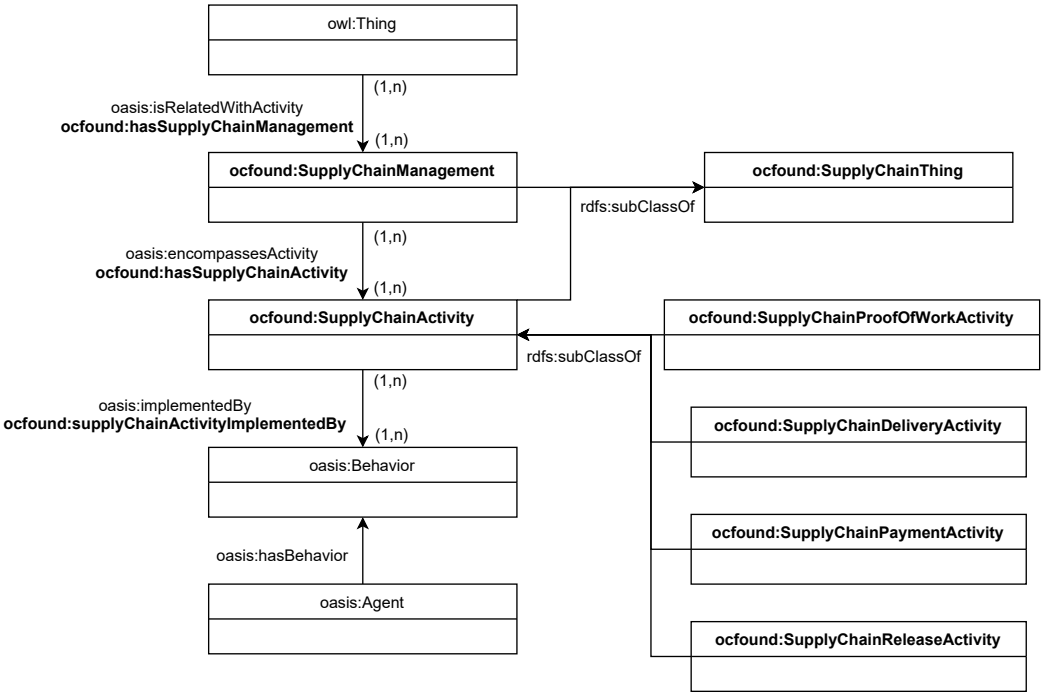

Fig. 5. UML diagram representing the OC-Found ontology

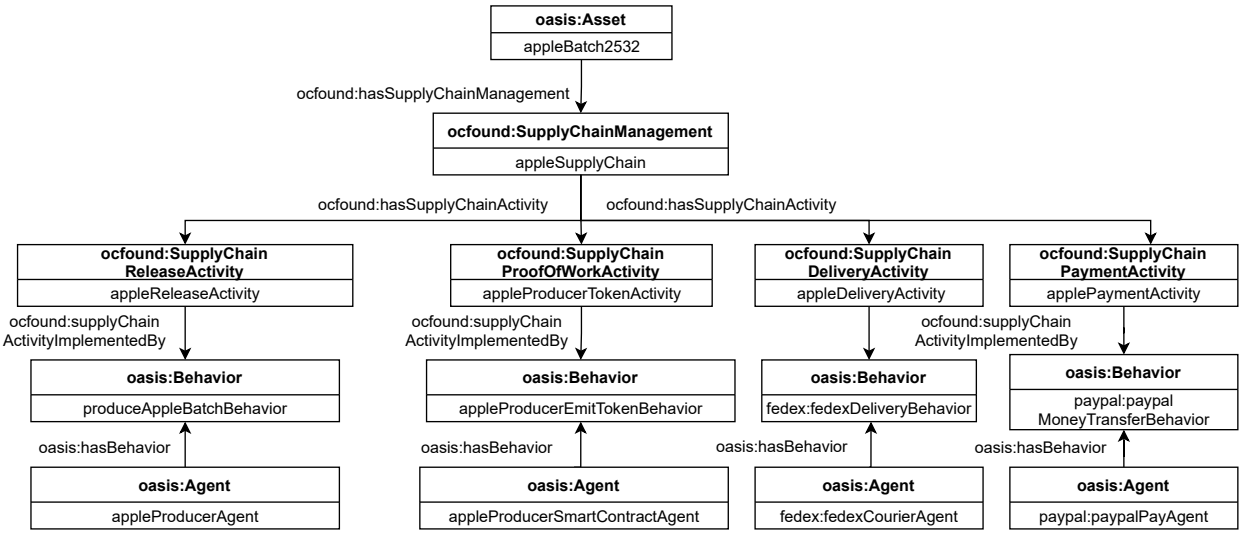

Fig. 6. UML diagram exemplifying an apple producer supply chain in OC-Found

The example in Figure 6 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 behavior 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 delivering activity of the batch which is carried out by the agent *fedex:fedexCourierAgent*.
- The fourth activity (individual *applePaymentActivity*) is related with activity 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 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 Figure 7 (entities that are newly introduced in OC-Found are reported in bold).

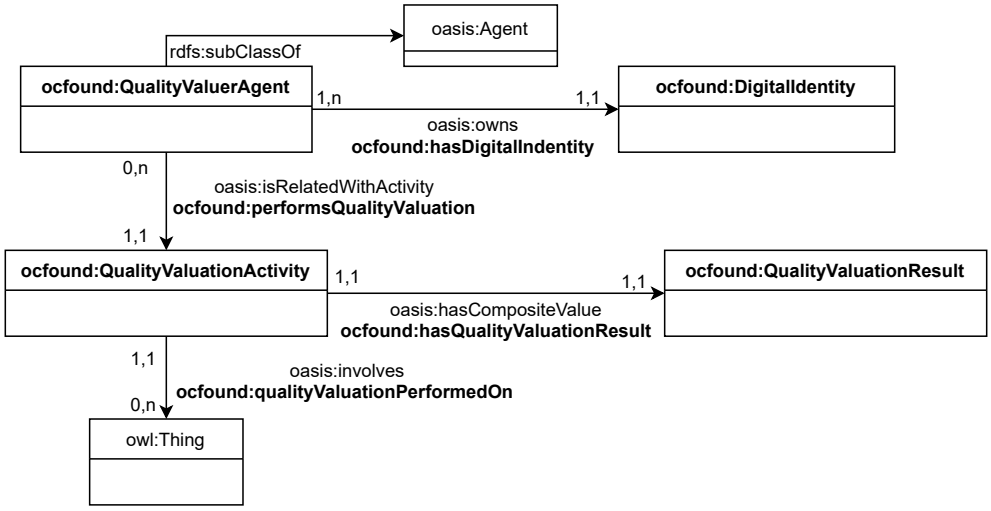

Fig. 7. UML diagram representing OC-Found quality valuation process

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 behavior, 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 a) with the resource on which the valuation is performed by means of the object-property *qualityValuationPerformedOn* (subproperty of the OASIS property *involves*) and b) with the result of the valuation, represented by an instance of the class *QualityValuationResult* (subclass of the OASIS class *CompositeValue*) by means of the object-property *hasQualityValuationResult* (subproperty of the OASIS property *hasCompositeValue*). An example of valuating the apple batch is illustrated in Figure 8, where the valuer agent *agriFoodValuer* performs a valuation activity on the apple batch *appleBatch2532*, assigning a total result of 5.

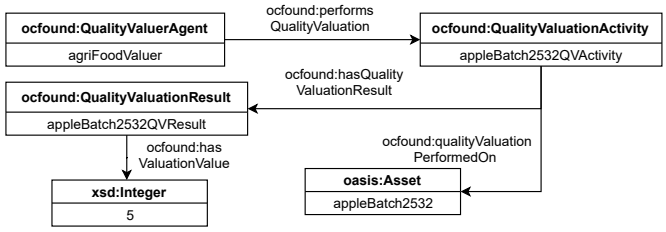

Fig. 8. UML diagram exemplifying the quality valuation on apple *batch 2563* in OC-Found

4.2. *The OC-Commerce ontology*

In this section, we first show the structure of GoodRelations and how some of its concepts are extended in OC-Commerce. The ontology OC-Commerce[2] conjoins OC-Found with many of the basic features provided by the GoodRelations ontology (the latter depicted in Figure 9) 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 offering 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 bargaining activities, auctions, modifications of offerings, valuations, and price determination mechanisms.

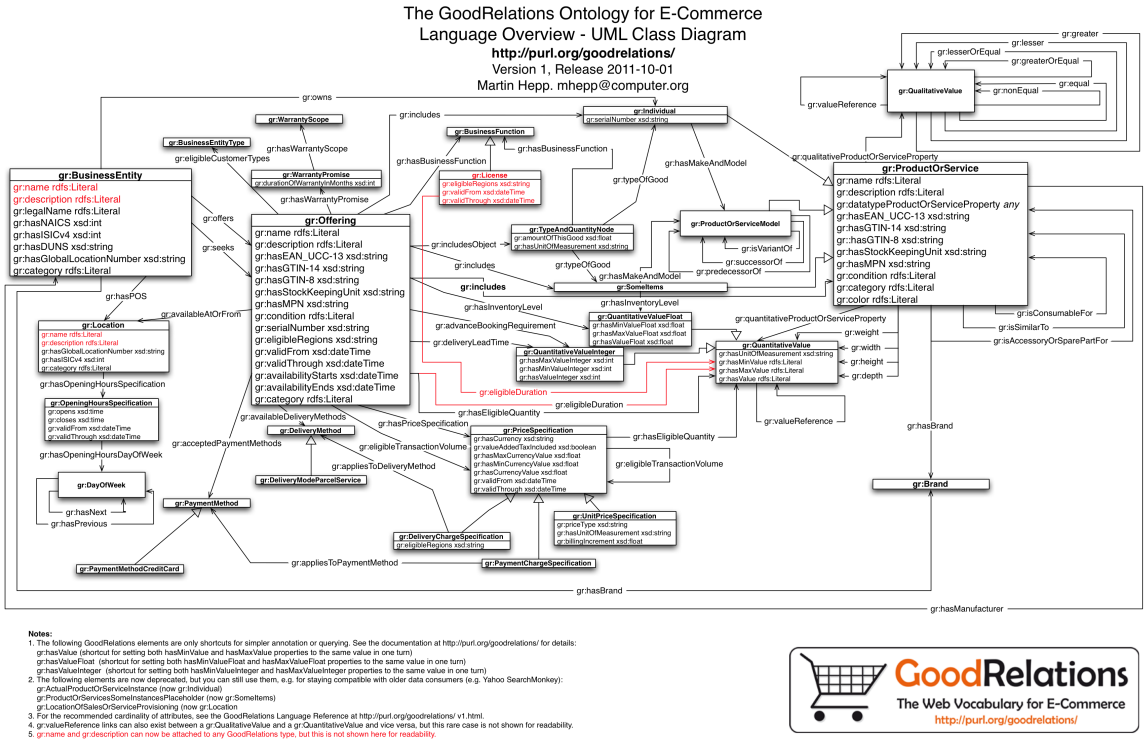

Fig. 9. UML diagram representing the GoodRelations ontology

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

The classes and object-properties introduced by OC-Commerce and the related mapping into GoodRelations are illustrated in Figures 10 and 11, 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 behavior 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 Figure 12. In an analogous way, agents enabled to request, modify, retract, close, accept, and reject offerings should manifest a suitable behavior 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;
- *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.

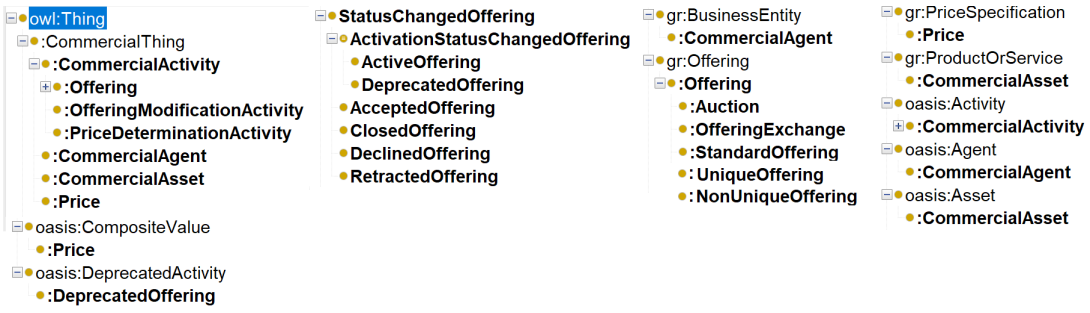

Fig. 10. Hierarchies of classes in the OC-Commerce ontology

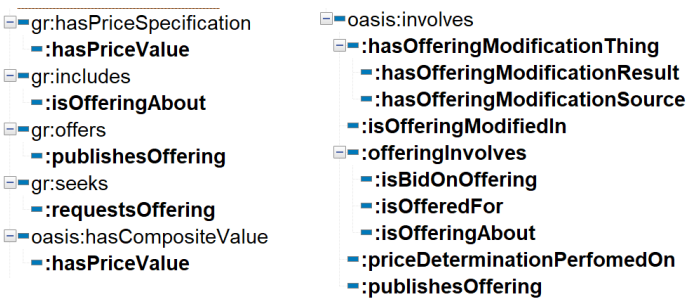

Fig. 11. Hierarchies of object-properties in the OC-Commerce ontology

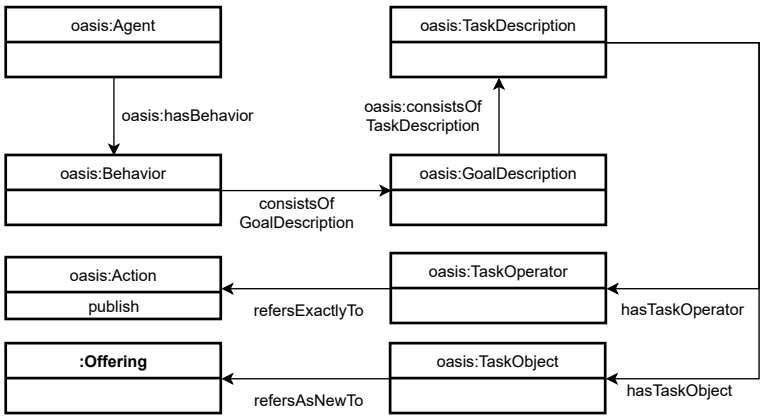

Fig. 12. UML diagram representing OC-Commerce offering publishing

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 instance, an apple producer, whose behavior is depicted in Figure 13, deploys a new asset of apples, namely *batch 2563*, and makes it available for trading, as reported in Figure 14.

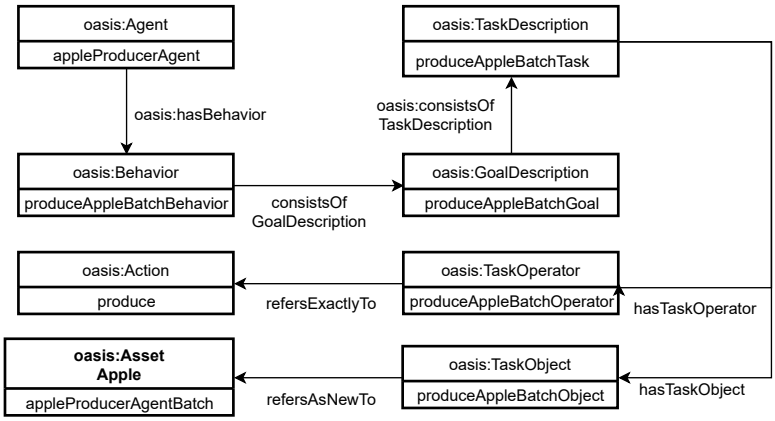

Fig. 13. UML diagram exemplifying the production of apple batch 2563 in the OC-Commerce ontology

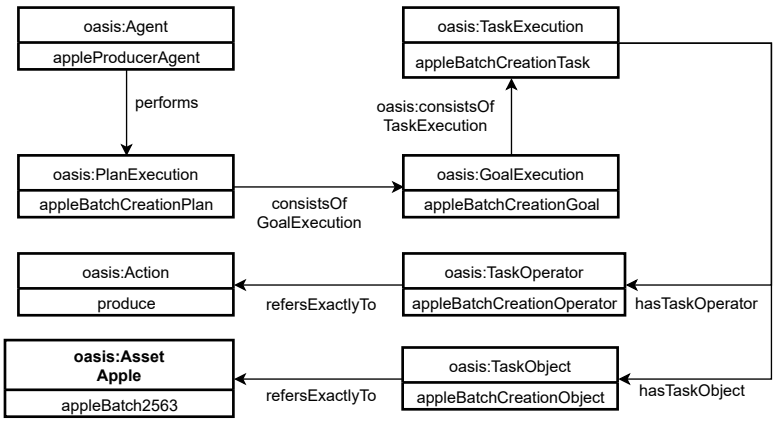

Fig. 14. UML diagram exemplifying the publication of apple batch 2563 in the OC-Commerce ontology

As a second step, the agent delegated to make the offering available publishes an action associated with the behavior responsible for publishing offerings. In the case of the apple producer, a new offering concerning the batch 2563 should be published as reported in Figure 15.

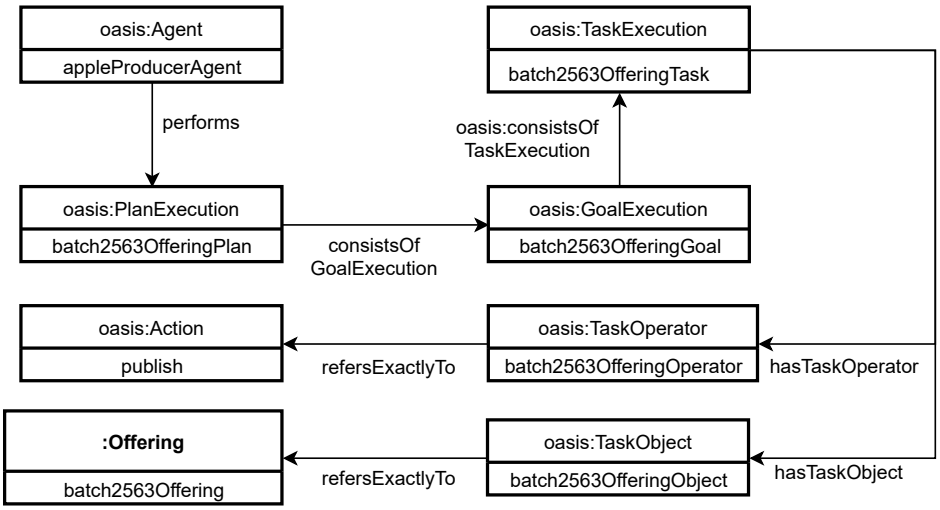

Fig. 15. UML diagram exemplifying the publication of an offering for apple batch 2563 in the OC-Commerce ontology

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 their turn, offerings 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 of Figure 16.

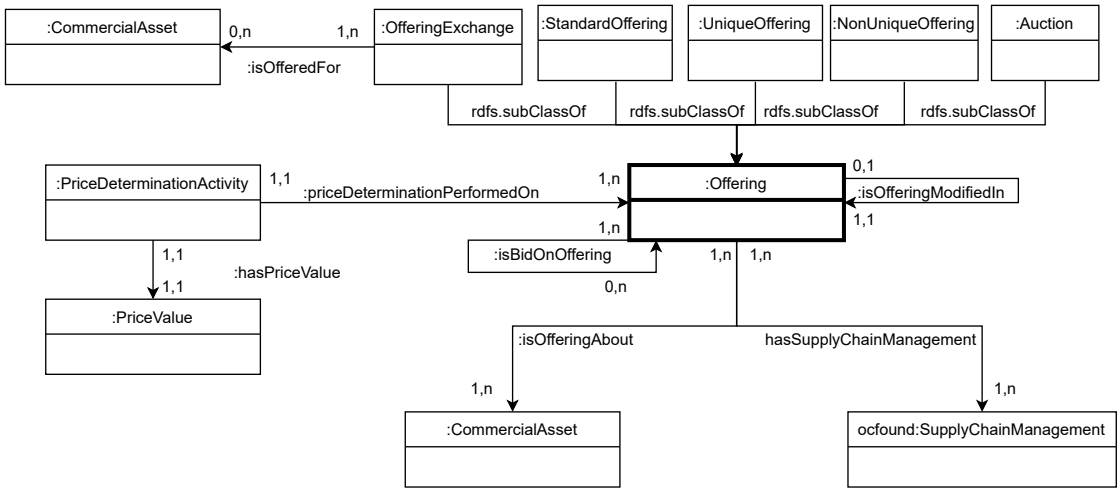

Fig. 16. UML diagram representing OC-Commerce offerings

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 behaviors for transferring crypto- of FIAT currencies, respectively. Offerings, in their 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 behavior.
- *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.
- *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 a) with the offering by means of the object-property *priceDeterminationPerformedOn* and b) with 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 defined also 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 Figure 17. 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 Figure 6 of Section 4.1. Then, the user Bob accepts the offering by performing the action depicted in Figure 18 and, as 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 Figure 19. 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 result of the action, a payment receipt is emitted. At this point, the Fedex agent ships the selected product, as described in Figure 20. The user destination is represented by an instance of the GeoNames ontology [68], 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 of the involved agents, by committing himself to an action as the one described in Section 4.1.

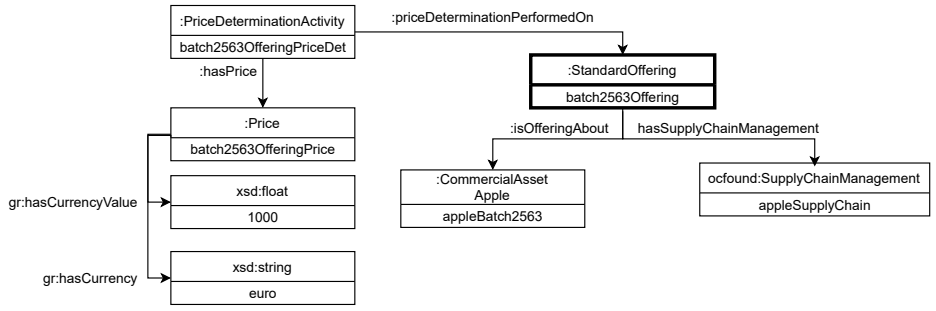

Fig. 17. UML diagram exemplifying the full details of an offering for apple batch 2563 in the OC-Commerce ontology

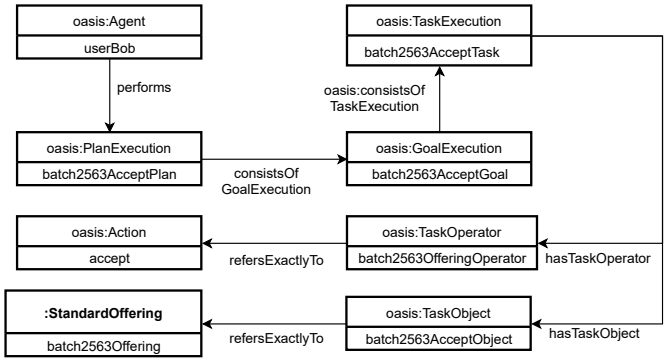

Fig. 18. UML diagram exemplifying the offering acceptance for apple batch 2563 in the OC-Commerce ontology

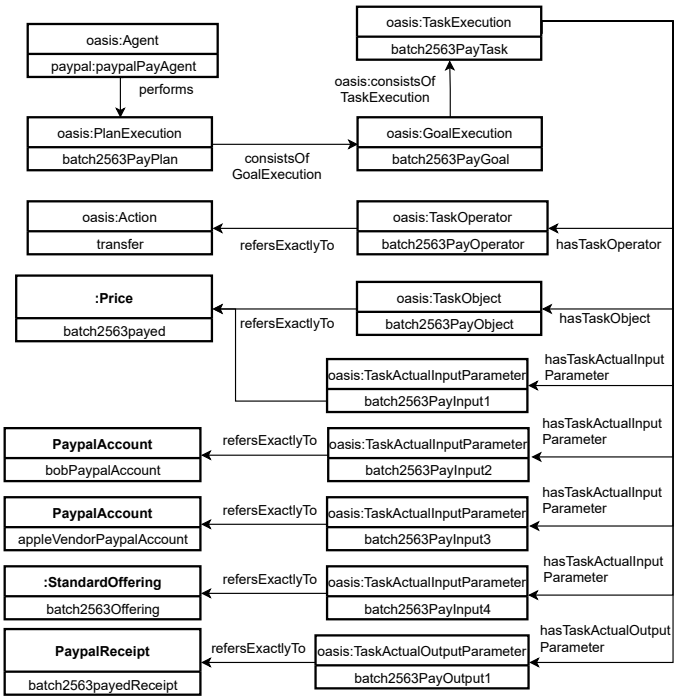

Fig. 19. UML diagram exemplifying the payment acceptance for apple batch 2563 in the OC-Commerce ontology

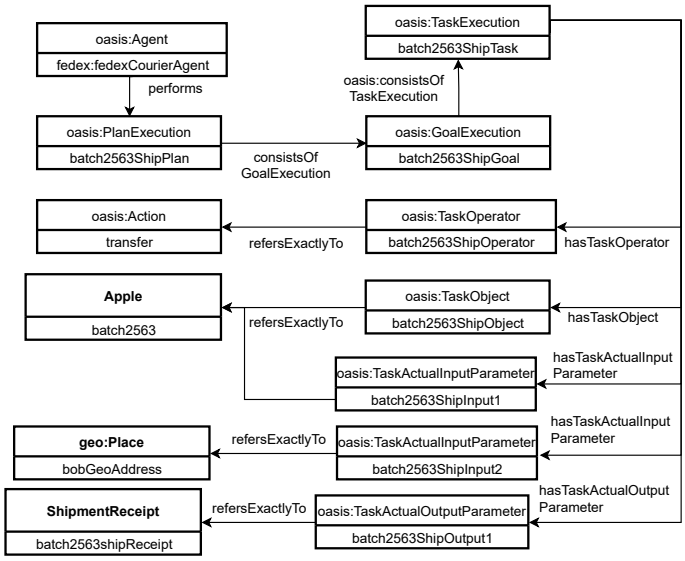

Fig. 20. UML diagram exemplifying the shipment confirmation for apple batch 2563 in the OC-Commerce ontology

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 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 Figure 21.

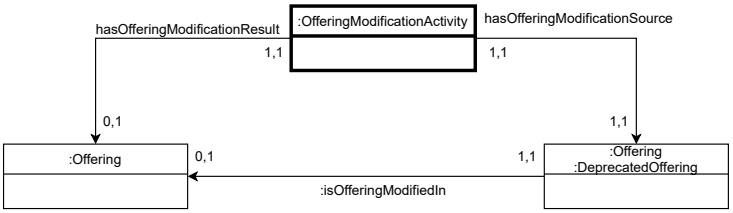

Fig. 21. UML diagram representing OC-Commerce offering modification

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 should 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, 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 Figure 22, in a way analogous as in counter-offerings.

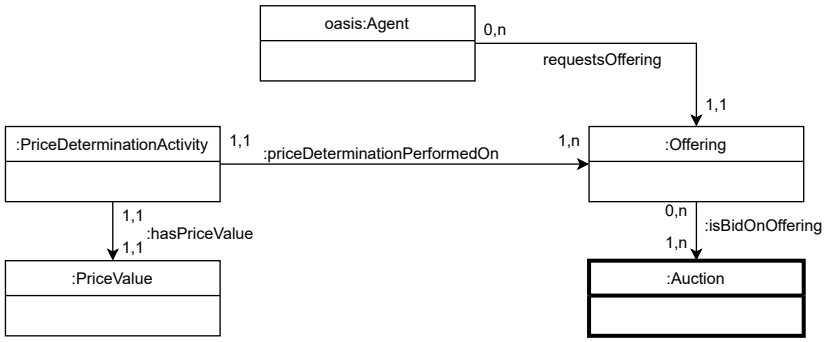

Fig. 22. UML diagram representing OC-Commerce auctions

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 [6] ontology. OC-Ethereum conjoins the BLONDiE ontology with the OC-Commerce ontology in order to provide a behavioristic vision of commercial activities, in particular of those exploiting the management of tokens for trading purposes carried out through the Ethereum blockchain.

The BLONDiE ontology is summarized in the UML diagram in Figure 23.

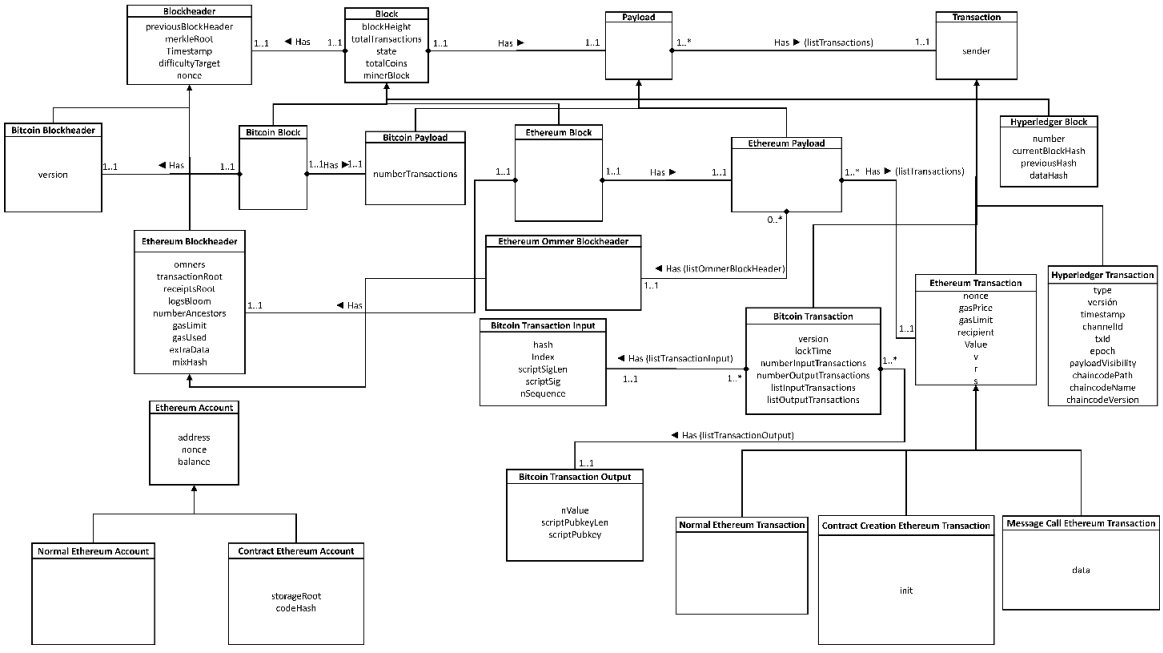

Fig. 23. UML diagram representing the BLONDiE ontology

Classes and properties defined in OC-Ethereum are depicted in Figure 24 and Figure 25, respectively.

Fig. 24. Hierarchies of classes in the OC-Ethereum ontology

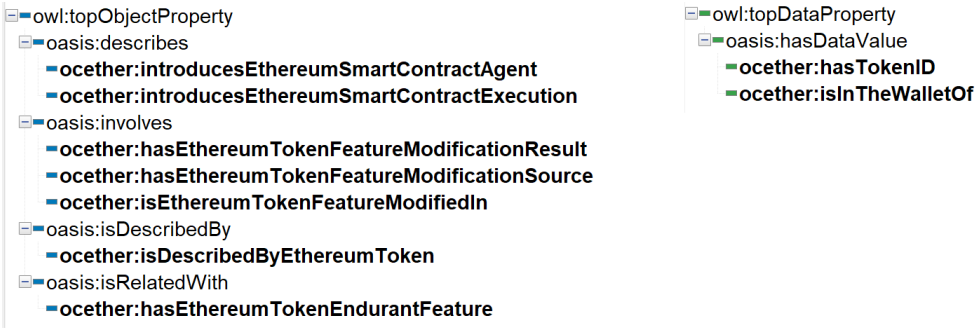

Fig. 25. Hierarchies of object-properties (on the left) and of data-properties (on the right) in the OC-Ethereum ontology

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 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 Figure 26. 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.

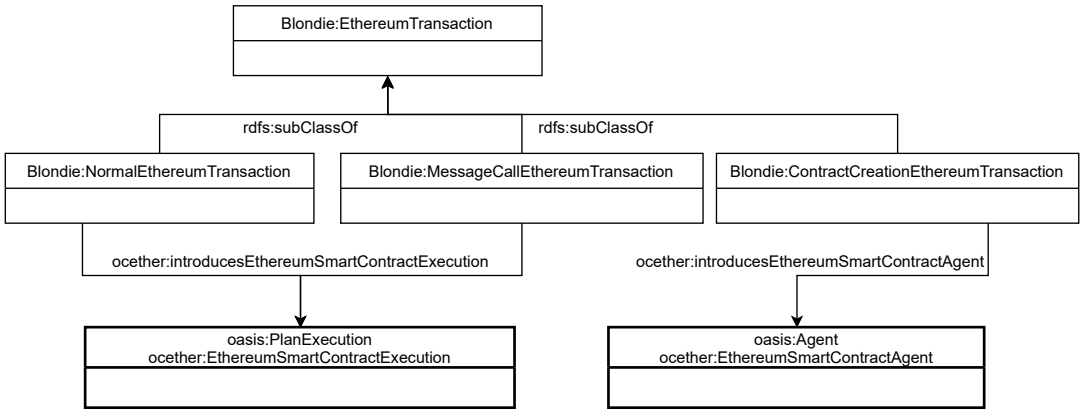

Fig. 26. UML diagram representing OC-Ethereum smart contracts

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

- *EthereumFungibleSmartContractAgent*, representing fungible smart contracts and containing the class *EthereumERC20SmartContractAgent* that represents smart contracts compliant with the ERC20 standard protocol;
- *EthereumNonFungibleSmartContractAgent*, representing non-fungible smart contracts and containing the class *EthereumERC721SmartContractAgent* that represents smart contracts compliant with the ERC721 standard protocol.
- *EthereumSemiFungibleSmartContractAgent*, representing semi-fungible smart contracts and containing the class *EthereumERC1155SmartContractAgent* 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 instance, the smart contract generated by our apple producer is represented by means of the fragment depicted in Figure 27.

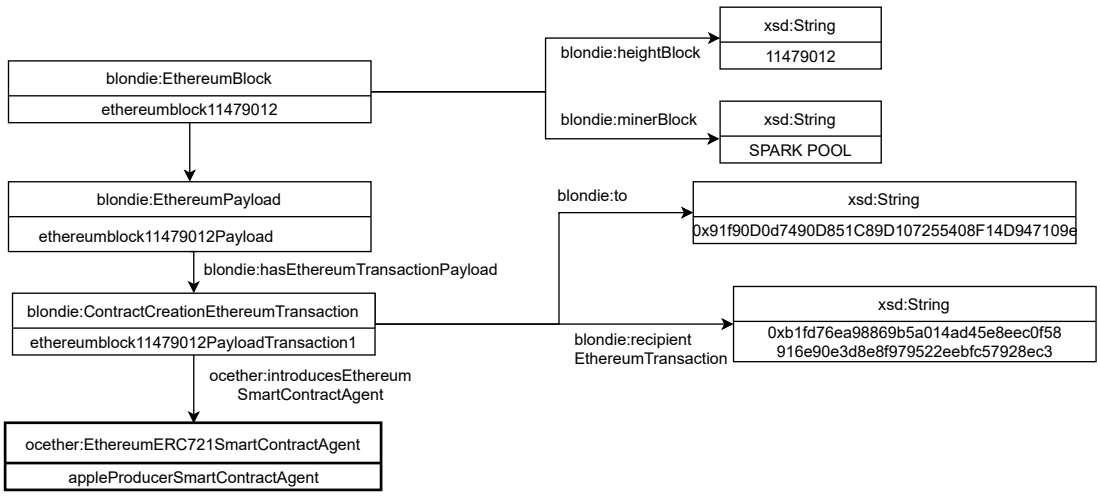

Fig. 27. UML diagram exemplifying an apple producer's smart contract in the OC-Ethereum ontology

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 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.

Moreover, the OC-Ethereum ontology extends BLONDiE by describing tokens generated and exchanged through the Ethereum blockchain, in particular, for commercial purposes. As illustrated in Figure 28, OC-Ethereum identifies four main types of token:

- *fungible tokens*, represented by instances of the class *EthereumFungibleToken*, the latter containing the class *EthereumTokenERC20* that represents fungible tokens compliant with the ERC20 standard protocol;
- *non-fungible tokens*, represented by instances of the class *EthereumSemiFungibleToken*, the latter containing the class *EthereumTokenERC721* that represents non-fungible tokens compliant with the ERC721 standard protocol;
- *semi-fungible tokens*, represented by instance of the class *EthereumSemiFungibleToken*, the latter 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*.

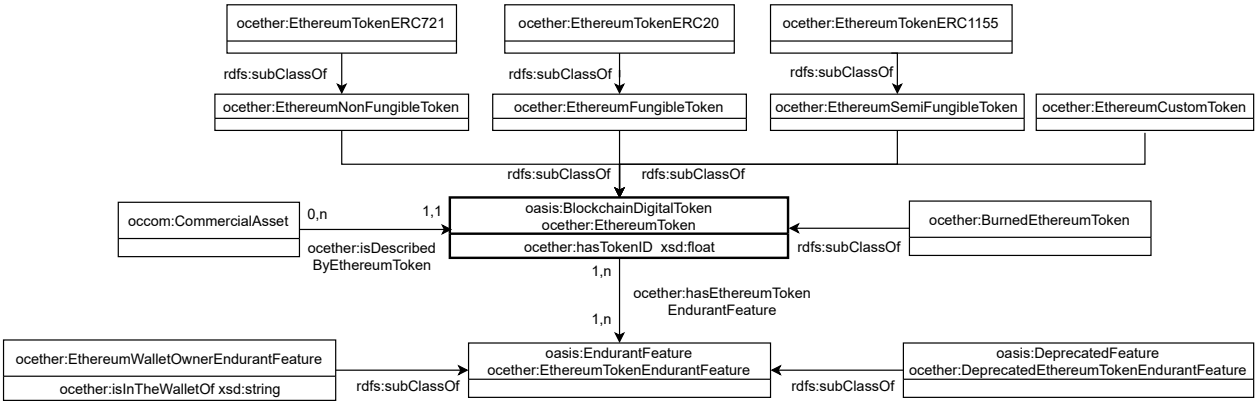

Fig. 28. UML diagram representing OC-Ethereum tokens

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

In OC-Ethereum, commercial assets (instance 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 [69], namely a) perdurant features, such as the token ID, that never change and are embedded with the entity representing the token, and b) endurant features that change during the life-span of the token and are associated with an instance of the OC-Ethereum class *EthereumTokenEndurantFeatures* (subclass of the OASIS class *EndurantFeature*) by means of the object-property *hasEthereumTokenEndurantFeature*. The most notable subclass of *EndurantFeature* is the class *EthereumWalletOwnerEndurantFeature*, which describes the wallet of the token's owner (by means of the data-properties *isInTheWalletOf*, having as range *XSD:string*). When the endurant features of a token are modified by the smart contract

managing it, they became deprecated and replaced by a new set of features by means of a modification activity. Those new features are introduced by a fresh instance of the class *EndurantFeature* as illustrated in Figure 29.

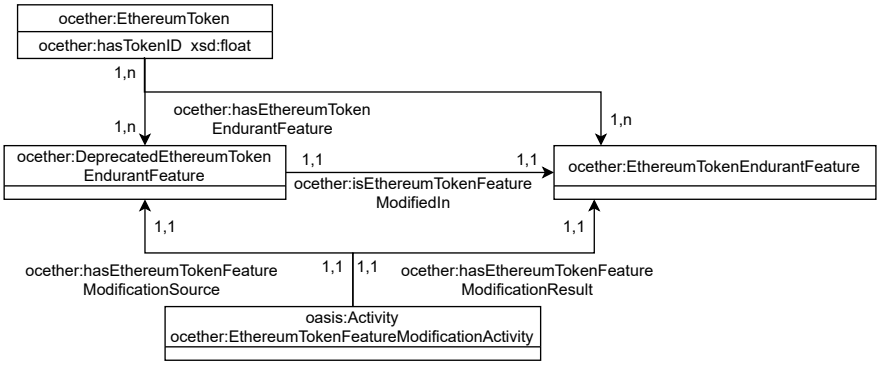

Fig. 29. UML diagram representing OC-Ethereum token modification

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

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

Moreover, the modified endurant feature is connected with the endurant feature that replaces it by means of the object-property *isEthereumTokenFeature ModifiedIn*, whereas the token embedding the features is connected with the new endurant feature by means of the object-property *hasEthereumTokenEndurantFeature*, as usual.

An example of representating tokens in OC-Ethereum is depicted in Figure 30, which shows a token emitted by the apple producer's smart contract.

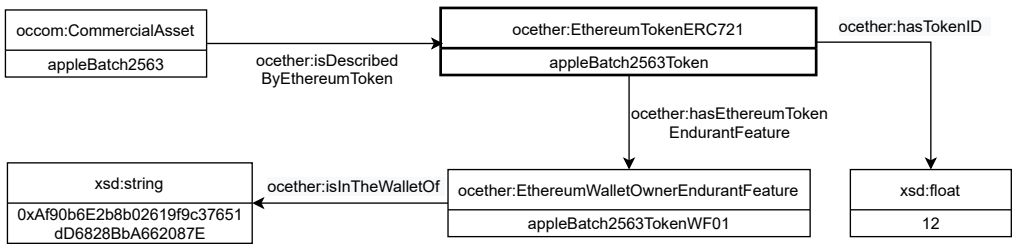

Fig. 30. UML diagram exemplifying the publication of the token for apple *batch 2563* in the OC-Ethereum ontology

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

5. Evaluation of the ONTOCHAIN ontological stack

The ontological stack was evaluated during the entire life-time of its 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 demon-

strating 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 that 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 exploiting the OntoQA approach [70]. OntoQA is a feature-based method for evaluating ontologies by applying techniques that do not require data training and that involve users in a minimal way. The evaluation took into account all the schema metrics defined in 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 other than class/subclass relations is richer than a taxonomy (with only class/subclass relationships). The value of relationship richness is a percentage representing how much relationships between classes are rich with respect to all of 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 measure can help to distinguish horizontal ontology either from a vertical ontology, or from an ontology with different levels of specialization. A horizontal (or flat) ontology has a small number of inheritance levels, and each class has a relatively large number of subclasses. On the contrary, a vertical ontology contains a large number of inheritance levels where classes have a small number of 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 much 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.

In addition, a suitable set of *competency questions* were defined for the ontological stack. 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. Competency questions are implemented into 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 competency questions defined and the related SPARQL queries 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 ontology OC-Found

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 competency questions and the related implementation as SPARQL queries.

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 [71], HermiT [72] and FaCT++ [73]. 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 of the entities provided by OASIS and hence the size different of the two ontology is appreciable.

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 1: Structural metrics on OC-Found

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

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%

Table 2: Ontological metrics on OC-Found

OC-Found reports a relationship richness of 57.14% that depicts a good balance between generic relationships and class hierarchies. The value obtained from OC-Found is higher than the one computed on OASIS since it consider 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 motivated by the fact 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 ontology concentrated on a relatively small domain is also confirmed by its low class richness (11.11%) and attribute richness (that is 0.61%), which are values close to those for OASIS. Finally, OC-Found provides a tree balance of 0.90% meaning that some hierarchies have been well described (e.g., *SupplyChainActivity*), 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. Competency questions

We now discuss the competency questions for the OC-Found ontology. Specifically, we defined the following competency questions for OC-Found (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 above question 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. The answer to CF1 is entailed by the following SPARQL query (QF1):

Query 1 The query QF1.

```
SELECT DISTINCT ?agent ?identity ?operation ?operationOn
WHERE { ?agent ocfound:hasDigitalIdentity ?identity .
        ?agent ocfound:hasDigitalIdentity ?identity.
        ?agent oasis:hasBehavior ?behavior.
        ?behavior 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 actions have been performed (including the agents responsible for the execution and the type of the action performed)?

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 life-span of the available agents. Hence, CF2 allows one to understand how the environment evolved as consequence of the available agents' commitments. The answer to CF2 is entailed by the following SPARQL query (QF2):

Query 2 The query QF2.

```
SELECT DISTINCT ?agent ?operation ?operationOn ?typeOf
WHERE { ?agent oasis:performs ?agentExe.
        ?agentExe oasis:hasTaskObject ?taskExe.
        ?agentExe oasis:hasTaskOperator ?operator.
        ?operator oasis:refersExactlyTo ?operation.
        ?taskExe oasis:refersExactlyTo ?operationOn.
        ?operationOn a ?typeOf.
        FILTER( ?typeOf != owl:NamedIndividual)
```

- 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. The answer to CF3 is entailed by the following SPARQL query (QF3):

Query 3 The query QF3.

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

- 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. The answer to CF4 is entailed by the following SPARQL query (QF4):

Query 4 The query QF4.

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

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

In line with CF3 and CF4, the competency questions CF5 investigates on the agents that are responsible for the supply chain of a given resource. The identity and the behavior of the agent is useful for the clients to establish whether they may trust the supply chain and, hence, if the resource is reliable. The answer to CF5 is entailed by the following SPARQL query (QF5):

Query 5 The query QF5.

```
Let resource be the resource of which the supply chain activity agents should be discovered
   SELECT ?agent ?supplychainActivity
   WHERE { resource ocfound:hasSupplyChainManagement ?supplychain.
                ?supplychain ocfound:hasSupplyChainActivity ?supplychainActivity.
                ?supplychainActivity ocfound:supplyChainActivityImplementedBy ?behavior.
                ?behavior a oasis:Behavior.
                ?agent oasis:hasBehavior ?behavior. }
```

- 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. The answer to CF6 is entailed by the following SPARQL query (QF6):

Query 6 The 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. }
```

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

Together with CF5 and CF6, the competency question assists the user on valuating the trustworthiness level on a resource by computing the average score of valuation for a given resource. The answer to CF7 is entailed by the following SPARQL query (QF7):

Query 7 The 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 ontology OC-Commerce

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 competency questions and their implementation as SPARQL queries.

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.

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 3: Structural metrics on OC-Commerce

The difference of dimension between OC-Commerce and GoodRelations is the result of the import 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.

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.

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%

Table 4: Ontological metrics on OC-Commerce

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 import 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. Competency Questions

We now discuss the competency questions defined for the OC-Commerce ontology. Specifically, we provide the following three main competency questions concerning how offerings and related information can be identified and consumed.

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

Competency question CC1 provides a complete set of the available offering that clients can consume. The answer to CC1 is entailed by the following SPARQL query (QC1):

Query 8 The query QC1.

```
1:  SELECT DISTINCT ?offering ?type ?value ?currency
2:   WHERE {
3:                ?taskExec a oasis:TaskExecution.
4:                ?taskExec oasis:hasTaskObject ?taskob.
5:                ?taskob oasis:refersExactlyTo ?offering.
6:                ?offering a ?offer.
7:                FILTER(?offer = occom:Offering)
8:                FILTER NOT EXISTS { ?offering a occom:DeprecatedOffering.}
9:                FILTER NOT EXISTS { ?offering a occom:ClosedOffering.}
10:                FILTER NOT EXISTS { ?offering a occom:RetractedOffering.}
11:                ?product a ?type.
12:                FILTER( ?type != owl:NamedIndividual)
13:                ?priceDetActivity occom:priceDeterminationPerformedOn ?offering.
14:                ?priceDetActivity occom:hasPriceValue ?price.
15:                ?price gr:hasCurrencyValue ?value.
16:                ?price gr:hasCurrency ?currency. }
```

- 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. The answer to CC2 is entailed by the following SPARQL query (QC2):

Query 9 The 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 ?behavior.
                ?agent oasis:hasBehavior ?behavior.}
```

- 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 life-span and hence what type of resources have been traded. These are entailed by the following SPARQL query (QC3):

Query 10 The 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 ontology OC-Ethereum

Next, we introduce the evaluation methods and related results obtained for the OC-Ethereum ontology. We first present the ontological metrics and then the competency questions and their implementations 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.

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 5: Structural metrics on OC-Ethereum

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.

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

Table 6: Ontological metrics on OC-Ethereum

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, 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 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 instance between the class hierarchy of *EthereumToken* and the class hierarchy of *EthereumTokenEndurantFeature*. 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. Competency Questions

We are now ready to discuss the competency questions for the OC-Ethereum ontology. OC-Ethereum answers at least to the four following competency questions, in addition to those provided by BLONDiE (see [6]).

- 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. The answer to CE1 is entailed by the following SPARQL query (QE1):

Query 11 The 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:hasEthereumTokenEndurantFeature ?feature.
       ?feature a ?ownerFeature.
       FILTER(?ownerFeature = ocether:EthereumWalletOwnerEndurantFeature)
       FILTER NOT EXISTS {?feature a ocether:DeprecatedEthereumTokenEndurantFeature.}
       ?feature ocether:isInTheWalletOf ?owner. }
```

- 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. The answer to CE2 is entailed by the following SPARQL query (QE2):

Query 12 The 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. }
```

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

The competency questions CE3 allows users to access the smart contracts that generate the tokens associated with the desired type of digital or physical asset. The answer to CE3 is entailed by the following SPARQL query (QE3):

Query 13 The 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 competency questions 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. The answer to CE4 is entailed by the following SPARQL query (QE4):

Query 14 The query QE4.

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

6. A real-world case study: mapping the iExec marketplace on the ONTOCHAIN ontological stack

In this section, we show how to apply the ONTOCHAIN ontological stack to map a real world use case, namely the iExec marketplace [15]. We first provide an overview of the basic concepts and functioning of the iExec marketplace, then we illustrate in detail the mapping of its main features to the ontological stack.

6.1. Outlining the iExec marketplace

The *iExec marketplace* connects buyers with sellers of cloud resources. Specifically, cloud resources are of three main types, namely *applications*, *datasets* and *computing resources*.

Applications are standalone computer programs that can be downloaded and executed 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 Figure 31, where

– `app` is a unique identifier for the application, that is the address of the smart contract associated with the 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 that will be described later on;
– `salt` is a random value to ensure order uniqueness;
– `sign` is the EIP712 cryptographic signature [74] of the order.

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

Fig. 31. App order's structure in the iExec marketplace

Dataset providers, namely actors providing datasets via the iExec marketplace, publish on the iExex marketplace *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 (Figure 32) is similar to the structure of app order, and hence their semantics fields can be easily deduced from the analogous app order fields.

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

Fig. 32. Dataset order's structure in the iExec marketplace

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*) [75, 76]. 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 [15].

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* (Figure 33). 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;

```
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. 33. Worker pool order's structure in the iExec marketplace

– `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, create a *request order* (see Figure 34) 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 [77] for details).

```
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. 34. Request order's structure in the iExec marketplace

6.2. Mapping the iExec marketplace

We now describe a map for representing the iExec marketplace by leveraging the ONTOCHAIN 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 map 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 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 Figure 34, executions have also some optional fields mapped to the suitable properties, grouped as *optional execution properties*. Specifically, the map 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 ONTOCHAIN ontological stack. For *AppOrder*, *DatasetOrder*, and *WorkerPoolOrder*, respectively illustrated in Figures 31, 32 and 33, 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. Thus, defining supply chains for these assets would not be appropriate. 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 Figure 35. The supply chain of the worker pool offering will be explained later on. The workerpool offering is accepted by clients through a client's *accept* behavior 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 mapping of a request order object to a 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 Figure 36. 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, a *iexec:DatasetOffering*, corresponding to a dataset orde,
– a *iexec:WorkerPoolOffering*, corresponding to a worker pool order,
– a *iexec:RequestOrder*, corresponding to the execution request.

The iExec clerk behavior 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 exploiting the iExec deal. The worker pool agent, whose behavior is depicted in Figure 37, 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 *ProofOfWork* supply chain of the worker pool order is related with the behavior of the iExec clerk agent (see Figure 36), whereas the Release supply chain consists of an activity exploiting the behavior of the worker pool agent (Figure 37). 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.

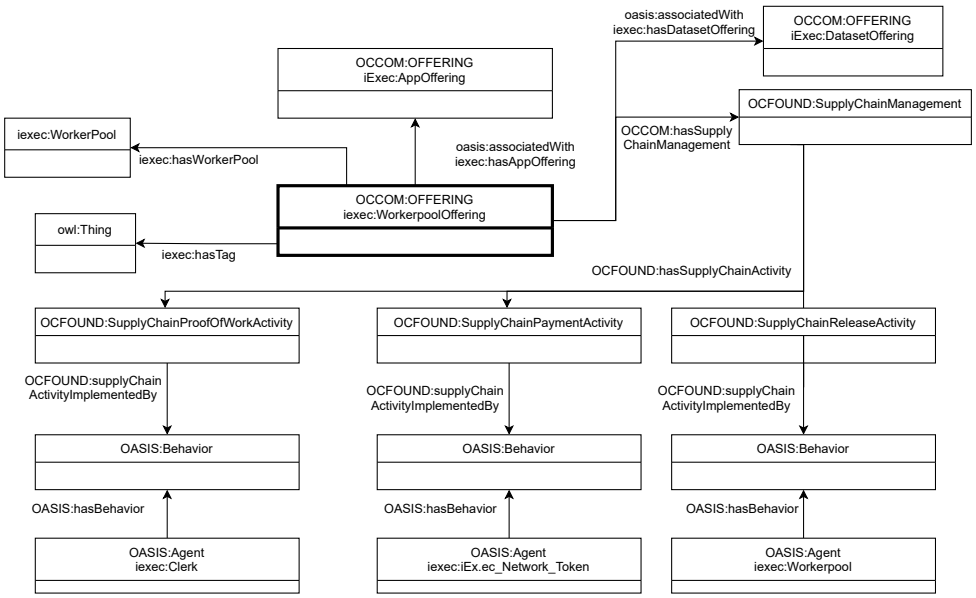

Fig. 35. UML diagram mapping the worker pool offering

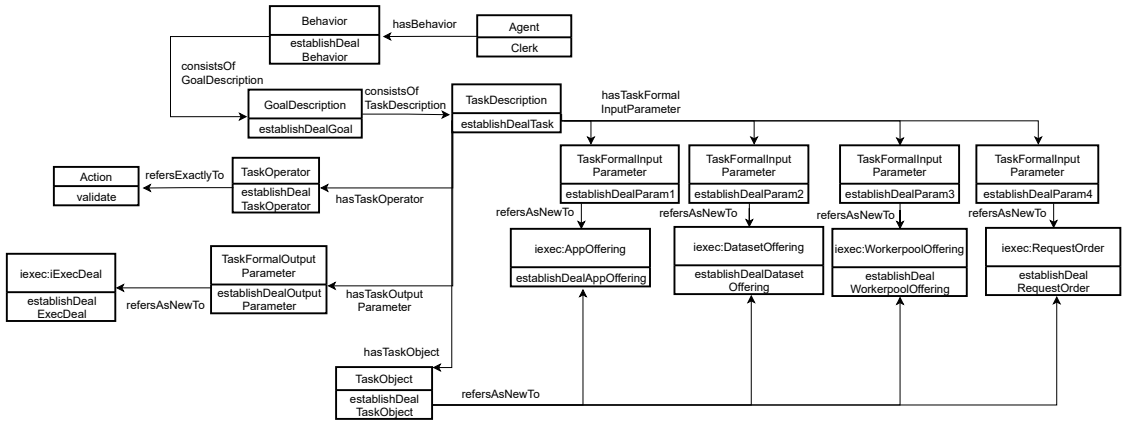

Fig. 36. UML diagram mapping the iExec clerk agent

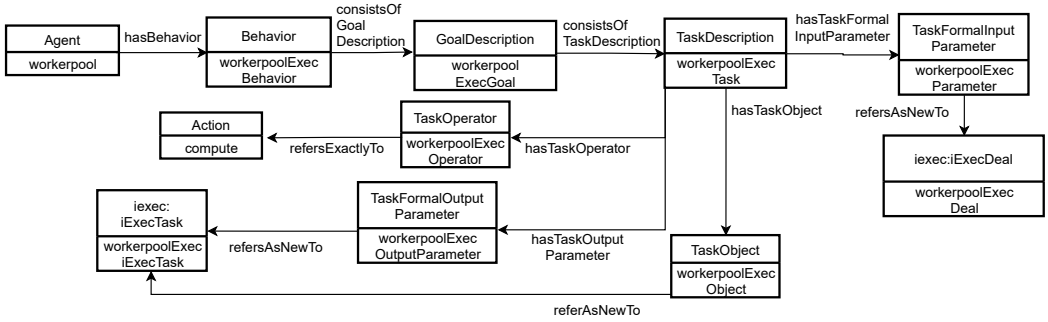

Fig. 37. UML diagram mapping the iExec worker pool agent

7. Conclusions

We presented a behavioristic approach for semantically representing blockchain-oriented e-commerce that uses the token generation and exchange mechanisms as decentralized and public proofs. The epistemological process results in a family of ontologies, namely the ONTOCHAIN ontological stack, that includes three ontologies. The first one, 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 ends 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.

Therefore, this article achieves a formal semantic representation capturing the smart contracts on the blockchain as well as the activities carried out on it. This facilitates the understanding of blockchain concepts, the interlinking with other out-of-chain information and also formal reasoning. A semantic behavioristic 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 facility of token exchange systems is indeed 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., of the type of exchanged asset, of the exchange of particular tokens at certain conditions, or of 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.

The results presented in this article are not conclusive. A semantic search engine is under development, currently standing at TRL3, in the context of the NGI-ONTOCHAIN project that funded the realization of the ontological stack. Other future work is clearly defined. OC-Found will include mechanisms for realising digital identities for people and applications selected by the consortium. The ontology OC-Commerce will be extended so as to be independent from GoodRelations with new modelling approaches for bargaining offerings. OC-Ethereum will be extended with new blockchains and behaviors 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 representation, for example in the style of the mapping presented in this article. The experience we have gained thus far as well as the overall flexibility of our approach make us optimistic on the successful pursuance of these research and development directions.

Acknowledgements

This work 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. We are also thankful to the Sicilian Wheat Bank S.p.A, and to Dr. Cristiano Longo in particular, for their relevant participation in our project.

References

[1] N. Goldmann, E-commerce, *Journal of Internet Banking and Commerce* **26**(3) (2021), 286–295.
[2] K. Christidis and M. Devetsikiotis, Blockchains and Smart Contracts for the Internet of Things, *IEEE Access* **4** (2016), 2292–2303.
[3] N. Szabo, Formalizing and Securing Relationships on Public Networks, *First Monday* **2**(9) (1997).
[4] 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/.
[5] 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. ISBN 978-3-540-87695-3. http://dblp.uni-trier.de/db/conf/ekaw/ekaw2008.html#Hepp08.
[6] H. Ugarte-Rojas and B. Chullo-Llave, BLONDiE: Blockchain Ontology with Dynamic Extensibility, *CoRR* **abs/2008.09518** (2020). https://arxiv.org/abs/2008.09518.
[7] J. Pfeffer, A. Beregszazi and S. Li, Ethon - an Ethereum ontology, 2016, available on-line: https://ethon.consensys.net/index.html.
[8] N.R. Jennings and M.J. Wooldridge (eds), *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag, Berlin, Heidelberg, 1998. ISBN 3540635912.
[9] Y. Shoham, Agent-oriented Programming, *Artificial Intelligence* **60**(1) (1993), 51–92.
[10] 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.
[11] 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: *Proceedings of the 20th Workshop From Objects to Agents, 26-28 June, 2019, Parma, Italy, CEUR Workshop Proceeding Vol. 2404*, 2019, pp. 119–126.
[12] NGI-ONTOCHAIN, ONTOCHAIN A new software ecosystem for trusted, traceable & transparent ontological knowledge, 2020. https://ontochain.ngi.eu/.
[13] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo and D.F. Santamaria, Semantic Representation as a Key Enabler for Blockchain-Based Commerce, *K. Tserpes et al. (Eds.): GECON 2021, Springer Lecture Note in Computer Science* **13072** (2021), 1–8.
[14] G. Bella, D. Cantone, C. Longo, M. Nicolosi Asmundo and D.F. Santamaria, POC4COMMERCE - Practical ONTOCHAIN for E-Commerce - Project Page, 2021. https://github.com/dfsantamaria/POC4COMMERCE.
[15] iExec Team, iExec, 2017. https://iex.ec/.
[16] J. Hendler, Agents and the Semantic Web, *IEEE Intelligent Systems* **16**(2) (2001), 30–37.
[17] World Wide Web Consortium, OWL-S: Semantic Markup for Web Services, 2004. https://www.w3.org/Submission/OWL-S/.
[18] 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. https://doi.org/10.1007/0-387-23344-X_3.
[19] N. Fornara and M. Colombetti, A Commitment-Based Approach To Agent Communication, *Applied Artificial Intelligence* (2004), 853–866.
[20] M. Esteva, Electronic Institutions: from specification to development, *IIIA Monograph Series. PhD Thesis* **19** (2003).
[21] M.P. Singh, *A Social Semantics for Agent Communication Languages*, North Carolina State University at Raleigh, USA, 1999.
[22] M. Hadzic, E. Chang and P. Wongthongtham, *Ontology-Based Multi-Agent Systems*, Springer Publishing Company, Incorporated, 2014. ISBN 3642425496, 9783642425493.
[23] 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.
[24] Q. Tran and G. Low, MOBMAS: A methodology for ontology-based multi-agent systems development, in: *Inf. Softw. Technol., 50(7-8)*, 2008, pp. 697–722.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] M.E. Bratman, *Intentions, Plans and Practical Reason*, Harvard University Press, 1987.

[30] A. Freitas, R.H. Bordini and R. Vieira, Model-driven engineering of multi-agent systems based on ontologies, *Applied Ontology* **12** (2017), 157–188.
[31] 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, Semantic Mashup of Biomedical Data.
[32] 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.
[33] 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.
[34] G. Bajaj, R. Agarwal, P. Singh, N. Georgantas and V.. Issarny, A study of existing Ontologies in the IoT-domain, *hal-01556256* (2017).
[35] 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.
[36] 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.
[37] 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.
[38] 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.
[39] World Wide Web Consortium, Web of Things (WoT) Thing Description, 2020. https://www.w3.org/TR/wot-thing-description/.
[40] J. Jovanovic and E. Bagheri, Electronic Commerce Meets the Semantic Web., *IT Prof.* **18**(4) (2016), 56–65. http://dblp.uni-trier.de/db/journals/itpro/itpro18.html#JovanovicB16.
[41] 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. ISBN 1-58113-332-4. http://dblp.uni-trier.de/db/conf/sigmod/sigmod2001.html#StonebrakerH01.
[42] 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.
[43] 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. ISBN 1-58113-680-3. doi:http://doi.acm.org/10.1145/775152.775250. http://portal.acm.org/citation.cfm?doid=775152.775250.
[44] 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, Beijing, China*, F.C.M. Lau, H. Lei, X. Meng and M. Wang, eds, IEEE Computer Society, 2005, pp. 572–581. doi:10.1109/ICEBE.2005.15.
[45] The United Nations Development Program (UNDP) and Dun & Bradstreet Corporation (D& B), United Nations Standard Products and Services Code (UNSPSC), 2003, (last visit 01/12/21). https://www.unspsc.org/.
[46] The ECLASS organization, Eclass – Standard For Master Data and Semantics For Digitalization, 2000, (last visit 01/12/21). https://www.eclass.eu/en/index.html.
[47] Electronic Commerce Code Management Association (ECCMA), ECCMA Data Requirements Registry (eDRR), 2000, (last visit 01/12/21). https://eccma.org/resources/.
[48] GS1, GS1 Initiative, 1971, (last visit 01/12/21). https://www.gs1.org.
[49] G.M. Organisations, Tag Data Standard 1.13, GS1, 2019. https://www.gs1.org/standards/epcrfid-epcis-id-keys/epc-rfid-tds/1-13.
[50] 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.
[51] 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.
[52] J. Ashraf, O.K. Hussain and F.K. Hussain, Empirical Analysis of Domain Ontology Usage on the Web: ECommerce Domain in Focus, *Concurr. Comput.: Pract. Exper.* **26**(5) (2014), 1157–1184. doi:10.1002/cpe.3089.
[53] 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. ISBN 978-3-319-11963-2. http://dblp.uni-trier.de/db/conf/semweb/iswc2014-1.html#Patel-Schneider14.
[54] 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, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, 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.
[55] M.D. English, S. Auer and J. Domingue, Block Chain Technologies & The Semantic Web: A Framework for Symbiotic Development, 2015.
[56] 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.
[57] J. de Kruijff and H. Weigand, Understanding the Blockchain Using Enterprise Ontology, in: *CAiSE*, 2017.
[58] 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.

[59] 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.
[60] H.E. Ugarte Rojas, A more pragmatic Web 3.0: Linked Blockchain Data, in: *Google Scholar*, 2017.
[61] 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.
[62] H.G. Fill, Applying the Concept of Knowledge Blockchains to Ontologies, in: *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*, 2019.
[63] 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, in: *To app. in: Proceedings of IDC 2021, The 14th International Symposium on Intelligent Distributed Computing, 16-18 September, Scilla, Reggio Calabria, Italy*, 2021.
[64] G. Bella, D. Cantone, C.F. Longo, M. Nicolosi-Asmundo and D.F. Santamaria, Ontological Smart Contracts in OASIS: Ontology for Agents, Systems, and Integration of Services, in: *To app. in: Proceedings of IDC 2021, The 14th International Symposium on Intelligent Distributed Computing, 16-18 September, Scilla, Reggio Calabria, Italy*, 2021.
[65] D. Cantone, C.F. Longo, M. Nicolosi-Asmundo, D.F. Santamaria and C. Santoro, OASIS-Abox ontology. http://www.dmi.unict.it/oasis-abox.owl.
[66] M.A. Musen, The protégé project: a look back and a look forward, *AI Matters* **1**(4) (2015), 4–12.
[67] 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.
[68] M. Wick, Geonames Ontology, 2015. http://www.geonames.org/about.html.
[69] 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. 166–181. ISBN 978-3-540-45810-4. doi:10.1007/3-540-45810-7_18. http://link.springer.de/link/service/series/0558/bibs/2473/24730166.htm.
[70] 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.
[71] 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.
[72] 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.
[73] D. Tsarkov and I. Horrocks, FaCT++ Description Logic Reasoner: System Description, in: *Automated Reasoning*, U. Furbach and N. Shankar, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 292–297. ISBN 978-3-540-37188-5.
[74] Ethereum, EIP-712 Protocol, 2018, (last visit 01/12/21). https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md.
[75] H. Croubois, PoCo Series #1 - About Trust and Agents Incentives, 2017. https://medium.com/iex-ec/about-trust-and-agents-incentives-4651c138974c.
[76] H. Croubois, PoCo Series #3 - Protocol update, 2018. https://medium.com/iex-ec/poco-series-3-poco-protocole-update-a2c8f8f30126.
[77] 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.