D3. 'Design specification with implementation approach'

>POC4COMMERCE<

Grant Agreement No.: 957338
Call: H2020-ICT-2020-1

Topic: ICT-54-2020
Type of action: RIA

>POC4COMMERCE<

D3. 'Design specification with implementation approach'

DUE DATE	06/08/2021
SUBMISSION DATE	06/08/2021
TEAM	POC4COMMERCE
VERSION	1.0
AUTHORS	Giampaolo Bella, Domenico Cantone, Cristiano Longo, Marianna Nicolosi Asmundo, Daniele Francesco Santamaria

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Executive Summary

Blockchains are disrupting the commerce foundations by pursuing new business models based on trustless and decentralized transactions where intermediaries and central authorities are discarded. One of the most notable means are the digital tokens emitted and exchanged on the blockchain to provide digital representations of assets that grant to the owners specific rights publicly verifiable by smart automatic contractual arrangements, namely *smart contracts*. Due to the growing of commercial mechanisms complexity, a clear unambiguous descriptions of commercial participants and their activities, carried out both inside and outside of the blockchain and on top of which trustworthy and affordable applications are constructed, is demanded. Even though such a form of commerce represents a novel business opportunity, a clear understanding of the actors, actions, and actuators involved in commercial activities is missing from literature, due also to the rapid evolution of the involved processes.

POC4COMMERCE innovates the ontological representation of blockchain-oriented digital commerce by integrating and extending the most representative ontologies for modelling, participants, in particular commercial actors, offers, products, and tokens emitted on the Ethereum blockchain as digital representation of exchanged assets: providing a semantic descriptions of smart contracts and related transactions, in particular of smart contracts related with tokens trading and associated with commercial means, lies at the core of POC4COMMERCE's vision. To achieve the goal, POC4COMMERCE pursues a deep level of interoperability of well-assessed ontologies for commercial participants, blockchains, and commercial offers. Specifically, POC4COMMERCE reuses and extends the most suitable and relevant ontologies in the domain, namely, OASIS [2] for the representation of commercial participants and smart contracts, GoodRelations [13] for representing commercial offers, and BLONDiE [24] for describing Ethereum essential elements: all these ontologies are conjoined and extended to also cover the gap missing from the literature on the representation of digital tokens, smart contracts, digital identities, valuation mechanisms, and auctions.

As result of the POC4COMMERCE effort, an ontological stack of three ontologies is deployed. The first ontology is **OC-Found**, modelling the stakeholders of the ONTOCHAIN ecosystem, vertically focusing on the commercial domain, and exploiting the OASIS ontology; the second ontology, called **OC-Commerce** and extending GoodRelations, is responsible for describing commercial offerings, assets, and means; the third ontology, called **OC-Ethereum** and exploiting BLONDiE, is focused on the representation of Ethereum smart contracts and related tokens compliant with the standard ERC20, ERC720, and ERC1155. The evaluation approach adopted for the ontological stack of POC4COMMERCE consists of three phases: **Consistency checking**, carried by three widespread *Web Ontology Language (OWL) 2* [14] (OWL 2) reasoners, verifies the consistency of both

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

the single ontologies and the entire stack; **Ontological and structural metrics**, required to check the assessment, domain coverage, and suitability of the ontologies, is carried by exploiting the most suitable criteria in literature [21]; finally, suitable competency questions are designed and implemented through SPARQL queries to verify whether the ontologies are being developed towards the project objectives and reaching the stated representational goals.

The next phase of the POC4COMMERCE project foresees a data-population step consisting of a small amount of data from iExec to check the ontological stack adherence to a real-word case. During this step, the semantic search engine **OC-Commerce Search Engine** is designed in light of the recent advancements of the ONTOCHAIN ecosystem.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

List of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

List of Tables

Abbreviations

IPFS - Interplanetary File System [18].
NFT - Non-fungible token.
OWL - Web Ontology Language [14].
RDF - Resource Description Framework [16].
SPARQL - SPARQL Protocol and RDF Query Language [27].
W3C - World Wide Web Consortium.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

1 Introduction

The Semantic Web is conceived to deliver appropriate representation languages that describe the elements of the observed world through a formally defined semantics. In particular, the modern digital commerce demands an ontological approach leveraging a hierarchical semantic modelling towards the effective and efficient interoperability of blockchain technology with the commercial domain, from actors to supply chains, including distribution, pricing, and selling mechanisms. POC4COMMERCE addresses the stated challenge of developing a consistent, unambiguous, and shared model supporting the semantic interoperability of the ONTOCHAIN heterogeneous stakeholders, by delivering a stack of the most appropriate ontologies for the building blocks of the ONTOCHAIN ecosystem in general and of eCommerce in particular. Remarkably, the ontological stack allows developers and users to easily extend the number of layers, thus covering different aspects of the digital commerce domain such as bidding mechanisms, payment systems, and delivering options, or modelling new domains such health, science, smart cities, and government.

The relevant literature is rich in representation models that can be combined and appropriately extended to meaningfully describe digital marketplaces. POC4COMMERCE is fully aware of the modern literature and embraces it in an ontological engineering process to promote an interoperable and sustainable shared eCommerce in the ONTOCHAIN ecosystem.

Concerning the description of services and agents, there are several approaches, such as OWL-S [26], an ontology for describing web services and the means by which they are accessed. OWL-S tries to enable several tasks such as automatic web services discovery, automatic web services invocation, and automatic web services composition and interoperation. As from 2000, many results concerning how agents enter and leave different interaction systems have been provided both by exploiting *commitment objects* [9] and *virtual institutions* [7]. Approaches analogous to those of *Agent-Oriented Software Engineering* (AOSE) [6, 5] have been applied to model ontologies for MAS. Other approaches inspired by *belief-desire-intention* (BDI) [19] agent architectures are proposed in [1] and [8]. In [10], 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 great enough detail. Moreover, an appropriate modelling of agent communication is missing. In [4], the authors introduce an approach to design scalable and flexible agent-based manufacturing systems that integrates automated planning with multi-agent oriented programming for the *Web of Things* (WoT), where autonomous agents synthesize production plans using semantic descriptions of web-based artifacts and coordinate with one another via multi-agent organizations.

To build a semantic digital commerce for ONTOCHAIN, POC4COMMERCE demands a representation system able to describe commercial partners, such as products and service providers, in

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

each phase of the supply chain. Indeed, a digital interconnected marketplace not only requires a complete description of all stakeholders such as providers and goods, but also of supply and selling mechanism, offers, auctions, payment tools, transportation, and other related services and activities. For this reason, POC4COMMERCE adopts the OASIS ontology and applies it to the realm of digital commerce related with blockchains. However, OASIS only provides the essential elements to describe agents in terms of their capabilities of contributing to a shared system. For this reason POC4COMMERCE extends OASIS as to include the description of commercial parties involved in the ONTOCHAIN ecosystem, supply chains, and quality valuation mechanisms. In this sense, the representation model provided by POC4COMMERCE gives a novel conception of digital commerce with respect to the state of the art, where actors are described through their roles in each step of the life-cycle of the commercial activity.

In terms of semantic representation of commercial activities, the most notable effort is *GoodRelations*, an OWL vocabulary to describe offerings of tangible goods and commodity services, recognized by major search engines such as Google,[1] Yahoo,[2] and Bing,[3] and supported by content management tools such as Joomla!,[4] osCommerce,[5] and Drupal.[6] In order to fully describe commercial offers and provide product descriptions, POC4COMMERCE adopts the GoodRelations ontology, which represents a landmark in the domain, and extends it with the modelling of product/service exchange, auctions, and digital representation of assets by means of tokens emitted on the blockchain.

Concerning blockchains, the *Blockchain Ontology with Dynamic Extensibility* (BLONDiE) [23] project provides a comprehensive vocabulary that covers the structure of different components (wallets, transactions blocks, accounts, etc.) of blockchain platforms (Bitcoin, Ethereum, and Hyperledger) and that can be easily extended to other alternative systems. It can be observed that the literature lacks a detailed description of smart contracts and their functionalities, including descriptions of digital tokens, but that are crucial for the digital marketplace conceived by ONTOCHAIN. POC4COMMERCE fills the gap left by literature by absorbing BLONDiE into a representation model describing smart contracts as agents participating to the ONTOCHAIN ecosystem and managing tokens compliant with the standard protocols ERC20, ERC721, ERC1155: no full-fledged blockchain-enabled marketplace will get established without the feature to associate tokens with product instances as well as with smart contracts that were used to generate, trade, and manipulate

[1] http://www.google.com
[2] www.yahoo.com
[3] www.bing.com
[4] https://www.joomla.org/
[5] http://www.oscommerce.com/
[6] https://www.drupal.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

those tokens.

2 The Approach

The semantic representation of the ONTOCHAIN ecosystem, with a focus on the commercial vertical domain, is pursued by POC4COMMERCE through an ontological stack constituted by three main layers of OWL 2 ontologies extendable to cover more knowledge domains or to characterize more precisely the available ones. POC4COMMERCE relies on well consolidated standard technologies to deliver the ontological stack. Specifically, the ontologies in the POC4COMMERCE stack are implemented by exploiting the Semantic Web language OWL 2, the *World Wide Web Consortium* (W3C) current standard for ontology engineering and developing, and serialized in RDF/XML, one of the standard serializations for both the *Resource Description Framework (RDF)* [16] and OWL languages. POC4COMMERCE provides an ecosystem of modular ontologies describing each semantic compartment of eCommerce, in particular, eCommerce carried out through and by means of blockchains. The advantage of ontological approaches relies on a high versatility and technological independence, allowing one to deliver ontologies by exploiting a wealth of different suitable implementation infrastructures. For instance, the family of ontologies developed by POC4COMMERCE remains adequate and flawlessly continues to provide the desired semantic representation whether the corresponding knowledge base management system is centralized on a single data provider or is distributed over a peer-to-peer network. The ontologies are constructed exploiting the TRL4 ontologies OASIS [2], BLONDiE [23], and the TRL9 ontology GoodRelations [13], thus ensuring the solid foundation of the ontological stack. Moreover, OASIS, BLONDiE, and GoodRelations are totally independent ontologies, freely adoptable for business purposes, and conceived for different missions: they are adopted by POC4COMMERCE to define a modular representation of the ONTOCHAIN world to be modelled, focused on the commercial domain but extremely relevant for any ONTOCHAIN deployment site. Indeed, the ontologies are adopted to define three distinguished, extendable, modifiable, and customizable layers for three knowledge domains, namely, agents and their interactions, digital commerce, and blockchains. The ontological stack is organized as follows.

The first layer, constituted by the **OC-Found** ontology, exploits the OASIS ontology to semantically describe the stakeholders of the ONTOCHAIN ecosystem, vertically focused on the commercial environment, including agents, participants, digital identities, supply chains, and quality valuation mechanisms. An ontology for the ONTOCHAIN stakeholders, and in particular for the main commercial actors, indeed, must be able to describe how participants contribute to the ecosystem, what they are capable of doing, and how they perform the proposed operations. This is crucial in terms

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

of commercial transactions, since the entire supply chain of exchanged assets, i.e., products and services provided, asset release mechanisms, and payment methods are fully semantically represented, hence unambiguously published and shared. The TRL4 ontology OASIS satisfies the objectives set by OC-Found, since it provides a general description of agents, conceived as entities performing certain actions in terms of the goals and tasks they accomplish: OASIS is adopted by OC-Found a) to provide the effective descriptions of the participant behaviors exploited to build the semantic representation of the agents joining the ecosystem; b) to model the actions that agents perform, thus entirely describing changes and evolution of the environment; c) to describe supply chains, in particular the ones associated with commercial offerings, in terms of the agents realizing the activities related with the supply chains. OC-Found additionally introduces digital identities connected with agents in such a way as to abstract from the implementation of the digital identity, and quality valuation both performed by specialized quality valuer agents and final commercial users, to provide the ecosystem with an experience-based trustworthiness mechanism.

The second layer, constituted by the **OC-Commerce** ontology, is responsible for describing commercial offers and assets by inheriting OC-Found and extending the GoodRelations ontology. Since GoodRelations is not capable of describing participants and how they provide supply chains, OC-Commerce conjoins the agent and supply chain representational model pursued by C-Found with the commercial offering schema introduced by GoodRelations. Moreover, OC-Commerce extends GoodRelations with auctions, bargains, and price determination mechanisms: OC-Commerce, from this point of view, is a specific extension of GoodRelations in support for the linking of offerings and product descriptions, on one side, with the related supply chain and agents responsible for it, on the other side with their digital representation on the blockchain.

Blokchains are the domain of the third layer, which absorbs OC-Commerce (and hence OC-Found) and the BLONDiE ontology to build the **OC-Ethereum** ontology. OC-Ethereum provides a clear description of the Ethereum blockchains including Ethereum smart contracts used to generate, destroy, and exchange tokens compliant with the Ethereum standards ERC20, ERC720, and ERC1155, in particular, those related with assets, such as products and services, involved in commercial exchanges, along with the blockchain operations performed to manage them.

Finally, on top of the ontological stack, POC4COMMERCE sketches a TRL2 semantic search engine called **OC-Commerce Search Engine** (OC-CSE, in short), designed on the most recent programming languages and Semantic Web APIs, together with SPARQL engines, to profitably find goods, products, information, and services, meeting the final user requirements and published by the wide array of ONTOCHAIN commercial participants. OC-CSE builds suitably constructed queries, implemented on well-founded competency questions for the three ontological layers, representing final user requirements and search desiderata that are injected to the underlying data-management systems:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

the search engine enables the interoperability of commercial parties, whose businesses would have been disconnected otherwise, favouring the spread of products and services through the ONTOCHAIN ecosystem and reducing economic inefficiencies.

The POC4COMMERCE ontological approach is evaluated during its entire lifetime through the KPIs stated in the original project proposal, namely a) *consistency check* performed by wide-spread OWL 2 compliant reasoners; b) *ontological and structural metric criteria* that guarantee a high-quality representation system for the ONTOCHAIN ecosystem; c) *competency questions* ensuring the suitability of the ontologies and related *SPARQL-SPARQL Protocol and RDF Query Language* [27] queries implementing the competency questions and performed over the ontology, which constitutes also regression and integration tests for the ontological stack.

3 Detailed Software Design

In this section we describe in details the ontological stack provided by POC4COMMERCE by presenting the general structure of each ontology, how to adopt the ontologies for the purposes of the ONTOCHAIN ecosystem, and introducing examples of ontological fragment exploiting the developed ontologies.

POC4COMMERCE semantically describes the ONTOCHAIN ecosystem, focusing on the commercial vertical domain, thanks to an ontological stack constituted by three principal layers extendable to cover more knowledge domains or to characterize more precisely the available ones. The first layer consists of the OC-Found ontology, which extends the OASIS ontology with supply chains describing the life-cycle of assets in any vertical domain, digital identities, and a general modelling of trust management based on agent valuations performed on assets, supply chains, and agents.

The second layer is represented by the OC-Commerce ontology, extending the modelling of offerings provided by *GoodRelations* [13] with auctions and supply chains thanks to the definitions provided by OC-Found.

Finally, the third layer is constituted by the ontology OC-Ethereum which extends the *BLONDiE* ontology [23] with the definition of Ethereum tokens and the related features by focusing on the commercial domain represented by the OC-Commmerce ontology.

The section opens with a brief introduction of the OASIS ontology exploited by the first ontological layer OC-Found. Then, we first present the OC-Found ontology and the OC-Commerce ontology together with an introduction to GoodRelations which is adopted by OC-Commerce. We conclude the section by presenting the third ontology OC-Ethereum, also providing a brief description of the BLONDiE ontology adopted by OC-Ethereum.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

3.1 The OASIS ontology

The ontology called *Ontology for Agents, Systems, and Integration of Services* (in short, OASIS),[7] is a foundational OWL 2 ontology modelling multi-agent systems by exploiting the mentalistic notion of *agent behavior* [1], namely, by characterizing agents in terms of the actions their 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 assignment of tasks, restrictions on them, and constraints used to establish responsibilities and authorizations among agents.

Recently, the ontology has been exploited to define an ontology-based protocol for the Internet of Agents (IoA) and to realize a transparent communication and information exchange system among agents via Semantic Web technologies [2]. The resulting protocol is based on the exchange of OASIS fragments, each consisting of a RDF description of a request that is checked, by means of suitably constructed queries, against the corresponding RDF description of the agent behavior selected to satisfy it. Moreover, to show the advantages of the proposed approach, notably its interoperability, scalability, and modularity, the authors also presented the case study of a TRL3 prototype version of a domotic assistant exploiting many features of the ontological protocol to activate and manage applications, devices, and users, interacting with each other within a home environment.

OASIS models agents by representing their behaviors which are publicly exposed. By exposing behaviors, agents report to the communication peers the set of operations that they are able to perform and, possibly, the type of data required to execute them, together with the expected output. The approach based on the representation of agents through their behaviors has many advantages since it permits to abstract from implementation details, thus making the task of discovering agents extremely transparent and automatic. Agents may join a collaborative environment in a *plug-and-play* way, because there is no need for third-party interventions, and as a consequence, users can freely choose products and services according to their needs since provision methods are clearly described and represented. Moreover, by means of Semantic Web technologies and of automated reasoners, data provides machine-understandable information which can be processed, integrated, and exchanged by any type of agent at a higher level: data consistency can be easily verified, and information can be inferred and retrieved by exploiting what is already provided by the knowledge base.

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.

[7]Ontology available at http://www.dmi.unict.it/oasis.owl

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- 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 one is an optional step consisting in defining the agent 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. 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 association, OASIS introduces the so called *plan execution*. In what follows, we describe how to perform the above mentioned steps.

3.1.1 Representing agent templates in OASIS

We now describe how agent templates are represented in OASIS and leveraged by the POC4COMMERCE ontological stack to provide the most suitable descriptions of participants and of their actions.

As stated above templates represent high level descriptions of abstract agent behaviors that can be adopted to define concrete agents. 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 OASIS agent representation, indeed, revolves around the description of atomic operations introduced by the instances of the class *TaskDescription*: atomic operations are the most simple actions that agents are able to materially perform. 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 describing physical actions which are

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

introduced by means of entity names in the form of infinite verbs and representing the actions (e.g., *produce*, *sell*, ...).[8] 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 exact entity that is involved in the task.

- Possibly, an instance of the class *TaskOperatorArgument*, connected by means of the object-property *hasTaskOperatorArgument* and representing additional specifications for the task operator 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 either the object-property *refersAsNewTo* or *refersExactlyTo*. Moreover, the classes *TaskInputParameterTemplate* and *TaskOutputParameterTemplate* are also subclasses of the class *TaskParameterTemplate*.

[8]Instances of *Action* are introduced in the *OASIS-Abox* ontology reachable at http://www.dmi.unict.it/oasis-abox.owl

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

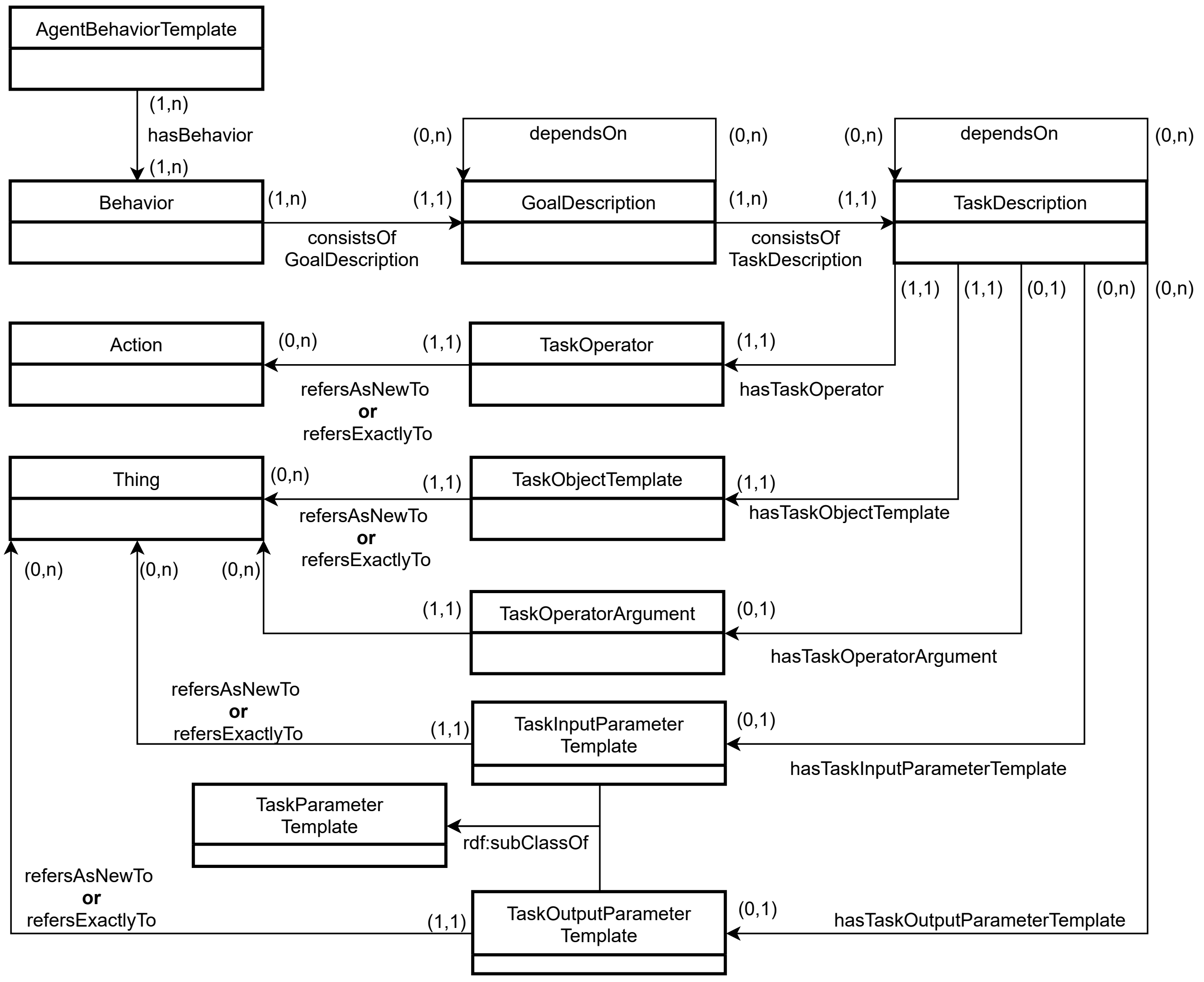


Figure 1: UML diagram of representing agent templates in OASIS

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

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- *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 can perform. Atomic operations are the most simple actions that agents are able to materially execute and, hence, they represent what agents can do. Atomic operations may depend on other atomic operations when the OWL 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 instances of *Action* are introduced as infinite verbs such as *buy*, *sell*, *compute*, and so on, drawn from a common, shared, and extendable vocabulary defined by OASIS.

- *TaskOperatorArgument*. This class defines an operator argument representing a subordinate characteristic of the task operator. 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, for example, the type of asset on which a quality valuation should

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

be performed. Tasks are possibly connected with the input parameters by means of the object-property *hasTaskInputParameterTemplate*.

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

Figure 2 illustrates an example of agent template describing an abstract agent that performs quality valuations on specific assets (in the diagram, classes are reported at the top of rectangles, whereas instances on the bottom). The agent template comprises a single behavior, constituted by a single goal that in its turn comprises a single task. The task, which represents the ability of the agent to perform a quality valuation, provides the following three elements:

- the task operator connected with the entity *perform*, instance of the class *Action*, by means of the object-property *refersExactlyTo*;
- the task operator argument connected with the entity, *quality_valuation*, instance of the class *DescriptionObject*, by means of the object-property *refersExactlyTo*;
- the task object template connected with an instance of the class *QualityValuationActivity* by means of the object-property *refersAsNewTo*, meaning that an instance of *QualityValuationActivity* is used as recipient of the action. Alternatively, a *punned entity*[9] can be used instead of the provided task object template.

Moreover, in order to describe the elements on which the agent can perform the quality valuation, the task introduces an instance of the class *TaskInputParameterTemplate* that is connected with an instance of the class *Asset* by means of the object-property *referAsNewTo*, meaning that instances of any subclass of *Asset* can be used as input of the action. Finally, the task introduces an instance of the class *TaskOutputParameterTemplate* connected with an instance of the class *QualityValuationResult*, which in its turn is connected with the recipient described above, since the output of the action to be returned is the quality valuation for the given asset. Moreover, the action recipient is also connected with the asset given as input in order to describe how the quality valuation activity must be carried on.

[9]https://www.w3.org/2007/OWL/wiki/Punning#References

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

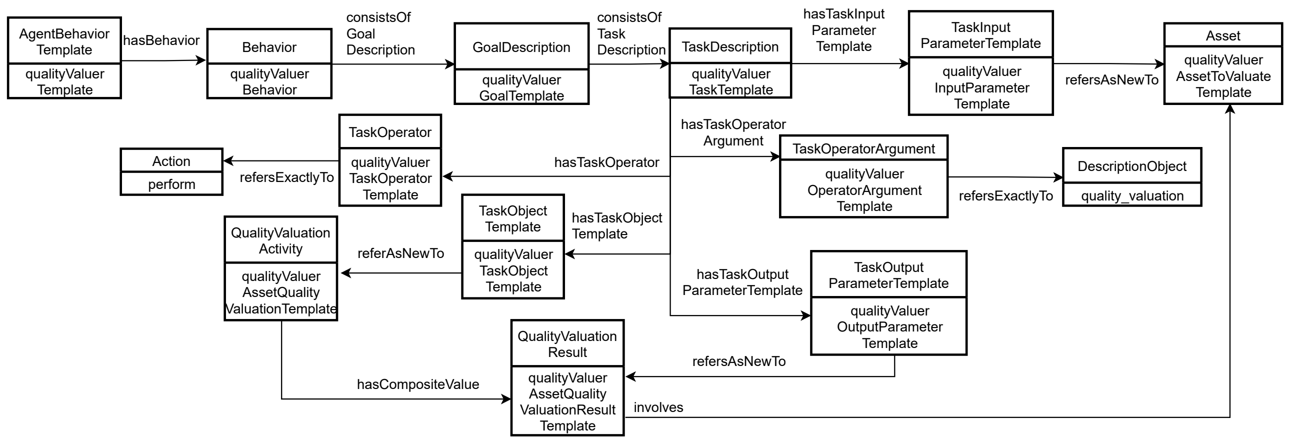

AgentBehavior Template
qualityValuer Template
hasBehavior
Behavior
qualityValuer Behavior
consistsOf Goal Description
GoalDescription
qualityValuer GoalTemplate
consistsOf Task Description
TaskDescription
qualityValuer TaskTemplate
hasTaskInput Parameter Template
TaskInput ParameterTemplate
qualityValuer InputParameter Template
refersAsNewTo
Asset
qualityValuer AssetToValuate Template
hasTaskOperator Argument
TaskOperatorArgument
qualityValuer OperatorArgument Template
refersExactlyTo
DescriptionObject
quality_valuation
TaskOperator
qualityValuer TaskOperator Template
hasTaskOperator
Action
perform
refersExactlyTo
TaskObject Template
qualityValuer TaskObject Template
hasTaskObject Template
QualityValuation Activity
qualityValuer AssetQuality ValuationTemplate
referAsNewTo
hasTaskOutput ParameterTemplate
TaskOutput ParameterTemplate
qualityValuer OutputParameter Template
QualityValuation Result
qualityValuer AssetQuality ValuationResult Template
refersAsNewTo
hasCompositeValue
involves

Figure 2: UML diagram of an example of an agent valuer template

3.1.2 Representing agents in OASIS

In OASIS, agents are represented according to the UML schema in Figure 3, which reflects the modelling pattern adopted for the representation of agent behavior templates described in Section 3.1.1, 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.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

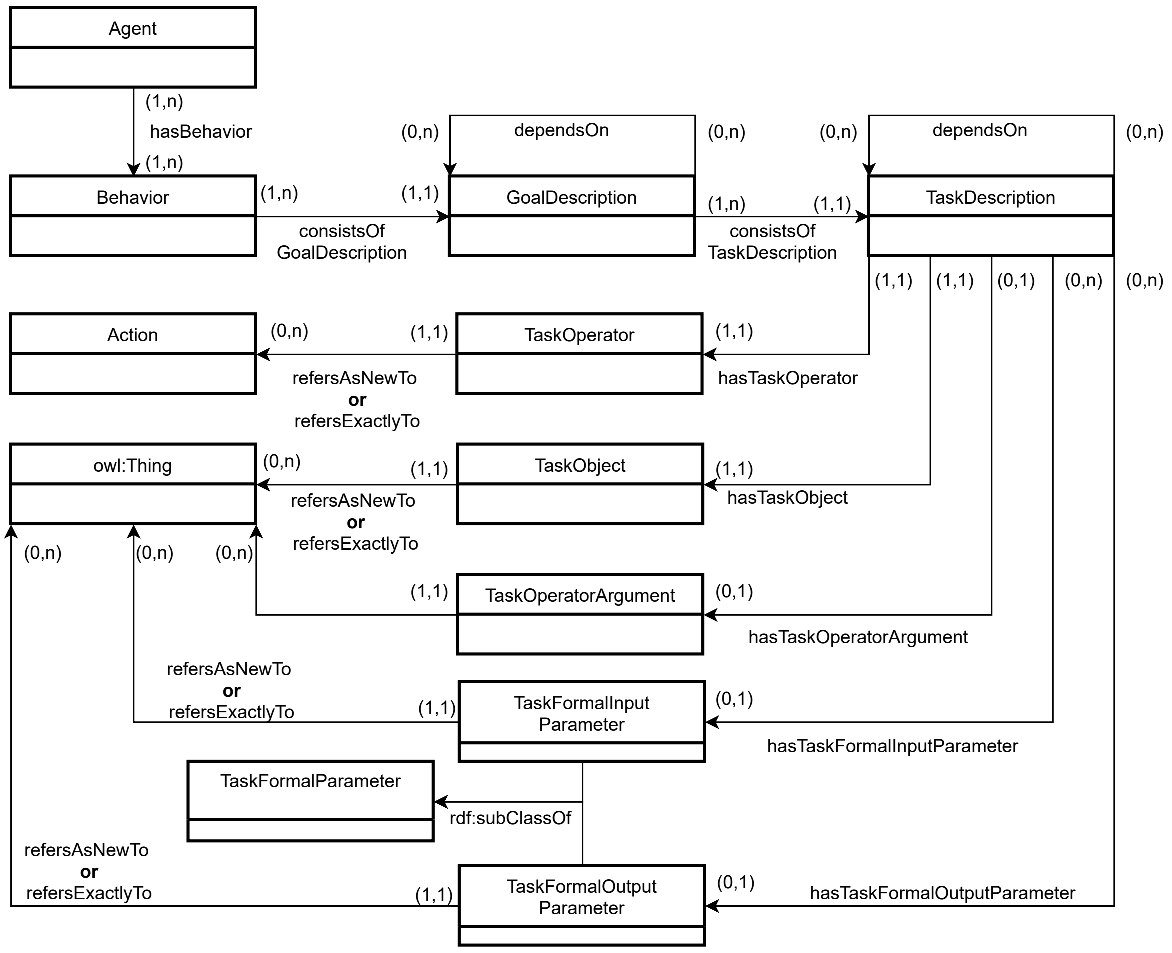


Figure 3: UML diagram of representing concrete agents in OASIS

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 an 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.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- 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.

Figure 4 shows an example of representing the agent *cloudServiceValuerAgent* that is an agent able to perform a quality evaluation on cloud services. The agent provides as task object the quality valuation activity, represented by a fresh individual of the class *QualityValuationActivity*, as task input parameter an instance of the class *CloudServiceAsset* by means of the object-property *refersAsNewTo*, meaning that it is able to evaluate only cloud services, and as task output parameter, the valuation result represented by an instance of the class *QualityValuationResult*. Finally, to ensure that *cloudServiceValuerAgent* adopts the agent template in Figure 2, the relationships in Figure 5 are also introduced, connecting the entities belonging to the concrete agent (on the right of Figure 5) with the entities belonging to the agent template (on the left of Figure 5).

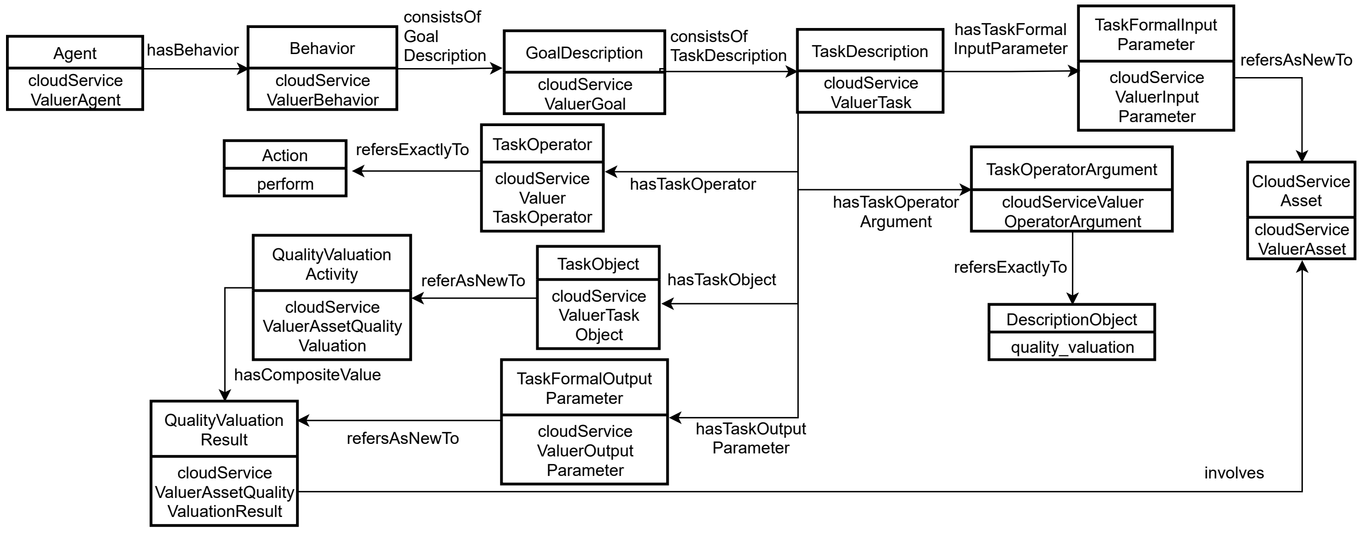


Figure 4: UML diagram of an example of concrete agents in OASIS.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

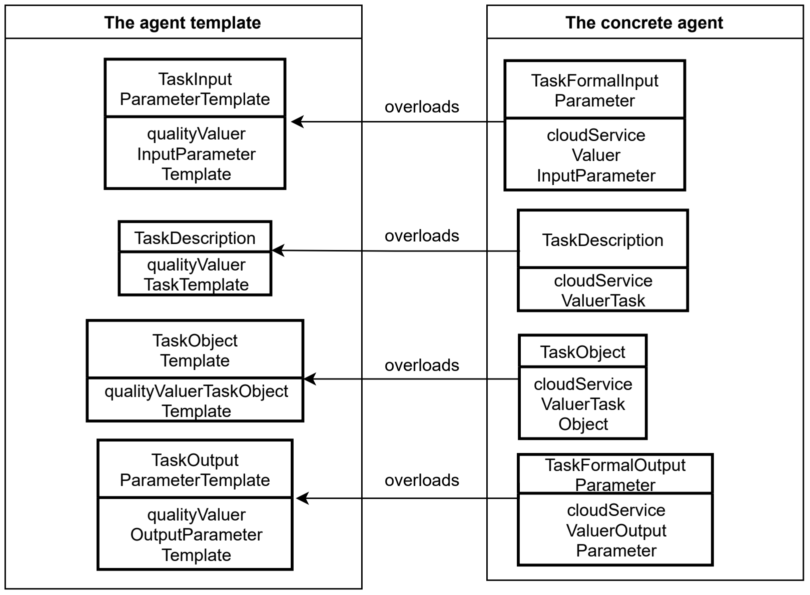

The agent template
TaskInput ParameterTemplate
qualityValuer InputParameter Template
TaskDescription
qualityValuer TaskTemplate
TaskObject Template
qualityValuerTaskObject Template
TaskOutput ParameterTemplate
qualityValuer OutputParameter Template
overloads
overloads
overloads
overloads
The concrete agent
TaskFormalInput Parameter
cloudService Valuer InputParameter
TaskDescription
cloudService ValuerTask
TaskObject
cloudService ValuerTask Object
TaskFormalOutput Parameter
cloudService ValuerOutput Parameter

Figure 5: UML diagram of relationships between a concrete agent and the related adopted template

3.1.3 Representing actions of agents in OASIS

The mentalistic notion of agent behavior pursued by OASIS allows one to describe the actions performed by an agent and related with its behavior, by entailing a description of the performed action to the behavior from which the action has been drawn. Agent actions are described by suitable graphs that retrace the schema of agent behavior. Specifically, as depicted in Figure 6, an agent action is primarily introduced by an instance of the class *PlanExecution*. 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 recipient 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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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.

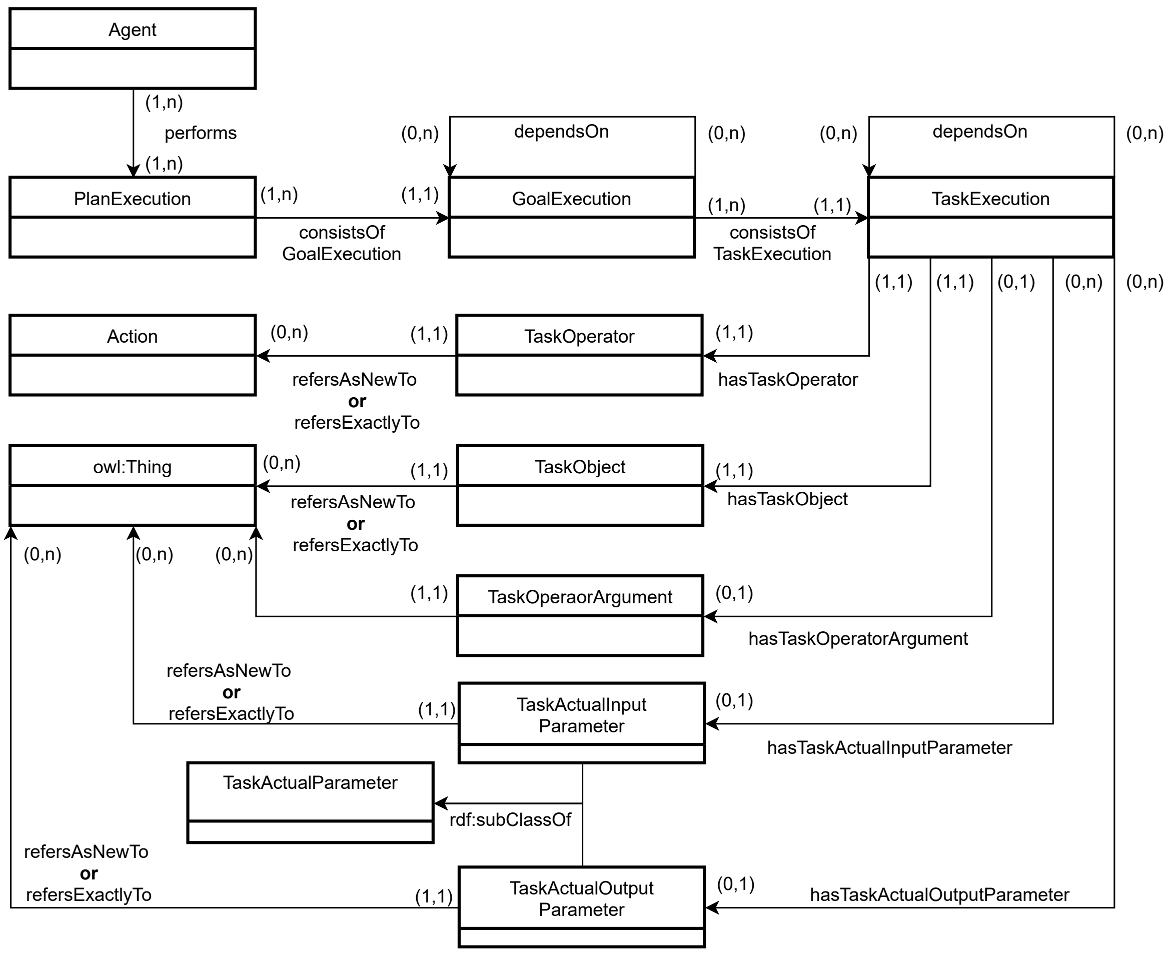


Figure 6: UML diagram of representing agent actions in OASIS

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- *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 action. Task execution are possibly connected with the output parameters by means of the object-property *hasTaskActualOutputParameter*.

Figure 7 shows an example of an action performed by the agent *cloudServiceValuerAgen* that evaluated the Dropbox cloud service. Specifically, the agent exploits the entity *DropboxCloudService* as input to perform an evaluation activity, i.e., the entity *cloudServiceDropboxValuation* (which is instance of the the class *QualityValuationActivity*). The evaluation activity is also connected with an instance of the class *QualityValuationResult*, representing the result of valuation. Notice that the object property *refersExactlyTo* has been used to connect a) the input parameter with the entity *DropboxCloudService* , b) the task object with the entity *cloudServiceDropboxValuation*, and c) the task output parameter with the entity *cloudServiceExecutionOutputParameter*.

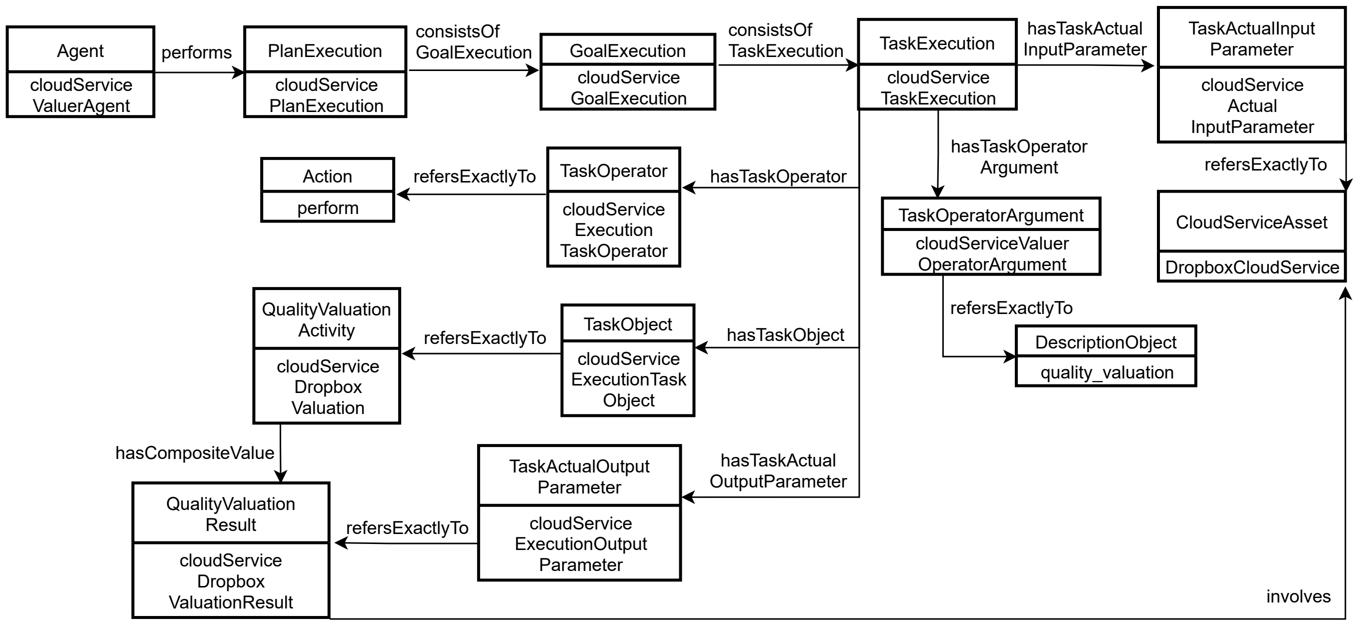

Agent
cloudService ValuerAgent
performs
PlanExecution
cloudService PlanExecution
consistsOf GoalExecution
GoalExecution
cloudService GoalExecution
consistsOf TaskExecution
TaskExecution
cloudService TaskExecution
hasTaskActual InputParameter
TaskActualInput Parameter
cloudService Actual InputParameter
refersExactlyTo
CloudServiceAsset
DropboxCloudService
hasTaskOperator Argument
TaskOperatorArgument
cloudServiceValuer OperatorArgument
refersExactlyTo
DescriptionObject
quality_valuation
Action
perform
refersExactlyTo
TaskOperator
cloudService Execution TaskOperator
hasTaskOperator
QualityValuation Activity
cloudService Dropbox Valuation
refersExactlyTo
TaskObject
cloudService ExecutionTask Object
hasTaskObject
hasCompositeValue
QualityValuation Result
cloudService Dropbox ValuationResult
refersExactlyTo
TaskActualOutput Parameter
cloudService ExecutionOutput Parameter
hasTaskActual OutputParameter
involves

Figure 7: UML diagram of an example of agent actions in OASIS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Finally, to unambiguously represent the fact that the action has been drawn by the behavior of the agent *cloudServiceValuerAgent*, the associations in Figure 8 are introduced: the task execution, the task object, the actual input parameter, and the actual output parameter of the agent action (on the left of Figure 8) are linked by means of the object-property *drawnBy* to the task description, the task object, the formal input parameter, the formal output parameter of the agent behavior (on the right of Figure 8), respectively.

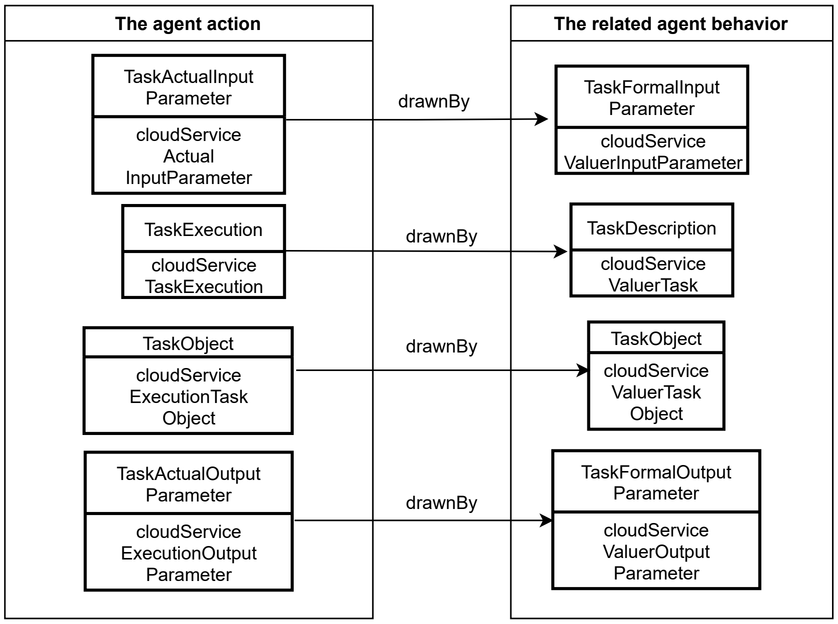

The agent action
TaskActualInput Parameter
cloudService Actual InputParameter
TaskExecution
cloudService TaskExecution
TaskObject
cloudService ExecutionTask Object
TaskActualOutput Parameter
cloudService ExecutionOutput Parameter
drawnBy
drawnBy
drawnBy
drawnBy
The related agent behavior
TaskFormalInput Parameter
cloudService ValuerInputParameter
TaskDescription
cloudService ValuerTask
TaskObject
cloudService ValuerTask Object
TaskFormalOutput Parameter
cloudService ValuerOutput Parameter

Figure 8: UML diagram of relationships between an agent action and the related agent behavior in OASIS

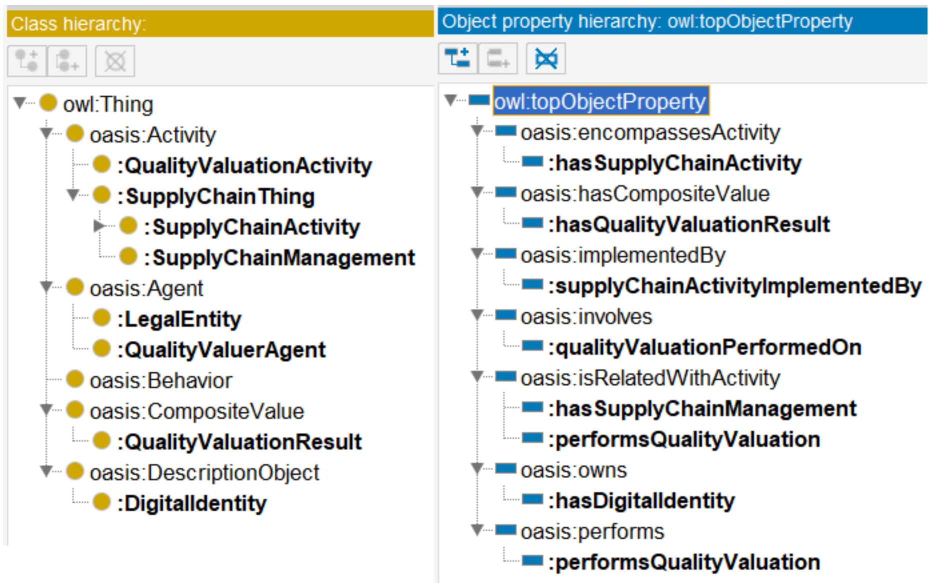
3.2 The OC-Found ontology

The first layer of the POC4COMMERCE stack is constituted by the OC-Found ontology,[10] which models the stakeholders of the ONTOCHAIN ecosystem, vertically focused on the commercial domain, including supply chains of resources for any deployment site and digital identities of agents. An ontology for the ONTOCHAIN stakeholders, and in particular for the main commercial actors, must be able to describe how participants contribute to the ecosystem, what they are capable of doing,

[10]The namespace of the ontology is http://www.ngi.ontochain/ontologies/oc-found.owl

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

and how they perform the proposed operations. This is crucial in terms of commercial transactions, since the entire supply chain of exchanged assets such as products and services provided, asset release mechanisms, and payment methods are fully semantically described, hence unambiguously published and shared. As described in Section 3.1, the OASIS ontology satisfies the objectives set by the first layer by providing a general description of agents, conceived as entities performing various actions in terms of the goals and tasks they accomplish: OASIS is imported and extended by OC-Found to provide the effective descriptions of the participant behaviors exploited to build the semantic representation of the actors joining the ecosystem. Specifically, the OC-Found ontology models supply chains and digital identities associated with agents as described in the OASIS fashion. Principal classes and properties introduced by OC-Found are depicted in Figure 9, obtained with the editor Protégé [17]. Entities having the prefix *oasis* are the ones imported from the OASIS ontology, whereas the other ones are defined in OC-Found.


Class hierarchy:

- owl:Thing
  - oasis:Activity
    - **:QualityValuationActivity**
    - **:SupplyChainThing**
      - **:SupplyChainActivity**
      - **:SupplyChainManagement**
  - oasis:Agent
    - **:LegalEntity**
    - **:QualityValuerAgent**
  - oasis:Behavior
  - oasis:CompositeValue
    - **:QualityValuationResult**
  - oasis:DescriptionObject
    - **:DigitalIdentity**

Object property hierarchy: owl:topObjectProperty

- owl:topObjectProperty
  - oasis:encompassesActivity
    - **:hasSupplyChainActivity**
  - oasis:hasCompositeValue
    - **:hasQualityValuationResult**
  - oasis:implementedBy
    - **:supplyChainActivityImplementedBy**
  - oasis:involves
    - **:qualityValuationPerformedOn**
  - oasis:isRelatedWithActivity
    - **:hasSupplyChainManagement**
    - **:performsQualityValuation**
  - oasis:owns
    - **:hasDigitalIdentity**
  - oasis:performs
    - **:performsQualityValuation**

Figure 9: Hierarchies of classes (on the left) and object-properties (on the right) in the ontology 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 type of digital identities such as public keys. Additionally,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

agents that are also legal entities are presented by instances of the class *LegalEntities*, subclass of the OASIS class *Agent*.

In OC-Found the life-cycle of assets is described by means of supply chains. Supply chains *encompasses all of those activities associated with moving goods from the raw-materials stage through to the end user* [15]. 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 resource and introduced by instances of one of the subclasses of the class *SupplyChainActivity*. Each subclass describes one of the phases of the life-cycle of the resource. Each phase, in its turn, is connected with the behavior of the agent responsible for its realization. Specifically, OC-Found includes at least the following subclasses of *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 the manufacturing, production, assembling, and so on.

Resources may express many supply chains that have to be considered as alternative supply chains. Moreover, supply chains may also specify one or more supply chain activities, depending on the particular 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 10. 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, OC-Found new entities are reported in bold.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Resources whose supply chain is introduced are related with an instance of the class *SupplyChainManagement* for each supply chain described, 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 example, the supply chain of the apple producer, as described in OC-Found, is depicted in Figure 11. Classes are reported in bold, whereas instances are reported below the related membership classes.

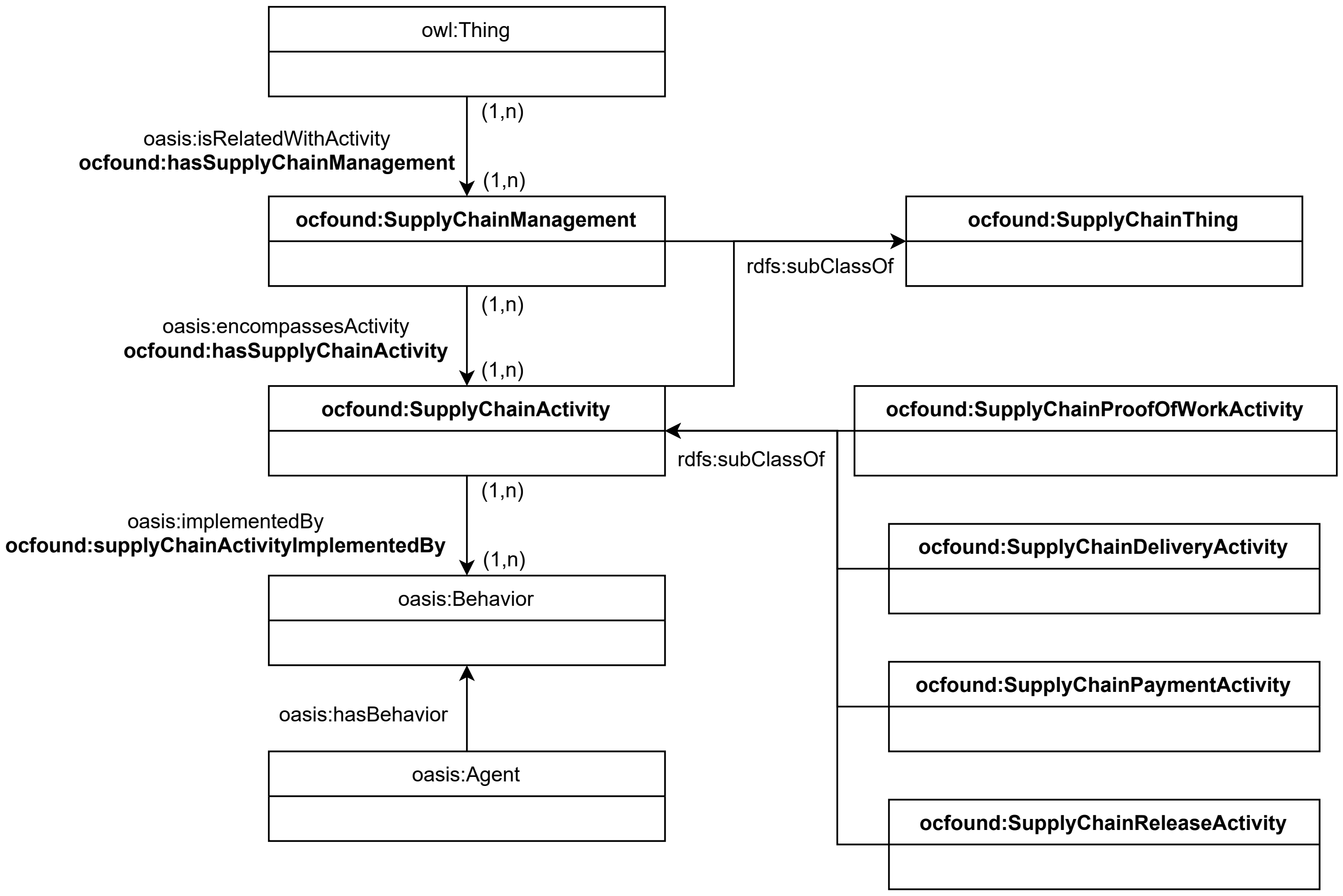

owl:Thing
(1,n)
oasis:isRelatedWithActivity
ocfound:hasSupplyChainManagement
(1,n)
ocfound:SupplyChainManagement
ocfound:SupplyChainThing
rdfs:subClassOf
(1,n)
oasis:encompassesActivity
ocfound:hasSupplyChainActivity
(1,n)
ocfound:SupplyChainActivity
ocfound:SupplyChainProofOfWorkActivity
rdfs:subClassOf
(1,n)
oasis:implementedBy
ocfound:supplyChainActivityImplementedBy
(1,n)
oasis:Behavior
ocfound:SupplyChainDeliveryActivity
ocfound:SupplyChainPaymentActivity
oasis:hasBehavior
oasis:Agent
ocfound:SupplyChainReleaseActivity

Figure 10: UML diagram of the OC-Found ontology.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

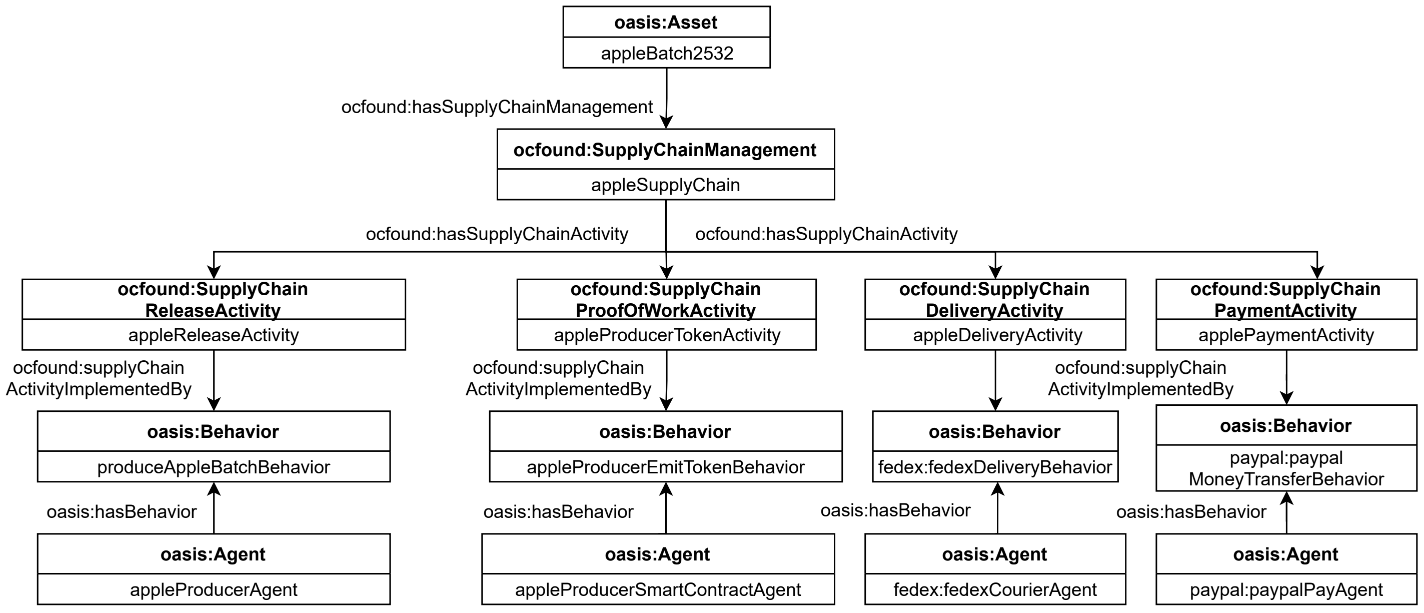

oasis:Asset
appleBatch2532
ocfound:hasSupplyChainManagement
ocfound:SupplyChainManagement
appleSupplyChain
ocfound:hasSupplyChainActivity
ocfound:hasSupplyChainActivity
ocfound:SupplyChain ReleaseActivity
appleReleaseActivity
ocfound:SupplyChain ProofOfWorkActivity
appleProducerTokenActivity
ocfound:SupplyChain DeliveryActivity
appleDeliveryActivity
ocfound:SupplyChain PaymentActivity
applePaymentActivity
ocfound:supplyChain ActivityImplementedBy
ocfound:supplyChain ActivityImplementedBy
ocfound:supplyChain ActivityImplementedBy
oasis:Behavior
produceAppleBatchBehavior
oasis:Behavior
appleProducerEmitTokenBehavior
oasis:Behavior
fedex:fedexDeliveryBehavior
oasis:Behavior
paypal:paypal MoneyTransferBehavior
oasis:hasBehavior
oasis:hasBehavior
oasis:hasBehavior
oasis:hasBehavior
oasis:Agent
appleProducerAgent
oasis:Agent
appleProducerSmartContractAgent
oasis:Agent
fedex:fedexCourierAgent
oasis:Agent
paypal:paypalPayAgent

Figure 11: Example of OC-Found: the apple producer supply chain.

The example in Figure 11 describes an apple farmer providing for its 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 apples batch is produced by connecting the activity with the behaviour responsible for producing apple and 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 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*.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

If dependency relationships among supply chain activities are required, the object-property of OASIS *depensOn* can be used to connect an activity with the activity it depends on. Hence, thanks to the description provided by OC-Found, the supply chain of the apples produced by the apple farmer is completely described and responsibilities of the agent entrusted for the activities that make up the supply chain are clearly defined. The class *Asset* introduced in Figure 11 is introduced in Section 3.3, where the OC-Commerce ontology is presented.

In order to build an ontological trust management system based on participants experiences and feedbacks, OC-Found models the quality valuation processes performed either by professional quality valuer agents or by standard users such as customers, and associated with resources. With this aim, OC-Found provides the schema depicted in Figure 12.

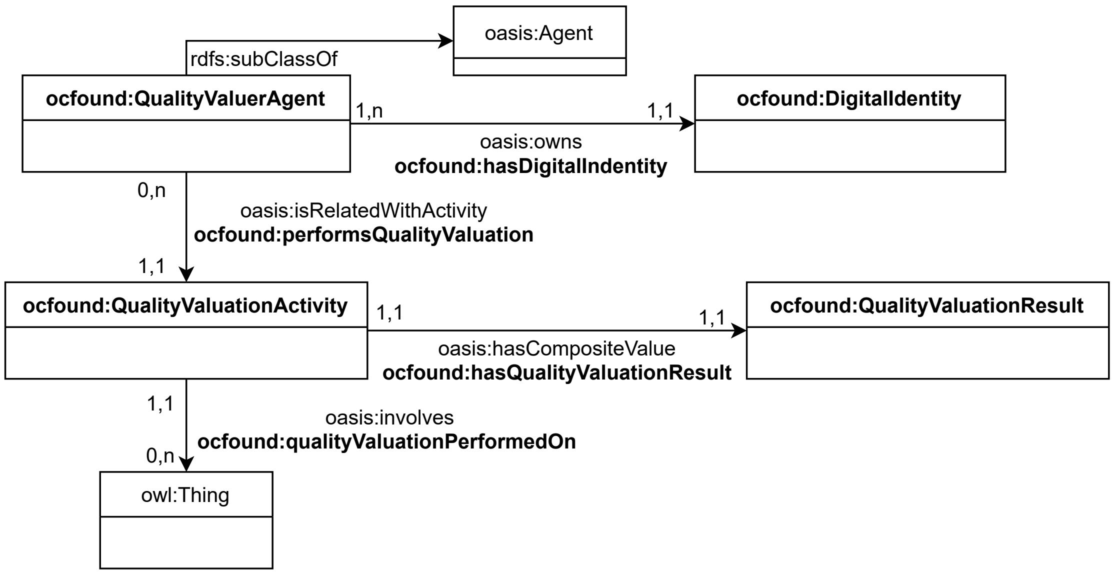

oasis:Agent
rdfs:subClassOf
ocfound:QualityValuerAgent
1,n
1,1
ocfound:DigitalIdentity
oasis:owns
ocfound:hasDigitalIndentity
0,n
oasis:isRelatedWithActivity
ocfound:performsQualityValuation
1,1
ocfound:QualityValuationActivity
1,1
1,1
ocfound:QualityValuationResult
oasis:hasCompositeValue
ocfound:hasQualityValuationResult
1,1
oasis:involves
ocfound:qualityValuationPerformedOn
0,n
owl:Thing

Figure 12: UML of quality valuation process in OC-Found. Entities newly introduced in OC-Found are reported in bold.

Agents performing quality valuations by profession are defined as instances of the OC-Found class *QualityValuerAgent*. When quality valuers perform actions (see Section 3.1.3) 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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

on which the valuation is performed by means of the object-property *qualityValuationPerformedOn* (subproperty of the OASIS property *involves*), 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 13, where the valuer agent *agriFoodValuer* performs a valuation activity on the apple batch *appleBatch2532*, assigning a total result of 5.

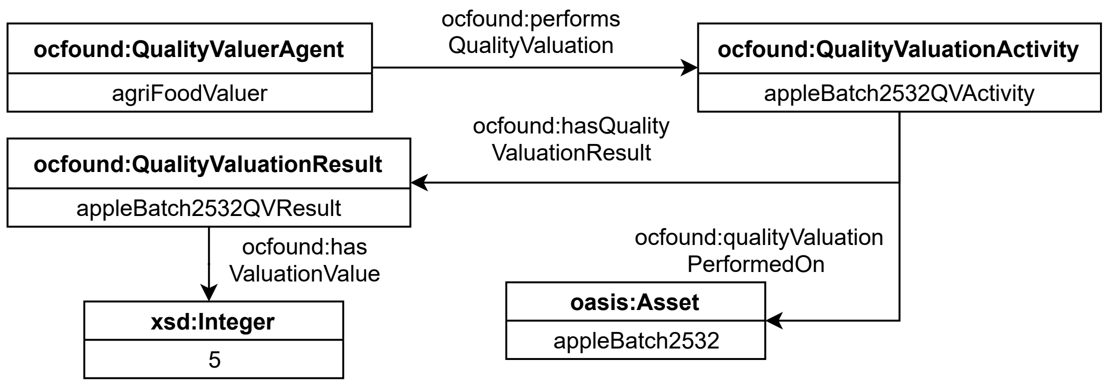

ocfound:QualityValuerAgent
agriFoodValuer
ocfound:performs QualityValuation
ocfound:QualityValuationActivity
appleBatch2532QVActivity
ocfound:hasQuality ValuationResult
ocfound:QualityValuationResult
appleBatch2532QVResult
ocfound:has ValuationValue
xsd:Integer
5
ocfound:qualityValuation PerformedOn
oasis:Asset
appleBatch2532

Figure 13: Example of a quality valuation process performed on the apple batch.

3.3 The OC-Commerce ontology

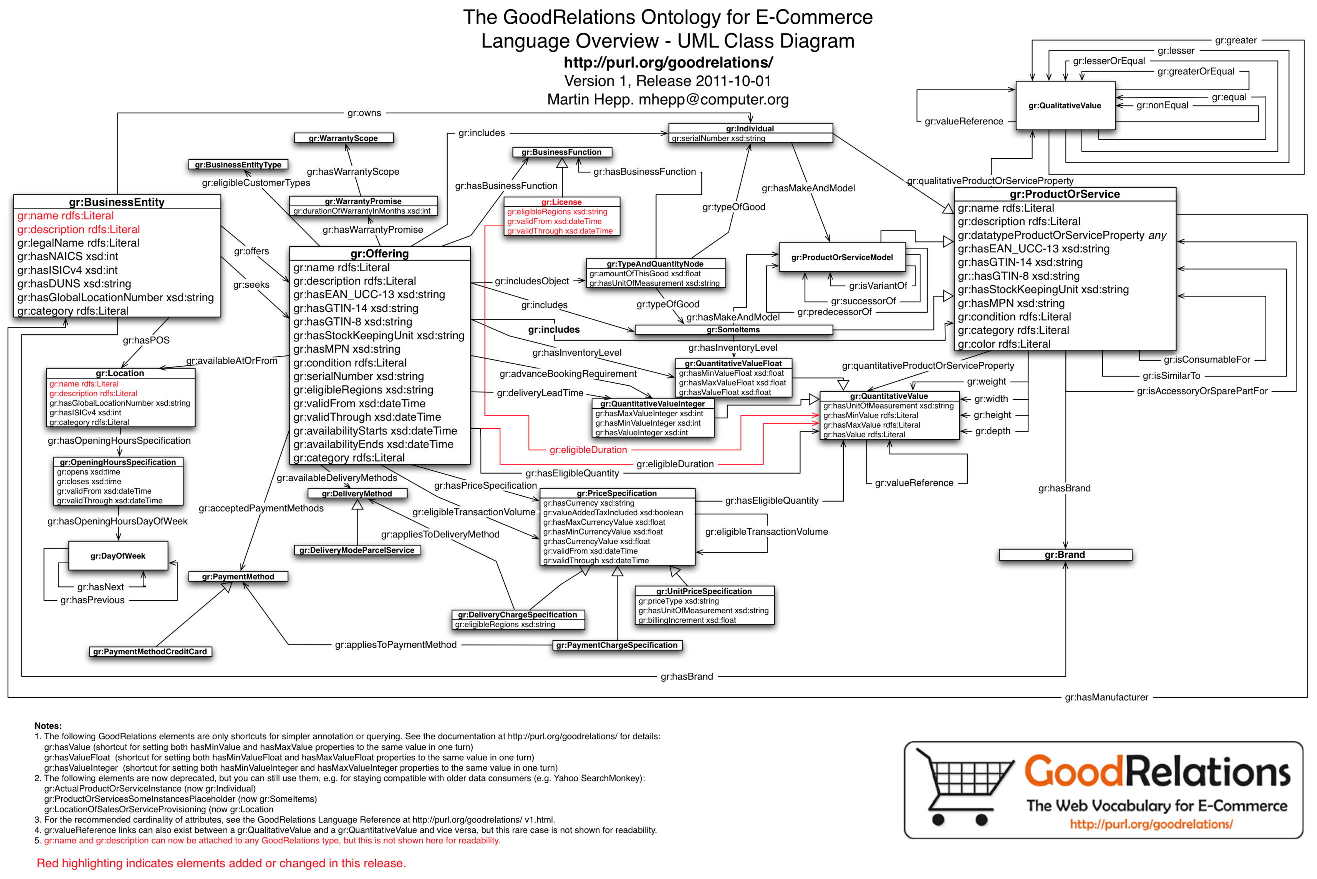
The ontology OC-Commerce[11] conjoins OC-Found with many features of the GoodRelations ontology to construct a means for representing offerings and auctions for the commercial domain together with the related supply chains, focusing in particular on the commerce carried through the Ethereum blockchain. Before exploring the OC-Commerce ontology, we briefly introduce GoodRelations. The general schema of GoodRelations is depicted in Figure 14.

GoodRelations is an OWL vocabulary describing offerings on products or services, legal entities involved, prices, offering terms and conditions. The core class of the GoodRelation vocabulary allows one to represent an *Offering*. An offering is an announcement of an agent providing a certain *business function*, which is 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 particular

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

commercial conditions. A *business entity*, i.e., a legal agent, can create such an offering or *seek* for someone else providing goods and terms under particular conditions.


The GoodRelations Ontology for E-Commerce
Language Overview - UML Class Diagram
http://purl.org/goodrelations/
Version 1, Release 2011-10-01
Martin Hepp. mhepp@computer.org
gr:BusinessEntity
gr:name rdfs:Literal
gr:description rdfs:Literal
gr:legalName rdfs:Literal
gr:hasNAICS xsd:int
gr:hasISICv4 xsd:int
gr:hasDUNS xsd:string
gr:hasGlobalLocationNumber xsd:string
gr:category rdfs:Literal
gr:Offering
gr:name rdfs:Literal
gr:description rdfs:Literal
gr:hasEAN_UCC-13 xsd:string
gr:hasGTIN-14 xsd:string
gr:hasGTIN-8 xsd:string
gr:hasStockKeepingUnit xsd:string
gr:hasMPN xsd:string
gr:condition rdfs:Literal
gr:serialNumber xsd:string
gr:eligibleRegions xsd:string
gr:validFrom xsd:dateTime
gr:validThrough xsd:dateTime
gr:availabilityStarts xsd:dateTime
gr:availabilityEnds xsd:dateTime
gr:category rdfs:Literal
gr:ProductOrService
gr:name rdfs:Literal
gr:description rdfs:Literal
gr:datatypeProductOrServiceProperty any
gr:hasEAN_UCC-13 xsd:string
gr:hasGTIN-14 xsd:string
gr::hasGTIN-8 xsd:string
gr:hasStockKeepingUnit xsd:string
gr:hasMPN xsd:string
gr:condition rdfs:Literal
gr:category rdfs:Literal
gr:color rdfs:Literal
gr:QualitativeValue
gr:Individual
gr:serialNumber xsd:string
gr:ProductOrServiceModel
gr:SomeItems
gr:BusinessFunction
gr:License
gr:eligibleRegions xsd:string
gr:validFrom xsd:dateTime
gr:validThrough xsd:dateTime
gr:WarrantyScope
gr:WarrantyPromise
gr:durationOfWarrantyInMonths xsd:int
gr:BusinessEntityType
gr:TypeAndQuantityNode
gr:amountOfThisGood xsd:float
gr:hasUnitOfMeasurement xsd:string
gr:QuantitativeValueFloat
gr:hasMinValueFloat xsd:float
gr:hasMaxValueFloat xsd:float
gr:hasValueFloat xsd:float
gr:QuantitativeValueInteger
gr:hasMaxValueInteger xsd:int
gr:hasMinValueInteger xsd:int
gr:hasValueInteger xsd:int
gr:QuantitativeValue
gr:hasUnitOfMeasurement xsd:string
gr:hasMinValue rdfs:Literal
gr:hasMaxValue rdfs:Literal
gr:hasValue rdfs:Literal
gr:Location
gr:name rdfs:Literal
gr:description rdfs:Literal
gr:hasGlobalLocationNumber xsd:string
gr:hasISICv4 xsd:int
gr:category rdfs:Literal
gr:OpeningHoursSpecification
gr:opens xsd:time
gr:closes xsd:time
gr:validFrom xsd:dateTime
gr:validThrough xsd:dateTime
gr:DayOfWeek
gr:hasNext
gr:hasPrevious
gr:PaymentMethod
gr:PaymentMethodCreditCard
gr:DeliveryMethod
gr:DeliveryModeParcelService
gr:PriceSpecification
gr:hasCurrency xsd:string
gr:valueAddedTaxIncluded xsd:boolean
gr:hasMaxCurrencyValue xsd:float
gr:hasMinCurrencyValue xsd:float
gr:hasCurrencyValue xsd:float
gr:validFrom xsd:dateTime
gr:validThrough xsd:dateTime
gr:UnitPriceSpecification
gr:priceType xsd:string
gr:hasUnitOfMeasurement xsd:string
gr:billingIncrement xsd:float
gr:DeliveryChargeSpecification
gr:eligibleRegions xsd:string
gr:PaymentChargeSpecification
gr:Brand
gr:owns
gr:includes
gr:offers
gr:seeks
gr:eligibleCustomerTypes
gr:hasWarrantyScope
gr:hasWarrantyPromise
gr:hasBusinessFunction
gr:typeOfGood
gr:hasMakeAndModel
gr:includesObject
gr:isVariantOf
gr:successorOf
gr:predecessorOf
gr:hasInventoryLevel
gr:advanceBookingRequirement
gr:deliveryLeadTime
gr:eligibleDuration
gr:hasEligibleQuantity
gr:eligibleTransactionVolume
gr:hasPOS
gr:availableAtOrFrom
gr:hasOpeningHoursSpecification
gr:hasOpeningHoursDayOfWeek
gr:availableDeliveryMethods
gr:acceptedPaymentMethods
gr:hasPriceSpecification
gr:appliesToDeliveryMethod
gr:appliesToPaymentMethod
gr:qualitativeProductOrServiceProperty
gr:quantitativeProductOrServiceProperty
gr:valueReference
gr:greater
gr:lesser
gr:lesserOrEqual
gr:greaterOrEqual
gr:equal
gr:nonEqual
gr:weight
gr:width
gr:height
gr:depth
gr:isConsumableFor
gr:isSimilarTo
gr:isAccessoryOrSparePartFor
gr:hasBrand
gr:hasManufacturer
GoodRelations
The Web Vocabulary for E-Commerce
http://purl.org/goodrelations/

Notes:
1. The following GoodRelations elements are only shortcuts for simpler annotation or querying. See the documentation at http://purl.org/goodrelations/ for details:
   gr:hasValue (shortcut for setting both hasMinValue and hasMaxValue properties to the same value in one turn)
   gr:hasValueFloat (shortcut for setting both hasMinValueFloat and hasMaxValueFloat properties to the same value in one turn)
   gr:hasValueInteger (shortcut for setting both hasMinValueInteger and hasMaxValueInteger properties to the same value in one turn)
2. The following elements are now deprecated, but you can still use them, e.g. for staying compatible with older data consumers (e.g. Yahoo SearchMonkey):
   gr:ActualProductOrServiceInstance (now gr:Individual)
   gr:ProductOrServicesSomeInstancesPlaceholder (now gr:SomeItems)
   gr:LocationOfSalesOrServiceProvisioning (now gr:Location
3. For the recommended cardinality of attributes, see the GoodRelations Language Reference at http://purl.org/goodrelations/ v1.html.
4. gr:valueReference links can also exist between a gr:QualitativeValue and a gr:QuantitativeValue and vice versa, but this rare case is not shown for readability.
5. gr:name and gr:description can now be attached to any GoodRelations type, but this is not shown here for readability.

Red highlighting indicates elements added or changed in this release.

Figure 14: UML schema of GoodRelations

An offering can either refer to

- a clearly specified instance (class *Individual*) or
- a set of anonymous instances of a given type (class *SomeItems*) or
- a product model specification (class *ProductOrServiceModel*).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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

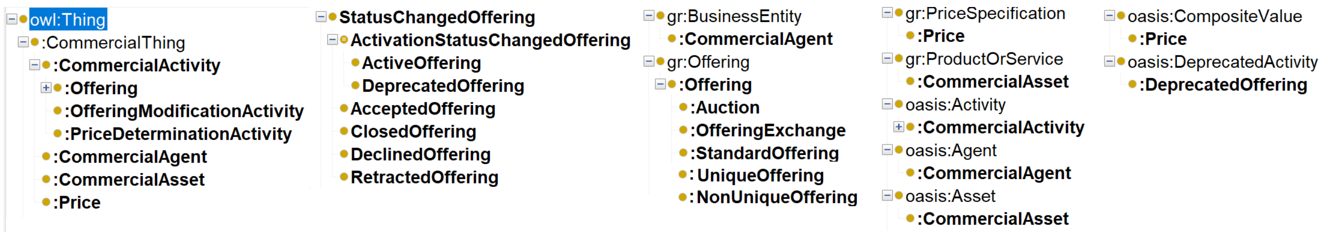
- the lower and upper limits of the eligible quantity,
- the monetary amount per unit (in combination with a currency), and
- whether this price includes local sales taxes, namely, VAT.

The time scope of an offering can be eventually specified and different accepted *payment methods* (eventually combined with additional *payment charge specifications*) may be available for an offering: in advance or after delivery, by credit card, cash or bank transfer, and so on.

OC-Commerce extends the definition of the offerings provided by GoodRelations, by integrating them with the definition of supply chain, with agents responsible for the execution of supply chain activities provided by OC-Found, including agent actions performed to publish, reject, accept, and close offerings. Moreover, OC-Commerce also introduces bargaining, auctions, editing of offerings, user valuations, and price specification mechanisms.

The classes and object-properties introduced by OC-Commerce and the related mapping into GoodRelations is illustrated in Figures 15 and 16, 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 17. 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:



```
owl:Thing
  :CommercialThing
    :CommercialActivity
      :Offering
      :OfferingModificationActivity
      :PriceDeterminationActivity
    :CommercialAgent
    :CommercialAsset
    :Price

StatusChangedOffering
  ActivationStatusChangedOffering
    ActiveOffering
    DeprecatedOffering
  AcceptedOffering
  ClosedOffering
  DeclinedOffering
  RetractedOffering

gr:BusinessEntity
  :CommercialAgent
gr:Offering
  :Offering
    :Auction
    :OfferingExchange
    :StandardOffering
    :UniqueOffering
    :NonUniqueOffering

gr:PriceSpecification
  :Price
gr:ProductOrService
  :CommercialAsset
oasis:Activity
  :CommercialActivity
oasis:Agent
  :CommercialAgent
oasis:Asset
  :CommercialAsset

oasis:CompositeValue
  :Price
oasis:DeprecatedActivity
  :DeprecatedOffering
```

Figure 15: Class hierarchy of OC-Commerce

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

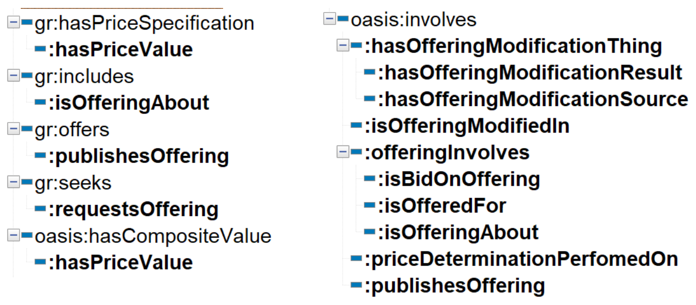


Figure 16: Object-property hierarchy of OC-Commerce

- *request*, whether the agent is enabled to seek for an offering;
- *modify*, whether the agent is enabled to change some features of a previously published offering;
- *retract*, whether 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*, whether the agent is enabled to close a previously published offering, meaning that the offering is expired or the publisher autonomously decide to close the offering;
- *accept*, whether 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*, whether the agent rejects a proposed offering.

Before publishing an offering, the traded asset should be available beforehand through a specific agent action that releases the asset as described in Section 3.1.3. For instance, an apple producer whose behavior is depicted in Figure 18, deploys a new asset of apples, which is now available for trading, by writing the fragment depicted in 19.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

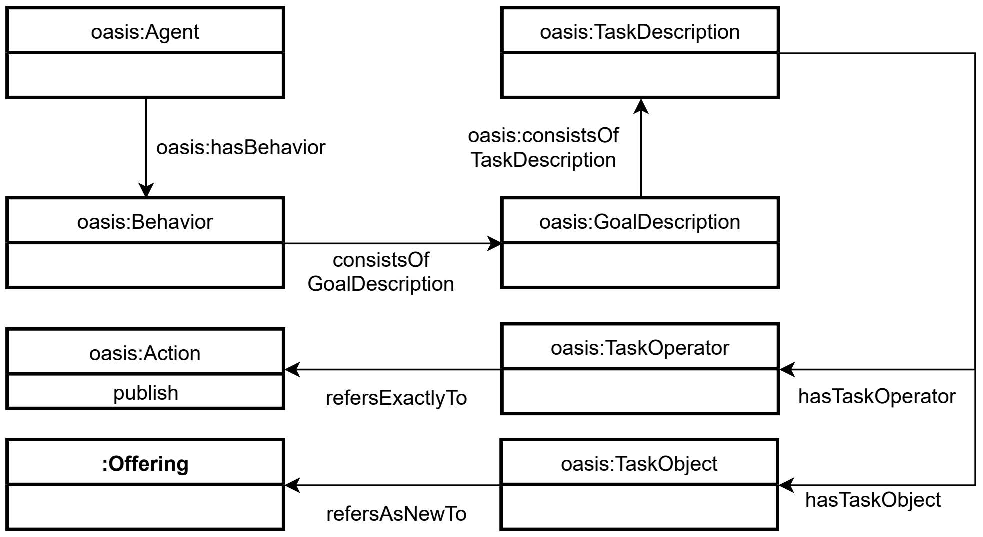


Figure 17: Behavior for publishing offering in OC-Commerce

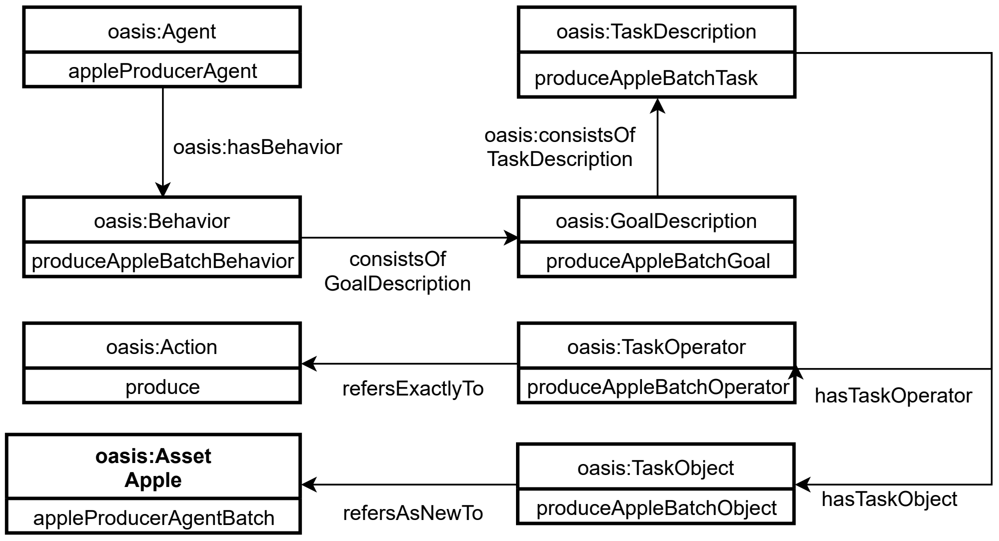


Figure 18: Apple producer behavior for producing asset of apples

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

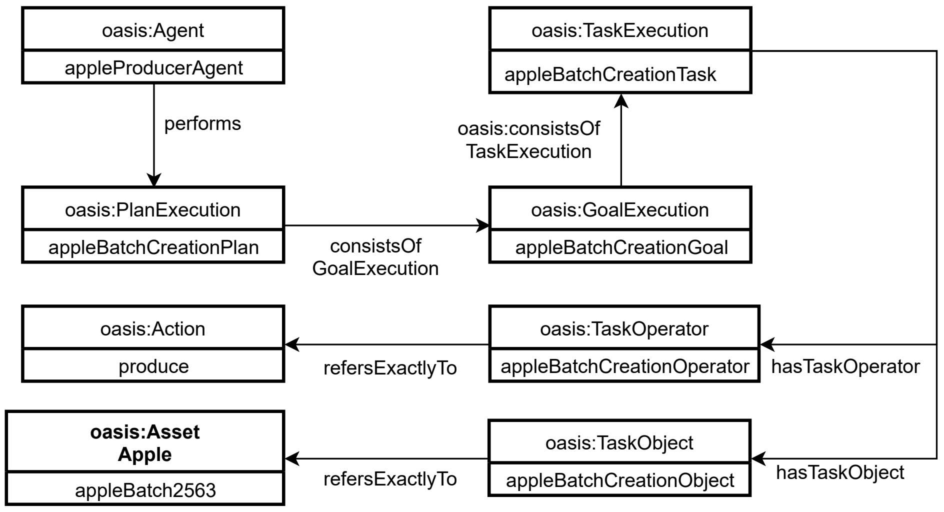


Figure 19: OC-Commerce example of making available a news assets of apples

As second step, the agent delegated to make the offering available, publishes the action related 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 depicted in Figure 20.

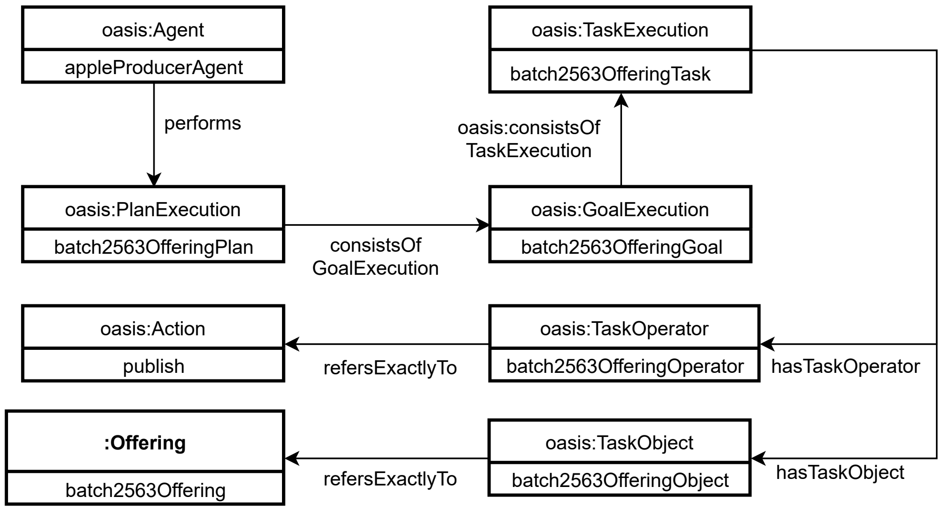


Figure 20: OC-Commerce example of publishing a new offering for the apple batch 2563

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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*. Offerings in their turn 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 schema of Figure 21.

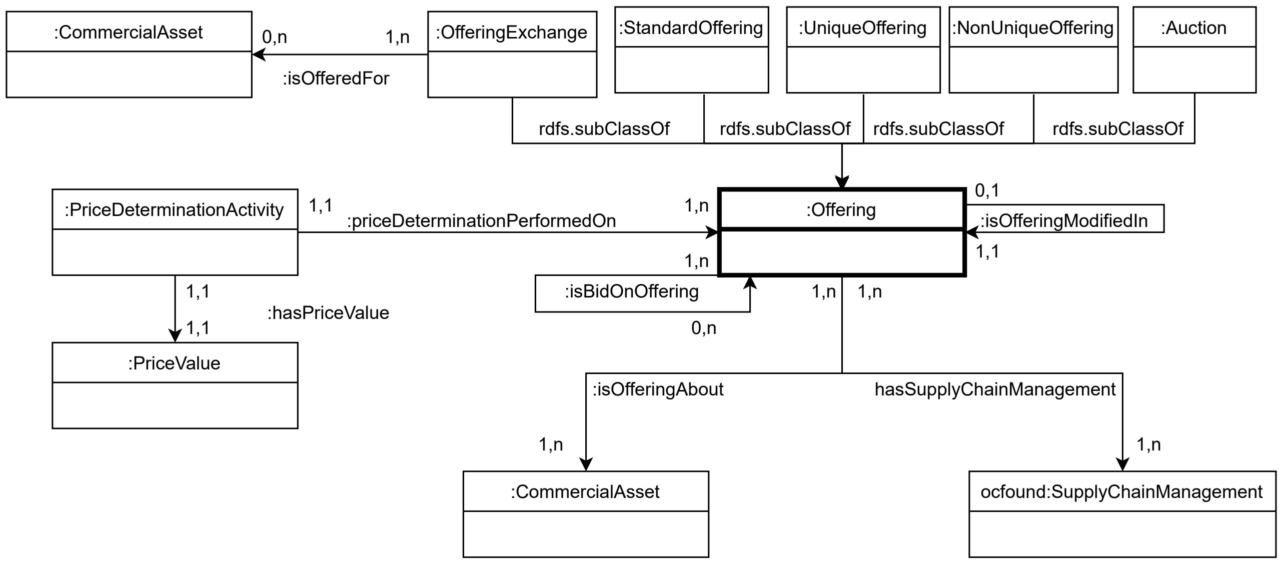


Figure 21: UML schema of offerings in OC-Commerce

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 FIAT or crypto-currencies and hence supply chain activities of payment involve agent behaviors for transferring FIAT or crypto- currencies. Offerings in their turn may receive counter-offerings, represented by instances *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 for other assets. The offering is related with the exchanged asset by means of the object-property *isOfferedFor*. Supply chain activities of payment related with exchange offerings involve the exchanged asset instead of an agent behavior.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- 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*, if the traded asset is uniquely identifiable, such as second-hand objects.
- *NonUniqueOffering*, if 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 3.2.

In OC-Commerce, prices are conceived as the result of some *price determination activities* carried either by the publisher of the offering or by suitable agents delegated to compute the value of specific assets (see Section 3.2). 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* and with the selected currency introduced as string by means of the GoodRelations dataproperty *hasCurrency*.

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

- *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.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- *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.

The offering concerning apples published by the apple producer is illustrated in Figure 22. 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 11 of Section 3.2. Then, the user Bob accepts the offering by performing the action depicted in Figure 23 and, as consequence, the publisher closes the offering. Subsequently the two agents indicated in the the supply chain perform the entrusted actions in order to register the payment and to ship the product, respectively. The payment action, preformed by the Paypal agent is illustrated in Figure 24. The action consists in transferring the established quantity of the selected currency from Bob's account to the apple producer account, in order to pay for the apple batch described in offering. As result, the action emits a payment receipt. Hereafter, the Fedex agent ships the selected product as described in Figure 25. The user destination is represented by an instance of the GeoNames ontology [25] and a suitable receipt is generated to track the shipment. Additionally, the user Bob can valuate his commercial transaction experience by valuating the quality of the offering or of the involved agents, by committing an action as the one described in Section 3.1, Figure 7.

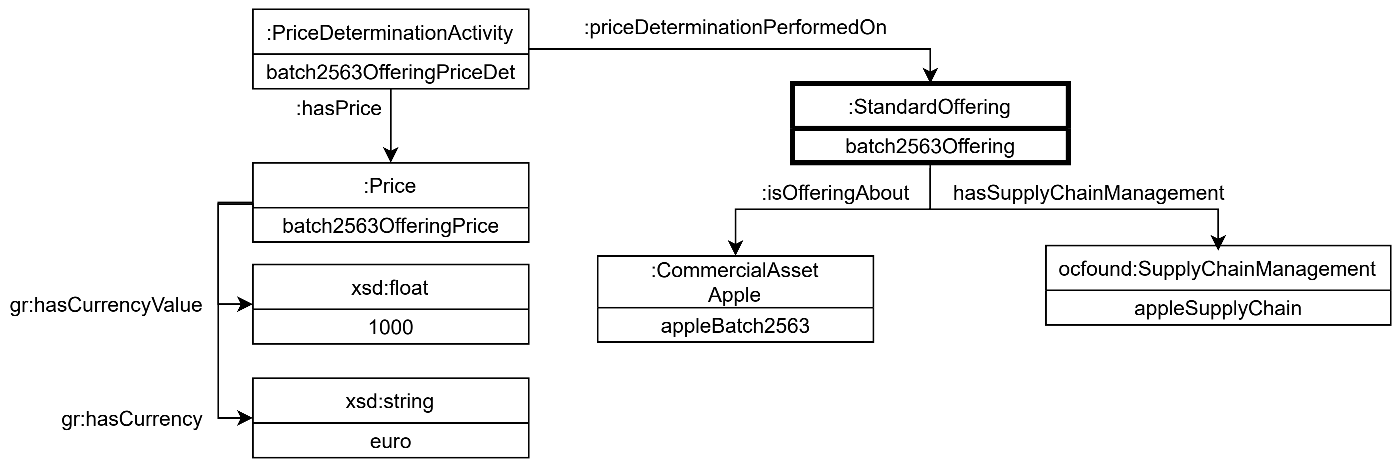


Figure 22: Example of representing offerings in OC-Commerce

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

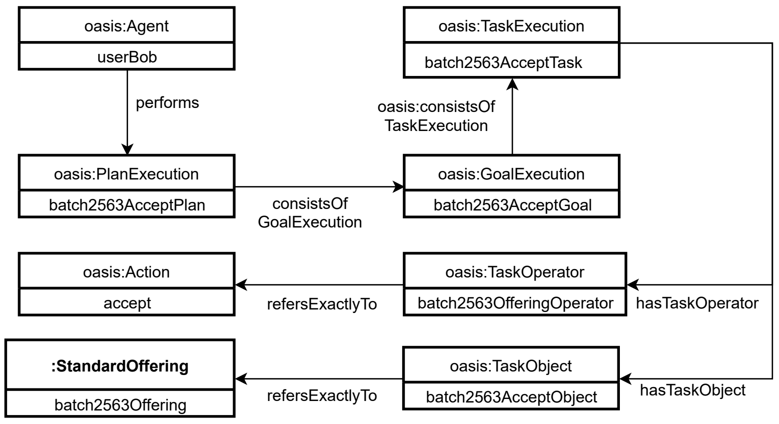


Figure 23: Example of accepting the offering for the apple batch 2563

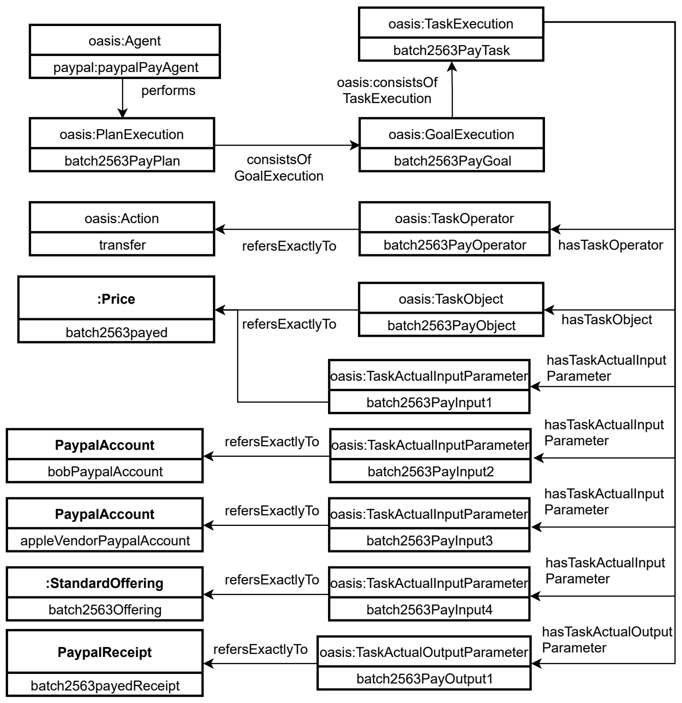


Figure 24: Example of accepted payment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

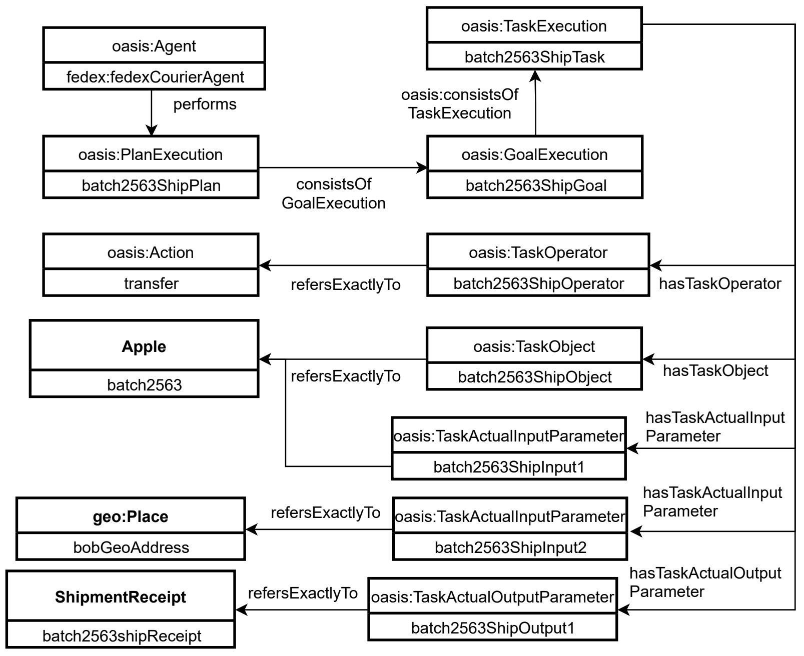


Figure 25: Example of performed shipment

Where allowed, offerings can be modified when some features, such as supply chain, 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 schema for offering modification is illustrated in Figure 26.

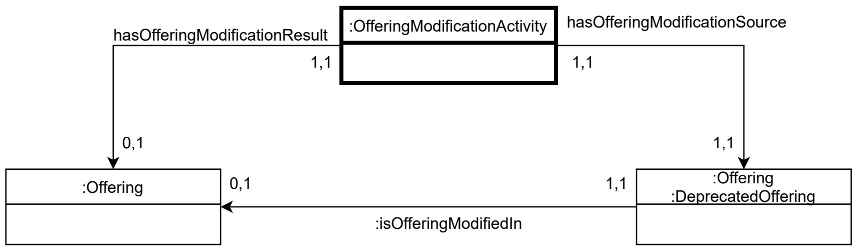

:OfferingModificationActivity
hasOfferingModificationResult
hasOfferingModificationSource
1,1
1,1
0,1
1,1
:Offering
0,1
1,1
:Offering
:DeprecatedOffering
:isOfferingModifiedIn

Figure 26: UML schema of offering modification in OC-Commerce

When offerings are replaced by new offerings, the deprecated ones are defined as instance of the class *DeprecatedOfferings* and must not be involved in any new commercial activity. Instead, a new

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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 eventually 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 27, in analogous way as in counter-offerings.

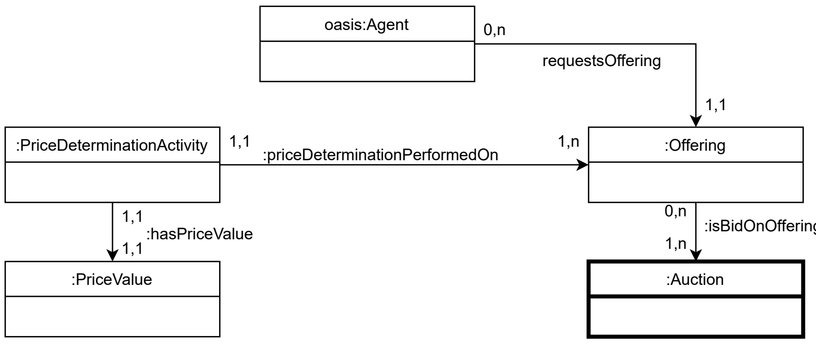


Figure 27: UML schema of auctions in OC-Commerce

3.4 The ontology OC-Ethereum

The OC-Ethereum ontology provides a semantic model of the essential elements of the Ethereum blockchain, smart contracts implemented on it, and related exchanged tokens. OC-Ethereum conjoins the BLONDiE ontology [23], which depicts foundational characteristics of blockchains such as blocks and transactions, with the OC-Commerce ontology, in order to provide an ontological conception of commercial activities, in particular of those exploiting the management of tokens for trading purposes, carried out through the Ethereum blockchain.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

The BLONDiE ontology is summarized in the UML schema of Figure 28.

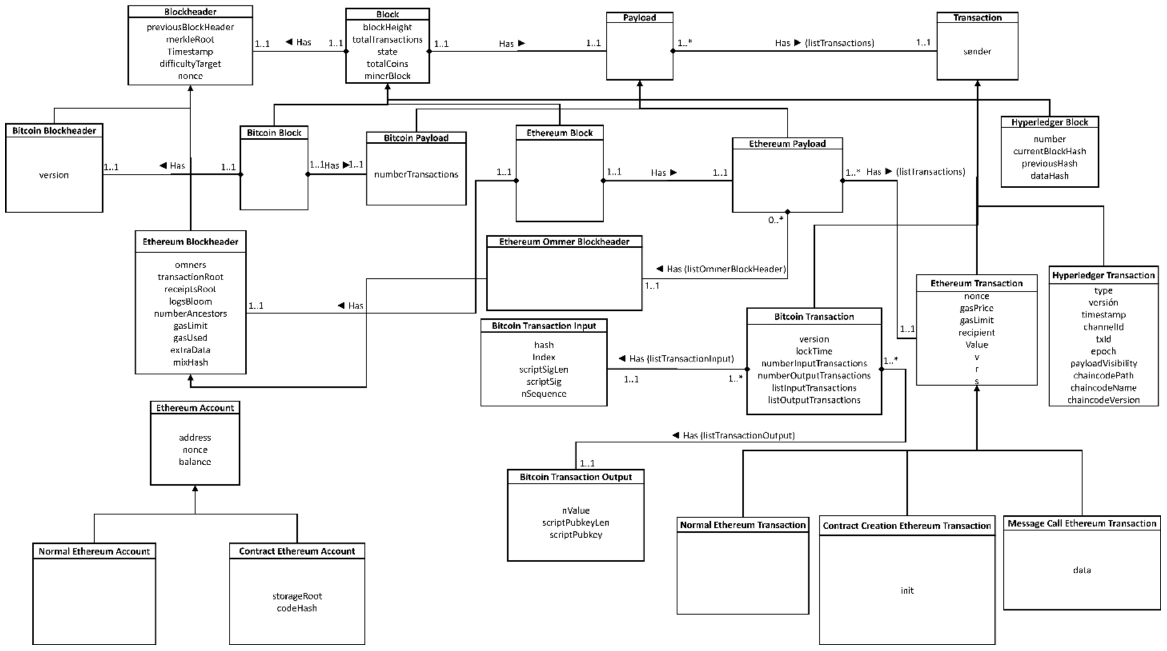


Figure 28: UML schema of the BLONDiE ontology

Specifically, OC-Ethereum mainly adopts the BLONDiE definitions of *EthereumBlock*, *Ethereum-Payload*, *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 29. As 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 *introducesEthereumS-*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

martContractExecution, 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.

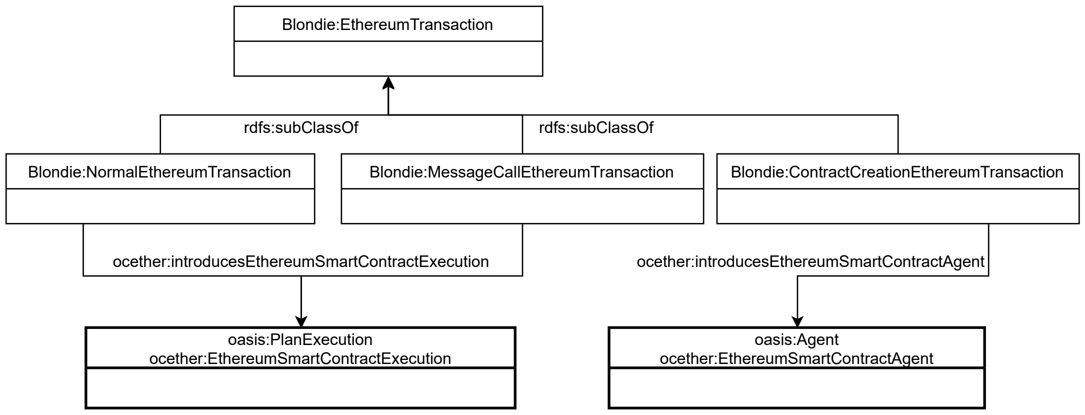

Blondie:EthereumTransaction
rdfs:subClassOf
rdfs:subClassOf
Blondie:NormalEthereumTransaction
Blondie:MessageCallEthereumTransaction
Blondie:ContractCreationEthereumTransaction
ocether:introducesEthereumSmartContractExecution
ocether:introducesEthereumSmartContractAgent
oasis:PlanExecution
ocether:EthereumSmartContractExecution
oasis:Agent
ocether:EthereumSmartContractAgent

Figure 29: OC-Ethereum representation of smart contracts and smart contract actions

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;
- *EthereumSemiFungibleSmartContractAgent*, representing semi-fungible smart contracts, and containing the class *EthereumERC1155SmartContractAgent* that represents smart contracts compliant with the ERC1155 standard protocol;
- *EthereumNonFungibleSmartContractAgent*, representing non-fungible smart contracts, and containing the class *EthereumERC720SmartContractAgent* that represents smart contracts compliant with the ERC720 standard protocol.
- *CustomEthereumSmartContractAgent*, representing user-defined smart contracts that are not compliant with the ERC standards.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

For example, the smart contract generated by the apple producer is represented by means of the fragment depicted in Figure 30.

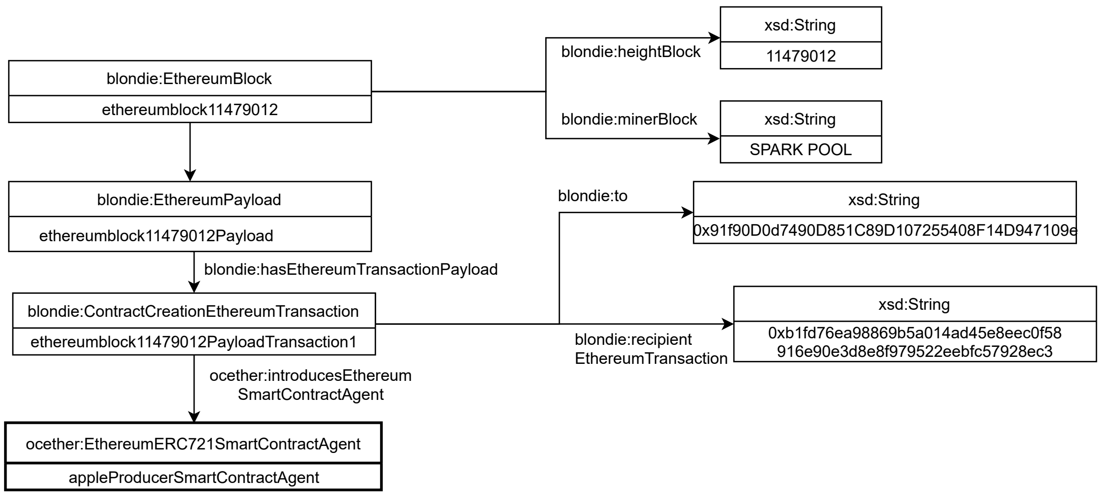

blondie:EthereumBlock
ethereumblock11479012
blondie:heightBlock
xsd:String
11479012
blondie:minerBlock
xsd:String
SPARK POOL
blondie:EthereumPayload
ethereumblock11479012Payload
blondie:hasEthereumTransactionPayload
blondie:to
xsd:String
0x91f90D0d7490D851C89D107255408F14D947109e
blondie:ContractCreationEthereumTransaction
ethereumblock11479012PayloadTransaction1
blondie:recipient EthereumTransaction
xsd:String
0xb1fd76ea98869b5a014ad45e8eec0f58 916e90e3d8e8f979522eebfc57928ec3
ocether:introducesEthereum SmartContractAgent
ocether:EthereumERC721SmartContractAgent
appleProducerSmartContractAgent

Figure 30: Representing the apple producer's smart contract in OC-Ethereum

The block and the transaction securing the apple producer smart contract 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, (in the case of the example, the entity *appleProducerSmartContractAgent*), by means of the OC-Ethereum object-property *introducesEthereumSmartContractAgent.*

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 31, OC-Ethereum identifies four main types of token:

- 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;

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- 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;
- 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, represented by instances of the class *EthereumCustomToken*.

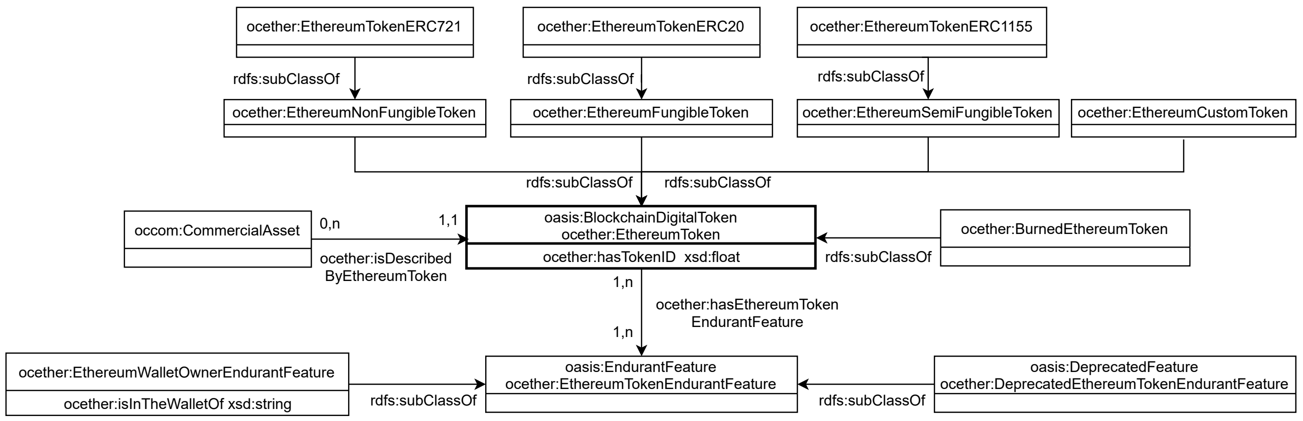

ocether:EthereumTokenERC721
ocether:EthereumTokenERC20
ocether:EthereumTokenERC1155
rdfs:subClassOf
rdfs:subClassOf
rdfs:subClassOf
ocether:EthereumNonFungibleToken
ocether:EthereumFungibleToken
ocether:EthereumSemiFungibleToken
ocether:EthereumCustomToken
rdfs:subClassOf
rdfs:subClassOf
occom:CommercialAsset
0,n
1,1
ocether:isDescribed ByEthereumToken
oasis:BlockchainDigitalToken
ocether:EthereumToken
ocether:hasTokenID xsd:float
ocether:BurnedEthereumToken
rdfs:subClassOf
1,n
ocether:hasEthereumToken EndurantFeature
1,n
ocether:EthereumWalletOwnerEndurantFeature
ocether:isInTheWalletOf xsd:string
rdfs:subClassOf
oasis:EndurantFeature
ocether:EthereumTokenEndurantFeature
rdfs:subClassOf
oasis:DeprecatedFeature
ocether:DeprecatedEthereumTokenEndurantFeature

Figure 31: Representing Ethereum tokens in OC-Ethereum

The four mentioned classes are defined as subclass 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 [11], 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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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 means of a fresh instance of the class *EndurantFeature* as illustrated in Figure 32.

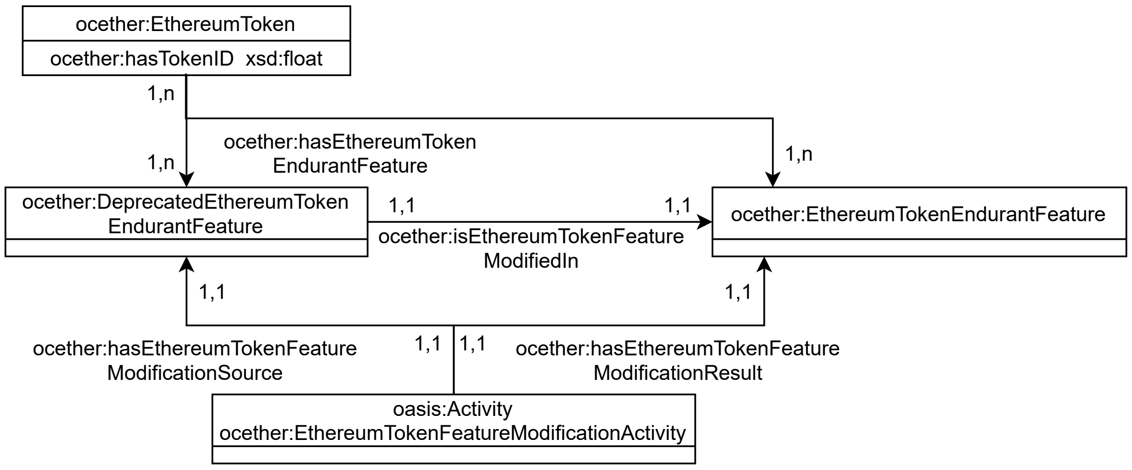

ocether:EthereumToken
ocether:hasTokenID xsd:float
1,n
ocether:hasEthereumToken EndurantFeature
1,n
1,n
ocether:DeprecatedEthereumToken EndurantFeature
1,1
1,1
ocether:isEthereumTokenFeature ModifiedIn
ocether:EthereumTokenEndurantFeature
1,1
1,1
ocether:hasEthereumTokenFeature ModificationSource
1,1
1,1
ocether:hasEthereumTokenFeature ModificationResult
oasis:Activity
ocether:EthereumTokenFeatureModificationActivity

Figure 32: Representing token modifications in OC-Ethereum

In OC-Ethereum, the modification of tokens is allowed only if it involves endurant features and hence perdurant features cannot change. 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 33, which shows a token emitted by the apple producer's smart contract.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

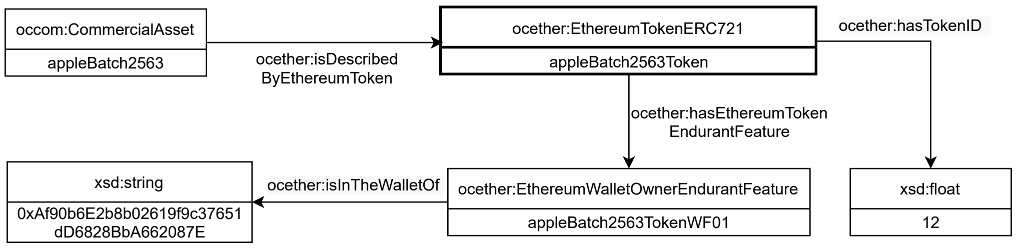
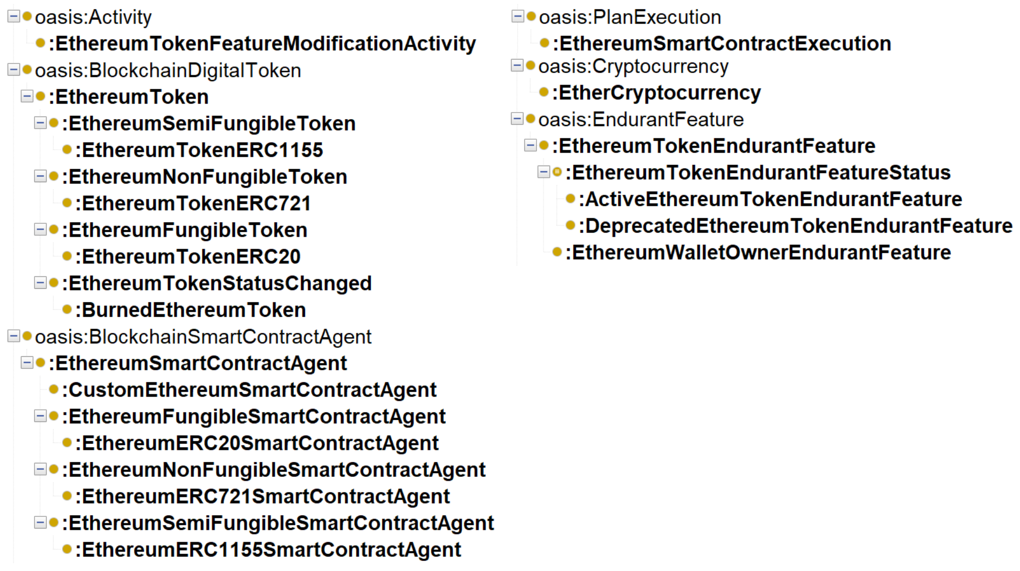

occom:CommercialAsset
appleBatch2563
ocether:isDescribed ByEthereumToken
ocether:EthereumTokenERC721
appleBatch2563Token
ocether:hasTokenID
ocether:hasEthereumToken EndurantFeature
xsd:string
0xAf90b6E2b8b02619f9c37651 dD6828BbA662087E
ocether:isInTheWalletOf
ocether:EthereumWalletOwnerEndurantFeature
appleBatch2563TokenWF01
xsd:float
12

Figure 33: Example of representing Ethereum tokens in OC-Ethereum

The token *appleBatch2563Token* and having identification code *12* is associated with the apple batch *appleBatch2563*. 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*.

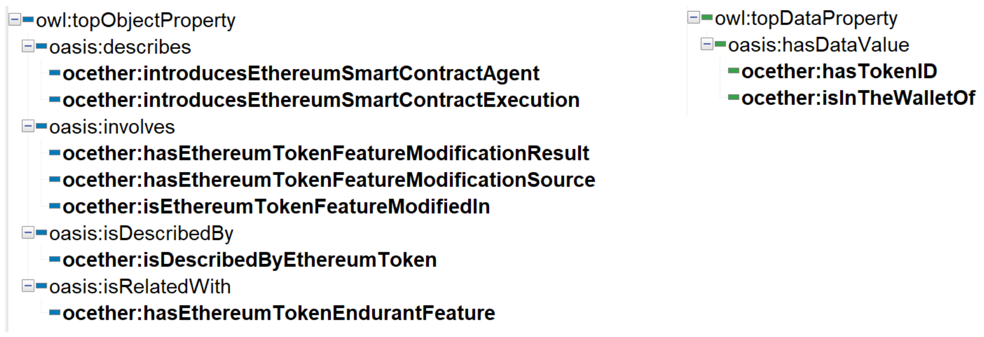
Classes and properties defined in OC-Ethereum are summarized in Figure 34 and Figure 35, respectively.



- oasis:Activity
  - **:EthereumTokenFeatureModificationActivity**
- oasis:BlockchainDigitalToken
  - **:EthereumToken**
    - **:EthereumSemiFungibleToken**
      - **:EthereumTokenERC1155**
    - **:EthereumNonFungibleToken**
      - **:EthereumTokenERC721**
    - **:EthereumFungibleToken**
      - **:EthereumTokenERC20**
    - **:EthereumTokenStatusChanged**
      - **:BurnedEthereumToken**
- oasis:BlockchainSmartContractAgent
  - **:EthereumSmartContractAgent**
    - **:CustomEthereumSmartContractAgent**
    - **:EthereumFungibleSmartContractAgent**
      - **:EthereumERC20SmartContractAgent**
    - **:EthereumNonFungibleSmartContractAgent**
      - **:EthereumERC721SmartContractAgent**
    - **:EthereumSemiFungibleSmartContractAgent**
      - **:EthereumERC1155SmartContractAgent**
- oasis:PlanExecution
  - **:EthereumSmartContractExecution**
- oasis:Cryptocurrency
  - **:EtherCryptocurrency**
- oasis:EndurantFeature
  - **:EthereumTokenEndurantFeature**
    - **:EthereumTokenEndurantFeatureStatus**
      - **:ActiveEthereumTokenEndurantFeature**
      - **:DeprecatedEthereumTokenEndurantFeature**
    - **:EthereumWalletOwnerEndurantFeature**

Figure 34: OC-Ethereum's classes

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338



```
⊟ owl:topObjectProperty
  ⊟ oasis:describes
      ocether:introducesEthereumSmartContractAgent
      ocether:introducesEthereumSmartContractExecution
  ⊟ oasis:involves
      ocether:hasEthereumTokenFeatureModificationResult
      ocether:hasEthereumTokenFeatureModificationSource
      ocether:isEthereumTokenFeatureModifiedIn
  ⊟ oasis:isDescribedBy
      ocether:isDescribedByEthereumToken
  ⊟ oasis:isRelatedWith
      ocether:hasEthereumTokenEndurantFeature
```

```
⊟ owl:topDataProperty
  ⊟ oasis:hasDataValue
      ocether:hasTokenID
      ocether:isInTheWalletOf
```

Figure 35: OC-Ethereum's object-properties (on the left) and data-properties (on the right)

4 Evaluation Methodology

The ontological stack of POC4COMMERCE is evaluated during the entire life-time through the KPIs stated in the original project proposal and confirmed in the Deliverable 2, which can be summarized in the following four points:

- **Consistency check**. Consistency check performed at least by two semantic web reasoners aims to demonstrate the consistency of the ontologies.
- **Structural metrics**. Structural metrics depict the number and type of elements used to define the ontologies, such as number of classes and object-properties, and provide a general evaluation of how much the ontologies are large and complex. We adopted Protégé to compute the structural metrics of the POC4COMMERCE ontologies.
- **Ontological metrics**. 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 are computed exploiting the OntoQA approach [21]. OntoQA is a feature-based method for evaluating ontologies applying techniques that do not require data training and that involve users in a minimal way. We take 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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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 a 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.

- **Competency questions**. 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 an explanation of the results,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

4.1 Evaluation of the OC-Found ontology

The principal structural metrics of OC-Found are reported in Table 1, whereas the consistency of OC-Found has been checked by the reasoners Pellet [20], HermiT [12], and FaCT++ [22]. The metrics are referred to the new entities introduced by OC-Found and hence do not include the classes and properties defined in the imported ontology OASIS.

Metric type	OC-Found metric value
Axiom count	68
Logical axiom count	32
Declaration axiom count	19
Class count	17
Object-property count	14
Subclass axiom count	12
Sub-object-property axiom count	8
Object property domain axiom count	6
Object property range axiom count	6
Annotation assertion	17

Table 1: Structural metrics of OC-Found

The ontological metrics for OC-Found are reported in Table 2.

Evaluation criteria	OC-Found value
Relationship Richness	53.84
Inheritance richness	2
Tree balance	0.90
Attribute richness	0.61
Class richness	11.11

Table 2: Ontological metrics for OC-Found computed by OntoQA.

OC-Found reports a relationship richness of 53.84%, depicting a good balancing between generic relationships and class hierarchies, whereas the inheritance richness stands at 2%, confirming the vertical nature of the ontology, which is mainly focused on supply chains (we recall that agent descriptions are inherited from OASIS). The foundational nature of the ontology focused on a relatively

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

small domain (namely, supply chains) is confirmed by its low class richness (11.11%) and attribute richness (0.61%). Finally, OC-Found provides a tree balance of 0.90% meaning that some hierarchies have been well described (e.g., *SupplyChainActivity*) whereas others remain very general (e.g., *DigitalIdentity*).

OC-Found answers at least to the following competency questions:

- CF1: Which are the participants currently available (including the associated digital identities and operations they can perform)?
- CF2: Which actions have been performed (including the agents responsible for the execution and the type of action performed)?
- CF3: Given the resource *resource*, which is, if any, the supply chain of *resource*?
- CF4: Given the resource *resource*, how the supply chain of *resource* is constituted?
- CF5: Given the resource *resource*, which are the agents responsible for *resource*'s supply chain activities?
- CF6: Given the resource *resource*, which are the valuations (including the valuer agents) performed on *resource*?
- CF7: Given the resource *resource*, which is the average score of valuation of *resource* and how many valuations there are?

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) }
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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)
```

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 .}
```

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.}
```

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. }
```

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

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. }
```

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. }
```

4.2 Evaluation of the OC-Commerce ontology

Consistency of OC-Commerce has been checked by the reasoners Pellet, HermiT, and FaCT++. The principal structural metrics of OC-Commerce involving the newly introduced entities, and hence excluding the imported ontology OC-Found, are reported in Table 3.

Metric type	OC-Commerce metric value
Axiom count	113
Logical axiom count	61
Declaration axiom count	29
Class count	30
Object property count	18

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Subclass axiom count	25
Sub-object-property axiom count	15
Object property domain axiom count	10
Object property range axiom count	10
Annotation assertion	23

Table 3: Structural metrics of OC-Commerce

The ontological metrics for OC-Commerce computed by OntoQA are reported in Table 4.

Evaluation criteria	OC-Commerce value
Relationship Richness	38.29
Inheritance richness	2.07
Tree balance	1.35
Attribute richness	0.5
Class richness	6.25

Table 4: Ontological metrics for OC-Commerce computed by OntoQA.

Relationship richness of OC-Commerce is less than the one of OC-Found (38.29%), due to the fact that OC-Commerce inherits and extends GoodRelations, but it has still a good balancing between description of properties and class hierarchies. Also inheritance and attribute richness (2.07% and 0.5%, respectively) are closed to the ones computed on OC-Found, confirming that also OC-Commerce is a vertical ontology, since it is focused on offering descriptions. Tree balance in OC-Commerce, as expected, is slightly higher than the one computed on OC-Found, because the domain of commercial offerings is less general and demands more details, but there is also some intrinsic differences on the deepness level between class hierarchies such as *Offerings* and *PriceDeterminationActivity*. Finally, class richness remains low since data are not present within the ontology, due to its higher-level nature.

OC-Commerce answers at least to the following competency questions:

- CC1: Which are the available offerings (including related details)?
- CC2: Given an offering *offering*, which is the supply chain related with *offering*?
- CC3: Which are the accepted offerings?

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

The answer to CC1 is entailed by the following SPARQL query (QC1):

Query 8 The query QC1.

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

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.}
```

The answer to CC3 is 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) }
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

4.3 Evaluation of the OC-Ethereum ontology

The principal structural metrics concerning the new entities introduced by OC-Ethereum, and hence excluding OC-Commerce, are reported in Table 5. As in the case of OC-Found and OC-Commerce, consistency check has been carried out by the reasoners Pellet, HermiT, and FaCT++.

Metric type	OC-Ethereum metric value
Axiom count	149
Logical axiom count	65
Declaration axiom count	58
Class count	45
Object property count	11
Data property count	3
Subclass axiom count	28
Sub-object-property axiom count	7
Object property domain axiom count	7
Object property range axiom count	8
Data property domain axiom count	2
Data property range axiom count	2
Annotation assertion	26

Table 5: Structural metrics of OC-Commerce

The ontological metrics for OC-Ethereum computed by OntoQA are reported in Table 6.

Evaluation criteria	OC-Ethereum value
Relationship Richness	29.16
Inheritance richness	1.36
Tree balance	0.83
Attribute richness	0.59
Class richness	9.09

Table 6: Ontological metrics for OC-Commerce computed by OntoQA.

Relationship richness of OC-Ethereum is close to the one computed on OC-Commerce and OC-Found (29.16%), due to the richness of class hierarchies regarding tokens and smart contracts. There-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

fore, the ontology still provides a sufficient balancing between descriptions of properties and class hierarchies. Inheritance and attribute richness (1.36% and 0.59%, respectively) are closed to the ones 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, for example, between the class hierarchy of *Ethereum-Token* and the class hierarchy of *EthereumTokenEndurantFeature*. Finally, class richness remains low (9.09%) since data are not included within the ontology.

OC-Ethereum answers at least to the following competency questions, in addition to the one provided by BLONDiE (see [23]):

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

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

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

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

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. }
```

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Query 12 The query QE2.

```
1: Let *token* be the token of which block number and transaction hash should be discovered
2:    SELECT ?blockNumber ?hash
3:    WHERE {
4:        ?block blon:heightBlock ?blockNumber.
5:        ?block blon:hasEthereumPayloadBlock ?payload.
6:        ?payload blon:hasEthereumTransactionPayload ?transaction.
7:        ?transaction blon:recipientEthereumTransaction ?hash.
8:        ?transaction ocether:introducesEthereumSmartContractExecution ?action.
9:        ?action oasis:consistsOfGoalExecution ?goal.
10:       ?goal oasis:consistsOfTaskExecution ?agentExe.
11:       ?agent oasis:performs ?agentExe.
12:       ?agentExe oasis:hasTaskObject ?taskExe.
13:       ?agentExe oasis:hasTaskOperator ?operator.
14:       ?operator oasis:refersExactlyTo oabox:mint.
15:       ?taskExe oasis:refersExactlyTo *token*. }
```

The answer to CE3 is entailed by the following SPARQL query (QE3)

Query 13 The query QE3.

```
1: Let *assetType* the type of asset related with the smart contract to be discovered
2:    SELECT DISTINCT ?agent ?hash ?address
3:    WHERE {
4:        ?agent oasis:performs ?agentExe.
5:        ?agentExe oasis:hasTaskObject ?taskExe.
6:        ?agentExe oasis:hasTaskOperator ?operator.
7:        ?operator oasis:refersExactlyTo oabox:mint.
8:        ?taskExe oasis:refersExactlyTo ?token.
9:        ?asset ocether:isDescribedByEthereumToken ?token.
10:       ?asset a *assetType*.
11:       ?block blon:heightBlock ?blockNumber.
12:       ?block blon:hasEthereumPayloadBlock ?payload.
13:       ?payload blon:hasEthereumTransactionPayload ?transaction.
14:       ?transaction blon:recipientEthereumTransaction ?hash.
15:       ?transaction ocether:introducesEthereumSmartContractAgent ?agent.
16:       ?transaction blon:to ?address. }
```

The answer to CE4 is entailed by the following SPARQL query (QE4)

Query 14 The query QE4.

```
1:    SELECT ?owner (COUNT(?operationOn) as ?tokenCounter) ?assetType
2:    WHERE {
3:        ?asset a ?assetType.
4:        FILTER(?assetType != owl:NamedIndividual).
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

```
?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
```

5 Detailed Demonstration Design and WorkFlow

The high versatility of the POC4COMMERCE solution allows one to implement a plethora of different use cases where sellers expose the offerings concerning their available assets together with the related supply chains, and potential buyers probe the ontological knowledge bases to profitably find products and services according to their desiderata. Typical use cases figured out by POC4COMMERCE are summarized in Figure 36. A typical use case depicts a green apples vendor called *AppleBay* that wants to sell his assets by granting to the buyers an Ethereum ERC721 compliant non-fungible token (NFT) assigning ownership rights for the specific batch of apples purchased. A suitable smart contract is published on the Ethereum blockchain to mint and transfer ERC721 compliant tokens representing apple batches. The seller also manifests that the available payment system is *PayPal* and that shipment of merchandises is entrusted to the *Fedex* courier.

On the other side, a potential user buyer *Bob* would purchase apples using FIAT currency through a digital payment platform such as PayPal. To complete the purchase, a token corresponding to the apples batch brought by Bob should be mint and transferred to his Ethereum wallet as a proof of the quality and quantity of product purchased, and of the payment received by the seller. Then, once the seller have received the payment for the batch of apples and have transferred the related token to the buyer, the product shipment process can be finalized through the shipment service indicated by the seller.

As preliminary step, to join the commercial ecosystem, both the participants AppleBay and Bob obtain from the ONTOCHAIN ecosystem their digital identities. Hence, participants make available through one of the semantic knowledge bases the OC-Found-compliant ontological descriptions of both the agent representing them and the related digital identities. The participants also require a suitable API assisting them with generating the agent behaviors and publishing in the knowledge base. We recall that the deployed solution abstracts from the implementation strategy adopted for the knowledge base: indeed the latter can be realized either by exploiting a) a combination of Ethereum and the *Interplanetary File System* (IPFS) [18], as in the case illustrated in [3], and maintained by the participants, b) a RDF triple-store such as Neo4J or OpenLink Virtuoso maintained by a centralized

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

and certified authority, c) by a combination of a) and b), d) adopting the side-chains solution realized by the winning project partner GraphChain and consisting in a combination of the Hyperledger Bezu client[12] and the Blazegraph[13] triple-store. Specifically, exploiting the description model provided by OC-Found (see Section 3.2),

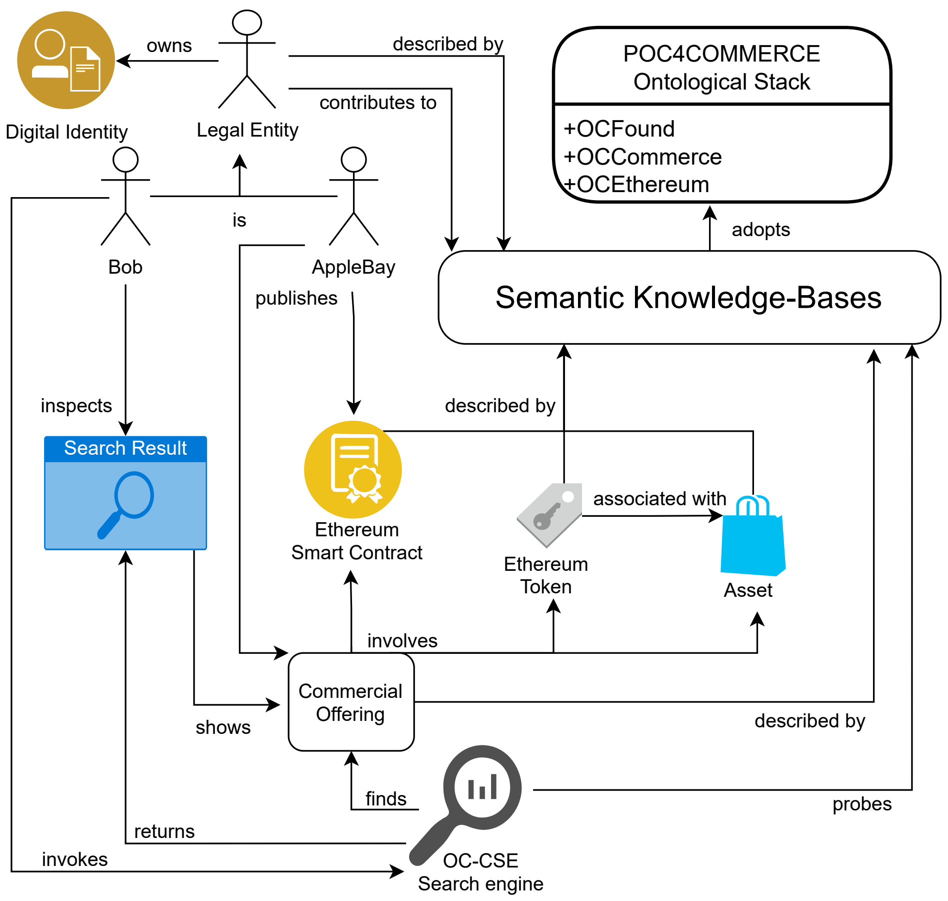


Figure 36: A typical POC4COMMERCE use case

- AppleBay adopts at least three behaviors, the *produce apple behavior*, the *offering publish behavior*, the *close offering behavior*, the *price determine behavior*.

[12]https://www.hyperledger.org/use/besu
[13]https://blazegraph.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- AppleBay publishes an ERC721 compliant smart contract that adopts at least the *mint token behavior*.
- Bob adopts at least the *accept offering behavior*. Additionally, he can adopts a *perform quality valuation behavior*.
- The PayPal agent adopts at least the *money transfer behavior* that includes as parameters the buyer account, the seller account, and the payed offering.
- The Fedex courier adopts at least the *ship product behavior*.

Each agent must be provided with a digital identity, modelled thanks to the OC-Found representation schema introduced in Section 3.2. Moreover, AppleBay and Bob must embed into their digital identities the Ethereum wallet adopted to store tokens and sign transactions.

In the second phase, AppleBay produces at least one batch of apples and makes available an offering concerning the released product. Specifically, AppleBay

- exploiting OC-Found, publishes a RDF fragment representing a *produce apple action* connected with its *produce apple behavior*. External ontologies such as the *AgroVoc* vocabulary[14] may be used to provide a description of the features of the apple batch released;
- exploiting OC-Commerce (see Section 3.3), publishes a RDF fragment representing a new offering by means of a *publish offering action* connected to its *publish offering behavior*. The offering includes the supply chain as described by OC-Found and specifying a) as behavior connected to the *release activity*, the AppleBay *produce apple behavior*; b) as behavior connected to the *proof of work activity*, the AppleBay *mint token behavior*; c) as behavior connected to the *payment activity*, the PayPal *money transfer behavior*; d) as behavior connected to the *delivery activity*, the Fedex *ship product behavior*;
- exploiting OC-Found, publishes a RDF fragment representing a *price determine action* connected to its *price determine behavior*, in order to provide the offering with the selling price.

Next, Bob exploits the OC-CSE search engine to find the products he desires to buy. Assuming that the offering previously published by AppleBay satisfies Bob's requirements, and hence Bob would complete the purchase of the apple batch, Bob publishes

[14]http://aims.fao.org/agrovoc

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- exploiting OC-Found, a RDF fragment representing an *accept offering action* connected to its *accept offering behavior*.

Once the offering is accepted and Bob payed for the offering, the following steps are carried out:

- PayPal publishes a RDF fragment representing a *money transfer action* connected with its *money transfer behavior* involving as parameters the AppleBay account as buyer, the Bob account as seller, and AppleBay offering as payed offering. A receipt witnessing the completed purchase is published and related with the action as an output parameter.
- AppleBay publishes a RDF fragment representing a *close offering action* connected with its *close offering behavior*, as manifest of the completed purchase.
- AppleBay, exploiting OC-Ethereum, publishes a RDF fragment representing a *mint token action* connected with its smart contract *mint token behavior*. On the blockchain Ethereum the smart contract should be invoked in order to mint and transfer to his wallet a token corresponding to the apple batch purchased by Bob. The knowledge base is updated by connecting the OC-Ethereum description of the emitted token to the asset purchased by Bob (see Section 3.4. The description of the token also includes information about block and transaction storing the token on the blockchain.
- The Fedex courier publishes a RDF fragment representing a *ship product action* connected with its *ship product behavior*, including as output parameter the tracking code of the shipping.

Finally, Bob can check the status of the purchase at any phase, for example, he can check whether the token has been mint for the product he payed for by exploiting the SPARQL query CE1 (see Section 3.4. Additionally, Bob can valuate his purchase experience by publishing a *perform quality valuation action* connected with his *perform quality valuation behavior* involving the offering of the purchased batch of apples or one of the agent involved in the supply chain of the asset.

Moreover, other extensions or variants of the above use case can be considered. For example, there may be an alternative supply chain for the apple batch that considers the transferring of a previously existing token instead of minting a new one, or the Fedex courier may publish an offering for booking pickups and deliveries of products.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

6 Software as Part of the ONTOCHAIN Ecosystem

POC4COMMERCE delivers a set of three modular ontologies describing each semantic compartment of eCommerce, from participants, assets, and offerings, to supply chains, smart contracts, and digital tokens. The ontological stack is released together with the SPARQL queries implementing the defined competency questions that enable users and developers to meaningfully probe the knowledge bases they contribute to construct. The advantage of the ontological stack relies on a high versatility and technological independence, for example, of the underlying storage services, of the serialization languages, and of the reasoning and query mechanisms. Moreover, the modularity of the ontological allows user and developers to adopts the required semantic representations for their purposes. API libraries can be built a) to simplify the semi-automatic generation of suitable RDF fragments adopting the semantic model delivered by the POC4COMMERCE ontological stack, in order to provide a high-level interface for populating and enriching the semantic knowledge base of the ONTOCHAIN ecosystem, and b) to probe the user-defined knowledge bases.

Specifically, API libraries are required to:

- generate RDF fragments representing participants, such as services and smart contracts, in the shape of agent behaviors as illustrated in Section 3.1.2. Additionally, agent behavior templates (see Section 3.1.1) can be defined in advance for sets of specific agents commonly used in the ONTOCHAIN ecosystem;
- generate RDF fragments representing agent actions connected with the corresponding agent behaviors, for example to make available assets, as illustrated in Section 3.1.3;
- generate RDF fragments representing commercial offerings and related supply chains together with price determination activities, as described in Section 3.3;
- generate RDF fragments representing tokens related with assets and smart contract actions responsible for their management, as described in Section 3.4;

API libraries are also required to probe the user-defined knowledge bases for retrieving information about

- agents, their actions, and supply chains of resources and offerings, as described in Section 4.1, competency questions CF1–CF7.
- available commercial offerings and related supply chains, as described in Section 4.2, competency questions CC1–CF3.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- tokens, smart contracts, and smart contract transactions related with tokens associated with assets, as described in Section 4.3, competency questions CE1–CE4.

For what concerns the integration of the ontological stack with the other winning project partners, we recall that

- The *CopyrightLY* project can adopts the OC-Found ontology to describe the agents responsible for the release of copyrights;
- the *OntoSsiValut* project can adopts the OC-Found ontology to describe digital identities of participants and the agents responsible for releasing digital identities;
- the *DR-HIBI* project can exploit the digital identity representation mechanism of OC-Found to provide information regarding the association between agent and Ethereum wallet owned;
- the *GraphChain* project can provide the tools and technologies for serializing and storing both the ontological stack and the knowledge bases leveraging it;
- the *Reputable* project can enjoy the quality valuation mechanism introduced by OC-Found to store the performed quality valuations and to represent the agents responsible for publishing valuations;
- the *KnowledgeX* can exploits the SPARQL queries for the competency questions defined in Section 4 to extract meaningfully information concerning agents, offerings and tokens. Moreover, the OC-CSE search engine can be also integrated with the trusted data-driven knowledge extraction application implemented by KnowledgeX.

7 Conclusions

POC4COMMERCE innovates the ontological representation of Ethereum blockchain-oriented commerce by integrating and extending the most representative ontologies for modelling commercial actors, offerings, and assets such as services and products. To achieve the goal, POC4COMMERCE provides an ontological stack as representation means to describe the ONTOCHAIN participants, their roles and capabilities related (not only) with the commercial domain, ranging from service and product providers to clients, including assets and related supply chains, together with an experience-oriented trustworthiness mechanism. In particular, concerning commercial participants, the POC4COMMERCE

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ontological stack describes how they are supposed to provide goods and related selling conditions. Commercial offers are integrated in the ontological stack provided by POC4COMMERCE as to include the description of commercial agents in the fashion of the mentalistic notion of agent behavior. Also Ethereum smart contracts and, in particular, smart contracts for exchanging tokens for commercial purposes are suitably represented together with blockchain transaction information. The value of the solution of POC4COMMERCE is very aptly evidenced by several points and confirms what have been stated in the project proposal:

- *Semantic formalisation of agents independently from their implementation details and configurations representing ONTOCHAIN actors and participants, in particular those relevant for the digital commerce domain, focusing on products and services supply chains.* POC4COMMERCE conceives active web resources as agents that are in turn modelled through their respective behaviors, i.e., their ability to contribute to the environment.

- *Facilitation and support for users and applications to access semantically discernible, data-oriented web content.* POC4COMMERCE supports each stakeholder's publication of its semantic "manifesto", expressing, e.g., what types of data or services the stakeholder treats or exposes and how anyone can seamlessly interact with those.

- *Enablement of users and applications to locate, select, employ, compose, and monitor web-based services automatically and autonomously.* POC4COMMERCE demonstrates this, in particular, but not limited, over a digital marketplace, providing a complete description not only of key elements such as providers, buyers and assets, but also of eCommerce logics such as price determination mechanisms including auctions, payment systems, transportation means and all other activities aimed at facilitating the match of demand and offer, including agent experience-based trustworthiness mechanism.

- *Wide semantic formalisation of Ethereum blockchain and related smart contracts.* POC4COMMERCE may cover a virtually infinite plethora of agent types and, notably, the Ethereum blockchain, including fungible, non-fungible, and semi-fungible tokens, in particular those related to (physical) products or services involved in commercial exchanges, along with the smart contracts for minting, destroying, and managing exchanges of tokens. Moreover, tokens are meaningfully associated with traded assets so as to provide a semantic probing of the blockchain and of cross-chain infrastructure.

- *Provision of knowledge base necessary for automated web service discovery, invocation, composition and interoperation.* POC4COMMERCE also sketches a foundational tool to prof-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

itably probe the conceived semantic blockchain, namely, the OC-CSE search engine. OC-CSE will be designed on Semantic Web API and reasoning service interfaces to generate suitable and parametric SPARQL queries implemented on well-founded competency questions defined on the three POC4COMMERCE ontologies, required by users and applications to probe the ONTOCHAIN ecosystem and in particular the digital marketplaces realized leveraging the POC4COMMERCE ontological stack.

At the time of writing, the ontological stack is deployed at TRL3 and the representation goals indicated in the project proposal have been achieved as also witnessed both by the competency questions designed on the three ontologies and implemented through suitable SPARQL queries and by the metrics criteria. The ontologies, as any epistemological process, will evolve also in light of the development of the ONTOCHAIN ecosystem and its requirements, and hence, modification at any level of the ontological stack can be committed during the final stages of the project. In an analogous way, competency questions can be also extended, or even modified, to answer the challenges arisen from the future requirements of the ecosystem. As consequence, the related SPARQL queries will be updated.

To totally fulfil the project proposal, a small set of data from iExec shall be adopted as workbench for the ontological stack in the final stage of the current phase. As consequence, the OC-CSE search engine shall be designed leveraging the latest versions of the ontological stack and exploiting the results of real world data testing, in light of the collaboration with the winner project partners and the POC4COMMERCE coaches.

References

[1] 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):203–236, 2004.

[2] 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*, pages 119–126, 2019.

[3] 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*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

app. in: Proceedings of IDC 2021, The 14th International Symposium on Intelligent Distributed Computing, 16-18 September, Scilla, Reggio Calabria, Italy, 2021.

[4] 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, pages 813–822, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[5] M. Cossentino, M. Gleizes, A. Molesini, and A. Omicini. Processes Engineering and AOSE. In Marie-Pierre Gleizes and Jorge J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, pages 191–212, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[6] K. H. Dam and M. Winikoff. Comparing Agent-Oriented Methodologies. In Paolo Giorgini, Brian Henderson-Sellers, and Michael Winikoff, editors, *Agent-Oriented Information Systems*, pages 78–93, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[7] M. Esteva. Electronic institutions: from specification to development. *IIIA Monograph Series. PhD Thesis*, 19, 2003.

[8] L. Fichera, F. Messina, G. Pappalardo, and C. Santoro. A Python Framework for Programming Autonomous Robots Using a Declarative Approach. *Science of Computer Programming*, 139:36–55, 2017.

[9] N. Fornara and M. Colombetti. A commitment-based approach to agent communication. *Applied Artificial Intelligence*, pages 853–866, 2004.

[10] A. Freitas, R. H. Bordini, and R. Vieira. Model-driven engineering of multi-agent systems based on ontologies. *Applied Ontology*, 12:157–188, 2017.

[11] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. *Sweetening Ontologies with DOLCE*, pages 166–181. Springer, 2002.

[12] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[13] M. Hepp. Goodrelations: An ontology for describing products and services offers on the web. In Aldo Gangemi and Jérôme Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2008.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

[14] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language Primer. W3C Recommendation, World Wide Web Consortium, October 2009.

[15] P.D. Larson and D.S. Rogers. Supply chain management: Definition, growth and approaches. *Journal of Marketing Theory and Practice*, 6(4):1–5, 1998.

[16] F. Manola and E. Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium, February 2004.

[17] M.A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[18] Protocol Labs. The Interplanetary File System (IPFS). https://ipfs.io/.

[19] 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*, pages 312–319, 1995.

[20] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics*, 5(2):51–53, 2007.

[21] 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.

[22] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 292–297, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[23] H. Ugarte-Rojas and B. Chullo-Llave. Blondie: Blockchain ontology with dynamic extensibility. *CoRR*, abs/2008.09518, 2020.

[24] H. E. Ugarte Rojas. A more pragmatic web 3.0: Linked blockchain data. In *Google Scholar*, 2017.

[25] M. Wick. Geonames ontology, 2015.

[26] World Wide Web Consortium. OWL-S: Semantic Markup for Web Services, 2004.

[27] World Wide Web Consortium. SPARQL 1.1 Query Language, 2013.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338