



care, judgment, dexterity

CRAEFT

Maker-Material-Negotiation Model and CAP

Project Acronym	Craeft
Project Title	Craft Understanding, Education, Training, and Preservation for Posterity and Prosperity
Project Number	101094349
Deliverable Number	D2.2, second release
Deliverable Title	Maker-Material-Negotiation Model and CAP
Work Package	WP2
Authors	Carlo Meghini, Valentina Bartalesi, Nicolò Pratelli, Voula Doulgeraki, Ioanna Demeridou, Nikolaos Partarakis, Xenophon Zabulis
Number of pages	119



This project has received funding from the European Commission, under the Horizon Europe research and innovation programme, Grant Agreement No 101094349.

<http://www.craeft.eu/>

Executive summary

The present deliverable reports the work done in Tasks T2.3 “Maker-Material-Negotiation model” and T2.4 “CRAEFT Authoring Platform” during the second year of the project. It is an update of the version of deliverable D2.2 released at M12. The updates to the ontology are all additions, which means that the updated ontology is backwards compatible with the previous one. These additions are substantial and have been made in response to the inadequacies of the ontology that have been discovered by using the ontology to represent realistic craft instances.

For self-containedness and readability, the present document does not merely report the updates, but it is structured as its first version so that a reader can find in here all the information without having to read the first version. For the reader who has already read the first version, here is a summary of the main additions and the rationale behind them.

- Generalised properties have been introduced to empower the ontology for the representation of domain relations that apply to several classes, having a different range for each such class. A generalised property is axiomatized following a specific pattern.
- A top class has been added for representing the entities that pertain to each of the three levels in which the ontology is articulated: the schema, the virtual and the real level. This addition permits a better structuring of the ontology and eases the expressions of axioms on the domain and range of properties.
- For identical reasons, one specific type of narrative has been added for the representation of crafts.
- One property for re-using step schemas within process schemas has been added, allowing us to preserve all constraints on the structuring of schemas while easing their specification.

Other, minor additions have also been made, such as the addition of a duration property for step schemas. These additions do not deserve a specific mention as they belong to the routine maintenance of the ontology. Readers are anyway advised to re-read Sections 2 and 3 of the document.

The document is divided into two main parts: the first part comprises Section 2, which describes the Maker-Material-Negotiation model; the second part comprises Section 3, which describes the CRAEFT Authoring Platform.

Section 2, after introducing notational conventions (Section 2.1) and describing the structure of IRIs used for the identification of CRAEFT resources (Section 2.2) is structured into the following main parts:

- Section 2.3 recapitulates the Narrative ontology, which forms an important conceptual basis of the CRAEFT ontology since it axiomatizes the vocabulary for describing processes and actions from a very general point of view. The Narrative Ontology has been used for craft representation in the Mingei innovation action, the predecessor of the CRAEFT project, and provides basic classes and properties for representing structured courses of events (called *fabulae*) documented through knowledge objects of multimedia nature (called *narrations*) linked to events by an *ad hoc* function (called *reference function*). To achieve its goals, the Narrative ontology provides fundamental notions such as agents, time and space, and does so by relying on a core ontology, the CIDOC CRM,

and domain ontologies, such as OWL Time, all standards *de jure* or *de facto*. Standards are also the language in which the Narrative Ontology is expressed (OWL 2 DL) and the notation used (RDF Turtle). Needless to say, the CRAFT Ontology adopts the standard-based approach of Narrative Ontology to achieve maximum interoperability. Section 2.3 is organized into several sub-sections: after the sub-sections presenting the main classes and properties, it includes sub-sections dealing with specific aspects, such as the representations of actor roles in events, time modelling, space, and time-varying properties. The Narrative ontology has been enriched with new classes and properties to account more properly for presentation fragments.

- Section 2.4 gives a conceptualisation of these entities, forming the basis of the ontology introduced in the subsequent section. The conceptualisation is strongly influenced by the specific context of the project, which requires three levels of representation: the schema level, where processes and actions are described by plans providing the general structure; the virtual level, where processes and actions schemas are instantiated by simulating them for given input parameters; the real level, where virtual processes and actions are enacted in a real context. Process and action schemas are modelled based on the standard set by activity diagrams of the unified Modelling language, that is graphs comprised of activity nodes and transitions. A subset of the machinery provided for activity diagrams is employed by the conceptualisation, adequate for the representation of the general structure of crafts. Virtual processes and actions are modelled to be aligned with the simulator that computes their outcomes. Real processes and actions, finally, are modelled as fabulae and events, respectively, of the Narrative Ontology. This Section includes the above-mentioned changes from the first version.
- Section 2.5 provides an axiomatization of the conceptualisation presented in the previous Section, in the OWL 2 DL language, which is the most expressive language of the OWL family retaining decidability. An important result of this axiomatization is that it covers all the laws that link the concepts involved except one, the law establishing the correct relation between the properties declared in an action schema and the properties used in a virtual action that is an instance of that schema. This law cannot be expressed in OWL 2 DL because properties in an action schema are treated as individuals. This Section is structured in two main parts: the first part covers activity nodes of process schemas, and includes actions; the second part covers control nodes of process schemas. This Section is changed as a consequence of the changes in the conceptualisation reported in the previous Section.
- Section 2.6 covers the usage of popular names for specific groups of entities drawn from widely used dictionaries, namely, the Art and Architecture Thesaurus of the Getty Foundation, the Catalogue of Art Collections, the Thesaurus of Geographic Names, and the Union List of Artist Names. The source code for these enhancements can be found at <https://zenodo.org/records/10532597>.
- Section 2.7, which in the previous version presented the representation of several craft processes and actions in the CRAFT Ontology, has been removed from this version, since at this stage of the project several craft instances have been completed and are reported elsewhere.

Section 3 outlines the main features of the Web-based, online, multiple-user authoring platform for the documentation of traditional crafts. The Craeft Authoring Platform (CAP) is an extension of the Mingei Online Platform (MOP) developed in the Mingei Innovation Action. The extension provides support for the representation of processes and actions based on the Craeft Ontology. Although it is functional and accessible online, the CAP is still under development and, thus not yet in its final form. The present deliverable describes the initial activities performed on the CAP, concerning important aspects that could

not be tackled in Mingei due to lack of time and others that were discovered during the present research. Specifically,

- Sections 3.1 and 3.2 contextualise the progress until the start of Craeft and briefly describe the MOP. Section 3.3 describes an overview of the planned improvements planned.
- In Section 3.4, we describe the addition of multilingualism features.
- In Section 3.5, we describe the features that automate the establishment of cross-references between knowledge elements in the CAP, to check the integrity and find mistakes, such as unreferenced entities and dead links.
- In Section 3.6, we describe how we use the European OpenAIRE infrastructure Zenodo to store our digital assets and submit them to Europeana extensively so that other heritage institutions can follow the same technique. In Section 3.7, we describe technical enhancements, such as the export of knowledge elements to file (Section 3.7.1) and the export of files for automatic ingestion of assets to Europeana (Section 3.7.2).
- Finally, in Section 3.7.4, we present an upgrade of the way that 3D models are viewed online in the CAP.

Finally, Section 4 concludes and summarises this deliverable.

Document history

Date	Author	Affiliation	Comment
4/2/2025	Carlo Meghini	CNR	First draft
9/2/2025	Xenophon Zabulis	FORTH	Revised

Abbreviations

3D	Three dimensional
AAT	Arts and Architecture Thesaurus
CAP	Craeft Authoring Platform
CH	Cultural Heritage
CONA	Catalogue of Art Collections
HTML	Hypertext Markup Language
MoCap	Motion Capture
MOP	Mingei Online Platform
TGN	Thesaurus of Geographic Names
RDF	Resource Description Framework
RS	Research Space
ULAN	Union List of Artist Names



UNESCO	United Nations Educational, Scientific and Cultural Organization
URL	Uniform Resource Locator
XML	Extensible Markup Language

Table of contents

Executive summary	2
Document history	4
Abbreviations	4
Table of contents	6
1 Introduction	8
2 The CRAEFT Ontology.....	9
2.1 Notational conventions.....	11
2.2 Resource identification	11
2.3 The Narrative Ontology.....	12
2.3.1 Main Classes.....	13
2.3.2 Main Properties	16
2.3.3 Representing actor roles in events	20
2.3.4 Modelling presentation fragments	21
2.3.5 Modelling Time	22
2.3.6 Modelling Space	24
2.3.7 Modelling time-varying properties	27
2.4 Modelling processes	28
2.4.1 Processes.....	28
2.4.2 Actions.....	30
2.4.3 Schemas	32
2.5 An ontology of processes and actions	35
2.5.1 Activity Schemas	36
2.5.2 Transitions.....	72
2.5.3 Reusing step schemas	80
2.6 Linking to standard dictionaries.....	83
2.6.1.1 AAT	83
2.6.1.2 CONA.....	87
2.6.1.3 TGN and Geonames	88
2.6.1.4 ULAN	90
2.6.1.5 Additional dictionaries	92
2.6.1.6 Implementation	94
3 The CRAEFT Authoring Platform	96
3.1 Representation of knowledge in the MOP/CAP	97



3.2 User Interface	97
3.3 Progress in Craeft	101
3.4 Multilingualism	101
3.4.1 Objectives.....	101
3.4.2 Methodology.....	102
3.4.3 Implementation notes	103
3.5 Cross-references	103
3.5.1 Objectives.....	104
3.5.2 Methodology.....	104
3.6 Zenodo storage	107
3.6.1 Technical advantages	108
3.6.2 Ecological advantages	109
3.7 Other enhancements	109
3.7.1 Knowledge element to file.....	109
3.7.2 Export for Europeana ingestion	110
3.7.3 Online viewing of 3D models	111
3.7.4 Media object types	112
3.7.5 Mosaics	113
4 Conclusions	116
References	118

1 Introduction

This document provides a detailed account of the Maker-Material-Negotiation model, termed “CRAEFT Ontology”, and of the CRAEFT Authoring Platform. It is divided into two main parts, each devoted to one of these two topics.

The first part, after some introductory information, recapitulates the narrative ontology and then presents an ontology of processes and actions, which is required to achieve the CRAEFT objectives. The latter part is the novel contribution of CRAEFT. It extends the Mingei Ontology by providing richer representations and is tightly coupled with the SIMULIA Abaqus software employed by the project to implement the mathematical models that compute the effects of actions. SIMULIA Abaqus is a suite of software applications for finite element analysis (FEA) and computer-aided engineering (CAE). Developed by Dassault Systèmes, Abaqus is widely used for engineering simulations, including stress analysis, heat transfer, and fluid dynamics. Abaqus is commonly used in aerospace, automotive, energy, and biomedical engineering industries to design, test, and optimise products and processes through virtual prototyping and simulation. The linking to standard vocabularies concludes this part.

The second part gives general information about the CRAEFT Authoring Platform, which at this stage is a minimal extension of the Mingei Online Platform that can manage the new concepts provided by the CRAEFT Ontology.

2 The CRAEFT Ontology

The CRAEFT Ontology (from now on, simply “CrO”) is used to represent the knowledge about crafts that the CRAEFT project is targeted to represent and preserve.

Following the approach successfully adopted in Mingei, the CrO views a craft as a narrative, composed by a fabula and a narration [6].

- The fabula of the craft is the *craft process*, that is, the complex activity situated in time and space that is carried out by a group of people to produce a craft. For simplicity, from now on we will use the term “process” with the meaning of “craft process”, unless otherwise specified.
- The narration of a craft is given by the documentation of the crafting process, consisting of texts, videos, images and data, collected during the execution of the craft, and related to the events of the fabula in the same way media objects are related to the events of a fabula.

At the same time, CrO tries to respond to the representational requirements that have emerged during the collaboration with craft experts striking a balance amongst several, often conflicting aspects: expressive power, usability, interoperability and efficiency. In particular,

- for expressive power, CrO is expressed in OWL 2 DL [1], currently the most expressive decidable language of the OWL family;
- for usability, CrO expresses a conceptualisation directly derived from the interaction with craft experts, to reflect the vision of the practitioners of the craft domain;
- for interoperability, CrO rests on several standards, the most prominent of which are: the very language in which it is expressed, OWL 2 DL, a standard of the W3C, and the top ontology CIDOC CRM [2][3], from a few decades an ISO standard with wide adoption in the Cultural Heritage domain [4]. Other standards on which CrO relies will be noted in due course; the axioms linking the CrO classes and properties to the CRM vocabulary are presented together with those defining them;
- for efficiency, CrO is implemented on top of a platform that allows fine-tuning of performances.

Technically, CrO is an OWL 2 DL application ontology (according to the terminology introduced in [5]) that rests on the top ontology CIDOC CRM as its conceptual backbone¹ and on several domain ontologies:

- The Narrative Ontology, a domain ontology focused on the representation of narratives [6];
- OWL Time, a domain ontology recommended by W3C for the representation of time [7];
- Dublin Core for simple resource description [8].

In addition, CrO includes extensions to the above ontologies needed to model specific aspects of reality relevant to CRAEFT, such as process schemas for which CrO relies on UML Activity Diagrams, a *de facto* standard [9]. CrO also uses the Semantic Web languages for modelling knowledge, in particular:

- the Resource Description Framework (RDF) for basic knowledge representation [10];

¹ CrO uses the OWL 2 DL expression of the CIDOC CRM, named “CRM Erlangen”, available from: <https://www.erlangen-crm.org/current-version>

- the RDF Schema Vocabulary for simple ontology modelling [11];
- OWL 2 DL for rich ontology modelling [1];
- XML Schema for datatypes [12].

Table I below provides the namespaces of these ontologies and the prefix we use in the CrO for each of these namespaces.

Table 1 The ontologies used in CrO, their prefixes and their namespaces

<i>Prefix</i>	<i>Ontology</i>	<i>Namespace</i>
craeft:	The CRAEFT Ontology	< https://craeft.eu/ontology/current/ >
narr:	The Narrative Ontology	< https://dlnarratives.eu/ontology# >
rdf:	Resource Description Framework (RDF) Vocabulary	< http://www.w3.org/1999/02/22-rdf-syntax-ns# >
rdfs:	RDF Schema Vocabulary	< http://www.w3.org/2000/01/rdf-schema# >
owl:	Web Ontology Language (OWL) Vocabulary	< http://www.w3.org/2002/07/owl# >
xsd:	XML Schema	< http://www.w3.org/2001/XMLSchema# >
ecrm:	The CIDOC CRM Ontology (Erlangen expression)	< http://erlangen-crm.org/current/ >
time:	OWL Time Ontology	< http://www.w3.org/2006/time# >
geo:	OGC GeoSPARQL	< http://www.opengis.net/ont/geosparql# >
crmgeo:	CRMgeo Spatiotemporal model	< https://www.cidoc-crm.org/crmgeo/sites/default/files/CRMgeo_v1_2.rdfs >

The rest of this Section is structured as follows:

- Section 2.1 introduces some notational conventions followed in the presentation of the ontology.
- Section 2.2 presents the identification of resources in the CrO.
- Section 2.3 briefly recapitulates the narrative ontology.
- Section 2.4 presents a conceptualisation of the notions central to the CRAEFT project, namely processes and actions.
- Section 2.5 axiomatizes the conceptualisation given in the previous Section.
- Section 2.6 extends the CrO with links to standard vocabularies.

2.1 Notational conventions

Any symbol belonging to the OWL 2 DL language, whether logical or non-logical, is written in **this font**. International Resource Identifiers (IRIs, from now on) is written in the standard prefixed notation, in which the absence of a prefix indicates that the **craeft:** prefix is used.

The non-prefix parts of class names are singular nouns of the English language reflecting the class semantics and are always capitalised. For instance, the class of processes is **craeft:Process**. Similarly, the non-prefix part of property names are singular expressions of the English language reflecting the property semantics and are written in the so-called Camel notation: the first letter is not capitalised while any other English word in the name is capitalised. For instance, the property connecting a process to its start event is **craeft:hasProcessStart**, or simply **hasProcessStart**.

The conceptualization of the various aspects of the ontology is presented in the text. The ensuing axioms are stated using the following notation:

Every axiom is given in a box like this and is written in a terse natural language very close to the logical expression of the axiom in the OWL 2 DL language.

For perspicuity, a subset of the axioms is given graphically, according to the following notation, well-known in database design:



Expressions of the OWL 2 DL languages that are stated in the functional notation [1] according to the following notation:

OWL 2 DL axiom

2.2 Resource identification

CrO conforms to the best practices for semantic interoperability, in the context of the Linked Data paradigm that the CRAEFT project adopts for the data that it collects or creates to reach its objectives. A crucial issue in this respect is the identification of resources, which concerns the policy followed by CRAEFT for assigning IRIs to the resources it manages.

The policy is given by the following principles:

1. A new IRI from the CRAEFT namespace is minted for every resource referenced in the CRAEFT Knowledge Base. This IRI has the form <https://craeft-project.eu/resource/N>, where: <http://craeft-project.eu/> identifies the CRAEFT namespace, and N is a unique progressive number, identifying this resource in the CRAEFT namespace. In this way, each resource is assigned a unique

integer number N, regardless of the class to which the resource belongs, which gives rise to a unique IRI.

2. For popular resources that have one or more identifiers in other Linked Data datasets, such as Wikidata, VIAF, *etc.*, an axiom of the form

```
ObjectPropertyAssertion( craeft-IRI owl:sameAs other-IRI )
```

is asserted in the CRAFT knowledge base, where *craeft-IRI* is the IRI of the resource described in the previous point and *other-IRI* is any other popular IRI of the resource.

The rationale for these principles is to follow the recommendations on Linked Data², which are:

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
- Include links to other URIs. so that they can discover more things.

In particular, the first principle follows the first three recommendations, since a CRAFT identifier is an HTTP IRI the de-referencing of which is under the control of the CRAFT project. In this way, the project has the opportunity to deliver a CRAFT web page or an RDF graph for describing the resource to the human or digital users, respectively, who ask for any CRAFT IRI.

The second principle follows the fourth recommendation on Linked Data by linking the Mingei knowledge base to other graphs containing knowledge about the same resources.

2.3 The Narrative Ontology

A narrative is conceptualised as consisting of two main elements:

- The *narration* is a symbolic object giving how the author tells the story of the narrative; historically, a narration is expressed in natural language, whether written or oral, and is rendered as a text; in recent years, we have more and more digitally born narrations; and
- The *fabula* is the way the story of the narrative happened in reality and consists of the events making up the story of the narrative in chronological order.

Fragments of the narration may be linked to the events of the fabula, to illustrate those events and, in general, the story told by the narrative. These links show the third component of a narrative according to narratology [6], the *plot* of the narrative, that is how the narrator has chosen to arrange the real events in the narration to make the story worth to be narrated.

In both cases, we view a narration as an arrangement of symbols, which can be the digitisation of a natural language narration, or a born-digital narration. Fragments of a narration may be linked to the events of

² <https://www.w3.org/DesignIssues/LinkedData.html>

the fabula, to illustrate those events and, in general, the story told by the narrative. This way of conceptualising narratives stems from basic studies in narratology [6] and is used to develop the Narrative ontology.

In summary, the main classes and properties of the Narrative Ontology are depicted in Figure 1. A more detailed account of the main classes and properties is given in the remaining parts of this Section.

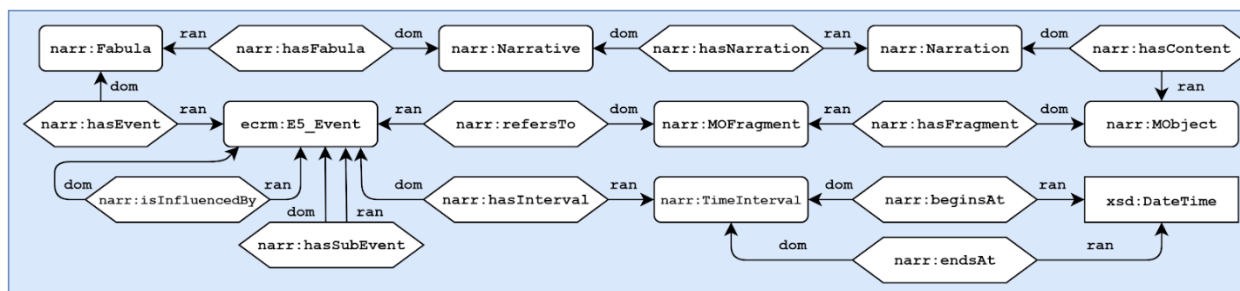


Figure 1 Main classes and properties of the Narrative Ontology.

2.3.1 Main Classes

A narrative is an instance of class **narr:Narrative**, a subclass of **ecrm:E73_Information_Object**, the CRM class that comprises³ “*identifiable immaterial items, such as poems, jokes, data sets, images, texts, multimedia objects, procedural prescriptions, computer program code, algorithm or mathematical formulae, that have an objectively recognizable structure and are documented as single units*”. As such, is **ecrm:E73_Information_Object** a natural fit for narratives.

narr:Narrative is a class

narr:Narrative is a subclass of **ecrm:E73_Information_Object**

A narration is the part of the narrative that narrates the narrative’s fabula using a natural, in the sense of non-formal, language. In CrO, a narration is an instance of class **narr:Narration**, a subclass of **ecrm:E89_Propositional_Object**, which “*comprises immaterial items, including but not limited to stories, plots, procedural prescriptions, algorithms, laws of physics or images that are, or represent in some sense, sets of propositions about real or imaginary things and that are documented as single units or serve as a topic of discourse*”.

narr:Narration is a class

³ All quotations about CRM classes or properties are from the CIDOC CRM v. 7.3 (May 2023) Official Specification, retrievable from: https://www.cidoc-crm.org/sites/default/files/cidoc_crm_version_7.2.3%5B4%20Sep%202023%5D.pdf

narr:Narration is a subclass of **ecrm:E89_Propositional_Object**

Narrations are composed of fragments which may be meaningful units, such as the chapters in a book or the scenes in a movie, or just arrangements of symbols that are factored out for some purpose, such as the trigrams occurring in a text or a sequence of photograms in a movie. Such fragments are instances of class **narr:NarrationFragment**, a subclass of **ecrm:E90_Symbolic_Object**. Class **narr:Narration** is a subclass of **narr:NarrationFragment**, since a whole narration is just a special case of a piece of narration. As **ecrm:E73_Information_Object** is a subclass of **ecrm:E90_Symbolic_Object**, an instance of **narr:Narrative** is also an instance of **ecrm:E90_Symbolic_Object**. Semantically, though, a narration and a narrative are disjoint objects, thus CRO declares disjointness of **narr:NarrationFragment** and **narr:Narrative**.

narr:NarrationFragment is a class
narr:NarrationFragment is a subclass of ecrm:E90_Symbolic_Object
narr:Narration is a class
narr:Narration is a subclass of narr:NarrationFragment
narr:NarrationFragment is a disjoint from narr:Narrative

Similarly to a narration, a presentation is an illustration of the fabula in a way that is somewhat alternative to the narration, which is the illustration of the fabula chosen by the author. For this reason, the class of presentations, **narr:Presentation**, and of their fragments, **narr:PresentationFragment**, are subclasses of **narr:Narration** and **narr:NarrationFragment**, respectively. As a consequence, they are both subclasses of **ecrm:E90_Symbolic_Object**, disjoint from **narr:Narrative**.

narr:PresentationFragment is a class
narr:PresentationFragment is a subclass of narr:NarrationFragment
narr:Presentation is a class
narr:Presentation is a subclass of narr:PresentationFragment
narr:Presentation is a subclass of narr:Narration

Graphically (the “D” diamond stands for disjointness of the connected classes):

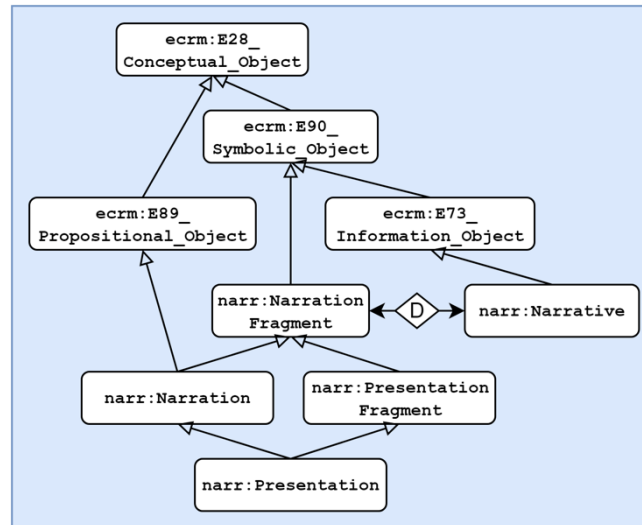


Figure 1 Presentations and narrations.

The actual contents of narration fragments (hence those of narrations, presentations and presentation fragments) are instances of class **narr:MObject**, a subclass of **RealObject** (to be introduced later) and of **ecrm:E90_Symbolic_Object**, having as instances multimedia objects. Media objects are disjoint from both narrations and narratives. Types of media objects are instances of class **narr:MOType**, a subclass of **ecrm:E55_Type**.

narr:MObject is a class
narr:MObject is a subclass of RealObject
narr:MObject is a subclass of ecrm:E90_Symbolic_Object
narr:MObject is a disjoint from narr:Narrative
narr:MObject is a disjoint from narr:Narration
narr:MOType is a class
narr:MOType is a subclass of ecrm:E55_Type

A fabula consists of the events narrated by a narrative, in chronological order. As such it is an instance of class **narr:Fabula**, a subclass of class **ecrm:E4_Period**, which “*comprises sets of coherent phenomena or cultural manifestations occurring in time and space*”. As **ecrm:E4_Period** is disjoint from **ecrm:E77_Persistent_Item** and **ecrm:E77_Persistent_Item** is a super-class of **ecrm:E90_Symbolic_Object**, **narr:Fabula** turns out to be disjoint from all classes introduced so far.

narr:Fabula is a class

narr:Fabula is a subclass of **ecrm:E4_Period**

To represent the events of a fabula, the CRO re-uses class **ecrm:E5_Event**, which “*comprises distinct, delimited and coherent processes and interactions of a material nature, in cultural, social or physical systems, involving and affecting instances of E77 Persistent Item in a way characteristic of the kind of process. Typical examples are meetings, births, deaths, actions of decision taking, making or inventing things, but also more complex and extended ones such as conferences, elections, the building of a castle, or battles*”. Events also include actions, which are viewed as events with an intention.

2.3.2 Main Properties

The connection between a narrative and its fabula is captured by property **narr:hasFabula**. The relation captured by this property is a generic semantic relation, in that it connects an information object with a set of phenomena described by that object. Therefore it is a sub-property of **ecrm:E89_P129_is_about**, which “*describes the primary subject or subjects of an instance of E89 Propositional Object*”. We note the **narr:Narrative** is a subclass of **ecrm:E73_Information_Objects**, therefore every narrative is also an instance of **ecrm:E89_Propositional_Object**. For convenience, also the inverse property **narr:isFabulaOf** is introduced. Narratives and fabulae are one-to-one.

narr:hasFabula is an object property

narr:hasFabula is a subproperty of **ecrm:P129_is_about**

The domain of **narr:hasFabula** is class **narr:Narrative**

The range of **narr:hasFabula** is class **narr:Fabula**

narr:isFabulaOf is an object property

narr:isFabulaOf is the inverse property of **narr:hasFabula**

An instance of class **narr:Narrative** is connected by property **narr:hasFabula** to exactly one instance of class **narr:Fabula**

An instance of class **narr:Fabula** is connected by property **narr:isFabulaOf** to exactly one instance of class **narr:Narrative**

The connection between a narrative and its narration is captured by property **narr:hasNarration**. This relation is an inheritance relation, linking a whole to one of its components, therefore it is a sub-property

of **ecrm:P148_has_component**, which “associates an instance of E89 Propositional Object with a structural part of it that is by itself an instance of E89 Propositional Object”. We note that both narratives and narrations are instances of **ecrm:E73_Information_Object**, hence of **ecrm:E89_Propositional_Object**. For convenience, also the inverse property **narr:isNarrationOf** is introduced. Narratives and narrations are one-to-one.

narr:hasNarration is a property
narr:hasNarration is a subproperty of ecrm:P148_has_component
The domain of narr:hasNarration is class narr:Narrative
The range of narr:hasNarration is class narr:Narration
narr:isNarrationOf is an object property
narr:isNarrationOf is the inverse property of narr:hasNarration
An instance of class narr:Narrative is connected by property narr:hasNarration to exactly one instance of class narr:Narration
An instance of class narr:Narration is connected by property narr:isNarrationOf to exactly one instance of class narr:Narrative

The connection between a fabula and anyone of its events is captured by property **narr:hasEvent**. The CRM offers property **ecrm:P9_consists_of**, which is the composition property for temporal entities, as it “associates an instance of E4 Period with another instance of E4 Period that is defined by a subset of the phenomena that define the former”. Thus **narr:hasEvent** is a subproperty of **ecrm:P9_consists_of**. We note that both fabulae and events are instances of **ecrm:E4_Period**.

narr:hasEvent is a property
narr:hasEvent is a subproperty of ecrm:P9_consists_of
The domain of narr:hasEvent is class narr:Fabula
The range of narr:hasEvent is class ecrm:E5_Event

The connection between a narrative and a presentation is captured by the property **narr:hasPresentation**, a sub-property of **narr:hasNarration**, since presentations are special kinds of

narrations. We consider this connection simply a composition relation thus we make **narr:hasPresentation** a sub-property of **ecrm:P148_has_component**, coherently with the fact that both narrations and presentations are instances of **ecrm:E89_Propositional_Object**.

narr:hasPresentation is a property
narr:hasPresentation is a subproperty of narr:hasNarration
The domain of narr:hasPresentation is class narr:Narrative
The range of narr:hasPresentation is class narr:Presentation

The connection between a narration fragment and the event it describes is captured by property **narr:refersTo**. In the CRM, this role is played by property **ecrm:P129_is_about**, which “documents that an E89 Propositional Object has as subject an instance of E1 CRM Entity”. Therefore, **narr:refersTo** is a sub-property of **ecrm:P129_is_about**.

narr:refersTo is a property
narr:refersTo is a subproperty of ecrm:P129_is_about
The domain of narr:refersTo is class narr:NarrationFragment
The range of narr:refersTo is class ecrm:E5_Event

The connection between a narration and any of its fragments is captured by property **narr:hasFragment**. The CRM offers property **ecrm:P106_is_composed_of** which “associates an instance of E90 Symbolic Object with a part of it that is by itself an instance of E90 Symbolic Object, such as fragments of texts or clippings from an image”. Therefore, **narr:hasFragment** is a sub-property of **ecrm:P106_is_composed_of**.

narr:hasFragment is a property
narr:hasFragment is a subproperty of ecrm:P106_is_composed_of
The domain of narr:hasFragment is class narr:Narration
The range of narr:hasFragment is class narr:NarrationFragment

The connection between a media object and its type is captured by property **narr:hasMObjectType**, a subproperty of the CRM property **ecrm:P2_has_type**.

	narr:hasMObjectType is a property
	narr:hasMObjectType is a subproperty of ecrm:P2_has_type
	The domain of narr:hasMObjectType is class narr:MObject
	The range of narr:hasMObjectType is class narr:MObjectType

The connection between a narration and the media object giving its content is captured by property **narr:hasContent**. The CRM offers property **ecrm:P129i_is_subject_of**, which “documents that an instance of E89 Propositional Object has as subject an instance of E1 CRM Entity”. For these reasons, **narr:hasContent** is a sub-property of **ecrm:P129i_is_subject_of**.

	narr:hasContent is a property
	narr:hasContent is a subproperty of ecrm:P129i_is_subject_of
	The domain of narr:hasContent is class narr:NarrationFragment
	The range of narr:hasContent is class narr:MObject

The connection between an event and any composing sub-event of it is captured by property **narr:hasSubevent**. In the CRM, the same role is played by property **ecrm:P9_consists_of**, which “describes the decomposition of an instance of E4 Period into discrete, subsidiary periods. The sub-periods into which the period is decomposed form a logical whole - although the entire picture may not be completely known - and the sub-periods are constitutive of the general period”. Therefore, the **narr:hasSubevent** property is a sub-property of **ecrm:P9_consists_of**.

	narr:hasSubevent is a property
	narr:hasSubevent is a subproperty of ecrm:P9_consists_of
	The domain of narr:hasSubevent is class ecrm:E5_Event
	The range of narr:hasSubevent is class ecrm:E5_Event

The connection between an event and any other entity on which the event causally depends is captured by properties `narr:causallyDependsOn`, whose inverse is defined for convenience and is given by `narr:isInfluencedBy`. The CRM does not address the causal dependency between proper events, but offers a property that subsumes causality, namely `ecrm:P15_was_influenced_by`, which “*captures the relationship between an instance of E7 Activity and anything, that is, an instance of E1 CRM Entity that may have had some bearing upon it*”. As it turns out, `ecrm:P15_was_influenced_by` is fairly generic having as domain class `ecrm:E7_Activity` and as range the most generic CRM class `ecrm:E1_CRM_Entity`. Therefore we assert `narr:causallyDependsOn` as a sub-property of `ecrm:P15_was_influenced_by`.

	<code>narr:causallyDependsOn</code> is a property
	<code>narr:causallyDependsOn</code> is a subproperty of <code>ecrm:P15_was_influenced_by</code>
	The domain of <code>narr:causallyDependsOn</code> is class <code>ecrm:E5_Event</code>
	The range of <code>narr:causallyDependsOn</code> is class <code>ecrm:E1_CRM_Entity</code>
	<code>narr:isInfluencedBy</code> is a property
	<code>narr:isInfluencedBy</code> is the inverse property of <code>narr:causallyDependsOn</code>

For the connection between an event and its spatial region of occurrence, we use the CRM property `ecrm:P7_took_place_at`, which “*describes the spatial location of an instance of E4 Period*”. Similarly, for the connection between an event and its temporal region of occurrence, we use the CRM property `ecrm:P4_has_time_span`, which “*associates an instance of E2 Temporal Entity with the instance of E52 Time-Span during which it was on-going*”. Finally, for the connection between an event and the agent(s) that performed it, we use the CRM property `ecrm:P14_carried_out_by`, which “*describes the active participation of an instance of E39 Actor in an instance of E7 Activity*”.

2.3.3 Representing actor roles in events

In CRAEFT there was the need to specify a role for an actor in an event (e.g., Pliny the Elder is an *observer* of the Vesuvius eruption). In natural language, the relation between actor, role and event is a triadic relation, since it involves three individuals. On the other hand, ternary properties are not allowed in Semantic Web languages. The CRM solves this problem by allowing properties of properties (e.g., P14.1, see below). But again, this solution cannot be adopted in CRAEFT since CRAEFT is committed to Semantic Web languages. To overcome this problem, CrO provides class `narr:ActorWithRole` and three properties to properly connect its instances to events, actors and roles. The three properties are:

- property `narr:hadParticipant` links events to instances of `narr:ActorWithRole`; this is a subproperty of the CRM property `ecrm:P12_occurred_in_the_presence_of`, which “*describes the active or passive presence of an E77 Persistent Item in an instance of E5 Event without implying any specific role*”;

- property **narr:hasSubject** links instances of **narr:ActorWithRole** to instances of class **ecrm:E39_Actor**, giving the person or the person group that participates in the event; this is a subproperty of the CRM property **ecrm:P148_has_component**, which is used to link propositional objects to their components;
- property **narr:hasRole** links instances of **narr:ActorWithRole** to a type giving the role of the actor; also this is a subproperty of the CRM property **ecrm:P148_has_component**.

	narr:ActorWithRole is a class
	narr:hadParticipant is an object property
	narr:hadParticipant is a subproperty of ecrm:P12_occurred_in_the_presence_of
	The domain of narr:hadParticipant is class ecrm:E5_Event
	The range of narr:hadParticipant is class narr:ActorWithRole
	narr:hasSubject is an object property
	narr:hasSubject is a subproperty of ecrm:P148_has_component
	The domain of narr:hasSubject is class narr:ActorWithRole
	The range of narr:hasSubject is class ecrm:E39_Actor
	narr:hasRole is an object property
	narr:hasRole is a subproperty of ecrm:P148_has_component
	The domain of narr:hasRole is class narr:ActorWithRole
	The range of narr:hasRole is class narr:Role

2.3.4 Modelling presentation fragments

A presentation fragment is a meaningful part of a presentation, an instance of class **narr:PresentationFragment**, introduced above. Presentation segments are characterised by a few data properties, namely:

- Starting and ending points, which locate a segment of a presentation into the virtual space of its playing. As such, they are not related to effective temporal entities, which are involved only when the presentation is executed on a specific device;
- Duration;
- Order;
- Channel, giving the output channel in which a segment belongs.

The corresponding data properties are: **narr:startsAtPoint**, **narr:endsAtPoint**, **narr:hasPresentationDuration**, **narr:hasPresentationSegOrder** and **narr:refersToChannel**. All these properties are subproperties of **ecrm:P3_has_note**, which “is a container for all informal descriptions about an object that has not been expressed in terms of CIDOC CRM constructs”.

2.3.5 Modelling Time

The main classes for the representation of time are **narr:TimePoint** and **narr:TimeInterval**, representing points and intervals of time, respectively. **narr:TimePoint** is equivalent to the OWL Time class **owltime:Instant**, while **narr:TimeInterval** is equivalent to the OWL Time class **owltime:ProperInterval** and the CRM class **ecrm:E52_TimeSpan**.

	narr:TimePoint is a class
	narr:TimePoint is equivalent to owltime:Instant
	narr:TimeInterval is a class
	narr:TimeInterval is equivalent to owltime:ProperInterval
	narr:TimeInterval is equivalent to ecrm:E52_Time_Span

As already pointed out, time intervals are connected to events by property **ecrm:P4_has_Time_SpanP4**, which associates an event with its interval of occurrence in time. In turn, a time interval is associated with its starting and ending time points by properties **narr:beginsAt** and **narr:endsAt**, respectively. These properties are equivalent to the properties **owltime:hasBeginning** and **owltime:hasEnd**, respectively, while the CRM does not provide any analogous property to map to.

	narr:beginsAt is an object property
	narr:beginsAt is an equivalent property to owltime:hasBeginning
	The domain of narr:beginsAt is class narr:TimeInterval

	The range of narr:beginsAt is class narr:TimePoint
	narr:endsAt is an object property
	narr:endsAt is an equivalent property to owltime:hasEnd
	The domain of narr:endsAt is class narr:TimeInterval
	The range of narr:endsAt is class narr:TimePoint

In addition to the above properties, the ontology offers properties for associating actual data values with time points. To this end, we adopt from the OWL Time ontology⁴ the following properties, all having time instants as a domain. The following properties provide alternative ways to describe the temporal position of an instant:

- **owltime:inXSDDate**, ranging on **xsd:date**
- **owltime:inXSDDateTimeStamp**, ranging on **xsd:dateTimeStamp**
- **owltime:inXSDgYear**, ranging on **xsd:gYear**
- **owltime:inXSDgYearMonth**, ranging on **xsd:gYearMonth**

The datatypes where these properties range also provide the ordering relations between time points (<, =, >). In addition, we used the temporal relation primitives based on fuzzy boundaries introduced in CRM 7.1.2 (official version) to compare two events. These temporal relation primitives are reported in the following table, where E^{start} and E^{end} are, respectively, the starting and the ending time point of event E, while “<” and “≤” are, respectively, the “less than” and the “less than or equal” relation between time points:

Table 2 Interval temporal properties from the CRM

⁴ Time Ontology in OWL. Candidate W3C Recommendations, 15 November 2022, <https://www.w3.org/TR/owl-time/>

	Property	Interpretation
1	P173 starts before or with the end of	$A^{\text{start}} \leq B^{\text{end}}$
2	P174 starts before the end of	$A^{\text{start}} < B^{\text{end}}$
3	P175 starts before or with the start of	$A^{\text{start}} \leq B^{\text{start}}$
4	P176 starts before the start of	$A^{\text{start}} < B^{\text{start}}$
5	P182 ends before or with the start of	$A^{\text{end}} \leq B^{\text{start}}$
6	P183 ends before the start of	$A^{\text{end}} < B^{\text{start}}$
7	P184 ends before or with the end of	$A^{\text{end}} \leq B^{\text{end}}$
8	P185 ends before the end of	$A^{\text{end}} < B^{\text{end}}$

2.3.6 Modelling Space

To represent spatiotemporal knowledge, the CrO does not define any class or property but entirely relies on two standards: the CRMgeo, an extension of the CRM for spatiotemporal knowledge, and GeoSPARQL⁵, a semantic-web aware standard for geographical information. Happily, the CRMgeo relates its classes and properties to the classes, topological relations and encodings provided by GeoSPARQL and thus allows spatiotemporal analysis offered by geoinformation systems based on the semantic distinctions of the CIDOC CRM. The CRMgeo is a formal ontology intended to be used as a global schema for integrating spatiotemporal properties of temporal entities and persistent items. It aims to provide a schema consistent with the CIDOC CRM to integrate geoinformation. CRMgeo uses the conceptualizations, formal definitions, encoding standards and topological relations defined by the Open Geospatial Consortium (OGC)⁶. The rest of this Section briefly introduces the classes and properties of the CRMgeo that are relevant to the CRAEFT project, and how these classes and properties are related to the corresponding classes and properties of GeoSPARQL.

All the classes declared in the CRMgeo were given both a name and an identifier constructed according to the conventions used in the CIDOC CRM model. For classes that identifier consists of the letter SP followed by a number. The resulting properties were also given a name and an identifier constructed according to the same conventions. That identifier consists of the letter Q followed by a number, which in turn is followed by the letter “i” every time the property is mentioned “backwards”, i.e., from target to domain.

The proposed geospatial representation complies with the OGC geoSPARQL standard, allowing geospatial queries on knowledge bases containing geographic data in Well-Known Text (WKT)⁷, a markup language for representing vector geometry objects, or Geography Markup Language (GML)⁸, an XML grammar defined by the OGC to express geographical features format.

In the CIDOC CRM, an event is an instance of class **ecrm:E5_Event** while a place is an instance of class

⁵ <https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html>

⁶ <https://www.ogc.org/>

⁷ <https://www.ogc.org/standard/wkt-crs/>

⁸ <https://www.ogc.org/standard/gml/>

ecrm:E53_Place. An event is associated with its place of occurrence through property **ecrm:P7_took_place_at**. In the CRMgeo, **ecrm:E53_Place** has as subclass **crmgeo:SP2_Phenomenal_Place**, which “comprises instances of E53 Place (S) whose extent (U) and position is defined by the spatial projection of the spatiotemporal extent of a real-world phenomenon that can be observed or measured. The spatial projection depends on the instance of SP3 Reference Space onto which the extent of the phenomenon is projected”. An instance of class **crmgeo:SP2_Phenomenal_Place** represents any place identified by an IRI in a standard gazetteer, such as Geonames for modern places, Pleiades for ancient places. **crmgeo:SP3_Reference_Space** “comprises the (typically Euclidian) Space (S) that is at rest (I) in relation to an instance of E18 Physical Thing and extends (U) infinitely beyond it. It is the space in which we typically expect things to stay in place if no particular natural or human distortion processes occur” (e.g., the space inside and around the Earth).

As Figure 2 shows, in CrO an instance of **ecrm:E5_Event** is directly linked to its (instance of) **crmgeo:SP2_Phenomenal_Place** through property **ecrm:P7_took_place_at**.

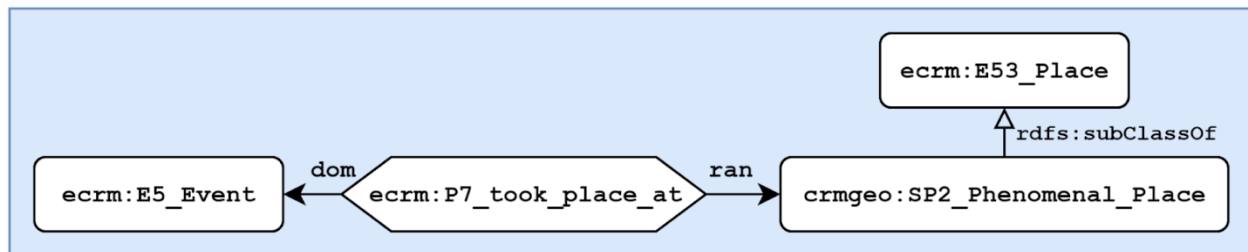


Figure 2 The classes of CRMgeo SP2 Phenomenal Place and its link with CIDOC CRM.

An instance of **crmgeo:SP2_Phenomenal_Place** is linked to an instance of class **crmgeo:SP5_Geometric_Place_Expression**, which “comprises definitions of places by quantitative expressions. An instance of SP5 Geometric Place Expression can be seen as a prescription of how to find the location meant by this expression in the real world, which is based on measuring where the quantities referred to in the expression lead to, beginning from the reference points of the respective reference system. A form of expression may be geometries or map elements defined in a SP4 Spatial Coordinate Reference System that unambiguously identify locations in a SP3 Reference Space”.

Since **crmgeo:SP2_Phenomenal_Place** and **crmgeo:SP5_Geometric_Place_Expression** are subclasses of GeoSPARQL classes **geosparql:Feature** and **geosparql:Geometry**, respectively, we can use the following properties also to directly link SP2 and SP5 (see Figure 3):

- **geosparql:hasDefaultGeometry**, which links an instance of class **geosparql:Feature** with its default instance of class **geosparql:Geometry**. The default geometry is the geometry that should be used for spatial calculations in the absence of a request for a specific geometry;
- **geosparql:hasGeometry**, which links an instance of class **geosparql:Feature** with an instance of class **geosparql:Geometry** that represents its spatial extent;

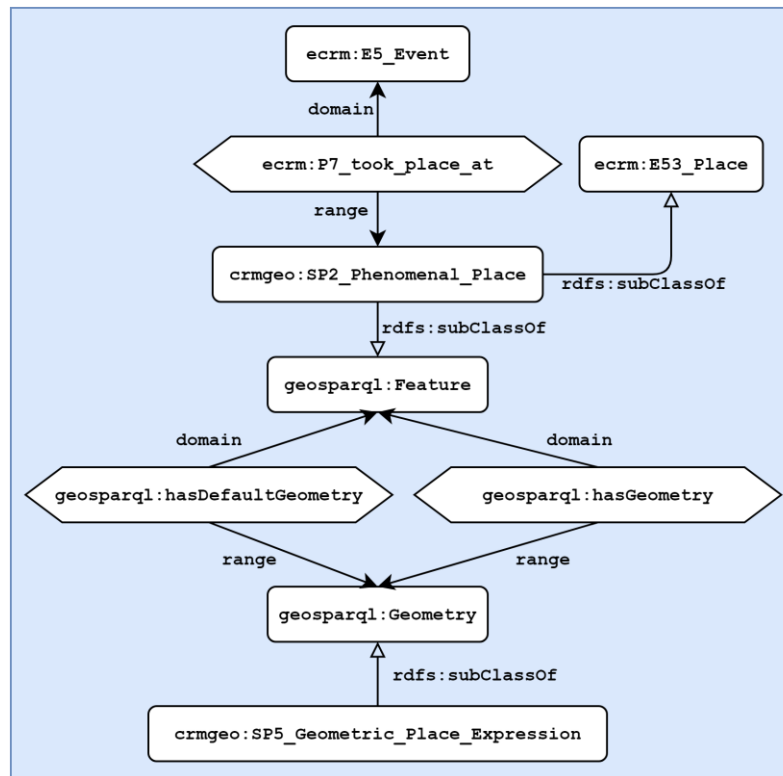


Figure 3 Relation between CRMgeo and Geo SPARQL classes.

An instance of `ecrm:SP2_Phenomenal_Place` is linked through the property `ecrm:P1_is_identified_by` to an instance of class `ecrm:E41_Appellation` that provides a name for the place in a natural language.

An instance of `ecrm:SP5_Geometric_Place_Expression` is linked through the property `ecrm:Q9_is_expressed_in_term_of` to an instance of class `ecrm:SP4_Spatial_Coordinate_Reference_System` that provides spatial coordinate reference system that is used by the geometry.

An instance of `ecrm:SP4_Spatial_Coordinate_Reference_System` is linked to an instance of `ecrm:SP3_Reference_Space` through property `ecrm:Q7_describes`.

An instance of `ecrm:SP5_Geometric_Place_Expression` is linked to its serialisation format through the property `geosparql:hasSerialization`, which has two subproperties corresponding to two different kinds of literal:

1. `geosparql:asWKT`, linking to a `wktLiteral`
2. `geosparql:asGML`, linking to a `gmlLiteral`

In the past, Latitude/Longitude coordinates in the WGS84 coordinate reference system have been commonly used to encode geospatial data. However, the GeoSPARQL standard has intentionally opted for a more adaptable approach by using various encoding formats that can accommodate different coordinate reference systems and geometry shapes. Thus, in CrO it is possible to represent the latitude

and longitude of a geographic point as a WKT literal, e.g., the longitude and latitude of the city of Pisa (IT) are represented as:

"<http://www.opengis.net/def/crs/OGC/1.3/CRS84> POINT (10.401 43.715)"^^geosparql:wktLiteral

The classes and properties reported above are shown in Figure 4.

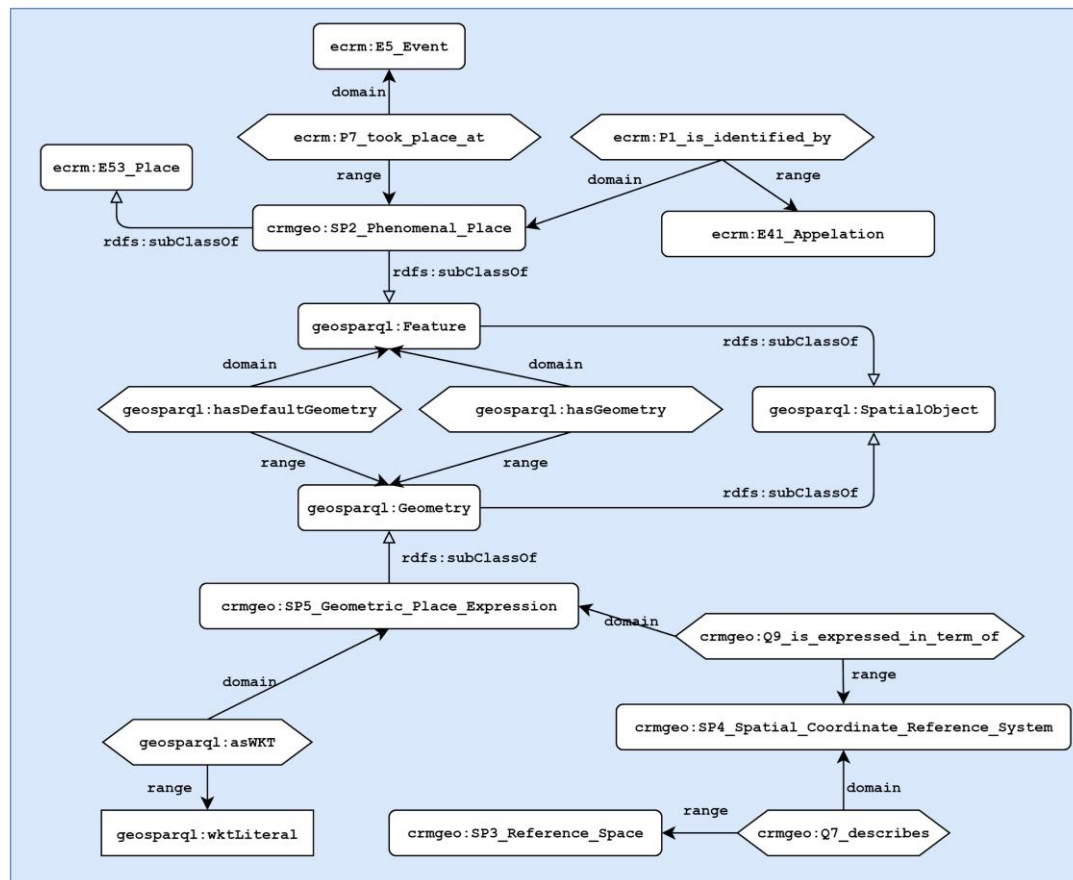


Figure 4 The classes and properties used to represent geospatial knowledge.

2.3.7 Modelling time-varying properties

The representations built with the CrO ontology are *diachronic*, in the sense that they do not refer to only one point in time; rather, these representations take into account the changes that may occur in the represented reality by modelling those changes via events. However, to be usable CrO does not require representing *every* change through one or more events.

This Section presents a method to model changes without using events. This method is well-known in knowledge representation and is called *4D-fluents*, based on the name given to time-changing properties by the fathers of AI. A more elaborate exposition of the method can be found in [14].

The basic assumption of the method is that every entity can be thought of as a four-dimensional~~four-dimension~~ space-time worm whose temporal parts are slices of the worm. For instance, the island of Chios

was ruled by the Republic of Genoa from 1363 to 1566 and by the Ottoman Empire from 1566 to 1912; in 1912 Chios became part of Greece. We can then think of Chios as being an object constituted by at least three slices:

- chios-1, the Chios dominated by the Republic of Genoa and existed from 1363 to 1566,
- chios-2, the Chios dominated by the Ottomans and existed from 1566 to 1912, and
- chios-3, the Chios included in the Greek Republic and existed from 1912 to date.

Figure 5 below shows how Chios and one of the above three slices (**chios-2**) can be modelled using the CrO. Green arrows are all labelled by the **rdf:type** property (**tin-2** is the interval of existence of **chios-2**, having **tp21** (i.e., the year 1566) and **tp22** (i.e., the year 1912) as the starting and end point, respectively).

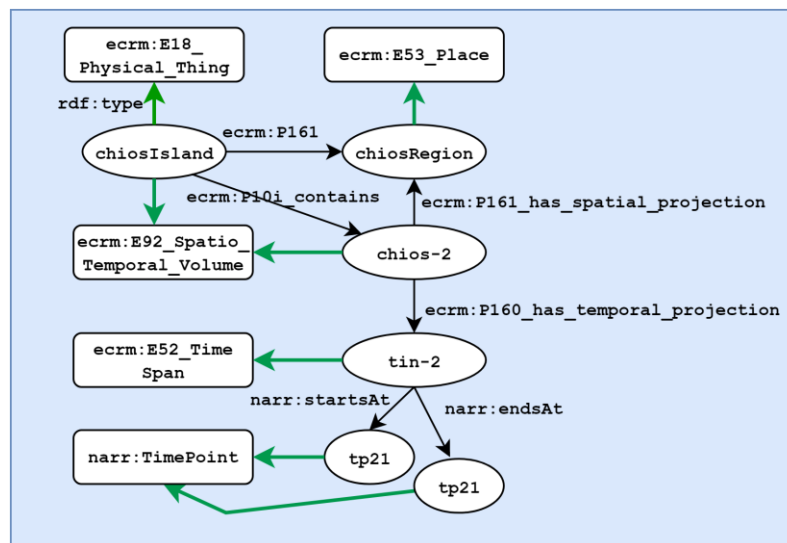


Figure 5 Chios and its time-varying properties.

The class **ecrm:E92_Spacetime_Volume** has temporal slices of things (the Chios Island in this case) as instances. These instances are related to the object representing Chios by property **ecrm:P10i_contains**, which is the inverse property of **ecrm:P10_falls_within**, while their spatial and temporal regions are linked via properties **ecrm:P161_has_spatial_projection** and **ecrm:P160_has_temporal_projection**, respectively.

2.4 Modelling processes

2.4.1 Processes

A craft process is composed of *steps*. A step can be either an *action* performed by one or more agents, or an event, that is something that happens without the direct intervention of the agents involved in the craft; for example, the growth of a plant or the provision of a piece of material for the craft are events. To support the modularisation of processes, we will also allow a step to be a whole process, which is to be

understood as a sub-process of the process where the step belongs. As customary, we will use two special steps, the *initial* step and the *final* step, to properly represent the beginning and the end of processes.

Following standard practice, also adopted by the Unified Modelling Language (UML, for short) activities, the steps in a process are interconnected by *transitions*. Consistently with UML, we will use the term “transition tail” (or simply “tail”) for any step where the transition hyperedge originates and “transition head” (or just “head”) for any step where the transition ends. Based on the experience gained in the Mingei project, CRAEFT processes will use transitions of the following kinds:

- Simple, to directly connect one tail to one head;
- Decision, to represent decision points in processes; a decision has one tail and two or more heads, each with an associated predicate;
- Merge, to represent the converging of several alternative paths in a single step; a merge has two or more tails and one head;
- Fork, to represent the division of a flow into two or more parallel flows; a fork has one tail and two or more heads;
- Join, to represent the converging of several parallel paths in a single step; a join has two or more tails and one head.

Consequently, a process is represented as a directed hypergraph, whose nodes represent steps and whose hyperedges represent transitions. Craft processes are deterministic, therefore the hypergraphs modelling them satisfy the following conditions:

1. any step other than the final one is in the tail of just one transition, and
2. any step other than the initial one is in the head of just one transition.

The following figures give the graphical form of process elements.

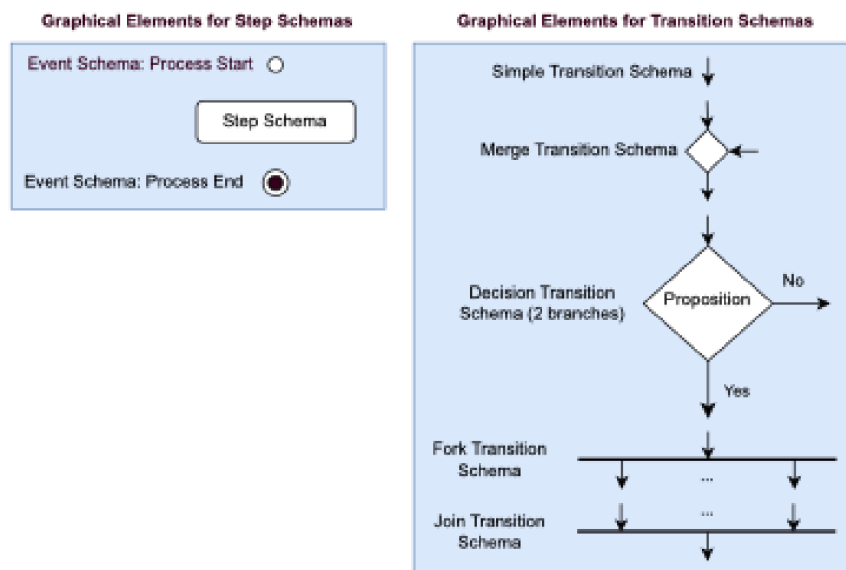


Figure 2 Graphical elements of processes

2.4.2 Actions

Generally, the CrO views an action as an application of forces to some objects, performed by one or more agents possibly with the aid of specific tools, resulting in transformations of the involved objects. Sometimes the result of an action is one or more objects that did not exist before and are produced by assembling one or more existing objects.

From an ontological point of view, actions are temporal entities called *perdurants* (for more details on perdurants, see for instance [15]) performed by agents endowed with *intentions*. Generally, the intention of the performer is a proposition giving the state of affairs that the performer aims to bring about by executing the action. Based on the intention of the performer, we can distinguish *correct* actions as those whose outcome reflects the intention of the performer, from the *incorrect* ones whose outcome does *not* reflect the performer's intention. An action can be incorrect, or fail, in many ways. For instance, the action of inserting a chisel into a piece of wood may fail if the applied force is not sufficient for the chisel to penetrate the wood, or if the applied force is excessive to break the chisel or the piece of wood. Unless otherwise specified, actions are considered to be of the correct kind. Certain incorrect actions stem from common mistakes of craft performers, thus the representation of those actions is as important as that of correct actions to properly train a craft performer.

An action can also be *uncertain*, if (and only if) the intention of the action performer does not identify a unique state of affairs. In more practical terms, the performer of an uncertain action has in mind several potential outcomes, all of which are acceptable to them. For instance, considering again our previous example, the performer may regard as acceptable any penetration of the chisel ranging from 2 to 5 millimetres, simply because the depth of the penetration depends on factors related to the particular materials employed or to the circumstances in which the action is executed. Uncertain actions are very relevant to the present purposes, as we observe that a great many, if not most actions involving physical materials and dexterity do involve a certain amount of uncertainty.

Objects involved in actions have a certain number of characteristics that distinguish them from one another. These characteristics can be usefully divided into two disjoint categories: time-independent and time-dependent characteristics. The former are the characteristics that do not depend on time, such as the identity of an object. In contrast, time-dependent characteristics may change over time, such as the temperature, the position or the shape of an object. Amongst the time-independent characteristics, a prominent role is played by the *composition* of an object, that is the relation of inherence that links an object to other objects, called *parts* or *components*. In addition, we regard composition as time-independent because an object is born composite and travels in time with the same components: if, at any time, some components of an object are added or removed, a new object is obtained. In contrast, the time-dependent characteristics of an object may vary without changing the identity of the object. Collectively, the time-dependent characteristics of an object form the object *state*.

One important requirement set by the CRAEFT project is that the KB containing knowledge about crafts must act as a *diachronic* representation of those crafts, that is, it must keep track of the craft process as it develops in time so that it is always possible to reconstruct the stages through which the process has evolved. In more practical terms, object states resulting from any actions *must not replace* the states of the same objects before that action; instead, old and new states must be *simultaneously* represented in the KB, each state linked with the information that permits to correctly determine the role played by the state in the context of a craft.

Based on these considerations, an action is modelled as a complex entity consisting of the following main parts:

- The agents that act, called the *performers*;
- the entities that cause the action, called *causing* entities, and the *duration* of the application of the causing entities;
- the objects involved in the action called the *affected* objects, whose states are changed as a result of the action;
- if any, the objects *created* by the action; these are the objects whose existence is a product of the action. Created objects may be composite, in this case, their components are amongst the affected objects, or they may be simple if the action just decomposes or breaks one affected object;
- the *action function*, that is the function that connects the input of the action to its output. In particular, an action function takes as input the causing entities and the initial states of the affected entities and produces as output the final states of the affected entities and the compositions and states of the created entities;
- the time when and the place where the action is performed.

As an example, the action of inserting a chisel into a piece of wood by hitting it with a hammer consists of:

- the person driving the hammer as a performer;
- the place and the time of the action;
- the force with which the hammer is driven (which implies the duration), as causing entities;
- the hammer, the chisel, and the piece of wood as affected objects; the action changes the position of all these objects and may change their shape;
- the piece of wood with the chisel inserted in it is the created object;
- the action function associates the force and the initial state of the involved entities (hammer, chisel and piece of wood) to the final states of these entities and the composition of the created object.

Causing entities are typically perdurants, as they develop in time, while affected objects are typically endurants which may be created or destroyed by an action, or which can change their states as the result of an action.

Notice that the above conceptualisation is simply the vision of action from a craft point of view, based on the principles of mechanics but ultimately tailored to the requirements set by the project on the representative craft instances. Thus,

- the causing entities in an action do not include all world objects that may in any way exert a causal role in the action, but only those entities whose causal role is relevant concerning the modelling of the craft at hand;
- the affected objects in an action do not include all world entities that may in any way be altered by the action, but only those entities whose participation in the action is relevant concerning the modelling of the craft at hand;
- the representation of an object in terms of composition and state is not expected to provide a complete model of the object but simply reflects the requirements of the project as to object representation.

As mentioned above, in the context of a craft process an event is something that happens without the direct intervention of the agents involved in the craft. Consequently, the representation of events is much simpler than that of actions, only consisting of the amount of time that an event takes to occur. Other features of events, specific to the kind of event occurring in a process, have an accidental nature and are not part of the general characterisation provided here.

For reasons that will be clarified later below, we need to consider another kind of action, which will be called *virtual actions*. A virtual action is a mathematical model, built analytically by considering the forces and the properties of the objects involved in the action to the end of computing the effects of the action in a theoretical way, like, *e.g.*, an engineer would do upon designing some artefact. To distinguish virtual actions from the actions we know from everyday experience, we will use the term *real* actions for the latter. The virtual actions that we will be considering in the present context will be simulations, performed by the SIMULIA Abaqus software. SIMULIA Abaqus is a suite of software applications for finite element analysis (FEA) and computer-aided engineering (CAE). Developed by Dassault Systèmes, Abaqus is widely used for engineering simulations, including stress analysis, heat transfer, and fluid dynamics. Abaqus is commonly used in aerospace, automotive, energy, and biomedical engineering industries to design, test, and optimise products and processes through virtual prototyping and simulation.

A virtual action is structured in the same way as a real action, but the involved entities are different. In particular,

- the action function of a virtual action is the function realised by the simulation, called the *simulation function*;
- the causing, the affected and, if any, the created entities of a virtual action are mathematical models which will be termed “virtual” to distinguish them from their real counterparts; in particular, virtual actions are caused by virtual entities and affect virtual objects. A virtual object is therefore a mathematical model of a real object, providing the information needed by the simulator to simulate the use of the object.

In the same way, actions can be simulated by virtual actions, events can be simulated by virtual events and transitions can be simulated by virtual transitions. A virtual process is a process solely consisting of virtual events, actions or sub-processes, connected by virtual transitions.

2.4.3 Schemas

In general, a schema of a phenomenon is a description that captures the salient aspects of that phenomenon for a specific class of applications. Any occurrence of that phenomenon is said to be an *instance* of the schema and provides specific values for the aspects encompassed in their corresponding schema. Schemas are created for *descriptive* and *prescriptive* reasons:

- from a descriptive point of view, schemas are representations that help understand phenomena in a general sense, before and independently from specific occurrences;
- from a prescriptive point of view, schemas help keep an information system under control, by providing templates to which instances must adhere.

In CRAEFT, schemas are introduced for both kinds of reasons. Descriptively, they provide general representations of crafts that highlight the main steps and their inter-relations, so that information consumers can familiarise themselves with crafts following a top-down approach; prescriptively, they allow managing the creation, storage and access of the representative craft instances, providing strong constraints on how these crafts are organised and structured.

Craft process schemas are no exceptions: they are formal representations that capture the salient aspects of processes for the CRAEFT project, as they have been identified above. Schemas are, introduced in CRAEFT for both kinds of reasons: descriptively, they provide general representations of crafts that highlight the main steps and their inter-relations, so that information consumers can familiarise themselves with crafts following a top-down approach; prescriptively, they allow managing the creation, storage and access of the representative craft instances, providing strong constraints on how these crafts are organised and structured.

A process schema is a directed hypergraph composed of step schemas and transition schemas. In particular,

- a step schema can be an action, an event or a (sub)process schema;
- a transition schema describes the conditions that control the flow of execution of instances of the process. There are different types of transition schema for each type of transition: simple transition schema, decision transition schema, merge transition schema, fork transition schema and join transition schema.

More specifically, an action schema describes the action function along with entities involved in the action, each with its role. Agents in schemas are represented by roles, which give the kinds of individuals acting. There are many possible languages for describing an action function, for instance as a set of mathematical equations, giving the evolution in space and time of the entities involved in the action. Given the nature of the actions dealt with by the CRAEFT project, the CRO uses two languages for describing actions: for real actions, it uses the natural language encoding the narrations of the craft practitioners, including not only the words of these persons but also their gestures as recorded in media objects; for virtual actions, it uses the language used by the SIMULIA Abaqus simulation tool, that is, the language of mathematics. It follows that agents of human actions are persons, while the agent of any virtual action is the SIMULIA Abaqus simulation tool.

Action schemas are instantiated by virtual actions, as the latter provides an abstract, mathematical execution of an action. Each virtual action encodes an application of the simulation code given by the schema, providing the input parameters identifying the application and recording the output parameters resulting from the application. Following the representation of the action given above, the input parameters of a virtual action are the (mathematical representations of the) causing entities and of the initial states of the affected objects; output parameters are the (mathematical representations of the) final states of the affected objects and the state of the created objects, if any. Instantiation is a one-to-many relation: an action schema can be instantiated by any number of virtual actions, each determined by a different set of input parameters. There cannot be two virtual actions that are instances of the same action schema and have the same input parameters but different output parameters. Conversely, each virtual action is an instance of exactly one schema.

Real actions are *enacted* by human agents that perform them in a real setting, that is, as real transformations to real objects, at a given place and at a given time. Enactment requires that the enacting

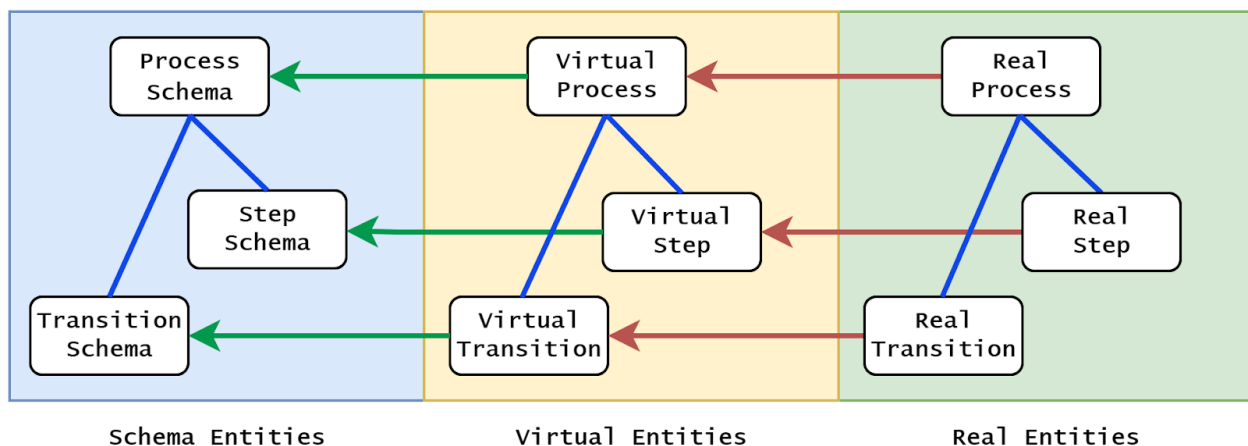
real action strictly mirrors the enacted virtual action, providing accurate realisations of the representations that make up the virtual action. Also, enactment is a one-to-many relation: A virtual action can be enacted any number of times, each performed by an agent and situated in its spatiotemporal region, different from the spatiotemporal region of all other actions. Conversely, each real action is an enactment of exactly one virtual action. Notice that, as there cannot be two virtual action instances of the same schema with the same input parameters, there cannot be two enactments of the same virtual actions within the same spatio-temporal region. Fortunately, this condition is enforced by nature.

The same instantiation/enactment structure applies to process schemas: by instantiating all action and transition schemas in a process schema, a *virtual process* is obtained, which can then be enacted by a real process. In sum,

- process schemas are composed of step and transition schemas and are instantiated into virtual processes, composed of virtual steps and virtual transitions, which are instances of the corresponding schemas;
- virtual processes are composed of virtual steps and virtual transitions and are enacted into processes, composed of steps and transitions, which are enactments of the corresponding virtual steps and virtual transitions, respectively.

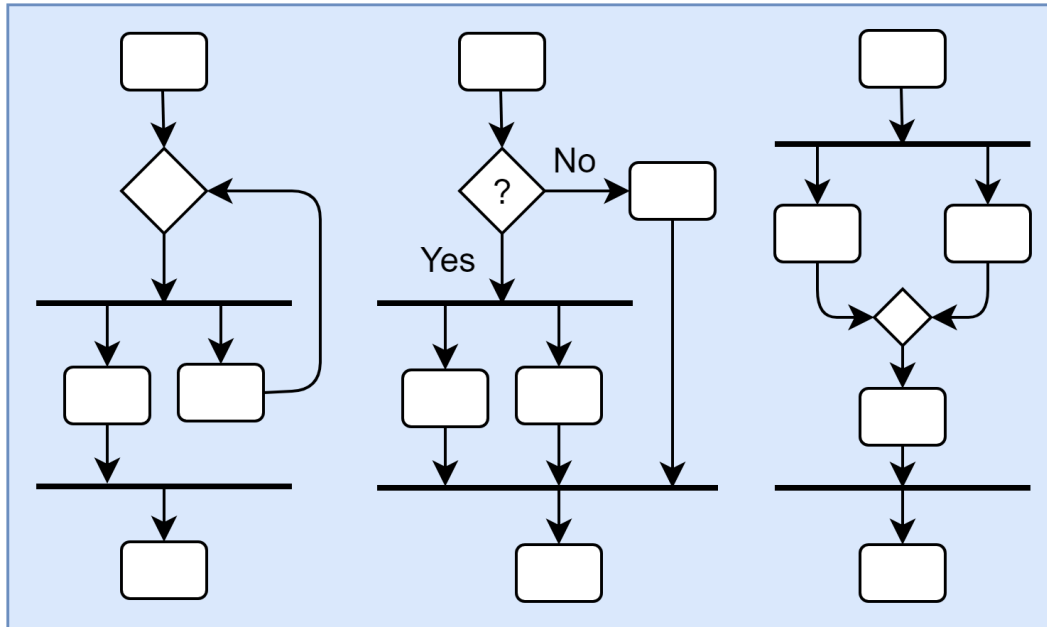
The following diagram illustrates graphically the notions introduced so far: the blue lines going down from the topmost entities represent decomposition; green arrows represent instantiation; and finally, red arrows represent enactment.

The following diagram illustrates the notions introduced so far.



2.4.3.1 Theoretical considerations on process schemas

The process model introduced thus far allows the representation of “odd” process schemas, that is schemas which hardly make any sense from an application point of view or that exhibit evident anomalies. These odd schemas stem from an improper usage of fork and join transitions, either alone or in combination with other kinds of transitions. A few examples (see Figure below) will suffice to illustrate some basic cases, from which more complex cases can be derived by composition.



The schema on the left exhibits a simple transition that crosses the boundary of a fork-join pair, giving raise to a virtual process that expands indefinitely. The schema in the centre shows a similar situation, with a simple transition that crosses the boundary of a fork leading directly to the matching join transition. This schema can be re-written into an equivalent one that does not present this anomaly, since the instantiation of the decision transition before the fork produces a simple transition that either leads to the fork (“yes” branch) or leads to a step outside the fork (“no” branch) and which does not need to be synchronised with anything. Finally, in the schema on the right two parallel flows merge before being synchronised, which violates the basic semantics of merge transitions which cannot combine parallel threads. This schema too can be rewritten to avoid the anomaly. By combining these basic cases, and more odd cases, a large subset of meaningless schemas can be obtained.

In principle, the subset of odd schemas should be mathematically characterised accurately, to make sure that only “good” schemas are allowed by the CRAEFT Authoring Platform. However, this is not necessary in the present context. The reason is that the schemas that the CRAEFT project develops and that are going to be stored on the CRAEFT Authoring Platform are all derived bottom-up from descriptions of the experts and documented by real processes, representing real craft instances. Thus, there is no possibility that this approach leads to the definition of odd schemas such as the ones shown above.

2.5 An ontology of processes and actions

This Section presents the part of the CRO ontology concerning processes and actions, axiomatising the conceptualisation given in the previous Section. The Section is organised as follows. First, generalised properties are introduced in Section Generalised properties, then the ontology proper is given, starting from activity schemas (Section Activity Schemas), which inform the other concepts and relations of the ontology, and then moving to activities (Sections from Activities to Enactment), and transitions (Section Transitions).

Generalised properties

From a methodological point of view and for reasons of clarity and economy, the ontology relies as much as possible on *Generalised Properties* (GPs, for short). A GP is an OWL 2 DL object property that represents a single domain relation that applies to several classes, having a different range for each such class. For example, a composition relation in the mechanical domain may apply to several kinds of machines, having a different component type for each type of machine it applies to. Some ontology languages, such as RDF Schema, are not powerful enough to allow a knowledge engineer to represent this fact, leaving no choice but to define one different property for each class of application. This causes a significant loss in the semantic quality of the ontology, as these different properties all stand for the same relation. Indeed, the proliferation of unnecessary properties increases the complexity of using and managing the ontology. The damage can be partially remedied by making these properties sub-properties of a special property that stands for the relation. However, such a remedy can only be applied if the domains of the sub-properties are subclasses of a single class and the same for the ranges. But this is not always the case, and the proliferation problem remains anyway. In short, GPs reduce the number of properties of the ontology to those that are strictly necessary to capture the relations underlying the conceptualisation.

GPs are axiomatised in OWL 2 DL according to a specific pattern, named the *GP pattern*, consisting of the following axioms. Let us assume that the GP applies to the classes C_1, C_2, \dots, C_n , and when applied to class C_i , the GP has class R_i as range. Then the axioms are as follows:

- One axiom declares the GP as an object property.
- One axiom declaring the inverse of GP as an object property.
- One axiom asserts that the domain of the GP is class C , where C is the most specialised class that generalises classes C_1, C_2, \dots , and C_n . In the worst case, C equals **owl:Thing**, which is to say that the domain of the GP is the whole universe; but in any other case, the definition of the domain by a specific class helps circumscribing the applicability of the GP, which is desirable.
- One axiom asserts that the range of the GP is class R , where R is the most specialised class that generalises classes R_1, R_2, \dots , and R_n . The considerations done above on the existence of C apply also to R .
- n axioms asserting that the range of the GP when applied to class C_i is class D_i and n axioms for the converse. These $2n$ axioms are usually called *localisation* axioms.

For convenience, these axioms will be given for each GP introduced in what follows.

2.5.1 Activity Schemas

ActivitySchema is the most general class having schemas as instances. Following the structure of hypergraphs, **ActivitySchema** is the disjoint union of two classes: **StepSchema** and **TransitionSchema**, the former having nodes of process schemas' hypergraphs as instances and the latter having hyperedges of process schemas' hypergraphs as instances. Each of these subclasses is further specialised based on the kinds of steps and transitions that can be found in a process hypergraph. Thus, **StepSchema** is the disjoint union of three classes:

- **EventSchema**, having schemas of events as instances;
- **ActionSchema**, having schemas of actions as instances;
- **ProcessSchema**, having schemas of processes as instances.

TransitionSchema is the disjoint union of five classes, corresponding to the different types of transitions included in the conceptualisation:

- **SimpleTransitionSchema**,
- **DecisionTransitionSchema**,
- **MergeTransitionSchema**,
- **ForkTransitionSchema** and
- **JoinTransitionSchema**.

The fact that **ProcessSchema** is a subclass of **StepSchema** allows us to include process schemas as steps of other process schemas, giving them the status of sub-processes. This applies to any process schema, even those which are not included in other process schemas; any process schema is therefore “ready” to become a step in another process schema.

Following the UML standard, every process is enclosed between a start and an end node. In the CrO vision of processes, these nodes are events, as they do not include any action. Accordingly, the CRO defines the disjoint classes **ProcessStartSchema** and **ProcessEndSchema** as subclasses of **EventSchema**, having as instances the descriptions of start and end nodes.

Schemas are kinds of *plans*, that is, descriptions of structured objects, whether these objects are steps or transitions. The CIDOC CRM offers the class **ecrm:E29_Design_or_Procedure** that “*comprises documented plans for the execution of actions to achieve a result of a specific quality, form or contents. In particular, it comprises plans for deliberate human activities that may result in new instances of E71 Human-Made Thing or for shaping or guiding the execution of an instance of E7 Activity*”. All the above schemas satisfy this description, therefore class **ActivitySchema** is a subclass of **ecrm:E29_Design_or_Procedure** and so are by consequence all schema classes introduced so far.

ActivitySchema is a class
ActivitySchema is a subclass of ecrm:E29_Design_or_Procedure
StepSchema is a class
EventSchema is a class
ProcessStartSchema is a class
ProcessStartSchema is a subclass of EventSchema
ProcessEndSchema is a class
ProcessEndSchema is a subclass of EventSchema

ProcessStartSchema is disjoint from ProcessEndSchema
ActionSchema is a class
ProcessSchema is a class
StepSchema is the disjoint union of EventSchema , ActionSchema and ProcessSchema
TransitionSchema is a class
SimpleTransitionSchema is a class
DecisionTransitionSchema is a class
MergeTransitionSchema is a class
ForkTransitionSchema is a class
JoinTransitionSchema is a class
TransitionSchema is the disjoint union of SimpleTransitionSchema , DecisionTransitionSchema , MergeTransitionSchema , ForkTransitionSchema , and JoinTransitionSchema
ActivitySchema is the disjoint union of StepSchema and TransitionSchema

To capture the composition relation of processes, the CrO introduces the property **hasProcessStart**, connecting a process schema to its start event step schema. Once the connection with the starting step is established, the structure of the process is fully determined by using transitions to individuate all the remaining steps of the process. For convenience, the inverse property of **hasProcessStart**, **isProcessStartOf**, is also introduced. For economy and conceptual homogeneity, **hasProcessStart** is also used for the composition of virtual and real processes, therefore **hasProcessStart** and **isProcessStartOf** are GPs. Since these properties apply to both schemas, instances of **ecrm:E29_Design_or_Procedure** and to fabulae, instances of **ecrm:E4_Activity**, their domain is the most specialised class that generalises both, that is, **ecrm:E1_CRM_Entity**. No mapping of **hasProcessStart** to a property of CRM is possible, as the CRM introduces, amongst others, two properties for part-whole relationships:

- **ecrm:P69_has_association_with**, for part-whole relationships between instances of **ecrm:E29_Design_or_Procedure**;
- **ecrm:P5_consists_of** for part-whole relationships between instances of **ecrm:E3_Condition_State**.

We should therefore make both **hasProcessStart** sub-property of a disjunction of two CRM properties. However, this is not supported by OWL 2 DL, which only provides object and data properties as property expressions. Despite this inconvenience, which applies also to the properties for the composition of transitions (see Section Transitions), the CrO prefers to use a single property.

hasProcessStart is an object property
The domain of hasProcessStart is class ecrm:E1_CRM_Entity
The range of hasProcessStart is class ecrm:E1_CRM_Entity
isProcessStartOf is an object property
isProcessStartOf is inverse property of hasProcessStart

The axioms given next complete the definition of the GPs **hasProcessStart** and **isProcessStartOf**, and provide the obvious cardinality relation: a process schema always connects to exactly one start process schema and vice-versa.

An instance of ProcessSchema is connected by hasProcessStart only to instances of ProcessStartSchema
An instance of ProcessSchema is connected by hasProcessStart to exactly one instance of ProcessStartSchema
An instance of ProcessStartSchema is connected by isProcessStartOf only to instances of ProcessSchema
An instance of ProcessStartSchema is connected by isProcessStartOf to exactly one instance of ProcessSchema

For supporting the automation of instantiation, the ontology defines a property **hasProcessEndSchema**, associating a process schema with its process-end event schema. The values of the property can be computed by visiting a process schema, however, the computation cannot be performed axiomatically, thus the property needs to be explicitly defined to hold the values. The property is one-to-one and has no corresponding property in the CRM. The property is solely used for process schemas, thus it is not a GP.

hasProcessEndSchema is an object property
The domain of hasProcessEndSchema is class ProcessSchema
The range of hasProcessEndSchema is class ProcessEndSchema

An instance of **ProcessSchema** is connected by **hasProcessEndSchema** to exactly one instance of **ProcessEndSchema**

An instance of **ProcessEndSchema** is connected by the inverse of **hasProcessEndSchema** to exactly one instance of **ProcessSchema**

One important property of activities is their planned duration, which fixes the amount of time that must be used to execute them. For this purpose, the ontology defines property **duration** and sets its domain as **ActivitySchema**, so that the appropriate value can be to step or transition schemas. **duration** is a sub-property of the CRM property **ecrm:P43_has_dimension**, whose domain is class **ecrm:E70_Thing**, a superclass of **ecrm:E29_Design_or_Procedure** and therefore of all schema classes, and whose range is the CRM class **ecrm:E54_Dimension**, also the range of **duration**.

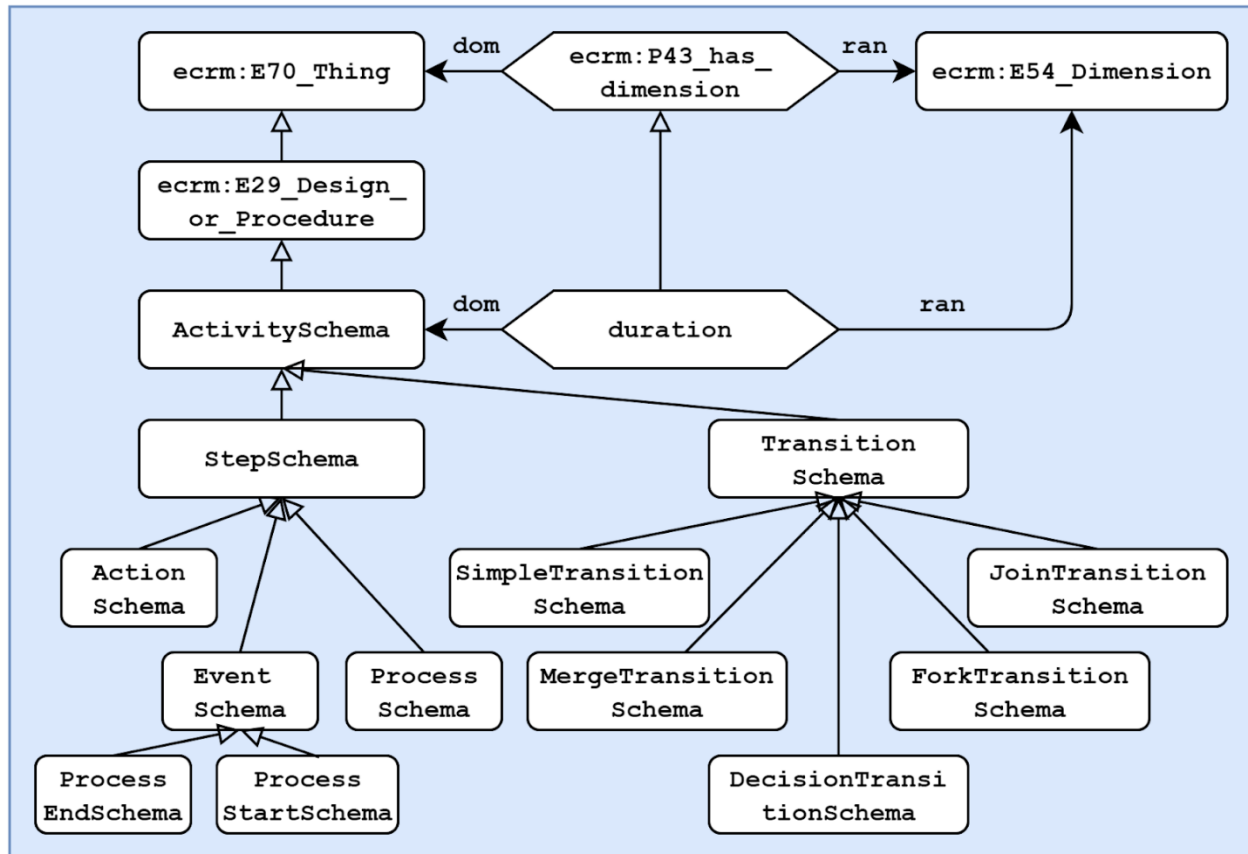
duration is an object property

duration is a sub-property of **ecrm:P43_has_dimension**

The domain of **duration** is class **ActivitySchema**

The range of **duration** is class **ecrm:E54_Dimension**

Graphically:



2.5.1.1 Activities

From a narratological point of view, processes are *fabulae*, since they are complex activities that form meaningful wholes, worth to be narrated because they lead to the production of relevant items. Craft processes, in particular, are very worth to be narrated because they embody important aspects of human nature and of the way humans relate to matter: the narration of craft processes is the very reason for the existence of the CRAFT project. To axiomatise processes, we need to distinguish between the *fabulae* of virtual processes and those of real processes, and consequently between the narratives about virtual processes and those of real processes. We will start the axiomatisation of processes from this distinction and introduce the class of **CraftNarrative**, having as instances the narratives of craft processes, and two disjoint subclasses of it: **VirtualNarrative** and **RealNarrative**, the former for virtual processes and the latter for real ones. Similarly, we introduce the class **CraftFabula**, having as instances all craft processes, and two disjoint subclasses of its: **VirtualFabula** and **RealFabula**, the former for virtual processes and the latter for real ones.

CraftNarrative is a class

CraftNarrative is a subclass of **narr:Narrative**

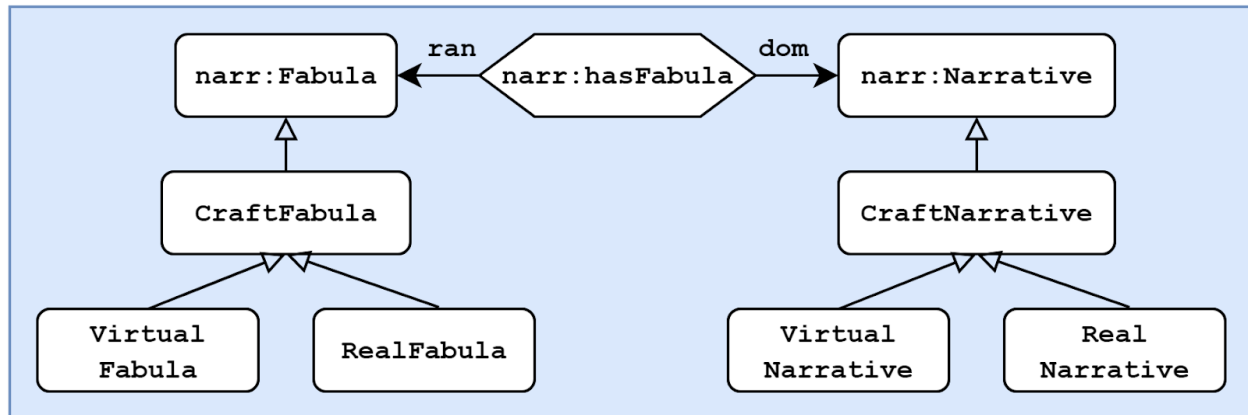
VirtualNarrative is a class

VirtualNarrative is a subclass of CraftNarrative
RealNarrative is a class
RealNarrative is a subclass of CraftNarrative
RealNarrative is disjoint from VirtualNarrative
CraftFabula is a class
CraftFabula is a subclass of narr:Fabula
VirtualFabula is a class
VirtualFabula is a subclass of CraftFabula
RealFabula is a class
RealFabula is a subclass of CraftFabula
RealFabula is disjoint from VirtualFabula

It follows that property **narr:hasFabula**, connecting an instance of class **narr:Narrative** to its instance of class **narr:Fabula**, becomes a GP, whose localisation axioms are given next. These axioms are not given for classes **CraftNarrative** and **CraftFabula**, which are abstract and are not meant to be directly instantiated but are defined solely to circumscribe their subclasses to the craft domain.

An instance of VirtualNarrative is connected by narr:hasFabula only to instances of VirtualFabula
An instance of VirtualFabula is connected by narr:isFabulaOf only to instances of class VirtualNarrative
An instance of RealNarrative is connected by narr:hasFabula only to instances of RealFabula
An instance of RealFabula is connected by narr:isFabulaOf only to instances of class RealNarrative

Graphically:



Activity is the most general activity class in crafts. For the reasons just given, **Activity** is a subclass of **CraftFabula**. **Activity** is specialised in two main classes: **VirtualActivity**, including all activities in virtual processes, and **RealActivity**, including all activities in real processes. Notice **VirtualActivity** is a subclass **VirtualFabula**, while **RealActivity** is a subclass **RealFabula**; disjointness of real from virtual fabulae implies that between real and virtual activities.

Following the structure already followed for schemas, **Activity** is the disjoint union of **Step**, the class of steps, and **Transition**, the class of transitions. Each of these is further specialised into the class of the corresponding virtual and real entities, so **Step** is specialised into **VirtualStep** and **RealStep**, while **Transition** is specialised into **VirtualTransition** and **RealTransition**. These pairs of classes are disjoint as a consequence of the disjointness of **VirtualActivity** and **RealActivity**. Each of these four classes is further refined following the structure seen in the previous Section. For instance, **RealStep** is the disjoint union of three classes: **RealAction**, **RealEvent** and **RealProcess**. We recall that decision and merge transition schemas are instantiated as simple virtual transitions, therefore there are neither virtual nor real merge or decision transitions.

Activity is a class
Activity is a subclass of CraftFabula
VirtualActivity is a class
VirtualActivity is a subclass of Activity
VirtualActivity is a subclass of VirtualFabula
RealActivity is a class
RealActivity is a subclass of Activity
RealActivity is a subclass of RealFabula

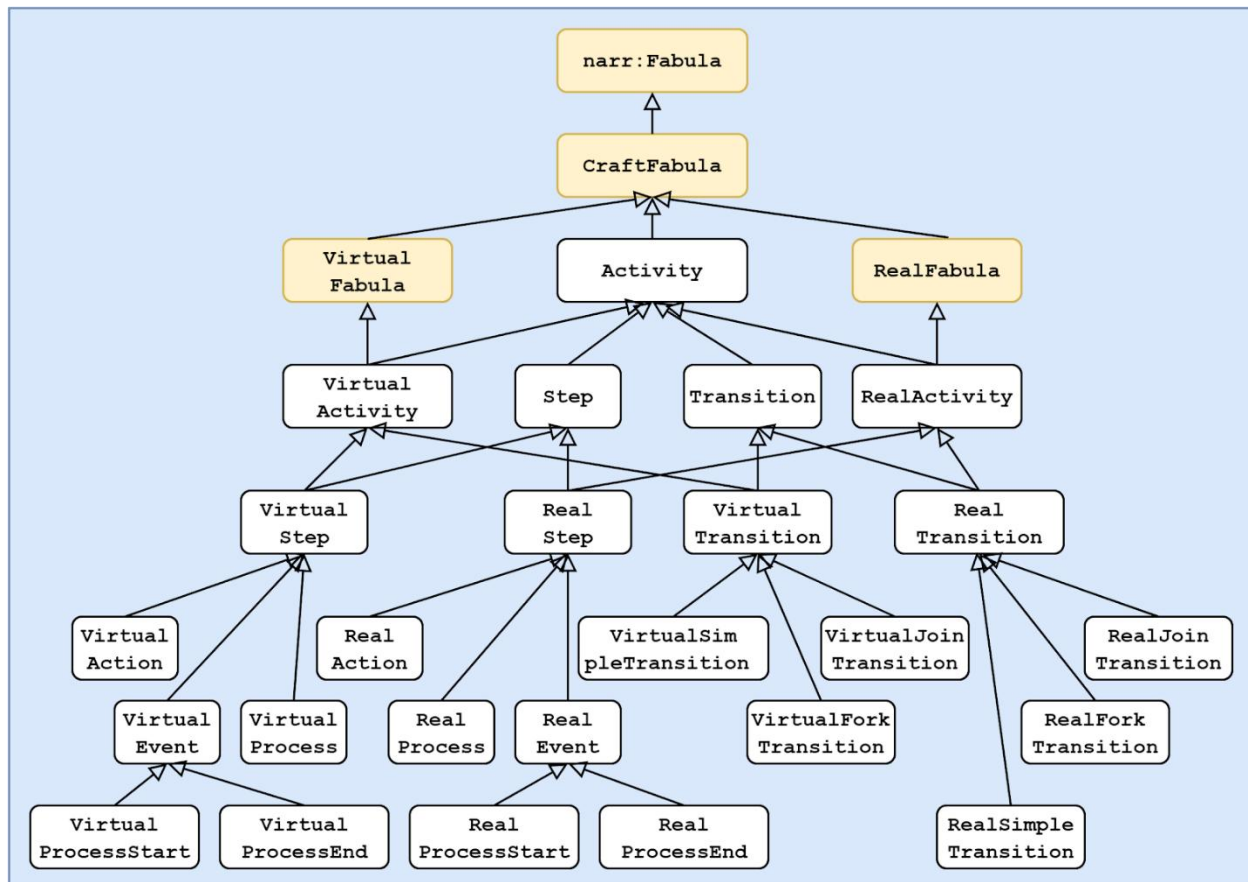
Step is a class
Transition is a class
Activity is the disjoint union of Step and Transition
VirtualStep is a class
VirtualStep is a subclass of Step
VirtualStep is a subclass of VirtualActivity
RealStep is a class
RealStep is a subclass of Step
RealStep is a subclass of RealActivity
VirtualEvent is a class
VirtualProcessStart is a class
VirtualProcessStart is a subclass of VirtualEvent
VirtualProcessEnd is a class
VirtualProcessEnd is a subclass of VirtualEvent
VirtualProcessEnd is disjoint from VirtualProcessStart
VirtualAction is a class
VirtualProcess is a class
VirtualStep is the disjoint union of VirtualEvent , VirtualAction and VirtualProcess
RealEvent is a class

RealProcessStart is a class
RealProcessStart is a subclass of VirtualEvent
RealProcessEnd is a class
RealProcessEnd is a subclass of VirtualEvent
RealAction is a class
RealProcess is a class
RealStep is the disjoint union of RealEvent , RealAction and RealProcess
VirtualTransition is a class
VirtualTransition is a subclass of Transition
VirtualTransition is a subclass of VirtualActivity
RealTransition is a class
RealTransition is a subclass of Transition
RealTransition is a subclass of RealActivity
VirtualSimpleTransition is a class
VirtualForkTransition is a class
VirtualJoinTransition is a class
VirtualTransition is the disjoint union of VirtualSimpleTransition , VirtualForkTransition and VirtualJoinTransition
RealSimpleTransition is a class
RealForkTransition is a class

RealJoinTransition is a class

RealTransition is the disjoint union of **RealSimpleTransition**, **RealForkTransition** and **RealJoinTransition**

Graphically (coloured boxes contain classes already introduced):



We now complete the axiomatisation of GPs A and B by giving the remaining localisation axioms, along with the corresponding cardinality axioms.

An instance of **VirtualProcess** is connected by **hasProcessStart** only to instances of **VirtualProcessStart**

An instance of **VirtualProcess** is connected by **hasProcessStart** to exactly one instance of **VirtualProcessStart**

An instance of **VirtualProcessStart** is connected by **isProcessStartOf** only to instances of **VirtualProcess**

An instance of VirtualProcessStart is connected by isProcessStartOf to exactly one instance of VirtualProcess
An instance of RealProcess is connected by hasProcessStart only to instances of RealProcessStart
An instance of RealProcess is connected by hasProcessStart to exactly one instance of RealProcessStart
An instance of RealProcessStart is connected by isProcessStartOf only to instances of RealProcess
An instance of RealProcessStart is connected by isProcessStartOf to exactly one instance of RealProcess

2.5.1.2 Step Schemas

This Section analyses schemas of steps, to the end of formalising the conceptualisation given above. We recall that a step can be an action, an event or a process. Since processes are formed by actions and events, their features will automatically result from those of their constituents and can be obtained via the appropriate queries. Moreover, since events are simpler kinds of actions, their representation is a simplification of that of actions. In what follows, then, we will focus on the representation of actions, indicating whether each analysed feature applies to events too.

An action is typed in three different ways:

- By a function in a standardised vocabulary of functions; the vocabulary is captured by class **ActionFunction**, subclass of **ecrm:E55_Type**; these individuals are linked to the action by using the property **hasActionFunction**, ranging over the action function class.
- By one of the following action types: *Add*, *Subtract*, *Interlock*, *Transform*. Each of these types is represented by an individual, instance of class **ActionType** that is a subclass of **ecrm:E55_Type**; these individuals are linked to the action by using the property **hasActionType**, ranging over the action type class.
- By one or more of the six Archimedean simple machines (e.g. Lever, Wheel and axle, Pulley, Inclined plane, Wedge, Screw) or a physical or chemical agent using the property **hasMachineType**. The Archimedean simple machines are individuals, instances of the class **MachineType** that is a subclass of **ecrm:E55_Type**.

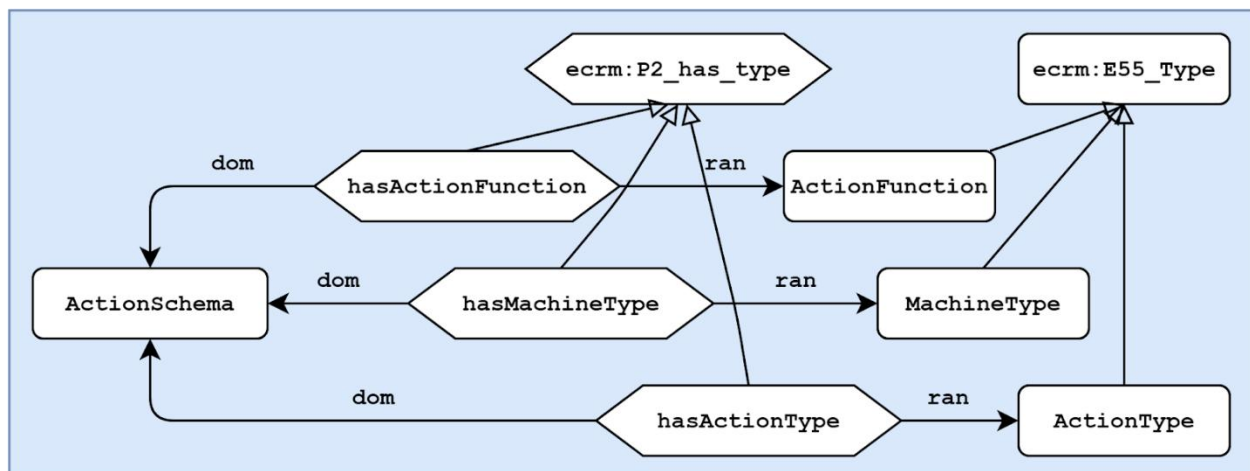
These properties are all sub-properties of **ecrm:P2_has_type**. They do not apply to events and have the same value for all virtual actions that are instances of the same action schema and for all real actions that enact these virtual actions; for this reason, they are associated with action schemas and are not replicated in virtual and real actions. Therefore they all have **ActionSchema** as their domain.

ActionFunction is a class

ActionFunction is a subclass of ecrm:E55_Type
eafunction_n is an instance of class ActionFunction
...
eafunction_n is an instance of class ActionFunction
hasActionFunction is an object property
hasActionFunction is a subproperty of ecrm:P2_has_type
The domain of hasActionFunction is class ActionSchema
The range of hasActionFunction is class ActionFunction
ActionType is a class
ActionType is a subclass of ecrm:E55_Type
add is an instance of class ActionType
subtract is an instance of class ActionType
interlock is an instance of class ActionType
transform is an instance of class ActionType
hasActionType is an object property
hasActionType is a subproperty of ecrm:P2_has_type
The domain of hasActionType is class ActionSchema
The range of hasActionType is class ActionType
MachineType is a class

MachineType is a subclass of ecrm:E55_Type
LeverMachineType is an instance of class MachineType
WedgeMachineType is an instance of class MachineType
ScrewMachineType is an instance of class MachineType
InclinedPlanMachineType is an instance of class MachineType
PulleyMachineType is an instance of class MachineType
WheelAxleMachineType is an instance of class MachineType
PhysicalAgent is an instance of class MachineType
ChemicalAgent is an instance of class MachineType
hasMachineType is an object property
hasMachineType is a subproperty of ecrm:P2_has_type
The domain of hasMachineType is class ActionSchema
The range of hasMachineType is class MachineType

Graphically:



Generalised Property Schemas

An action schema specifies the causing entities and the affected objects of the action by giving the properties that must be used to associate virtual actions that instantiate the schema to the involved virtual causing entities and virtual affected objects, and real actions that are enactments of those virtual actions to the involved real causing entities and real affected objects. These properties are specified via a special class, called **GPSchema**, each instance of which provides all required information about a GP. Since class **GPSchema** is used extensively in action schemas, we start by introducing it.

An instance of class **GPSchema** provides the following information about a property: each instance of which provides the following kinds of information:

1. the *property* itself, that is the symbol of the vocabulary used for the property. This information is provided by property **ofProperty**, having **GPSchema** as domain and **rdf:Property** as range, and being mandatory and functional;
2. the *cardinality* of the property, *i.e.*, how many entities of the same type the property connects. This information is provided by data property **hasCardinality**, having **GPSchema** as domain and **xsd:int** as range, also mandatory and functional;
3. the *domain* of the property, provided by object property **hasDomain**, having **GPSchema** as domain and **owl:Class** as range, also mandatory and functional;
4. the *range* of the property, provided by object property **hasRange**, having **GPSchema** as domain and **owl:Class** as range, also mandatory and functional;
5. the *applicability* of the property, *i.e.*, to which classes the property applies and, for each such classes, which classes the properties range over. This information is provided by property **hasApplicability**, having **GPSchema** as domain and class **Applicability** as range, mandatory and having at-least one as cardinality reflecting the fact that class **GPSchema** is used for the specification of GPs. In turn, every instance of **Applicability** provides a domain (via property **domainOfApplicability**) and a range (**rangeOfApplicability**). Both properties **domainOfApplicability** and **rangeOfApplicability** have **owl:Class** as domain and range, and are mandatory and functional. Any instance of **Applicability** must conform to the domain and range of the GP: thus, the class given as **domainOfApplicability** must be a subclass of the domain of the GP, given by property **hasDomain**; and the class given as **rangeOfApplicability** must be a subclass of the range of the GP, given by property **hasRange**.

Notice that all the information carried by an instance of **GPSchema** can be expressed using OWL 2 DL axioms. For instance, to express the fact that actions instances of class **C** have exactly two causing entities of type **E**, the axiom

```
SubClassOf(C ObjectExactCardinality(2 P E))
```

can be used, where **P** is the property for causing entities of type **E**. However, the cardinality information (2, in this case) embedded in this axiom cannot be recovered, *i.e.*, it is not possible to write a SPARQL query that returns that information. Now, this information is needed by the instantiation algorithm and for this reason, **GPSchema** must be used.

GPSchema is a subclass of **ecrm:E73_Information_Object**.

GPSSchema is a class
GPSSchema is a subclass of ecrm:E73_Information_Object
ofProperty is an object property
The domain of ofProperty is class GPSSchema
The range of ofProperty is class rdf:Property
An instance of GPSSchema is connected by ofProperty to exactly one instance of class rdf:Property
hasCardinality is a data property
The domain of hasCardinality is class GPSSchema
The range of hasCardinality is data range xsd:short
An instance of GPSSchema is connected by hasCardinality to exactly one instance of data range xsd:short
hasDomain is a data property
The domain of hasDomain is class GPSSchema
The range of hasDomain is is class owl:Class
An instance of GPSSchema is connected by hasDomain to exactly one instance of class owl:Class
hasRange is a data property
The domain of hasRange is class GPSSchema
The range of hasRange is class owl:Class
An instance of hasRange is connected by hasRange to exactly one instance of class owl:Class
GPApplicability is a class

GPAapplicability is a subclass of ecrm:E73_Information_Object
hasApplicability is an object property
The domain of hasApplicability is class GPSchema
The range of hasApplicability is class GPAapplicability
An instance of GPSchema is connected by hasApplicability to at least one instance of class GPAapplicability
domainOfApplicability is an object property
The domain of domainOfApplicability is class GPAapplicability
The range of domainOfApplicability is class owl:Class
An instance of GPAapplicability is connected by domainOfApplicability to exactly one instance of class owl:Class
rangeOfApplicability is an object property
The domain of rangeOfApplicability is class GPAapplicability
The range of rangeOfApplicability is class owl:Class
An instance of GPAapplicability is connected by rangeOfApplicability to exactly one instance of class owl:Class

Action Schemas

Let us now turn back to action schemas and to the representation of their causing and affected entities using the machinery just introduced.

Property **hasCausingEntityProperty** connects any action schema to the properties, called *causing entity properties* (CEPs), giving the causing entities of the virtual actions instantiating the schema and of the real actions enacting them. Events have no causing entities, therefore property **hasCausingEntityProperty** has **ActionSchema** as domain, while its range is class **GPSchema**, discussed above. Every CEP is a sub-property of the CRM property **ecrm:P15_was_influenced_by**, which “*captures the relationship between an instance of E7 Activity and anything, that is, an instance of E1 CRM Entity that may have had some bearing upon it*”. Since an action has at least one causing entity, property **hasCausingEntityProperty** is mandatory.

For instance, let us consider the action of hitting a chisel with a hammer, having just one causing entity, namely the force driving the hammer. Then, any virtual action must have one property, say CEP **hasForce**, connecting the action to a virtual force driving the hammer; similarly, any action that enacts that virtual action must be linked by property **hasForce** to a real force driving the hammer. To specify that, the schema of the action, let it be named **as1**, is connected by **hasCausingEntityProperty** to one instance of **GPSSchema**, let it be named **gp1**, describing the CEP **hasForce** as follows:

1. **gp1** is linked by property **ofProperty** to property **hasForce**;
2. **gp1** is linked by data property **hasCardinality** to the positive integer 1;
3. **gp1** is linked by data property **hasDomain** to the class **narr:Fabula**;
4. **gp1** is linked by data property **hasRange** to the class **CausingEntity** for the reasons explained in Section 3.3.4.1 Representing causing entities below;
5. **gp1** is linked by property **hasApplicability** to two instances of class **Applicability**, **app1** and **app2** as follows:
 - a. **app1** is linked by property **domainOfApplicability** to class **VirtualAction** and by property **rangeOfApplicability** to class **VirtualForce**;
 - b. **app2** is linked by property **domainOfApplicability** to class **RealAction** and by property **rangeOfApplicability** to class **RealForce**.

In English, the above statements read as “the action schema **as1** has one causing entity property named **hasForce**, having **narr:Fabula** as domain and **CausingEntity** as range, and connecting any instance of the schema to exactly one instance of **VirtualForce** and any real action that enacts an instance of the schema to exactly one instance of **RealForce**”. Notice that this statement can be directly expressed as a set OWL 2 DL axioms as follows:

1. **as1** is linked by property **hasCausingEntityProperty** to property **hasForce**;
2. every instance of schema **as1** is connected by **hasForce** only to instances of **VirtualForce**;
3. every instance of schema **as1** is connected by **hasForce** to exactly one instance of **VirtualForce**;
4. every enactment of any instance of schema **as1** is connected by **hasForce** only to instances of **RealForce**;
5. every enactment of any instance of schema **as1** is connected by **hasForce** to exactly one instance of **RealForce**;
6. the domain of **hasForce** is class **narr:Fabula**;
7. the range of **hasForce** is class **CausingEntity**.

In practice, an instance of class **GPSSchema** reifies the above set of OWL 2 DL axioms by using a set of properties to carry all the information needed to express those axioms. For this reason, we will call the former modelling style the *reified* approach and the latter the *direct* approach. The advantage of the reified approach is practical: it provides direct access to the information encoded in a **GPSSchema** instance, such as the cardinality of a CEP property, which is not possible to recover from the axioms used by the direct approach; on the other hand, the reified approach does not allow using consistency checking for maintaining the integrity of an ontology, as it omits the specification of the OWL 2 DL axioms that express the basic facts for a CEP. The situation is reversed for the direct approach. To have the advantages of both approaches, we adopt both of them using class **GPSSchema** to describe a CEP while including the equivalent OWL 2 DL axioms in a craft ontology. This presents a potential consistency issue, as the two specifications must be coherent with one another. To mitigate this problem, wherever possible redundancy will be

avoided; in practice, properties **hasDomain**, **hasRange** and **hasApplicability** will be used as little as possible and direct axioms will be used in their place. Also, user interface techniques will be used to control this issue as much as possible.

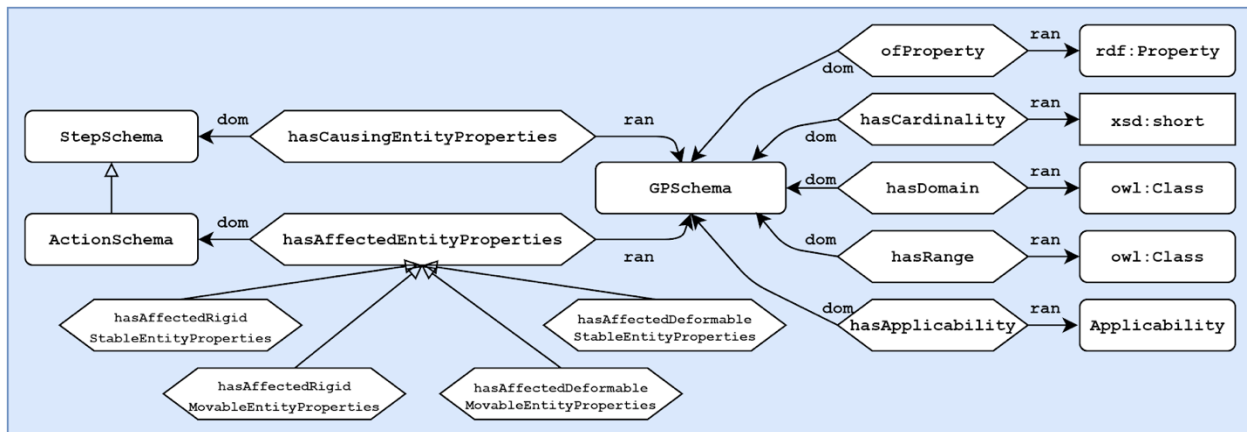
Similarly, property **hasAffectedEntityProperty** is used for indicating the properties giving the entities affected by the action. Such entities are further categorised along two orthogonal dimensions: the rigid/deformable and the stable/mobile dimensions. As a consequence, **hasAffectedEntityProperty** has four sub-properties, each capturing a combination of the two dimensions:

- the **hasAffectedRigidStableEntityProperty** property, for the rigid and stable entities;
- the **hasAffectedRigidMovableEntityProperty** property, for the rigid and movable entities;
- the **hasAffectedDeformableStableEntityProperty** property, for the deformable and stable entities;
- the **hasAffectedDeformableMovableEntityProperty** property, for the deformable and movable entities.

Each of these properties connects any action schema to the properties, called *affected entity properties* (AEPs), giving the affected entities the virtual actions instantiating the schema and the real actions enacting them. Events can have affected entities, therefore each property has **StepSchema** as a domain, while its range is class **GPSchema**. Every AEP is a sub-property of the CRM property **ecrm:P12_occurred_in_the_presence_of**, which “describes the active or passive presence of an E77 Persistent Item in an instance of E5 Event without implying any specific role”.

For instance, the action of hitting a piece of metal with a hammer has three affected entities: the hammer and the chisel, both rigid and movable; and the piece of wood, deformable and stable. As a consequence, the corresponding action schema has two GP schemas associated with it by the **hasAffectedRigidMovableEntityProperty** property, one defining property **hasHammer** and the other defining property **hasChisel**, and one GP schema associated with it by the **hasAffectedDeformableStableEntityProperty** property, defining property **hasPieceOfWood**.

Graphically:



hasCausingEntityProperty is an object property

The domain of hasCausingEntityProperty is class ActionSchema
The range of hasCausingEntityProperty is class GPSchema
An instance of ActionSchema is connected by hasCausingEntityProperty to at least one instance of class GPSchema
hasAffectedEntityProperty is an object property
The domain of hasAffectedEntityProperty is class StepSchema
The range of hasAffectedEntityProperty is class GPSchema
hasAffectedRigidStableEntityProperty is an object property
hasAffectedRigidStableEntityProperty is a subproperty of hasAffectedEntityProperty
hasAffectedRigidMovableEntityProperty is an object property
hasAffectedRigidMovableEntityProperty is a subproperty of hasAffectedEntityProperty
hasAffectedDeformableStableEntityProperty is an object property
hasAffectedDeformableStableEntityProperty is a subproperty of hasAffectedEntityProperty
hasAffectedDeformableMovableEntityProperty is an object property
hasAffectedDeformableMovableEntityProperty is a subproperty of hasAffectedEntityProperty

An action or an event is also connected to the states of the affected objects before and after the action or the event, and, possibly, to the states of the produced objects. Notice that an event may produce something even though it implies no action by the people involved in a craft; for instance, an event may produce a fruit that has grown during the event or something done by someone outside the craft, such as a tool or a piece of material. The CRO must therefore represent both these connections. Since they apply to any step, they are not specified by schemas, like the connections seen so far, but are axiomatised independently of schemas. However, their axiomatisation will have to wait for that of objects, which is done in the next Section.

Agents in actions

In this Section, we formalise the relationship between actions and actors. We recall that agents in action schema are represented by roles, instances of class **narr:Role**, which give the type of agent required by the action. In contrast, agents in virtual action are software tools, an instance of class **SoftwareTool**, the subclass of **VirtualTool**, which is part of the Common Craft Ontology (see below). Finally, agents in real actions are persons, instances of class **ecrm:E21_Person**, with the role established by the corresponding schemas.

Following the modelling style adopted thus far, the CrO defines a single property for connecting actions to agents, **hasAgent**, using localisation axioms to properly link the different kinds of actions to the different kinds of agents. Because of its generality, **hasAgent** cannot be mapped in any CRM property.

SoftwareTool is a class
SoftwareTool is a subclass of Tool
hasAgent is an object property
An instance of ActionSchema is connected by hasAgent only to instances of class narr:Role
An instance of ActionSchema is connected by hasAgent to at least one instance of class narr:Role
An instance of VirtualAction is connected by hasAgent only to instances of class SoftwareTool
An instance of VirtualAction is connected by hasAgent to at least one instance of class SoftwareTool
An instance of RealAction is connected by hasAgent only to instances of class ecrm:E21_Person
An instance of RealAction is connected by hasAgent to at least one instance of class ecrm:E21_Person

2.5.1.3 Objects

Like processes and their constituents, also the objects affected by actions come in two main kinds, as already pointed out in Section 3.2.2 Actions:

- virtual objects, that is mathematical models representing objects in the space of simulation, and
- real objects, that is semantic models representing objects in the real world.

Following the same approach adopted for steps and transitions, the CRO introduces class **Object**, having as instances all objects of interest for crafts, and two subclasses of its: **VirtualObject** and **RealObject**.

Object is a subclass of the CRM class **ecrm:E70_Thing**, which “*comprises discrete, identifiable, instances of E77 Persistent Item that are documented as single units, that either consists of matter or depend on being carried by matter and are characterised by relative stability*”. This definition applies to both the physical human-made objects instances of class **RealObject** and the mathematical representations that are instances of **VirtualObject**. To strengthen the link with the CRM the CRO further asserts class **VirtualObject** as a subclass of **ecrm:E73_Information_Object**, which comprises “*identifiable immaterial items, such as poems, jokes, data sets, images, texts, multimedia objects, procedural prescriptions, computer program code, algorithm or mathematical formulae, that have an objectively recognizable structure and are documented as single units*”, and class **RealObject** as a subclass of **ecrm:E24_Physical_Human-Made_Thing**, which comprises “*all persistent physical items of any size that are purposely created by human activity*”.

Object is a class
Object is a subclass of ecrm:E70_Thing
VirtualObject is a class
VirtualObject is a subclass of Object
VirtualObject is a subclass of ecrm:E73_Information_Object
RealObject is a class
RealObject is a subclass of Object
RealObject is a subclass of ecrm:E24_Physical_Human-Made_Thing

Classes for specific kinds of objects, such as for instance hammers or chisels, will be introduced in due course upon modelling the actions in which these objects are involved.

In addition, we introduce a class for composite objects, **CompositeObject**, having as instances the objects created in actions by assembling other objects, in ways that may vary from action to action. **CompositeObject** is a subclass of **Object** and specialises in **VirtualCompositeObject**, also a subclass of **VirtualObject**, and **RealCompositeObject**, also a subclass of **RealObject**.

CompositeObject is a class
CompositeObject is a subclass of Object
VirtualCompositeObject is a class

VirtualCompositeObject is a subclass of **CompositeObject**

VirtualCompositeObject is a subclass of **VirtualObject**

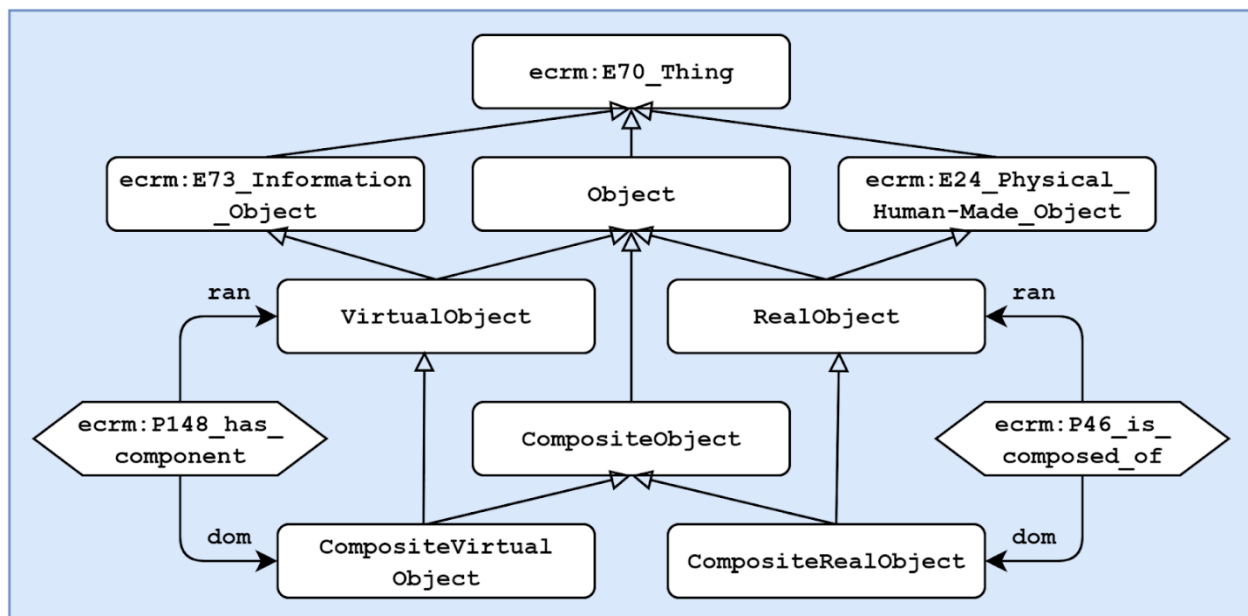
RealCompositeObject is a class

RealCompositeObject is a subclass of **RealObject**

RealCompositeObject is a subclass of **CompositeObject**

The properties for associating composite objects to their components are taken directly from the CRM, and are property **ecrm:P148_has_component** for virtual composite objects, and property **ecrm:P46_is_composed_of** for real composite objects.

Graphically:



As argued above, object characteristics are grouped into time-dependent and time-independent, the former giving the state of an object. To reflect this conceptualisation straightforwardly, the CRO introduces two classes: **ObjectProfile**, a subclass of **ecrm:E73_Information_Object** holding the time-independent characteristics of an object, and **ObjectState**, a subclass of **ecrm:E3_Condition_State** holding the time-dependent ones. Each of these classes is further specialised along the virtual/real dimensions, producing four classes:

- **VirtualObjectProfile**,

- **VirtualObjectState**,
- **RealObjectProfile** and
- **RealObjectState**.

ObjectProfile is a class
ObjectProfile is a subclass of ecrm:E73_Information_Object
ObjectState is a class
ObjectState is a subclass of ecrm:E3_Condition_State
VirtualObjectProfile is a class
VirtualObjectProfile is a subclass of ObjectProfile
VirtualObjectState is a class
VirtualObjectState is a subclass of ObjectState
RealObjectProfile is a class
RealObjectProfile is a subclass of ObjectProfile
RealObjectState is a class
RealObjectState is a subclass of ObjectState

Correspondingly, the CRO introduces two GPs for linking objects to their profiles and their states:

hasProfile, a subproperty of **ecrm:P67i_is_referred_to_by** having **Object** as domain and **ObjectProfile** as range; the inverse property **isProfileOf** is also introduced;

hasState, having **Object** as domain and **ObjectState** as range but no corresponding property in the CRM, since the CRM property linking an object to a state, a subproperty of **ecrm:P44_has_condition**, only applies to physical objects and is therefore not applicable to virtual objects. The inverse property **isStateOf** is also introduced.

These properties are further localised as required by the GP pattern. In addition, both GPs are one-to-many, a fact that is captured by cardinality axioms.

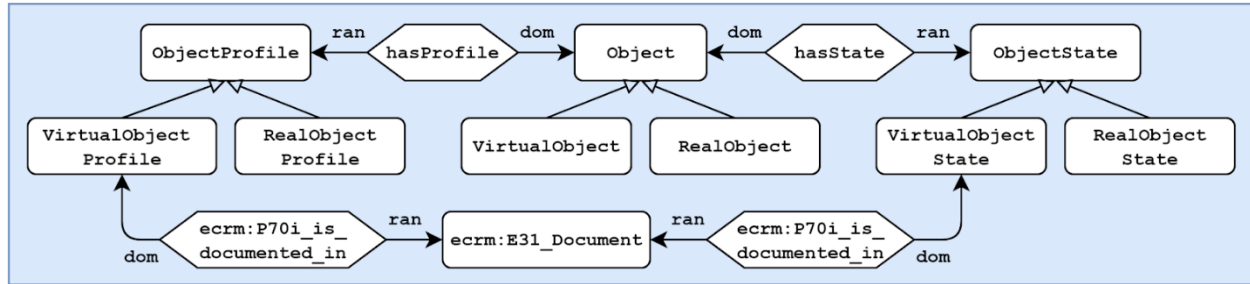
hasProfile is an object property
hasProfile is a super-property of ecrm:P67i_is_referred_to_by
The domain of hasProfile is Object
The range of hasProfile is ObjectProfile
isProfileOf is an object property
isProfileOf is the inverse of property hasProfile
An instance of VirtualObject is connected by property hasProfile only to instances of class VirtualObjectProfile
An instance of VirtualObject is connected by property hasProfile to exactly one instance of class VirtualObjectProfile
An instance of VirtualObjectProfile is connected by property isProfileOf only to instances of class VirtualObject
An instance of VirtualObjectProfile is connected by property isProfileOf to exactly one instance of class VirtualObject
An instance of RealObject is connected by property hasProfile only to instances of class RealObjectProfile
An instance of RealObject is connected by property hasProfile to exactly one instance of class RealObjectProfile
An instance of RealObjectProfile is connected by property isProfileOf only to instances of class RealObject
An instance of RealObjectProfile is connected by property isProfileOf to exactly one instance of class RealObject
hasState is an object property
The domain of hasState is Object

The range of hasState is ObjectState
isStateOf is an object property
isStateOf is the inverse of property hasState
An instance of VirtualObject is connected by property hasState only to instances of class VirtualObjectState
An instance of VirtualObjectState is connected by property isStateOf only to instances of class VirtualObject
An instance of VirtualObjectState is connected by property isStateOf to exactly one instance of class VirtualObject
An instance of RealObject is connected by property hasState only to instances of class RealObjectState
An instance of RealObjectState is connected by property isStateOf only to instances of class RealObject
An instance of RealObjectState is connected by property isStateOf to exactly one instance of class RealObject

Finally, let us consider how profiles and states are represented. Conceptually, profiles and states of virtual objects are mathematical representations of the corresponding profiles and states of real objects, respectively. However, for practical reasons, the profile of a virtual object is collected into a single document, an instance of class **ecrm:E31_Document**; and the same goes for the state. The rationale behind this choice is that the characteristics of virtual objects, whether time-independent or time-dependent, are produced by the simulator and stored in the CRAEFT KB exclusively to be later consumed by the same simulator, therefore the CRO does not need to enter the details of what characteristics are represented and how, but simply provides the means for catering to the needs of the simulator. The property connecting a virtual object profile or state to the document where it is represented is taken from the CRM and is property **ecrm:P70i_is_documented_in**, which “*describes the CRM Entities documented by instances of E31 Document*” and has class **ecrm:E1_CRM_Entity** as domain, therefore it applies to both virtual object profiles and states, while its range is **ecrm:E31_Document**.

In contrast, the characteristics of real objects are given a semantical representation, that is, they are expressed by linking the individuals representing the profile or the state of a real object (instances of **RealObjectProfile** or **RealObjectState**, respectively) to the relevant individuals via the appropriate properties. Since these individuals, the classes they belong to, and the properties linking them to an object are dependent on the object at the end, they will be introduced upon modelling specific (real) actions.

Graphically:



We are now ready to represent the association of a step with the states of the affected objects before and after the step, and with the objects produced by the step, if any. The former associations are captured by the GPs **AOSBefore** and **AOSAAfter**, both having **Step** as domain and **ObjectState** as range. Notice that these properties cannot be mapped to the CRM because they are used also to associate a virtual step with states of virtual objects which are not physical objects, as already pointed out above. The remaining axioms of the GP pattern are given below.

AOSBefore is an object property
The domain of AOSBefore is class Step
The range of AOSBefore is class ObjectState
An instance of VirtualStep is connected by property AOSBefore only to instances of class VirtualObjectState
An instance of VirtualObjectState is connected by the inverse of property AOSBefore only to instances of class VirtualStep
An instance of RealStep is connected by property AOSBefore only to instances of class RealObjectState
An instance of RealObjectState is connected by the inverse of property AOSBefore only to instances of class RealStep
AOSAAfter is an object property
The domain of AOSAAfter is class Step
The range of AOSAAfter is class ObjectState

An instance of VirtualStep is connected by property AOSAAfter only to instances of class VirtualObjectState
An instance of VirtualObjectState is connected by the inverse of property AOSAAfter only to instances of class VirtualStep
An instance of RealStep is connected by property AOSAAfter only to instances of class RealObjectState
An instance of RealObjectState is connected by the inverse of property AOSAAfter only to instances of class RealStep

Notice that the most important fact relating actions and affected objects' states is that, given an action schema AS and a virtual action VA instance of AS, properties **AOSBefore** and **AOSAAfter** should link VA with all and only the states of the objects that are linked to VA by the properties given by one of the four sub-properties of **hasAffectedEntityProperty** in VA. The same for any real action RA enacting VA. Unfortunately, this fact cannot be axiomatised because the properties linking an action to its affected objects are not known *a priori*, so they could only be denoted by object property expressions such as "the property that is linked to a virtual action via property **hasAffectedRigidStableEntityProperty**"; but the only property expressions allowed by OWL 2 DL are object properties and their inverses. The CRO can therefore provide the weaker axiomatisation given above, consisting of localisation axioms establishing that a virtual (resp. real) action can only be connected by properties **AOSBefore** and **AOSAAfter** to states of virtual (resp. real) objects, with no guarantee that these objects are amongst the affected objects of the action.

To represent the association of a step with the created entities, the property **produces** is introduced in the CRO, also a GP property, axiomatised below. If an action or an event does not produce any new object, the property **produces** will connect that action or event with no object at all. As already remarked in [3.3.3 Step schemas](#), the domain of **produces** is class **Step**, as also events may result in a new object being produced, while its range is class **Object**. Moreover, **produces** is a sub-property of the CRM property **ecrm:P92_brought_into_existence**, which "links an instance of E63 Beginning of Existence to the instance of E77 Persistent Item brought into existence by it". This relationship automatically classifies the steps that produce something as instances of the CRM class **ecrm:E63_Beginning_of_Existence**, that is as "events that bring into existence any instance of E77 Persistent Item". This is an intended consequence.

produces is an object property
produces is a sub-property of property ecrm:P92_brought_into_existence
The domain of produces is class Step
The range of produces is class Object

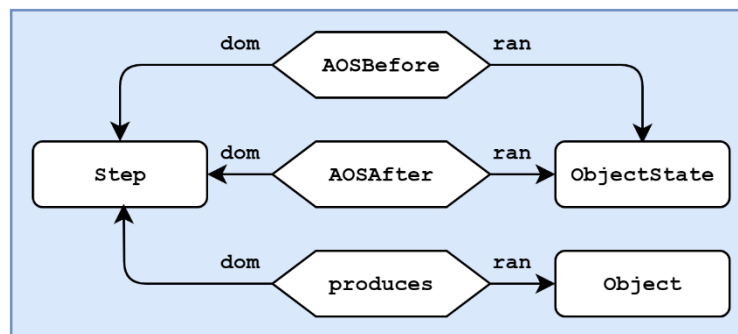
An instance of **VirtualStep** is connected by property **produces** only to instances of class **VirtualObject**

An instance of **VirtualObject** is connected by the inverse of property **produces** only to instances of class **VirtualStep**

An instance of **RealStep** is connected by property **produces** only to instances of class **RealObject**

An instance of **RealObject** is connected by the inverse of property **produces** only to instances of class **RealStep**

Graphically:



Representing causing entities

As explained above, causing entities are perdurants that play a causal role in actions. In this Section, we discuss how causing entities are represented in the CRO.

Following the approach adopted for affected objects, the CRO could represent causing entities as objects endowed with time-independent properties, connected to their causing entities via property **hasProfile**, and with time-dependent properties represented as states, connected to their causing entities via property **hasState**. The time-independent properties of, *e.g.*, a force could be the nature, the type, the measurement unit, and so on, while its time-dependent properties could be the vectors representing the force along the time axis. While this representation is suitable for virtual actions, as the simulator needs to know how to apply the force per unit of time, there is no application requirement to represent the individual states of a force from a semantic point of view. Consequently, the CRO introduces a class, named **CausingEntity**, a subclass of **ecrm:E5_Event**, having as instances causing entities in actions. **CausingEntity** is further specialised in **VirtualCausingEntity** and **RealCausingEntity** for which no further representation is offered. For virtual entities, the CRM property **ecrm:P70i_is_documented_in** can be used to link a virtual-causing entity to a document that gives the mathematical representation of the entity. For real entities, a semantic representation can be given hinged on the real causing entity.

CausingEntity is a class

CausingEntity is a subclass of ecrm:E5_Event
VirtualCausingEntity is a class
VirtualCausingEntity is a subclass of CausingEntity
RealCausingEntity is a class
RealCausingEntity is a subclass of CausingEntity

2.5.1.4 Instantiation

We now axiomatise the GP **instanceOf**, which represents the instantiation relation between virtual activities and the corresponding schemas. For convenience, we introduce also the inverse property of **instanceOf**, **hasInstance**. The domain of **instanceOf** is the most specialised class that includes both virtual steps and virtual transitions, which is the class **VirtualActivity** while its range is the class **ActivitySchema**. Mapping **instanceOf** to a CRM property is not possible due to the generality of the former.

instanceOf is an object property
The domain of instanceOf is class VirtualActivity
The range of instanceOf is class ActivitySchema
hasInstance is an object property
hasInstance is an inverse property of instanceOf

The following localisation axioms apply the instantiation property to steps. They are the axioms of the GP pattern and cardinality axioms.

An instance of VirtualProcess is connected by instanceOf only to instances of ProcessSchema
An instance of VirtualProcess is connected by instanceOf to exactly one instance of ProcessSchema
An instance of ProcessSchema is connected by hasInstance only to instances of VirtualProcess

An instance of ProcessSchema is connected by hasInstance to at least one instance of VirtualProcess
An instance of VirtualAction is connected by instanceOf only to instances of ActionSchema
An instance of VirtualAction is connected by instanceOf to exactly one instance of ActionSchema
An instance of ActionSchema is connected by hasInstance only to instances of VirtualAction
An instance of ActionSchema is connected by hasInstance to at least one instance of VirtualAction
An instance of VirtualEvent is connected by instanceOf only to instances of EventSchema
An instance of VirtualEvent is connected by instanceOf to exactly one instance of EventSchema
An instance of EventSchema is connected by hasInstance only to instances of VirtualEvent
An instance of EventSchema is connected by hasInstance to at least one instance of VirtualEvent
An instance of VirtualProcessStart is connected by instanceOf only to instances of ProcessStartSchema
An instance of VirtualProcessStart is connected by instanceOf to exactly one instance of ProcessStartSchema
An instance of ProcessStartSchema is connected by hasInstance only to instances of VirtualProcessStart
An instance of ProcessStartSchema is connected by hasInstance to at least one instance of VirtualProcessStart
An instance of VirtualProcessEnd is connected by instanceOf only to instances of ProcessEndSchema
An instance of VirtualProcessEnd is connected by instanceOf to exactly one instance of ProcessStartSchema
An instance of ProcessEndSchema is connected by hasInstance only to instances of VirtualProcessEnd

An instance of **ProcessEndSchema** is connected by **hasInstance** to at least one instance of **VirtualProcessEnd**

Similarly, the following axioms apply the instantiation property to transitions. They are the axioms of the GP pattern and cardinality axioms.

An instance of **SimpleVirtualTransition** is connected by **instanceOf** only to instances of **SimpleTransitionSchema** or **DecisionTransitionSchema** or **MergeTransitionSchema**

An instance of **SimpleVirtualTransition** is connected by **instanceOf** to exactly one instance of **SimpleTransitionSchema** or **DecisionTransitionSchema** or **MergeTransitionSchema**

An instance of **SimpleTransitionSchema** is connected by **hasInstance** only to instances of **SimpleVirtualTransition**

An instance of **SimpleTransitionSchema** is connected by **hasInstance** to at least one instance of **SimpleVirtualTransition**

An instance of **DecisionTransitionSchema** is connected by **hasInstance** only to instances of **SimpleVirtualTransition**

An instance of **DecisionTransitionSchema** is connected by **hasInstance** to at least one instance of **SimpleVirtualTransition**

An instance of **MergeTransitionSchema** is connected by **hasInstance** only to instances of **SimpleVirtualTransition**

An instance of **MergeTransitionSchema** is connected by **hasInstance** to at least one instance of **SimpleVirtualTransition**

An instance of **ForkVirtualTransition** is connected by **instanceOf** only to instances of **ForkTransitionSchema**

An instance of **ForkVirtualTransition** is connected by **instanceOf** to exactly one instance of **ForkTransitionSchema**

An instance of **ForkTransitionSchema** is connected by **hasInstance** only to instances of **ForkVirtualTransition**

An instance of **ForkTransitionSchema** is connected by **hasInstance** to at least one instance of **ForkVirtualTransition**

An instance of JoinVirtualTransition is connected by instanceOf only to instances of JoinTransitionSchema
An instance of JoinVirtualTransition is connected by instanceOf to exactly one instance of JoinTransitionSchema
An instance of JoinTransitionSchema is connected by hasInstance only to instances of JoinVirtualTransition
An instance of JoinTransitionSchema is connected by hasInstance to at least one instance of JoinVirtualTransition

The relation between schemas and the corresponding virtual entities with respect to the composition property is as follows:

Let VP be a virtual process and PS be a process schema such that VP is an instance of PS . Then, for every virtual step VS that is a step of VP , there exists a step schema SS that is a step of VS , such that VS is an instance of SS . Conversely, for every step schema SS that is a step of PS , there exists a virtual step VS that is a step of VP , such that VS is an instance of SS .

The axiom is an abbreviation for three axioms, one for each of the three entities that can be a step: event, action or process. Now, the cardinality axioms on instantiation guarantee that a virtual step is an instance of exactly one schema, hence, given VP and VS above, there is a unique PS such that VP is an instance of PS , and a unique SS such that VS is an instance of SS . It follows that the above axiom is equivalent to the following axiom, which can be used to automatically derive the **instanceOf** property between virtual processes and the corresponding schemas:

Let VP be a virtual process and VS be a step-in VP . Then VP is an instance of the process schema PS that has as step the step schema of VS .

This axiom can be expressed as an OWL 2 DL complex role inclusion axiom as follows, where **hasStep** is the composition property of processes and **isStepOf** is its inverse (notice that these properties can be simply derived from the properties **hasHead** and **hasTail**):

The chain formed by **hasStep** and **instanceOf** and **isStepOf** is a subproperty of **instanceOf**

Unfortunately, this axiom would make **instanceOf** a composite property, on which no cardinality axiom can be stated, lest a violation of the OWL 2 DL global restriction on simple roles. For this reason, the axiom is not included in the CRO and the correct relationship between schemas and instances is checked procedurally.

Another important axiom on instantiation concerns the correct relation between the properties declared in an action of event schema and the properties used in virtual actions or events that are an instance of that schema. In particular, any property declared in an action schema via **hasCausingEntityProperty** or

any subproperty of **hasAffectedEntityProperty**, must be used in any instance of the schema. This axiom completes the axiom about the proper association between a step and the states of the affected objects, spelt out in Section Objects above, and cannot be expressed in OWL 2 DL for the same reasons.

2.5.1.5 Enactment

We finally axiomatize enactment by GP **enactmentOf**, which represents the enactment relation between real activities and the corresponding virtual activities. For convenience, we introduce also the inverse property of **enactmentOf**, **isEnactedBy**. For the reasons given in the previous Section, the domain of **enactmentOf** is class **RealActivity**, while its range is class **VirtualActivity**.

Similarly to the **instanceOf** property, mapping to a CRM property is not possible due to the generality of the **enactmentOf** property.

enactmentOf is an object property
The domain of enactmentOf is class RealActivity
The range of enactmentOf is class VirtualActivity
isEnactedby is an object property
isEnactedby is an inverse property of enactmentOf

Enacted entities are performed by agents and situated in space and time. To represent these aspects, we use the three CRM properties already mentioned in Section 2.3.2, namely **ecrm:P14_carried_out_by**, which “describes the active participation of an instance of E39 Actor in an instance of E7 Activity”; **ecrm:P7_took_place_at**, which “describes the spatial location of an instance of E4 Period”; and **ecrm:P4_has_time-span**, which “associates an instance of E2 Temporal Entity with the instance of E52 Time-Span during which it was on-going”.

The following axioms apply to the enactment property to steps. They are the axioms of the GP pattern and cardinality axioms.

An instance of RealProcess is connected by enactmentOf only to instances of VirtualProcess
An instance of RealProcess is connected by enactmentOf to exactly one instance of VirtualProcess
An instance of VirtualProcess is connected by isEnactedby only to instances of RealProcess
An instance of VirtualProcess is connected by isEnactedby to at least one instance of RealProcess

An instance of RealAction is connected by enactmentOf only to instances of VirtualAction
An instance of RealAction is connected by enactmentOf to exactly one instance of VirtualAction
An instance of VirtualAction is connected by isEnactedby only to instances of RealAction
An instance of VirtualAction is connected by isEnactedby to at least one instance of RealAction
An instance of RealEvent is connected by enactmentOf only to instances of VirtualEvent
An instance of RealEvent is connected by enactmentOf to exactly one instance of VirtualEvent
An instance of VirtualEvent is connected by isEnactedby only to instances of RealEvent
An instance of VirtualEvent is connected by isEnactedby to at least one instance of RealEvent
An instance of RealProcessStart is connected by enactmentOf only to instances of VirtualProcessStart
An instance of RealProcessStart is connected by enactmentOf to exactly one instance of VirtualProcessStart
An instance of VirtualProcessStart is connected by isEnactedby only to instances of RealProcessStart
An instance of VirtualProcessStart is connected by isEnactedby to at least one instance of RealProcessStart

Similarly, the following axioms apply to the enactment property to transitions. They are the axioms of the GP pattern and cardinality axioms.

An instance of SimpleTransition is connected by enactmentOf only to instances of SimpleVirtualTransition
An instance of SimpleTransition is connected by enactmentOf to exactly one instance of SimpleVirtualTransition
An instance of SimpleVirtualTransition is connected by isEnactedBy only to instances of SimpleTransition

An instance of SimpleVirtualTransition is connected by isEnactedBy to at least one instance of SimpleTransition
An instance of ForkTransition is connected by enactmentOf only to instances of ForkVirtualTransition
An instance of ForkTransition is connected by enactmentOf to exactly one instance of ForkVirtualTransition
An instance of ForkVirtualTransition is connected by isEnactedBy only to instances of ForkTransition
An instance of ForkVirtualTransition is connected by isEnactedBy to at least one instance of ForkTransition
An instance of JoinTransition is connected by enactmentOf only to instances of JoinVirtualTransition
An instance of JoinTransition is connected by enactmentOf to exactly one instance of JoinVirtualTransition
An instance of JoinVirtualTransition is connected by isEnactedBy only to instances of JoinTransition
An instance of JoinVirtualTransition is connected by isEnactedBy to at least one instance of JoinTransition

The enactment axiom concerning activities models the relation between virtual and the corresponding real entities with respect to the composition property. It is as follows:

Let P be a process and VP be a virtual process such that P is an enactment of VP . Then, for every step S of P , there exists a virtual step VS of VP , such that S is an enactment of VS . Conversely, for every virtual step VS of VP there exists a step S of P , such that S is an enactment of VS .

Also in this case, the cardinality axioms on enactment guarantee that a step is the enactment of exactly one virtual step, hence, given P and S above, there is a unique VP such that P is an enactment of VP , and a unique VS such that S is an enactment of VS . It follows that the above axiom is equivalent to the following:

Let P be a process and S be a step in P . Then P is an enactment of the virtual process VP that has as a step the virtual step enacted by S .

This axiom can be expressed as an OWL 2 DL complex role inclusion axiom as follows, where property **hasStep** and its inverse **isStepOf** are as described in the previous Section:

The chain formed by **hasStep** and **enactmentOf** and **isStepOf** is a subproperty of **enactmentOf**

Similarly, the **enactmentOf** relation can be extended to narratives, establishing the enactment of the relation between a real and a virtual narrative:

The chain formed by **hasFabula** and **enactmentOf** and **isFabulaOf** is a subproperty of **enactmentOf**

Unfortunately, these axioms make **enactmentOf** a composite property, on which no cardinality axiom can be stated, lest a violation of the OWL 2 DL global restriction on simple roles. For this reason, the axioms are not included in the CrO.

2.5.2 Transitions

Classes for transition schemas and virtual and real transitions have been already introduced. We now introduce two GPs to model the structure of transitions. Since a transition models a directed hyperedge, it has a set of nodes as the tail and a set of nodes as the head. We use GPs **hasTail** and **hasHead** to associate the nodes that are in the tail or the head to their transitions, respectively. For convenience, we also introduce their inverses **isTailOf** and **isHeadOf**, respectively. The domain and range of **hasTail** and **hasHead** is the most specialised class that includes transition schemas, virtual transitions and real transitions; now, transition schemas are instances of **ecrm:E29_Design_or_procedure**, while virtual and real transitions are instances of **ecrm:E5_Event**. The domain of **hasTail** and **hasHead** is therefore **ecrm:E1_CRM_Entity**. By the same argument, we have that also the range of **hasTail** and **hasHead** is therefore **ecrm:E1_CRM_Entity**. Again by the same argument, we have that It is not possible to map these properties to the CRM, since no CRM property captures the composition of any CRM entity. Indeed, the CRM uses property **ecrm:P106_is_composed_of** for the composition of plans and property **ecrm:P5_consists_of** for the composition of events. Thus, we should relate **hasTail** and **hasHead** to a disjunction of CRM properties; unfortunately, this kind of property expression is not supported by OWL 2 DL.

Notice that a transition may have another transition as tail or head, for instance, if a merge transition follows a decision transition. This situation may be avoided by introducing “dummy” activity nodes, corresponding to no action, created for the sole purpose of enclosing transitions between activity nodes. This would be necessary if transitions were modelled as properties, as it is not possible to have properties of properties. However, transitions are modelled as classes hence the introduction of such dummy activity nodes is not necessary.

hasHead is an object property

The domain of **hasHead** is class **ecrm:E1_CRM_Entity**

The range of **hasHead** is class **ecrm:E1_CRM_Entity**

isHeadOf is an object property
isHeadOf is an inverse property of hasHead
hasTail is an object property
The domain of hasTail is class ecrm:E1_CRM_Entity
The range of hasTail is class ecrm:E1_CRM_Entity
isTailOf is an object property
isTailOf is an inverse property of hasTail

Based on the assumptions given in Section Processes, an activity schema can be in the tail or in the head of at most one transition schema, and the same applies to virtual and real activities.

An instance of ActivitySchema is connected by isTailOf to at most one instance of class TransitionSchema
An instance of ActivitySchema is connected by isHeadOf to at most one instance of class TransitionSchema
An instance of Activity is connected by isTailOf to at most one instance of class Transition
An instance of Activity is connected by isHeadOf to at most one instance of class Transition

The axiomatisation of the various kinds of transitions is given in the remainder of this Section.

2.5.2.1 Simple transitions

A simple transition represents the sequential, unconditional passage from one step (the input step) to another one (the output step). This structure applies to all three levels at which simple transitions are represented (*i.e.*, to instances of **SimpleTransitionSchema**, to instances of **SimpleVirtualTransition** and instances of **SimpleTransition**) and can be modelled by appropriately axiomatizing the properties **hasHead** and **hasTail** already introduced. In particular: a simple transition schema has exactly one tail and one head, which is an activity schema, that is, a step schema or a transition schema. Conversely, an activity schema can be the tail or the head of at most one transition schema, of whatever kind.

An instance of SimpleTransitionSchema is connected by hasTail only to instances of class ActivitySchema
--

An instance of **SimpleTransitionSchema** is connected by **hasTail** to exactly one instance of class **ActivitySchema**

An instance of **SimpleTransitionSchema** is connected by **hasHead** only to instances of class **ActivitySchema**

An instance of **SimpleTransitionSchema** is connected by **hasHead** to exactly one instance of class **ActivitySchema**

A virtual simple transition has exactly one tail and one head, which is a virtual activity. Conversely, a virtual activity can be the tail or the head of at most one virtual transition, of whatever kind.

An instance of **VirtualSimpleTransition** is connected by **hasTail** only to instances of class **VirtualActivity**

An instance of **VirtualSimpleTransition** is connected by **hasTail** to exactly one instance of class **VirtualActivity**

An instance of **VirtualSimpleTransition** is connected by **hasHead** only to instances of class **VirtualActivity**

An instance of **VirtualSimpleTransition** is connected by **hasHead** to exactly one instance of class **VirtualActivity**

An instance of **VirtualActivity** is connected by **isTailOf** to at most one instance of class **VirtualTransition**

An instance of **VirtualActivity** is connected by **isHeadOf** to at most one instance of class **VirtualTransition**

Finally, a real simple transition has exactly one tail and one head, which is a real activity. Conversely, a real activity can be the tail or the head of at most one transition, of whatever kind.

An instance of **RealSimpleTransition** is connected by **hasTail** only to instances of class **RealActivity**

An instance of **RealSimpleTransition** is connected by **hasTail** to exactly one instance of class **RealActivity**

An instance of **RealSimpleTransition** is connected by **hasHead** only to instances of class **RealActivity**

An instance of **RealSimpleTransition** is connected by **hasHead** to exactly one instance of class **RealActivity**

An instance of **RealActivity** is connected by **isTailOf** to at most one instance of class **RealTransition**

An instance of **RealActivity** is connected by **isHeadOf** to at most one instance of class **RealTransition**

2.5.2.2 Decision transitions

A decision transition represents the selection of one of several alternatives, based on the evaluation of a predicate. Consequently, decision transition schemas have one tail schema and two or more head schemas, each associated with a predicate. In contrast, virtual decision transitions have one tail and one head, because they are instances of schemas in which the alternative has been already selected. Similarly, decision transitions have only one head and one tail. In addition to properties **hasTail** and **hasHead**, the CrO uses:

- class **craeft:DTAlternativeSchema** to represent each alternative in a decision transition schema; **craeft:DTAlternativeSchema** is a subclass of **ecrm:E29_Design_or_Procedure**, as its instances represents parts of plans;
- property **craeft:hasAlternative** to associate a decision transition schema to each alternative; **craeft:hasAlternative** is a subproperty of **ecrm:P69_has_association_with**, which as already remarked is the CRM composition property for plans;
- class **craeft:Predicate** to represent the predicate associated with each alternative; also **craeft:Predicate** is a subclass of **ecrm:E29_Design_or_Procedure**, as its instances represent parts of plans;
- properties **craeft:leadsTo** and **craeft:hasPredicate** to associate each alternative to a tail of the schema and a predicate, respectively. Each of these properties is a sub-property of **ecrm:P69_has_association_with**, as they connect plans to their parts. Notice that the link between a decision transition schema and each of its heads, which would be represented by property **craeft:hasHead**, could be deduced from the composition of **craeft:hasAlternative** and **craeft:leadsTo** by a complex role inclusion axiom. However, the inclusion of this axiom in the CrO would make **craeft:hasHead** a composite property, on which no cardinality axiom can be stated, lest a violation of the OWL 2 DL global restriction on simple roles. Since localisation of **craeft:hasHead** does require cardinality axioms, the complex role inclusion axiom is not included in the CrO. As a consequence, the knowledge of heads of Decision Transition Schemas is recovered procedurally.

The linguistic expression of a predicate is represented by a string of characters, which gives the expression in any convenient implementation language. That expression is associated with an instance of class **craeft:Predicate** by property **ecrm:P190_has_symbolic_content**.

The following Figure illustrates the just described representation of decision transition schemas, axiomatized next:

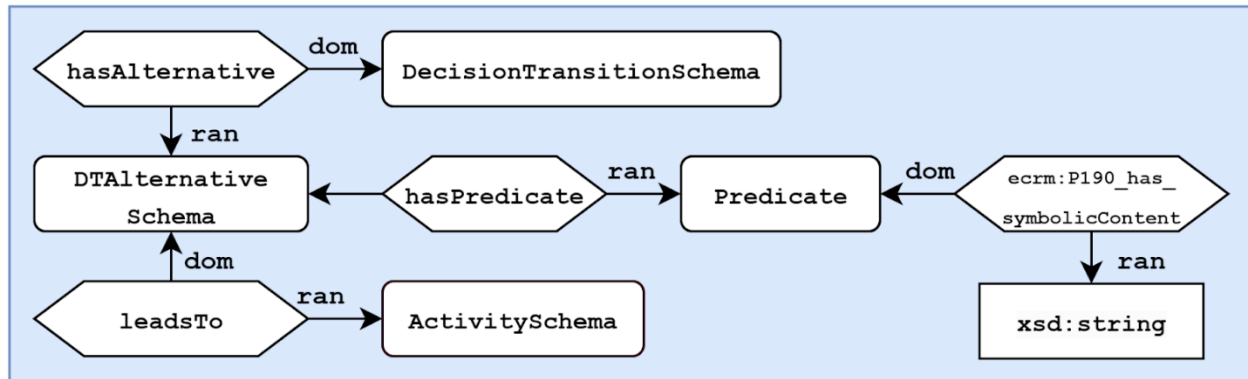


Figure 13 Representation of alternatives in Decision Transition Schemas.

DTAlternativeSchema is a class
DTAlternativeSchema is a subclass of ecrm:E29_Design_or_Procedure
leadsTo is an object property
leadsTo is a subproperty of ecrm:P69_has_association_with
The domain of leadsTo is class DTAlternativeSchema
The range of leadsTo is class ActivitySchema
An instance of DTAlternativeSchema is connected by leadsTo to exactly one instance of ActivitySchema
An instance of ActivitySchema is connected by the inverse of leadsTo to at most one instance of DTAlternativeSchema
Predicate is a class
Predicate is a subclass of ecrm:E29_Design_or_Procedure
hasPredicate is an object property
hasPredicate is a subproperty of ecrm:P69_has_association_with
The domain of hasPredicate is class DTAlternativeSchema
The range of hasPredicate is class Predicate

An instance of DTAlternativeSchema is connected by hasPredicate to exactly one instance of Predicate
An instance of Predicate is connected by the inverse of hasPredicate to exactly one instance of DTAlternativeSchema
hasAlternative is an object property
hasAlternative is a subproperty of ecrm:P69_has_association_with
The domain of hasAlternative is class DecisionTransitionSchema
The range of hasAlternative is class DTAlternativeSchema
An instance of DecisionTransitionSchema is connected by hasAlternative to at least two instances of DTAlternativeSchema
An instance of DTAlternativeSchema is connected by the inverse of hasAlternative to exactly one instance of DecisionTransitionSchema

We now localise **hasTail** to decision transition schemas. A decision transition schema has a single tail, which is either a step schema or a transition schema.

An instance of DecisionTransitionSchema is connected by hasTail only to instances of class ActivitySchema
An instance of DecisionTransitionSchema is connected by hasTail to exactly one instance of class ActivitySchema

2.5.2.3 Merge transitions

A merge transition schema brings together two or more alternative paths. Consequently, it has two or more tails and exactly one head, which can be a step schema or a transition schema.

An instance of MergeTransitionSchema is connected by hasTail only to instances of class ActivitySchema
An instance of MergeTransitionSchema is connected by hasTail to at least two instances of class ActivitySchema

An instance of **MergeTransitionSchema** is connected by **hasHead** only to instances of class **ActivitySchema**

An instance of **MergeTransitionSchema** is connected by **hasHead** to exactly one instance of class **ActivitySchema**

2.5.2.4 Fork transitions

A fork transition has a single tail and to or more heads, similar to a decision transition, except that there are no predicates associated with heads because all the paths outcoming from a fork are supposed to be executed, in parallel. Consequently, the same structure is replicated for fork transition schemas. In contrast, to represent fork transitions, whether virtual or real, it is sufficient to properly localise properties **craeft:hasTail** and **craeft:hasHead**.

A fork transition schema has exactly one tail and two or more heads, which can be step schemas or transition schemas.

An instance of **ForkTransitionSchema** is connected by **hasTail** only to instances of class **ActivitySchema**

An instance of **ForkTransitionSchema** is connected by **hasTail** to exactly one instance of class **ActivitySchema**

An instance of **ForkTransitionSchema** is connected by **hasHead** only to instances of class **ActivitySchema**

An instance of **ForkTransitionSchema** is connected by **hasHead** to at least two instances of class **ActivitySchema**

A fork virtual transition has exactly one tail and two or more heads, which can be a virtual step or a virtual transition.

An instance of **VirtualForkTransition** is connected by **hasTail** only to instances of class **VirtualActivity**

An instance of **VirtualForkTransition** is connected by **hasTail** to exactly one instance of class **VirtualActivity**

An instance of **VirtualForkTransition** is connected by **hasHead** only to instances of class **VirtualActivity**

An instance of **VirtualForkTransition** is connected by **hasHead** to at least two instances of class **VirtualActivity**

A fork transition has exactly one tail and two or more heads, which can be steps or transitions.

An instance of **RealForkTransition** is connected by **hasTail** only to instances of class **RealActivity**

An instance of **RealForkTransition** is connected by **hasTail** to exactly one instance of class **RealActivity**

An instance of **RealForkTransition** is connected by **hasHead** only to instances of class **RealActivity**

An instance of **RealForkTransition** is connected by **hasHead** to at least two instances of class **RealActivity**

2.5.2.5 Join transitions

Join transitions are similar to merge transitions, they have at least two tails and one head, except that their tails do not represent alternatives but are supposed to be all present because they are parallel courses of action that the join synchronises. Consequently, the same structure is replicated at all levels, similar to fork transitions. To represent join transitions, it is sufficient to properly localise properties **craeft:hasTail** and **craeft:hasHead**.

A join transition schema has two or more tails and one head, which can be step schemas or transition schemas.

An instance of **JoinTransitionSchema** is connected by **hasTail** only to instances of class **ActivitySchema**

An instance of **JoinTransitionSchema** is connected by **hasTail** to at least two instances of class **ActivitySchema**

An instance of **JoinTransitionSchema** is connected by **hasHead** only to instances of class **ActivitySchema**

An instance of **JoinTransitionSchema** is connected by **hasHead** to exactly one instance of class **ActivitySchema**

A joint virtual transition has two or more tails and one head, which can be a virtual step or a virtual transition.

An instance of VirtualJoinTransition is connected by hasTail only to instances of class VirtualActivity
An instance of VirtualJoinTransition is connected by hasTail to at least two instances of class VirtualActivity
An instance of VirtualJoinTransition is connected by hasHead only to instances of class VirtualActivity
An instance of VirtualJoinTransition is connected by hasHead to exactly one instance of class VirtualActivity

A join transition has two or more tails and one head, which can be a step or a transition.

An instance of RealJoinTransition is connected by hasTail only to instances of class RealActivity
An instance of RealJoinTransition is connected by hasTail to at least two instances of class RealActivity
An instance of RealJoinTransition is connected by hasHead only to instances of class RealActivity
An instance of RealJoinTransition is connected by hasHead to exactly one instance of class RealActivity

2.5.3 Reusing step schemas

In representing activities, the CrO is based on the assumption that a step occurs only once in a process. For instance, the axiom that an instance of **RealJoinTransition** is connected by **hasHead** to exactly one instance of class **RealActivity**, reflects the fact the same real activity cannot be the head of more than one join transition. By applying the same axiom to all kinds of transitions, the ontology guarantees that the same activity does not occur more than once in a process.

This assumption is justified by the fact that, in real processes, an action or an event is a unique phenomenon, situated in time and space, and the same action or event cannot occur in two different situations. Indeed, the assumption descends from the basic laws of Newtonian physics, such as the one sanctioning the impossibility of locating the same object in two different places at the same time (as we all know things are different in quantum physics but crafts are not quantum phenomena).

The same applies to virtual steps. Even though a virtual action, being a mathematical representation of an action, is bound to be deterministic, (therefore any execution of the simulation that models the action will always yield the same results, once given the same input parameters), the action of running the simulation

is a real action and is therefore unique, even though the simulation *per sé* is a deterministic process that will always be the same. The same applies to virtual events.

The assumption does not hold for schemas, though, because schemas do not *happen*: they are descriptions of things that happen, but descriptions themselves do not happen. As a consequence, the possibility for an action or an event schema to occur several times within a process schema is not only real, but quite common: in a glass-blowing craft, the craftsperson may need to instil a bubble of air into the gob of glass at several points of the craft: upon gathering the glass from the furnace, in case they need to gather a large quantity, and before giving the desired shape to the glass in the mould, in case the glass does not have the required volume or density. The schema describing how to insulfate a bubble of air into the gob is the same and it is not only convenient but fundamental to represent the fact that the same schema is applied in both cases.

We face therefore the necessity of allowing a step schema to occur several times in a process schema. This is not allowed by the axioms we have given so far, as the ontology treats in the same way step schemas, virtual steps and real steps. One possible solution would be to drop those axioms for step schemas. But if we did so, the models admitted by the ontology would be far more than just those desired, as step schemas would enjoy total freedom. To solve the problem without compromising the integrity of the knowledge graphs consistent with the ontology, the CrO adopts the following strategy: it keeps the axioms sanctioning the uniqueness of step schemas in process schemas and introduces one property representing the fact that two schemas are different occurrences of the. Same schema. The former ensures the syntactic correctness of process schemas, while the latter allows representing the multiple occurrences of the same step schema in a process schema. Since the just described relation between step schemas is a sameness relation, it is an equivalence relation, thus reflexive, symmetric and transitive, and creates equivalence classes, each including the step schemas that are clones of the same schema. The property that the CrO uses to represent this relation is called **sameStepSchemaAs**. **sameStepSchemaAs** has **StepSchema** as domain and as range and is declared to be reflexive, symmetric and transitive.

sameStepSchemaAs is an object property
The domain of sameStepSchemaAs is class StepSchema
The range of sameStepSchemaAs is class StepSchema
sameStepSchemaAs is reflexive
sameStepSchemaAs is symmetric
sameStepSchemaAs is transitive

Let us now discuss the consequences of sameness axioms. We recall that a step schema can be an action, an event, or a process schema.

If two action schemas describe the same action then they share all their characteristics, namely: action function, action type, machine type, and all the properties for associating causing and affected entities to the instances of the schemas. Happily, OWL 2 DL is expressive enough to allow stating that two action schemas that are connected by the **sameStepSchemaAs** property share these characteristics. The expression is based on the appropriate complex role Inclusion axioms, given next:

The chain formed by sameStepSchemaAs and hasActionFunction is a subproperty of hasActionFunction
The chain formed by sameStepSchemaAs and hasActionType is a subproperty of hasActionType
The chain formed by sameStepSchemaAs and hasMachineType is a subproperty of hasMachineType
The chain formed by sameStepSchemaAs and hasCausingEntityProperty is a subproperty of hasCausingEntityProperty
The chain formed by sameStepSchemaAs and hasAffectedEntityProperty is a subproperty of hasAffectedEntityProperty
The chain formed by sameStepSchemaAs and hasAffectedEntityProperty is a subproperty of hasAffectedEntityProperty
The chain formed by sameStepSchemaAs and hasAffectedRigidStableEntityProperty is a subproperty of hasAffectedRigidStableEntityProperty
The chain f. by sameStepSchemaAs and hasAffectedRigidMovableEntityProperty is a subproperty of hasAffectedRigidMovableEntityProperty
The chain f. by sameStepSchemaAs and hasAffectedDeformableStableEntityProperty is a subproperty of hasAffectedDeformableStableEntityProperty
The chain f. by sameStepSchemaAs and hasAffectedDeformableMovableEntityProperty is a subproperty of hasAffectedDeformableMovableEntityProperty

The net effect of these axioms is to “copy” one action schema *S* into any action schema *S'* that is the same as *S*, including *S* itself, due to the reflexivity of **sameStepSchemaAs**. In practice, it suffices to specify one schema *S* and the sameness of any schema *S'* to *S* to have the property values of *S* copied into *S'*. As explained above, the copying mechanisms realised by the axioms above allow using different step schemas in process schemas, to guarantee the correctness of the process schema specifications, while avoiding duplicating the schemas.

Event schemas are characterised only by the duration property, which can be “copied” between the same event schemas using the following complex role inclusion axiom:

The chain formed by **sameStepSchemaAs** and **duration** is a subproperty of **duration**

The same treatment cannot be applied to process schemas. This is because a process schema consists of transition and step schemas, and “copying” such schemas in an axiomatic way means producing axioms that have the copy transition and step schemas as a consequence. OWL 2 DL does not possess the expressivity required for that, so the CrO allows to state sameness between process schemas but does not compute the consequences of such a statement. This would have to be done programmatically, much in the same way instantiation, that is, the generation of a virtual process from a process schema is done. Instantiation is discussed in the next Section.

2.6 Linking to standard dictionaries

To provide the widest interoperability of the KB built by the CRAEFT project, the CrO provides classes and properties for linking to prominent semantic dictionaries. The considered dictionaries are *de facto* standards in CH; they are the Getty Art & Architecture Thesaurus (AAT), the Catalog of Art Collections (CONA), the Thesaurus of Geographic Names (TGN), and the Union List of Artist Names (ULAN).

In addition to widening the interoperability of the CRAEFT KB, linking to these dictionaries increases the system's capacity for precise and comprehensive documentation and eases data entry, by avoiding repeating the entry of information that already exists online.

The approach followed to link to these dictionaries is very simple. For every dictionary, the ontology provides:

- A specific class, a subclass of **ecrm:E55_Type**, including as instances the terms of the dictionary; the class is named **NNNTerm**, where **NNN** is a (popular) name of the dictionary, *e.g.*, **AAT**;
- A specific property, subproperty of **ecrm:P2_has_type**, for linking any individual to a term from the dictionary. The property is named **hasNNNTerm**, where **NNN** is as above. The domain of this property is the most specific CrO class that includes the individuals addressed by the dictionary, while its range is the class **NNNTerm**.

As can be seen, the approach is quite general and can be easily applied to other similar resources. Each dictionary is considered in the remaining subsections of this section. As this extraction of information from the aforementioned vocabularies may be useful to others, we provide their source code that implements it here: <https://zenodo.org/records/10532597>.

2.6.1.1 AAT

The AAT includes generic terms, and associated dates, relationships, and other information about concepts related to or required to catalogue, discover, and retrieve information about art, architecture, and other visual cultural heritage, including related disciplines dealing with visual works, such as archaeology and conservation, where the works are of the type collected by art museums and repositories for visual cultural heritage, or that are architecture.

The inclusion of Getty AAT was to enhance the system's vocabulary related to art, architecture, and material culture. The AAT dictionary is the basic dictionary used by the CRAEFT Authoring Platform, illustrated below. The class including the Getty terms is **GettyTerm**. The property for linking to the AAT is **hasGettyTerm**. Every knowledge entity in the system has an annotation from AAT, thus the domain of **hasGettyTerm** is left unspecified for generality.

	GettyTerm is a class
	GettyTerm is a subclass of ecrm:E55_Type
	hasGettyTerm is an object property
	hasGettyTerm is a subproperty of ecrm:P2_has_type
	The range of hasGettyTerm is class GettyTerm

The specification of the Getty term is ensured and automated by adding the pertinent annotation by default to each instantiated knowledge entity, at the data input form. Naturally, this default semantic annotation is generic and it is left to the user to specialise it, if needed and if known. The default annotations are provided in the table below. We recall that in the craft ontology, all digital assets are media objects, while they may differ in type (image, video, 3D).

Media object (image)	http://semantics.gr/authorities/digital-item-types/1332557240 - Born-digital photograph
Media object (video)	http://semantics.gr/authorities/digital-item-types/346883033 - Born-digital video
Media object (3D model)	http://vocab.getty.edu/aat/300411661 - virtual models
Person	http://vocab.getty.edu/aat/300024979 - people (agents)
Social groups	http://vocab.getty.edu/aat/300025948 - organizations (groups)
Location	http://vocab.getty.edu/aat/300248479 - location (physical position)
Tool	http://vocab.getty.edu/aat/300122241 - tools
Product	http://vocab.getty.edu/aat/300387427 - products

Event	http://vocab.getty.edu/aat/300069084 - events (activities) (Events (hierarchy name))
Process	http://vocab.getty.edu/aat/300138076 - processes

When the user documents any of these entities the appropriate annotation is by default entered. The user may specialise these annotations according to the available knowledge about the knowledge entity. The example below demonstrates such a specialisation, for a handcrafted musical instrument (a lute).

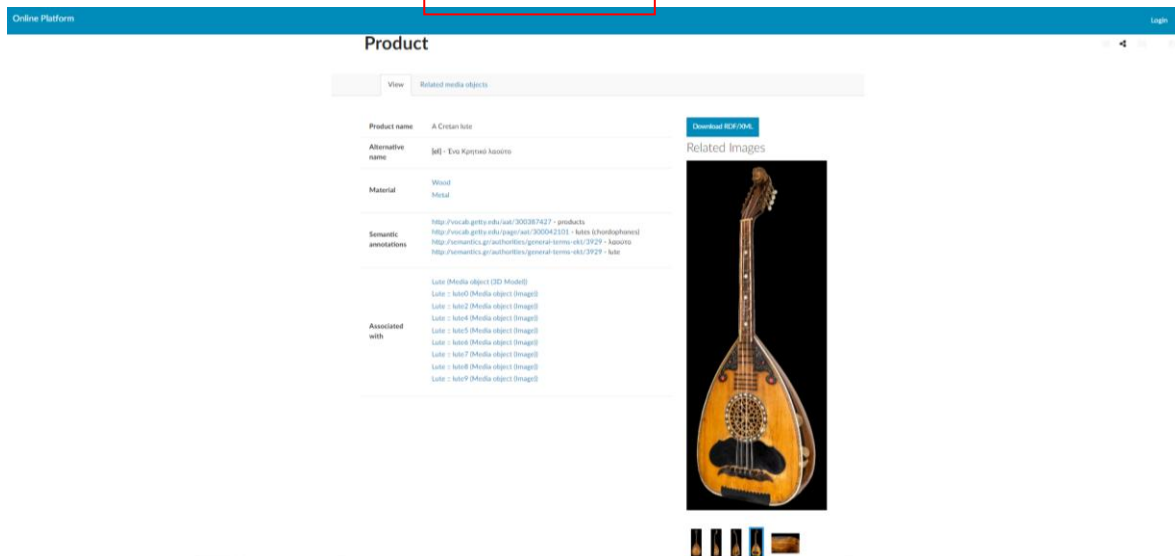


Figure 14. The page for a craft product with semantic annotations.

In the example above, the following semantic annotations have been added by the user to the default one.

<http://vocab.getty.edu/page/aat/300042101> - lutes (chordophones)

<http://semantics.gr/authorities/general-terms-ekt/3929> - λαούτο

<http://semantics.gr/authorities/general-terms-ekt/3929> - lute

Specifically, the term “lutes (chordophones)” specialises in the product type. In addition, the dictionary of the Greek National Aggregator has been used, so that the digital assets can be ingested in Europeana. This last semantic annotation has been added in both the Greek and the English languages.

It ought to be noted that although the AAT dictionary is more often used to document tangible heritage, we found it very useful for the documentation of intangible heritage as well. Verbs are equally well represented in the AAT and their hierarchical structure is quite useful in the semantic characterisation of actions and processes. We provide two examples below.

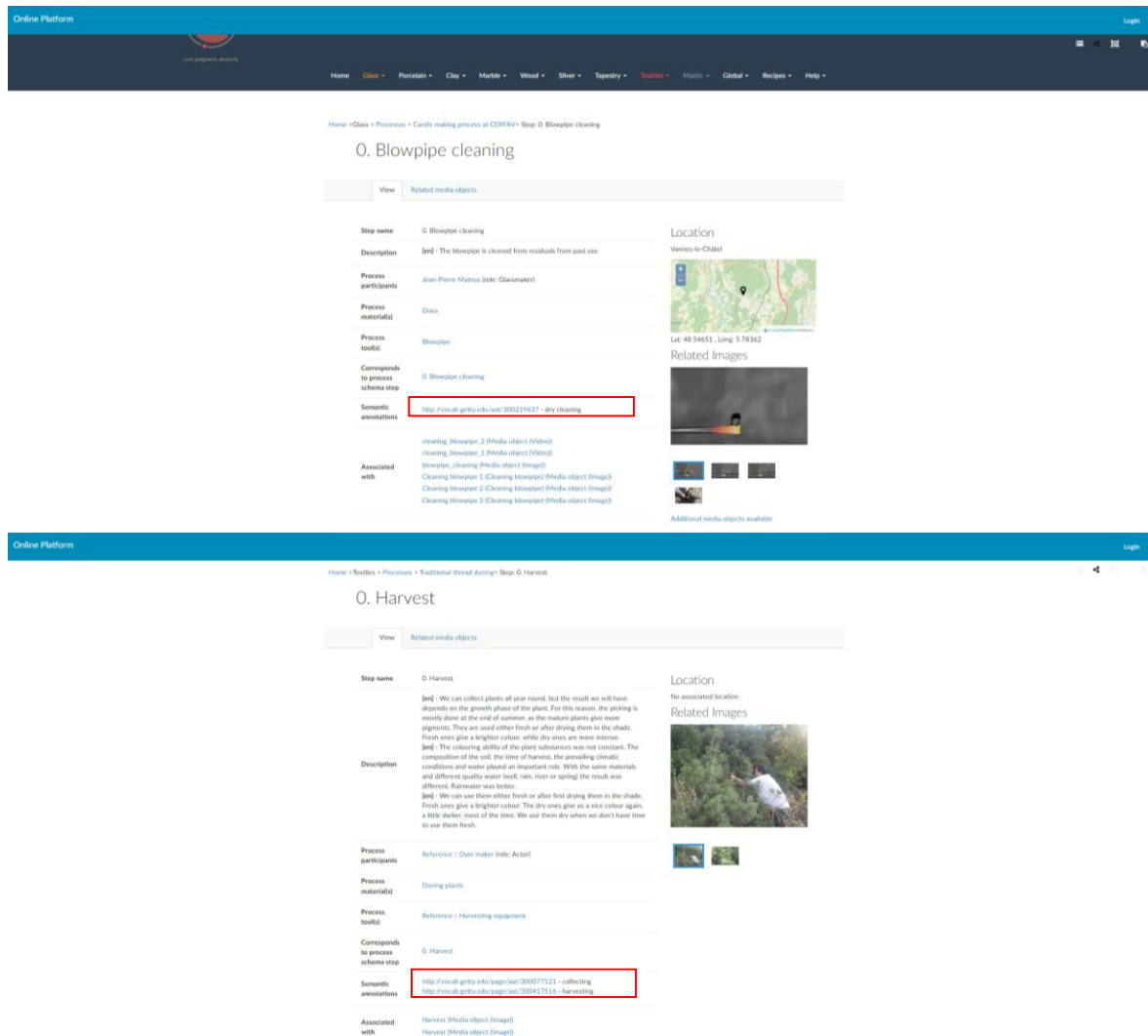


Figure 15. Two pages for craft process steps, with semantic annotations.

The examples show two process steps, one from glasswork and one from textile manufacturing. The first example regards the cleaning of a blowpipe, which is annotated as

<http://vocab.getty.edu/aat/300219637> - dry cleaning

since the cleaning takes place by scrubbing the blowpipe, without the use of water or any other cleaning substance. The second example regards the collection of plants for dyeing threads with natural ingredients and is annotated with two terms:

<http://vocab.getty.edu/page/aat/300077121> - collecting

<http://vocab.getty.edu/page/aat/300417516> - harvesting

to indicate the collection process by the human agent, as well as the fact that the collected items are plant “products” that grew over the year (the corresponding process schema indicates the appropriate time of the year for this harvesting).

Although we initially added the AAT annotations to be able to expose our digital assets for ingestion to Europeana, we noticed an added value advantage. The thesaurus structure of the AAT dictionary enables the discovery of very specific terms that may not necessarily be known to the user. The addition of the AAT annotations enables, thus, users to access a broader range of standardised terminology for documenting the represented knowledge.

2.6.1.2 CONA

The incorporation of CONA extends the system's capability to include detailed information on art collections, helping users connect and explore diverse art collections worldwide.

CONA compiles titles, attributions, depicted subjects, and other metadata about works of art, architecture, and cultural heritage, both extant and historical, physical and conceptual. Metadata is gathered and linked from museum collections, special collections, archives, libraries, scholarly research, and other sources.

CONA is focused on “unique” artworks, or better, artworks with individual names each that have been catalogued by museums, libraries, or regional authorities (i.e. for monuments, buildings, etc). For this reason, the domain of the property is class **ecrm:E22_Human-Made_Object**.

	CONATerm is a class
	CONATerm is a subclass of ecrm:E55_Type
	hasCONATerm is an object property
	hasCONATerm is a subproperty of ecrm:P2_has_type
	The domain of hasCONATerm is class ecrm:E22_Human-Made_Object
	The range of hasCONATerm is class CONATerm

CONA would seem rather irrelevant for craft artefacts. The reason is that although each craft item is unique, it rarely has a specific name, as opposed to a painting or statue. However, we found it very useful and we are using it for the following two cases. First, when a crafted artefact references another known artefact. Second, when a crafted artefact references a known event. We provide an example below, coming from a collection of handcrafted garments whose design was inspired by antiquities.

The example regards two items that were inspired by the same archaeological finding, that is the “Small snake goddess figurine”, <http://vocab.getty.edu/page/cona/700000112>.

Online Platform

Product name

Branding Heritage :: The Snake Goddess Category: Embroidered garment 2

Alternative name

[el] - Branding Heritage :: Η θεά των φιδιών: Κεντημένο ένδυμα 2

Description

[en] - Reason for Creation: Marios Schwab AW 2011 Collection Object of Inspiration: The "Snake Goddess", Heraklion Archaeological Museum Way of Acquisition: Donation to the organisation by the creator in 2019 Collection Number: BH/CM/18

[el] - Άντς Δημιουργίας: Μarios Schwab Fall 2011 Ready-to-Wear (LOOK 6) Αρτυρίσµατο Έμπνευση: Η θεά των φιδιών: Αρχαιολογικό Μουσείο Ηρακλείου Τρόπος Αποκτήσεως: Δωρεά στον οργανισμό από τον δημιουργό το 2019 Νόµισµα Σαλβολής: BH/CM/20

Material

Silk
Cotton

Creation information

Creation of Branding Heritage :: Embroidered garment

Creator

Marios Schwab

Semantic annotations

<http://vocab.getty.edu/aat/300387427-products>
<http://semantics.gr/authorities/ekt-item-types/ndyma-Ενδυµασία>
<http://vocabularies.unesco.org/thesaurus/concept12328-Textile industry>
[http://vocab.getty.edu/aat/300046159-dresses \(garments\)](http://vocab.getty.edu/aat/300046159-dresses (garments))
<http://vocab.getty.edu/aat/300020224-Minoan>
http://semantics.gr/authorities/thematic_tags/994297606-Mινωαϊκό πολιτισµός
<http://semantics.gr/authorities/ekt-item-types/ndyma-Ενδυµασία>
<http://vocab.getty.edu/page/cona/700000112-Small snake goddess figurine>

Download RDF/XML

Related Images

Additional media objects available

Location

Athens

Lat: 37.98376 , Long: 23.72784

Online Platform

Product name

Branding Heritage :: Minoan mosaic dress

Alternative name

[el] - Branding Heritage :: Μινωαϊκό µωσαϊκό φόρεµα

Description

[en] - Clay model of a dress found with the Snake Goddess

[el] - Πηλινό µοντέλο φορέµατος που βρέθηκε µε τη θεά του φιδιού

Material

Clay

Creation information

Creation of Branding Heritage::Minoan mosaic dress

Creator

Loukia Orfanou

Destruction information

N/A

Semantic annotations

<http://vocab.getty.edu/aat/300387427-products>
http://semantics.gr/authorities/thematic_tags/994297606-Mινωαϊκό πολιτισµός
<http://vocab.getty.edu/aat/300020224-Minoan>
<http://semantics.gr/authorities/ekt-item-types/ndyma-Ενδυµασία>
[http://vocab.getty.edu/aat/300046159-dresses \(garments\)](http://vocab.getty.edu/aat/300046159-dresses (garments))
<http://vocabularies.unesco.org/thesaurus/concept12328-Textile industry>
<http://semantics.gr/authorities/ekt-item-types/ndyma-Ενδυµασία>
<http://vocab.getty.edu/page/cona/700000112-Small snake goddess figurine>

Download RDF/XML

Related Images

Additional media objects available

Location

Athens

Lat: 37.98376 , Long: 23.72784

Figure 16. Two pages for two craft artefacts inspired by another artefact that is registered in CONA.

The CONA dictionary provides information on the particular archaeological artefact, thus relieving the user from documenting that as well.

2.6.1.3 TGN and Geonames

Integration of the TGN and Geonames dictionaries provides geospatial context, enriching the documentation system with standardised geographic names for locations relevant to cultural heritage. Thus the domain of both properties **hasTGNTerm** and **hasGeonamesTerm** is class **ecrm:E53_Place**.

	TGNTerm is a class
	TGNTerm is a subclass of ecrm:E55_Type
	hasTGNTerm is an object property
	hasTGNTerm is a subproperty of ecrm:P2_has_type
	The domain of hasTGNTerm is class ecrm:E53_Place
	The range of hasTGNTerm is class TGNTerm
	GeonamesTerm is a class
	GeonamesTerm is a subclass of ecrm:E55_Type
	hasGeonamesTerm is an object property
	hasGeonamesTerm is a subproperty of ecrm:P2_has_type
	The domain of hasGeonamesTerm is class ecrm:E53_Place
	The range of hasGeonamesTerm is class GeonamesTerm

The Geonames dictionary is more comprehensive and detailed than TGN. We have been using it since early versions of the Mingei Online Platform, in the Mingei project. In this version, we have automated the extraction of location coordinates (GPS) from the Geonames service, relieving the user from the task of entering this data, while at the same time, protecting system integrity from human errors.

Although we retrieved location coordinates from the Geonames dictionary, we added the TGN dictionary as well. The reason is that TGN offers descriptions of the locations that provide historical information.

The example below shows the entry for a city (Athens). As the city is referenced in both dictionaries, both of the annotations are included:

<https://sws.geonames.org/264371/> - Athens

<http://vocab.getty.edu/tgn/7001393> - Athens

However, in the case of less known locations, the Geonames directory is much more comprehensive and includes many more locations.

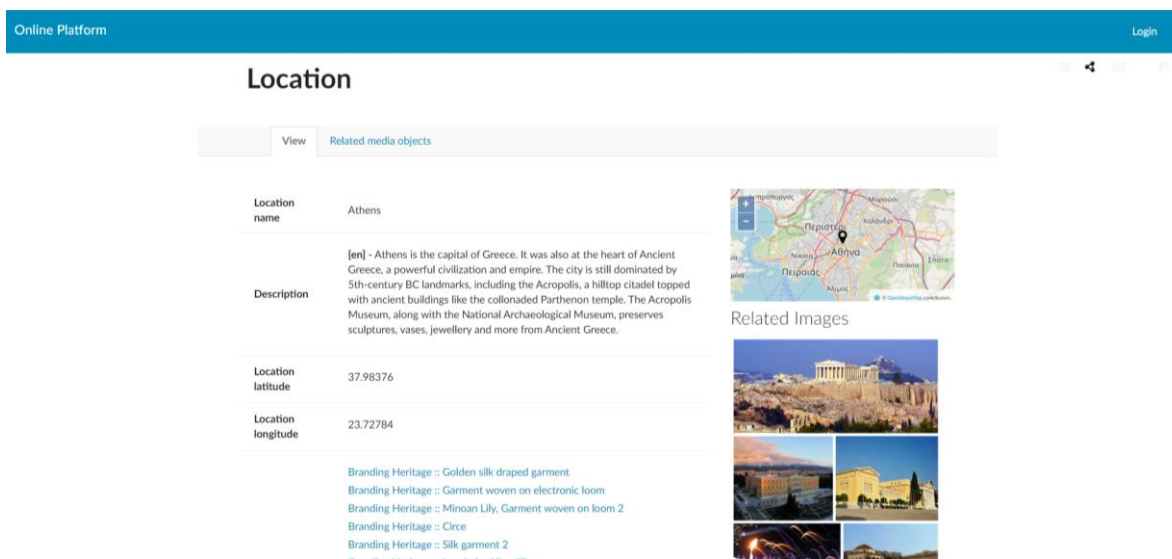


Figure 17. A page for a location knowledge entity, geosemantically annotated with Geonames and TGN labels.

2.6.1.4 ULAN

Linking to ULAN facilitates the identification and documentation of artists associated with cultural artefacts, ensuring a comprehensive understanding of the individuals behind the creations.

The integration with ULAN serves two purposes. The first is the comprehensive documentation of the represented knowledge pertinent to artisan names, used either in the documentation of artefacts or in the representation of narratives. The second is the automation of the completion of biographical information about the referenced persons. Specifically, we automated the retrieval of the following attributes from the ULAB service.

- Birth / Death dates.
- Alternative names
- Nationality
- Role(s)

	ULANTerm is a class
	ULANTerm is a subclass of <code>ecrm:E55_Type</code>
	hasULANTerm is an object property
	hasULANTerm is a subproperty of <code>ecrm:P2_has_type</code>
	The domain of <code>hasULANTerm</code> is class <code>ecrm:E21_Person</code>

The range of **hasULANTerm** is class **ULANTerm**

The example below regards a wood carving artist who created wooden sculptures for ecclesiastical altars, named Francisco Salzillo.

Online Platform
Login

Person

View
Related media objects


Person name	Francisco Salzillo	Related Images 
Alternative name	Alcaraz, Francisco Salzillo y Alcaraz, Francisco Zarcillo y Salzillo y Alcaraz, Francisco Salzillo, Francisco Sancillo y Alcaraz, Francisco Zarcillo y Alcaraz, Francisco Zarcillo y Alcaraz, Francisco Francisco Salzillo y Alcaraz	
Birth information	Birth of Francisco Salzillo	
Death information	Death of Francisco Salzillo	
Nationality	Spanish	
Person parent	Nicolas Salzillo	
Occupation information	Occupation of Francisco Salzillo	
Related Events	WCY :: Historical Tradition in Wood Carving WCY :: 1b :: Francisco Salzillo's Craftsmanship Creation of The Salzillo Nativity Scene Creation of The Last Supper Occupation of Francisco Salzillo	
Semantic annotations	http://vocab.getty.edu/aat/300024979 - people (agents) http://vocab.getty.edu/aat/300025181 - sculptors http://vocab.getty.edu/aat/300025368 - woodworkers http://vocab.getty.edu/aat/300025382 - woodcarvers (woodworkers) http://vocab.getty.edu/ulan/500035899 - Salzillo y Alcaraz, Francisco http://vocab.getty.edu/aat/300435120 - altarpiece sculptors	
Associated with	Francisco Salzillo (Media object (Image))	

Figure 18. The page for a wood sculptor (person), is semantically annotated with a ULAN label.

Using the ULAN entry for this person,

<http://vocab.getty.edu/ulan/500035899> - Salzillo y Alcaraz, Francisco

The aforementioned biographical information has been retrieved and has been entered into the system. We notice that several alternative names have been retrieved. Moreover, the AAT dictionary has been used to characterise the type of art produced by this person as follows:

<http://vocab.getty.edu/aat/300025181> - sculptors

<http://vocab.getty.edu/aat/300025368> - woodworkers

<http://vocab.getty.edu/aat/300025382> - woodcarvers (woodworkers)

<http://vocab.getty.edu/aat/300435120> - altarpiece sculptors

2.6.1.5 Additional dictionaries

Support is also provided for two more dictionaries, namely the UNESCO thesaurus and the dictionary of the Greek National Aggregator. No automatic retrieval of information is foreseen for these dictionaries, as the dictionaries mentioned earlier cover our semantic annotation needs. However, the UNESCO dictionary is required for ingestion by all National Aggregators and the SearchCulture.gr dictionary is required by the Greek Aggregator.

	UnescoThesaurusTerm is a class
	UnescoThesaurusTerm is a subclass of ecrm:E55_Type
	hasUnescoThesaurusTerm is an object property
	hasUnescoThesaurusTerm is a subproperty of ecrm:P2_has_type
	The range of hasUnescoThesaurusTerm is class UnescoThesaurusTerm
	UnescoThesaurusTerm is a class
	GNATerm is a subclass of ecrm:E55_Type
	hasGNATerm is an object property
	hasGNATerm is a subproperty of ecrm:P2_has_type
	The range of hasGNATerm is class GNATerm

In Craeft, we plan to submit all of the digital assets collected through the Greek National Aggregator, as it participated in the Europeana project “CRAFTED”⁹ and welcomes digital assets pertinent to crafts, regardless of their national provenance. For this reason, the Greek National Aggregator provides also the “craft-item-types”¹⁰ sub-vocabulary which we also utilise. It ought to be noted, that when using national, non-English, vocabularies we use the annotation twice with the same URL: once for the national label and once for the English label. The example below demonstrates this case.

⁹ <https://pro.europeana.eu/project/crafted>

¹⁰ <http://semantics.gr/authorities/vocabularies/craft-item-types>

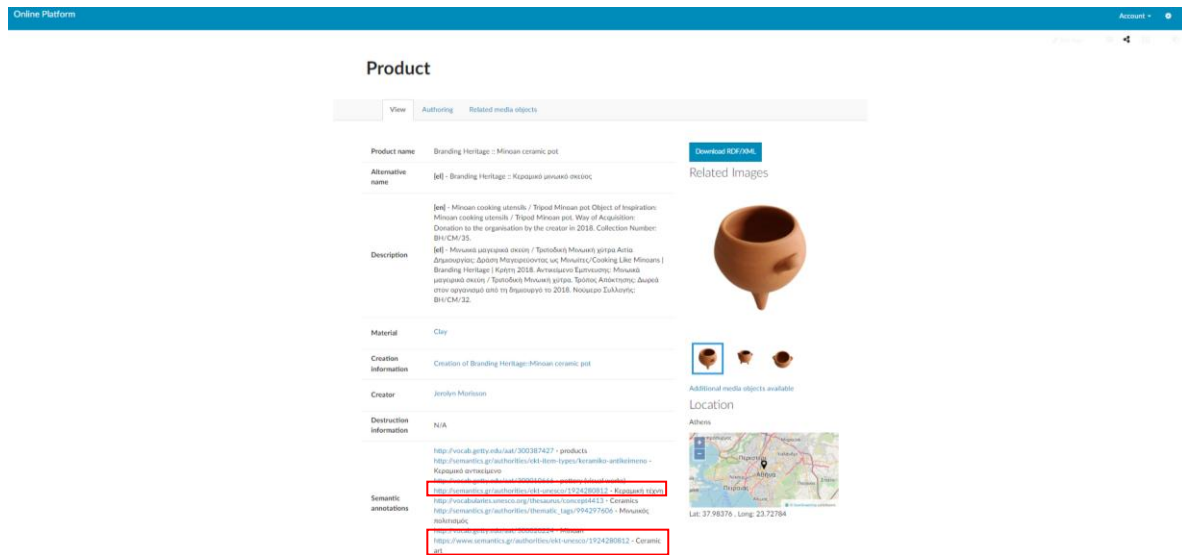


Figure 19. The page for a craft artefact, annotated with English and Greek metadata.

In the example, a semantic annotation is noted twice, that is

<http://semantics.gr/authorities/ekt-unesco/1924280812> - Κεραμική τέχνη

and

<http://semantics.gr/authorities/ekt-unesco/1924280812> - Ceramic art

comes from a national vocabulary that provides associations with the UNESCO vocabulary. Each label is associated with the corresponding language tag.

Finally, we note that the system is open to any vocabulary the user wishes to include. For example, The Library of Congress vocabulary can be used, if the aforementioned libraries do not suffice. The example below demonstrates such a case.

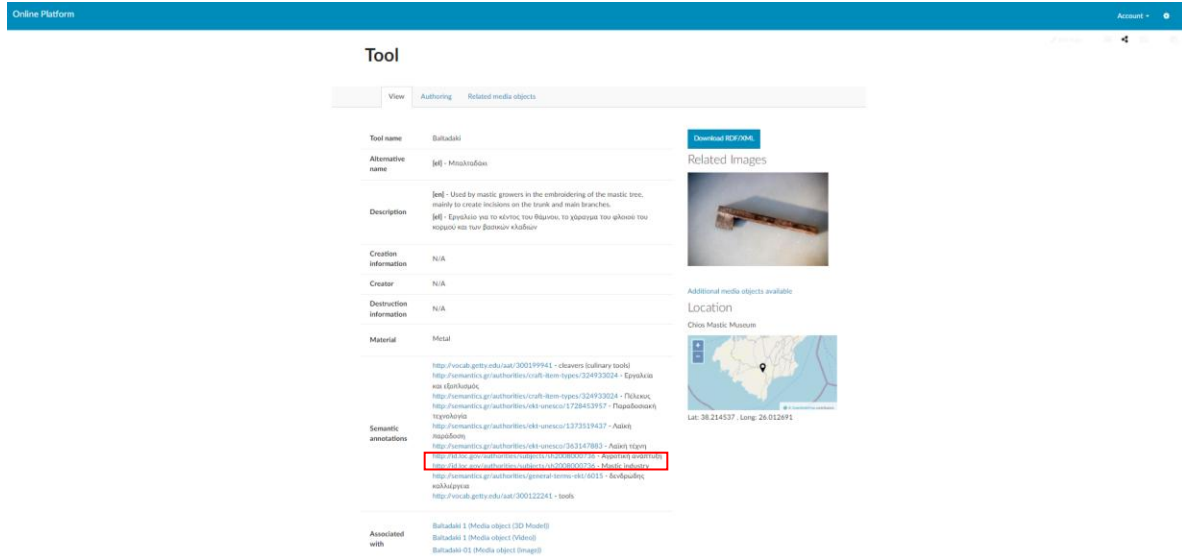


Figure 20. The page for a craft tool, with semantic annotations including annotations from the Library of Congress.

In the example, the term the vocabulary item

<http://id.loc.gov/authorities/subjects/sh2008000736> - Mastic industry

is used to enrich the contextualisation of the shown tool, as the specific term is not found in the aforementioned vocabularies.

When exporting assets for ingestion to Europeana the annotations of the additional dictionaries are also included.

2.6.1.6 Implementation

The implementation utilises the APIs provided by Getty to establish seamless connections with AAT, CONA, TGN, and ULAN. We considered two approaches to the implementation of data retrieval mechanisms to fetch and update information from these dictionaries.

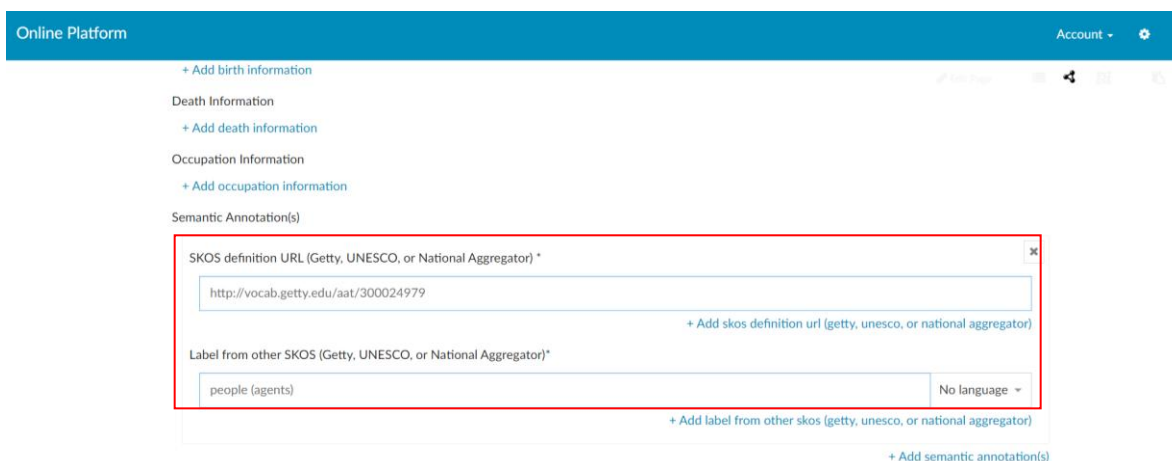
The offline scenario. In this usage scenario, the user copies the URL of the dictionary term to the system. A system process (a script) is run periodically in the background that retrieves the data from the dictionary and completes it offline.

The online scenario. In this usage scenario, we considered either a priori loading of the dictionaries or dynamically loading the parts of the dictionaries, in an autocomplete fashion.

We have technically reviewed prototypes of both implementation scenarios and selected the offline scenario. The reason is that in the online scenario, the following disadvantages are encountered.

- Copying the entire dictionary locally costs storage space and since this is an online system, energy and maintenance costs. Additionally, as these dictionaries are updated, we would have to perform this copy periodically.
- Retrieving parts of the dictionary dynamically, either in an autocomplete function or in creating appropriate UI items (e.g. popup menus) is dependent on the speed of the network connection. We found that the wait for this retrieval during data entry could be more friendly for the user.

The user interface implementation is shown in the figure below.



Online Platform Account -

+ Add birth information

Death Information

+ Add death information

Occupation Information

+ Add occupation information

Semantic Annotation(s)

SKOS definition URL (Getty, UNESCO, or National Aggregator) *

http://vocab.getty.edu/aat/300024979

+ Add skos definition url (getty, unesco, or national aggregator)

Label from other SKOS (Getty, UNESCO, or National Aggregator) *

people (agents) No language

+ Add label from other skos (getty, unesco, or national aggregator)

+ Add semantic annotation(s)

Figure 21. The GUI facility for default semantic annotations.

The example shows the default semantic annotations provided by the system, for the case of a person. The modification of the user interface accommodates the new data fields and presents information from the integrated dictionaries in a user-friendly manner.

3 The CRAEFT Authoring Platform

The Craeft Authoring Platform (CAP) is a Web-based, online, multiple-user authoring platform for the documentation of traditional crafts that is under development. It is functional and accessible online, although it is not yet in its final form. The CAP is based on the Mingei Online Platform (MOP) developed in the Mingei Innovation Action.

Authoritative reports on crafts and conservation [1, 2] identify tangible and intangible craft dimensions. Tangible dimensions regard materials, tools, and workspaces. Intangible dimensions refer to know-how and skill, but also collective memories, values, and traditions. We refer to contextual and crafting intangible dimensions. Contextual dimensions refer to the social and historical context. Crafting dimensions refer to the knowledge and judgement of the practitioner in handicraft activities.

We briefly review the MOP to provide context on the upgrades that are reported in the remainder of this deliverable.

The MOP represents traditional craft instances based on two axes:

1. Craft processes. The representation is based on the identification and digital representation of pertinent data, information, and knowledge that enable the understanding and reenactment of traditional craft processes.
2. The social and historic context of craft instances. The representation is based on documented narratives that support the presentation of social and historical context to diverse audiences.

The MOP is based on a systematic method for craft representation, the adoption, knowledge, and representation standards of the cultural heritage (CH) domain, and the integration of outcomes from advanced digitization techniques. More specifically, the MOP follows and implements the Mingei Craft Representation Protocol [3]. A handbook accompanies the MOP elaborating on good practices for its use [4].

The MOP has been documented in the following publications:

1. *A Web-Based Platform for Traditional Craft Documentation. Multimodal Technologies and Interaction*. 2022; 6(5):37, Partarakis N, Doulgeraki V, Karuzaki E, Galanakis G, Zabulis X, Meghini C, Bartalesi V, Metilli D. This publication presents the technical architecture and GUI of the MOP.
2. *Representation of socio-historical context to support the authoring and presentation of multimodal narratives: The Mingei Online Platform, (2021)*, N. Partarakis, P. Doulgeraki, E. Karuzaki, I. Adami, S. Ntoa, D. Metilli, V. Bartalesi, C. Meghini, Y. Marketakis, M. Theodoridou, D. Kaplanidi, X. Zabulis, *ACM Journal on Computing and Cultural Heritage*, DOI:10.1145/3465556. This publication presents the way that contextualisation narratives are represented in the MOP.
3. *Digitisation of traditional craft processes*, X. Zabulis, C. Meghini, A. Dubois, P. Doulgeraki, N. Partarakis, I. Adami, E. Karuzaki, A. Carre, N. Patsiouras, D. Kaplanidi, D. Metilli, V. Bartalesi, C. Ringas, E. Tasiopoulou, Z. Stefanidi, *ACM Journal on Computing and Cultural Heritage*, DOI:10.1145/3494675. This publication presents the way that traditional craft processes are represented in the MOP.

4. *The Mingei Handbook*, X. Zabulis, N. Partarakis, A. Argyros, A. Tsoli, A. Qammaz, I. Adami, P. Doulgeraki, E. Karuzaki, A. Chatziantoniou, N. Patsiouras, E. Stefanidi, Z. Stefanidi, A. Rigaki, M. Doulgeraki, A. Patakos, S. Manitsaris, A. Glushkova, B. Olivas-Padilla, D. Menychtas, D. Makrygiannis, C. Meghini, V. Bartalesi, D. Metilli, N. Magnenat-Thalmann, E. Bakas, N. Cadi, D. van Dijk, P. de Sterke, M. Wippoo, M. van der Vaart, C. Ringas, M. Fasoula, E. Tasiopoulou, D. Kaplanidi, L. Pannese, V. Nitti, C. Cuenca, A. Carre, A. Dubois, H. Hauser, C. Beisswenger, D. Blatt, I. Neumann, U. Denter. DOI:10.5281/zenodo.6580124. This handbook is a guide to good practices for representing crafts in the MOP.

As the CAP is based on the MOP, we henceforth refer to it as MOP/CAP. Moreover, because the MOP has already users and a “brand name” while its URL is mentioned in the literature, we have retained the same WWW address for the platform, which is <http://mop.mingei-project.eu>.

3.1 Representation of knowledge in the MOP/CAP

In MOP, knowledge is represented using the conceptualisation provided by an ontology, the Mingei Crafts Ontology (CrO) [5]. The ontology provides a vocabulary and axioms to align the vocabulary terms with the conceptualization. The ontology harmonizes in a coherent vision multiple sub-domain ontologies, re-using solid results in knowledge representation that have now become standards, such as (a) ‘Narrative’ modelling, based on an extension of the CIDOC-CRM [6, 7] with narratological concepts; (b) time, based on the OWL time ontology [8]; (c) content representation, based on RDF; and (d) 4D-fluents for the representation of time-varying properties. Also, we have designed the required mappings between CrO and Europeana Data Model (EDM). This will allow us to link particular instances of CrO with Europeana resources, enabling, therefore, the validation as well as the enrichment of resources and the ingestion of the latter in Europeana. Furthermore, the implementation of the ontology is based on standards: the Web architecture for identifying, storing and retrieving the basic resources using Internationalized Resource Identifiers (IRIs), whether media objects, formal concepts or individuals; RDF as the basic data model for knowledge; OWL as ontology Web language; and SPARQL as a knowledge extraction language.

3.2 User Interface

A set of GUI components enables the instantiation of knowledge entities, facilitating the entry of attribute data and the association with recordings that document them.

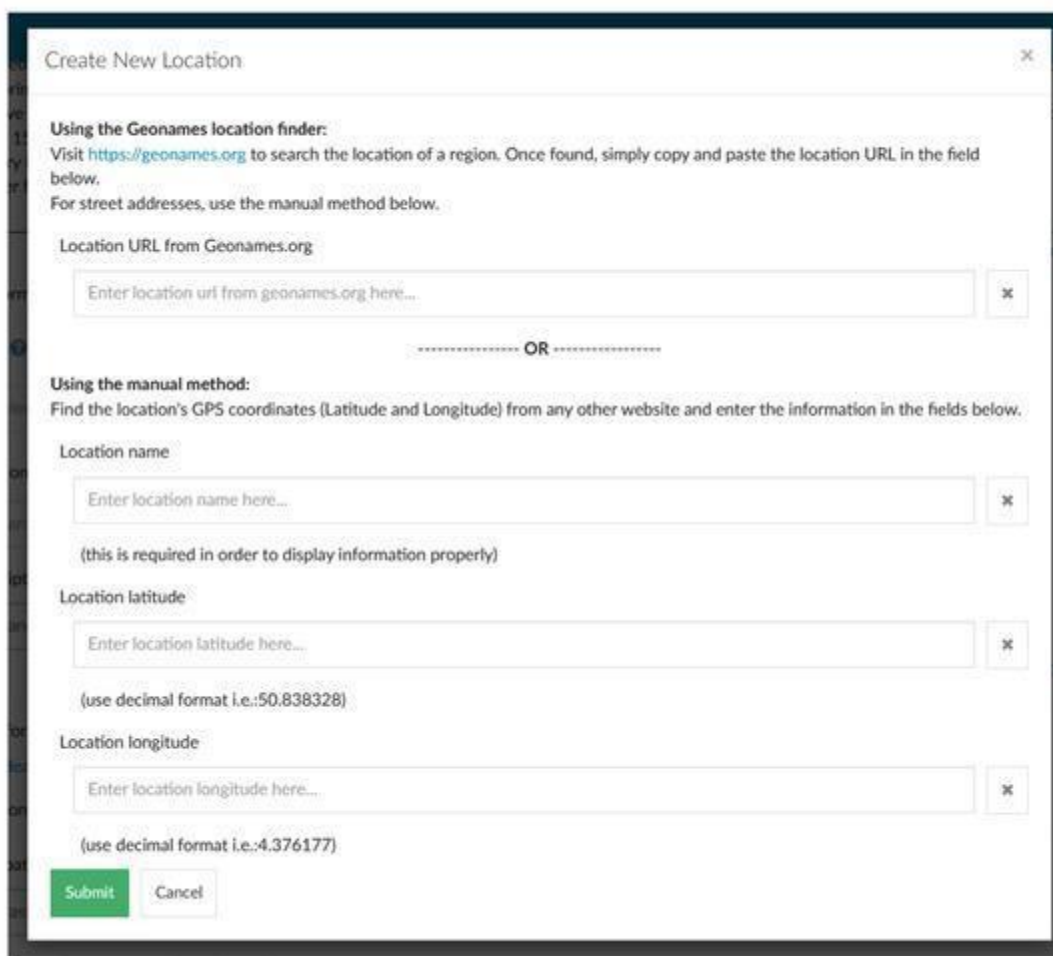
The formulation of basic data entries is systematised through an authoring environment that builds on top of an Ontology that adheres to knowledge representation standards in Cultural Heritage [9] and supports the representation of knowledge about ‘Persons’, ‘Social Groups’, ‘Places’, ‘Objects’, and related ‘Media Objects’. This facilitates the data curators in transforming verbal and visual content into data entries. Furthermore, in MOP, a user-friendly user interface for the data curators is offered for integrating digitisation results produced by modern digital media and digital capturing technologies including Motion Capture (MoCap) and 2D and 3D digitisation, thus enhancing their representation capacity.

The front end was implemented using the Research Space¹¹ (RS) toolkit, which provides HTML5 semantic components for structuring Web authoring forms, template pages, navigation menus, content panels, and

¹¹ <https://researchspace.org/>

other interaction and ‘Presentation’ elements (i.e., buttons, searches, drop-downs, table grids, etc.). It also provides ‘Presentation’ features such as interactive maps, a timeline component for visualising chronologically ordered events, and various image gallery components. The RS toolkit facilitated rapid prototyping in the first design iteration. Targeted design prototypes were produced thereafter, to visualize suggested design solutions and improvements stemming from the results of the design iterations.

UI templates: The ontology provides the semantics of the knowledge representation employed by the RS toolkit. For example, a representation of a particular location can be associated with the ontology (model) as being of type ‘Location’. In this context using the RS toolkit UI templates have been created to define generic views that are being automatically applied to entire sets of instances. An example of a data entry form for locations is shown in the figure below.



Create New Location [X]

Using the Geonames location finder:
Visit <https://geonames.org> to search the location of a region. Once found, simply copy and paste the location URL in the field below.
For street addresses, use the manual method below.

Location URL from Geonames.org
[Enter location url from geonames.org here...] [X]

***** OR *****

Using the manual method:
Find the location's GPS coordinates (Latitude and Longitude) from any other website and enter the information in the fields below.

Location name
[Enter location name here...] [X]
(this is required in order to display information properly)

Location latitude
[Enter location latitude here...] [X]
(use decimal format i.e.:50.838328)

Location longitude
[Enter location longitude here...] [X]
(use decimal format i.e.:4.376177)

[Submit] [Cancel]

Figure 34. Data entry form for a basic knowledge element (Location).

Application pages: For the ‘Presentation’ of a collection of knowledge such as, for example, the visualisation of a ‘Presentation’ of a ‘Narration’, application pages are used. These are pages that are not associated with any entity in the knowledge graph. Using application pages functionality that goes beyond associations with entities can be built. In application pages, the markup is bound with knowledge from the ontology. For application pages, HTML5 semantic components are used. The components are custom

HTML5 components that operate on the result of SPARQL queries executed over the knowledge graph. HTML5 components allow formatting and structuring the content of application pages and templates providing functionality beyond that of native HTML mark-up.

A human-comprehensible way to present the represented knowledge network is hypertext. The implementation employs a Web interface that dynamically generates Hypertext Markup Language (HTML) pages from the knowledge queries and a Web server that transmits them to the Web client (browser). An individual documentation page is provided for each entity. Semantic links are implemented as hyperlinks that lead to the pages of cited entities. This way, browsing and exploration of knowledge through semantic associations are enabled. Contents can be organised and presented spatiotemporally or thematically. A keyword-based search is provided.

Documentation pages for media objects contain links to digital assets, textual presentation of metadata, and previews of the associated digital assets. One or more URLs are provided on each page. For media objects, these links point to the source data files. For knowledge entities, the link points to the RDF encoding of that entity. For locations and events, specific UI modules are provided. For locations, embedded, dynamic maps are provided through OpenStreetMap [10]. Timeline and calendar views are available for events. The figure below, shown is an example of a data entry form for the presentation of a narrative (left) and the presentation output as seen by the user (right).

Presentation: The cultural heritage of mastic

Web template

View Authoring Narrative preview

Tips

- Always click the 'Save' button before leaving this form.
- A presentation can be broken down into multiple segments, i.e. content sections, by clicking on the respective link button.

Presentation name*

The cultural heritage of mastic

Description

Enter description here...

+ Add description

Related media object

Search or create a media

+ Create new

+ Add related media object

Presentation Segment

Presentation segment name*

Patron saint of mastic trees

Viewing order on page

1

Text

According to tradition, the skinos trees on Chios shed tears owing to the miracle of Saint Isidore. He was born around 230 A.D. in Alexandria. While serving as an officer at the Roman navy in Chios he confessed his faith before admiral Numerius. He was jailed, tortured and finally beheaded. His martyrdom during transport from Chora to Nechori, the beheading and the tearing to pieces of his body remained indelibly in the collective memory of local society, and his memory is honoured with great glory on 14 May, as he is considered the patron saint of mastic trees. Churches and chapels dedicated to Saint Isidore can be found at the villages Neochori, Nenita, Kallimasia, Armolia, Pyrgi, Mesta, Lithi, Elata, Korni, Agios Georgios Sykousis and elsewhere on Chios.

+ Add text

Related media object

Search or create a media

+ Create new

+ Add related media object

+ Add presentation segment

Save Reset

Presentation: The cultural heritage of mastic

Web template

View Authoring Narrative preview

Saint Isidore of Chios

Narration

The story of Saint Isidore of Chios

Presentation

The cultural heritage of mastic

Key Actors

Saint Isidore of C...

The cultural heritage of mastic

Patron saint of mastic trees

According to tradition, the skinos trees on Chios shed tears owing to the miracle of Saint Isidore. He was born around 230 A.D. in Alexandria. While serving as an officer at the Roman navy in Chios he confessed his faith before admiral Numerius. He was jailed, tortured and finally beheaded. His martyrdom during transport from Chora to Nechori, the beheading and the tearing to pieces of his body remained indelibly in the collective memory of local society, and his memory is honoured with great glory on 14 May, as he is considered the patron saint of mastic trees. Churches and chapels dedicated to Saint Isidore can be found at the villages Neochori, Nenita, Kallimasia, Armolia, Pyrgi, Mesta, Lithi, Elata, Korni, Agios Georgios Sykousis and elsewhere on Chios.

Timeline

Map of events

Figure 35. Data entry form (left) and presentation (right) for a narrative.

Moreover, the elements stored in the knowledge base are semantically annotated using the Getty Arts and Architecture Thesaurus. These annotations are presented as hyperlinks for the user to access their definitions as well as find their place in the conceptual hierarchy. Two examples are provided in the figure below.

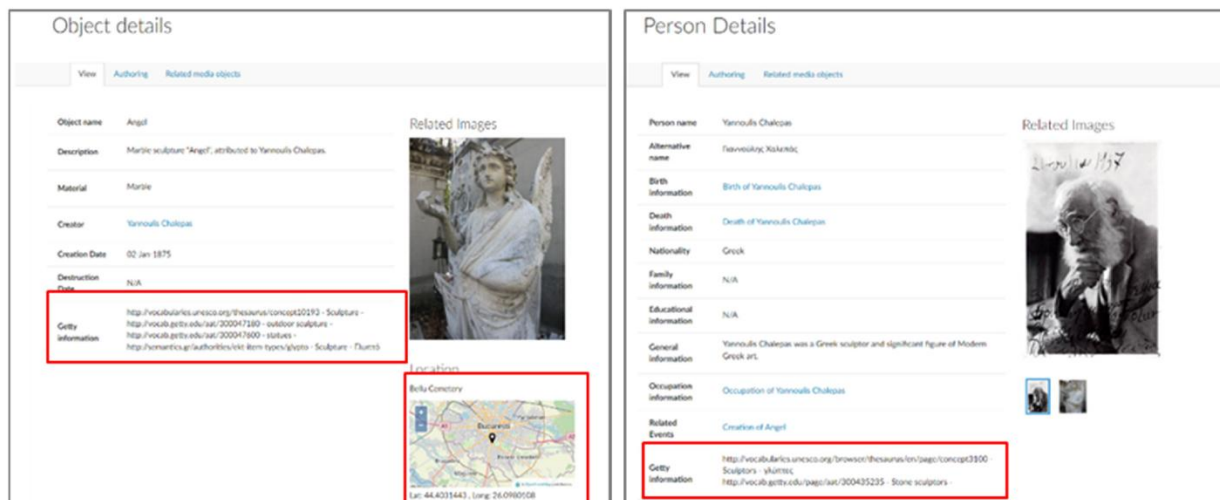


Figure 36. Two examples of semantic annotations using controlled vocabularies, one for a craft product (left) and one for a person (right).

3.3 Progress in Craeft

In Craeft we intend to extend MOP in two directions

1. Accommodate the extended ontology to be developed in Craeft to accommodate new concepts pertinent to craft simulations. Accordingly, extend the GUI to accommodate the authoring and presentation of the new concepts.
2. Given our experience of the platform usage, to improve the functionalities and technical facilities of the MOP. This includes both facilities for the users, as well as technical improvements that enhance the integrity of the platform. Moreover, these extensions include the adoption of additional controlled vocabularies, as described below.

In addition, the MOP was extended to accommodate the new RCIs that were introduced in Craeft.

3.4 Multilingualism

Enhancements were made to the MOP/CAP system for traditional crafts, focusing on the implementation of multilingual support. The effort aimed to improve user experience by allowing the entry of textual data in multiple languages, thus accommodating the diverse linguistic backgrounds of users.

We consider the MOP/CAP as a valuable platform for preserving and sharing knowledge about traditional artisanal practices. With this enhancement, we wish to overcome limitations in supporting multiple languages that hindered its usability for a broader audience.

3.4.1 Objectives

The objectives of this task were the following:

- Multilingual Support: Enable users to enter textual data in multiple languages for all textual fields and entities in the system.
- User-Friendly Interface: Implement an intuitive interface for data entry through online forms with a convenient language selection mechanism.

3.4.2 Methodology

Our approach followed the two objectives separately.

3.4.2.1 Database schema

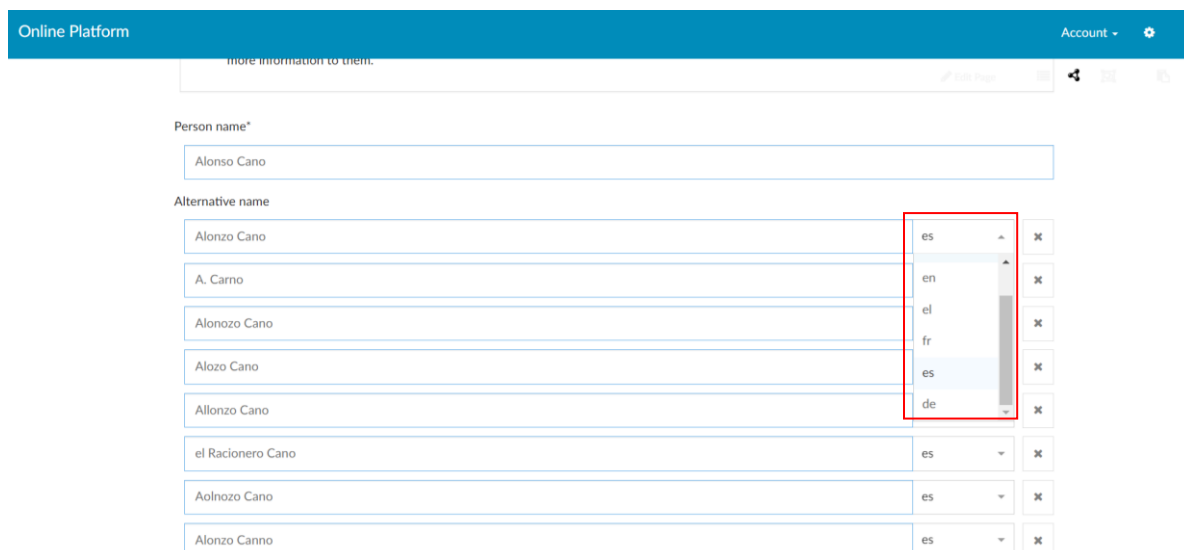
To accommodate multilingual support, modifications were made to the database schema. Each textual field and entity now supports multiple instances, allowing users to input data in different languages. The implementation of multilingual support involved a systematic approach to database modifications and user interface enhancements. Users can now input textual data in various languages, with the ability to add instances for each language.

3.4.2.2 User Interface

The data entry forms were updated to include a pop-up language selection menu alongside each textual field. This enables users to specify the language of the entered textual data easily. A pop-up menu enables users to select the language for each textual field, providing a clear and accessible language specification mechanism.

The panel that presents the documentation for each knowledge entity has also been updated so that it indicates the language in which each textual entry is written.

The images below illustrate these enhancements.



Online Platform Account ▾ ⚙

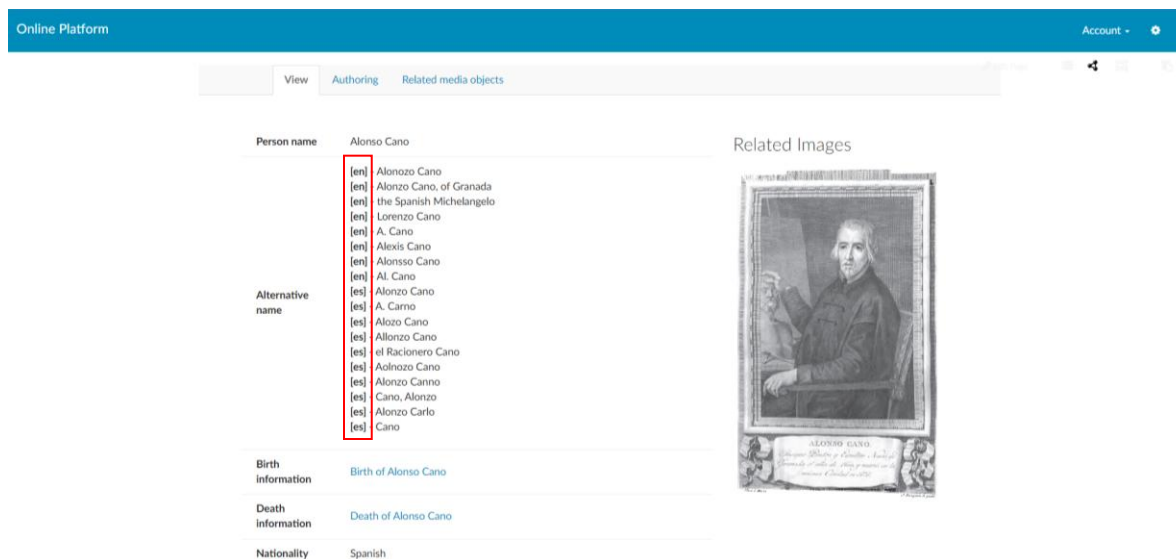
more information to them. ✎ Edit Page ⏏ ⏴ ⏵ ⏶ ⏷ ⏸

Person name*

Alonso Cano

Alternative name

Alonzo Cano	es	✕
A. Cano	en	✕
Alonzo Cano	el	✕
Alozo Cano	fr	✕
Allonzo Cano	es	✕
el Racionero Cano	de	✕
el Racionero Cano	es	✕
Aolnzo Cano	es	✕
Alonzo Canno	es	✕



The screenshot displays the 'Online Platform' interface. At the top, there are tabs for 'View', 'Authoring', and 'Related media objects'. The 'Authoring' tab is active, showing a form for a person named 'Alonso Cano'. The form includes a 'Person name' field with a dropdown menu showing various language-coded entries (e.g., (en) Alonso Cano, (es) Alonso Cano, (fr) Alonso Cano). Below this is an 'Alternative name' field with a similar dropdown. The form also includes fields for 'Birth information', 'Death information', and 'Nationality'. To the right of the form, there is a 'Related Images' section displaying a portrait of Alonso Cano.

Figure 37. The authoring data entry form (top) and the viewing page (bottom) for a person, illustrate the multilingual capabilities of the platform.

The first image (top) shows the pop-up menu. The second image (bottom) shows the view panel that presents the documentation. As can be observed, each textual entry is preceded by an indicator that specifies the language of the textual field's content.

3.4.3 Implementation notes

The database schema has been modified to support an arbitrary number of instances so that any number of languages can be supported. For now, the GUI and specifically the pop-up menu for the textual fields has five entries, which correspond to the native languages of the RCIs studied in Craeft. These are French, Spanish, Greek, and German, as well as English which is the default language. The reason we used only these four languages was usability. In the context of the Craeft project, only these four languages are used. As such, creating a pop-up menu with numerous language entries would hinder usability. In a future step, we plan to associate user and craft instance profiles with languages, so that the pop-up menu for language selection dynamically adapts to the user and the craft instance.

The only textual field for which our implementation slightly differs from what was already mentioned is the name of knowledge entities, which remains in English. The reason is that this is the default language of the system. Still, names can be entered in any language, using the alternative name field.

3.5 Cross-references

Updates were made to the MOP/CAP focusing on the improvement of knowledge entity relationships and the implementation of a computer-aided garbage collection and error correction mechanism. The goal is to enhance the user experience by providing comprehensive views of associations between knowledge elements and facilitating the identification and correction of unreferenced entities in the database.

Our motivation was the following. As the MOP/CAP evolved, it became imperative to strengthen the connections between knowledge entities. A computer-aided garbage collection mechanism was introduced to identify and rectify unreferenced entities, often resulting from data entry errors.

In the following, we call cross-references of knowledge elements the database relations (pointers) that point to this element from other knowledge elements.

3.5.1 Objectives

- **Enhanced Entity Relationships:** Enable users to easily navigate and view associations between different knowledge elements within the system.
- **Garbage Collection:** Implement a mechanism to identify and manage unreferenced entities for database cleanliness and accuracy.
- **User-Friendly Correction:** Introduce a new field in the data entry form to allow users to establish associations between knowledge entities for improved data integrity.

3.5.2 Methodology

3.5.2.1 Database Relations

The cross-references between knowledge entities were revisited and are now shown when viewing the knowledge elements. This enables users to navigate through associated elements. This enhancement promotes a better understanding of the interconnections within the traditional crafts documentation system. Users can now view all associated knowledge elements when viewing a specific entity, providing a more comprehensive and interconnected understanding of traditional crafts data.

3.5.2.2 Garbage Collection

Queries were developed to identify unreferenced entities in the database. The results of these queries are displayed on online Web pages, presenting hyperlinks to unreferenced elements along with a delete button for corrective actions. Online Web pages showcase unreferenced entities, enabling users to either delete these entities or follow links for correction.

3.5.2.3 Data Entry Forms

A new field, named "Associate with", was added to the data entry form of each knowledge entity. This field incorporates a pop-up menu with autocomplete functionality, allowing users to establish associations during the data entry process. The addition of the "Associate with" field in data entry forms streamlines the process of establishing associations between knowledge entities, reducing errors and improving data integrity.

3.5.2.4 Additional queries

Additional queries were formulated in the same spirit to increase the integrity of the knowledge base contents. Specifically, several queries were formulated to ease the checking of missing inputs or other

user errors. As such the following queries were formulated and added to a Web page, along with an additional query that presents analytics on the contents of the database.

1.	3D Models Source URLs
2.	3D Models WITHOUT Source URLs
3.	3D Models GLB Sources
4.	3D Models WITHOUT GLB Sources
5.	3D Models Thumbnail image source URLs
6.	3D Models WITHOUT Thumbnail image source URLs
7.	3D Models & Associated with objects
8.	3D Models WITHOUT Associated with objects
9.	3D Models WITHOUT Pilot
10.	3D Models with UNKNOWN Pilot
11.	3D Models WITHOUT Creator
12.	3D Models WITHOUT Creation Date
13.	3D Models WITHOUT Metrics
14.	3D Models WITHOUT Creative Commons licence
15.	3D Motions Source URLs
16.	3D Motions WITHOUT Source URLs
17.	3D Motions WITHOUT Pilot
18.	3D Motions with UNKNOWN Pilot
19.	3D Motions WITHOUT Creator
20.	3D Motions WITHOUT Creation Date
21.	3D Motions WITHOUT Creative Commons licence
22.	Embedded Videos Source URLs
23.	Embedded Videos WITHOUT source URLs
24.	Embedded Videos WITHOUT Pilot
25.	Embedded Videos WITHOUT Creator
26.	Embedded Videos WITHOUT Creation Date
27.	Embedded Videos with UNKNOWN Pilot
28.	Events
29.	Events WITHOUT Name
30.	Events WITHOUT Description
31.	Events WITHOUT Pilot
32.	Events with UNKNOWN Pilot
33.	Events WITHOUT Dates
34.	Fabulae
35.	Fabulae WITHOUT Description
36.	Fabulae WITHOUT Events
37.	Fabulae WITHOUT Associated with objects
38.	Fabulae WITHOUT Pilot
39.	Fabulae with UNKNOWN Pilot
40.	Images Thumbnails

41. Images WITHOUT Source URLs
42. Images Source URLs WITHOUT Associated with objects
43. Images WITHOUT Pilot
44. Images with UNKNOWN Pilot
45. Images WITHOUT Creator
46. Images WITHOUT Creation Date
47. Images WITHOUT Metrics
48. Images WITHOUT Creative Commons licence
49. Locations
50. Locations WITHOUT Name
51. Locations WITHOUT Image
52. Locations WITHOUT Pilot
53. Locations with UNKNOWN Pilot
54. Locations WITHOUT Associated with event(s)
55. Materials
56. Materials WITHOUT Name
57. Materials Associated with Objects
58. Materials WITHOUT Associated with objects
59. Materials WITHOUT Pilot
60. Materials with UNKNOWN Pilot
61. Narratives
62. Narratives WITHOUT Description
63. Narratives WITHOUT Fabula
64. Narratives WITHOUT Narration
65. Narratives WITHOUT Pilot
66. Narratives with UNKNOWN Pilot
67. Persons
68. Persons WITHOUT Name
69. Persons WITHOUT Image
70. Persons WITHOUT Pilot
71. Persons with UNKNOWN Pilot
72. Pilots
73. Process Schemas
74. Process Schemas WITHOUT Pilot
75. Process Schemas with UNKNOWN Pilot
76. Processes
77. Processes WITHOUT Description
78. Processes WITHOUT Pilot
79. Processes with UNKNOWN Pilot
80. Products
81. Products WITHOUT Name

82.	Products WITHOUT Description
83.	Products WITHOUT Pilot
84.	Products with UNKNOWN Pilot
85.	Products WITHOUT Material
86.	Products WITHOUT Image
87.	Social Groups
88.	Social Groups WITHOUT Pilot
89.	Social Groups with UNKNOWN Pilot
90.	Tools
91.	Tools WITHOUT Name
92.	Tools WITHOUT Description
93.	Tools WITHOUT Pilot
94.	Tools with UNKNOWN Pilot
95.	Tools WITHOUT Material
96.	Tools WITHOUT Image
97.	Videos Source URLs
98.	Videos WITHOUT source URLs
99.	Videos WITHOUT Pilot
100.	Videos with UNKNOWN Pilot
101.	Videos WITHOUT Creator
102.	Videos WITHOUT Creation Date
103.	Videos WITHOUT Metrics
104.	Videos WITHOUT Creative Commons licence
105.	Statistics

3.6 Zenodo storage

In collaboration and coordination with representatives of Europeana and Zenodo the digital assets provided by Craeft are stored in the Zenodo repository, which is funded by the European Commission. As such, only meta-data are stored in the local file system of the MOP/CAP. An example is shown in the figure below.

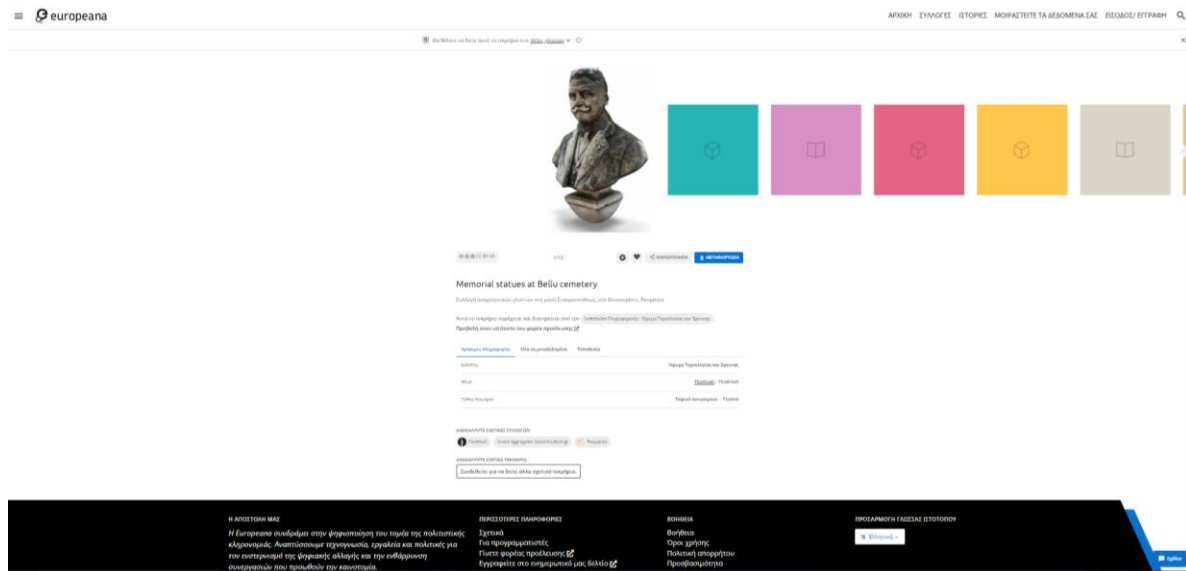


Figure 40. A Web page at Europeana, with a collection of 10 marble sculptures.

The example regards a collection of 10 marble craft artefacts which can be found on the following Europeana page

https://www.europeana.eu/el/item/911/https://www.searchculture.gr/aggregator/edm/mingei_000_155_c3ec5fe2_e189_4a4e_88f0_79e3718a1dcd

The 3D assets, as well as their image previews, are all stored on the Zenodo repository.

3.6.1 Technical advantages

Storing assets on Zenodo is important because the online storage space for digital assets of large storage capacity can be costly. It is therefore more important for small artisanal communities that may have limited resources.

Moreover, storing digital assets on an online repository (Zenodo) instead of our Web server storage offers several additional advantages.

- Zenodo is hosted as a cloud service, providing high availability and accessibility from anywhere with an internet connection. This ensures that digital assets are readily available. Moreover, Zenodo offers an API and, therefore, the stored content can be used directly by third-party applications.
- Zenodo implements backup and redundancy measures, ensuring the integrity and availability of digital assets. This helps protect against data loss due to hardware failures, disasters, or other unforeseen events.
- Zenodo has invested in security measures, including encryption, access controls, and regular security updates. This enhances the overall security of digital assets compared to managing our server. Moreover, Zenodo performs its own server maintenance, updates, and security patches, reducing the burden of keeping up with security updates.

- Zenodo provides DOIs for the submitted assets and also offers analytics of access, easing the distribution of assets and enabling the tracking of their access respectively. Moreover, Zenodo offers compliance certifications with the Creative Commons licences and adheres to industry-specific regulations, ensuring that data storage practices align with legal requirements and industry standards.

3.6.2 Ecological advantages

Using Zenodo and cloud storage services in general, instead of hosting digital assets on our Web server offers ecological benefits.

An advantage is the increased energy efficiency achieved by large-scale cloud providers through economies of scale and investments in optimised data centres. These providers often utilise advanced technologies and practices to reduce overall energy consumption, making their infrastructure more environmentally friendly compared to smaller, less efficient server setups.

Cloud services also promote better resource utilisation by consolidating data and applications on shared infrastructure. This can lead to more efficient use of hardware resources, resulting in lower energy consumption per unit of computing power. The use of server virtualization technologies by cloud providers allows multiple virtual servers to run on a single physical server, contributing to higher server utilisation rates and reduced energy consumption compared to traditional setups. Additionally, cloud services enable dynamic scaling of resources based on demand, ensuring that energy consumption is optimised. During periods of low demand, fewer servers are active, while additional resources can be provisioned during peak demand, resulting in a more efficient use of energy resources.

3.7 Other enhancements

Two types of exports have been automated and the online viewing of 3D models has been upgraded.

3.7.1 Knowledge element to file

When visiting a page that displays the contents of a knowledge element, it is often that users may wish to copy this content, e.g. to include it in a document that they are writing etc. Copying from the screen is possible, but rather inefficient and, at the same time, does not preserve the formatting hierarchy.

To ease users in this task, a button was added that provides the contents of the knowledge element in an XML file that contains the documentation of knowledge elements organised as per the RDF standard. This is demonstrated in the figure below.

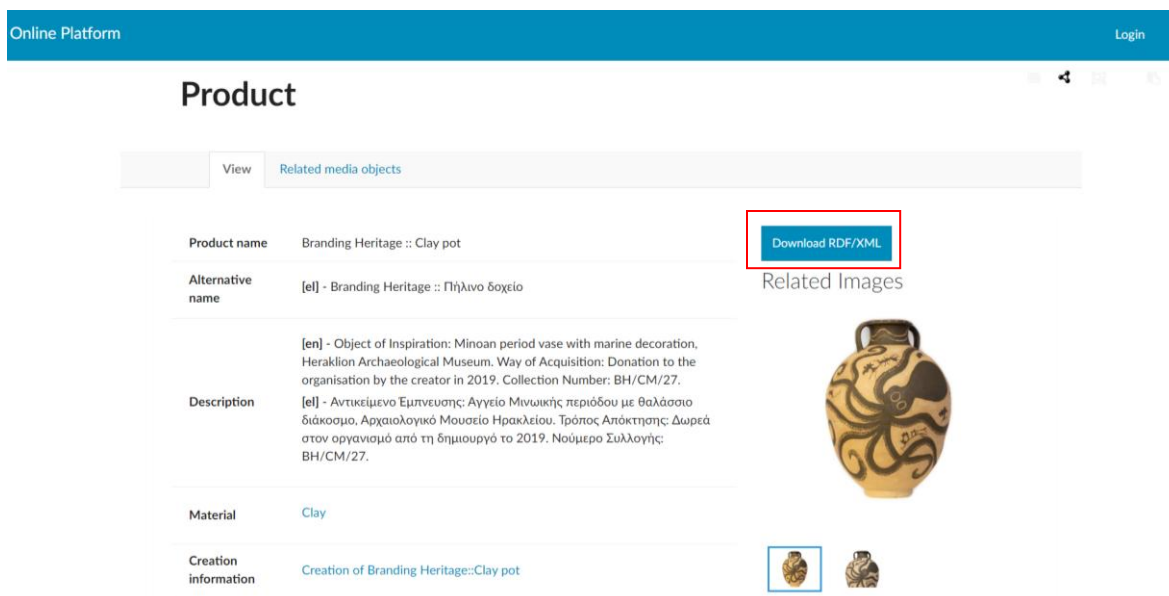


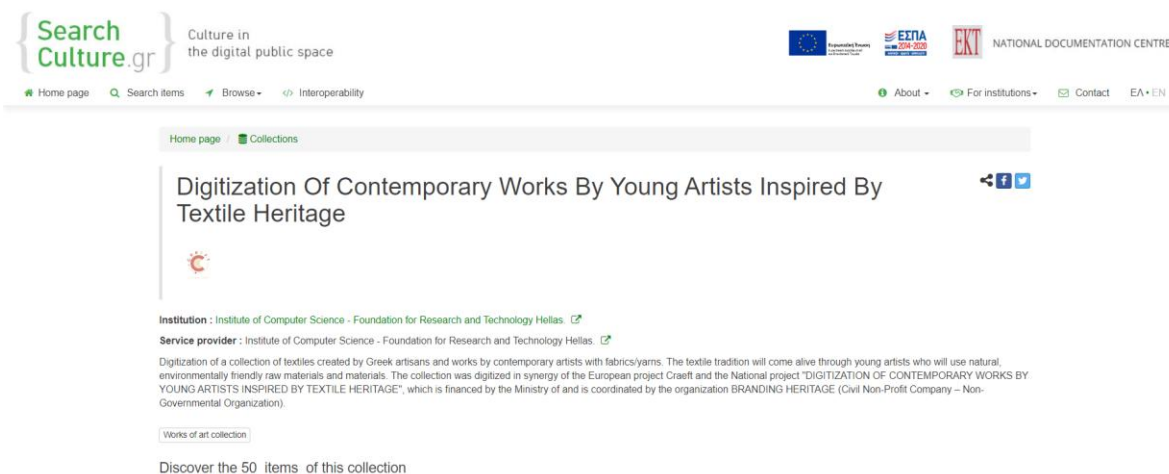
Figure 38. The page for a crafted artefact illustrates the export-to-file capabilities of the platform.

As the exported file is in XML format it can be directly incorporated by several word processing software suites, including the widely-used MS Word software, as well as Web browsers. Moreover, the XML file can be directly converted to HTML by several software utilities.

3.7.2 Export for Europeana ingestion

The RDF export mentioned in the previous subsection is utilised for the automation of meta-data exports for submitting them to National Aggregators and eventually being ingested by Europeana. System (MOP/CAP) administrators can define a collection of items to be submitted and using the aforementioned automation the meta-data are automatically exported in an online file that is sent to the national aggregator.

An example is shown in the figure below.



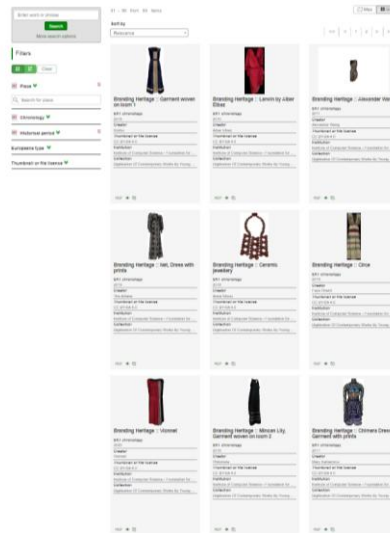


Figure 39. A Web page from the Greek National Aggregator, for a collection of 50 handcrafted garments.

The example shows a collection of 50 handcrafted garments on the page of the Greek National Aggregator, submitted in the context of Craeft. The digital assets for each garment are images and a 3D model obtained by photogrammetric reconstruction. This content was ingested by the National Aggregator on 18 December 2023 and has been forwarded for ingestion by Europeana. The automatically generated file for this collection can be found at the following link:

<http://api.mingei-project.eu/public/api/metadata?verb=ListRecords&metadataPrefix=edm&set=brandingHeritage>

while the page of the Greek National Aggregator for this collection can be found here:

<https://www.searchculture.gr/aggregator/portal/collections/brandingHeritage/search?page.page=1&scrollPositionX=4057&sortByCount=false&resultsMode=GRID&sortResults=SCORE>

3.7.3 Online viewing of 3D models

To enhance the digital presentation of our craft artefacts, we have transitioned our website's 3D viewing capabilities to the Babylon.js engine. This upgrade aligns with our commitment to leveraging cutting-edge technology to make cultural and artistic works more accessible and engaging to a global audience.

The primary objectives behind this upgrade were to:

- Improve the visual quality and interactivity of 3D models.
- Enhance user experience with smoother navigation and more intuitive controls.
- Enable advanced features such as real-time lighting, shadows, and animations.
- Ensure broad accessibility across various devices and browsers.
- Foster a deeper appreciation of craft artefacts through immersive digital experiences.

The upgrade process involved the following steps: (1) we evaluated existing 3D model presentation frameworks and identified their advantages and limitations; we chose Babylon.js based on its performance, feature set, and compatibility. Babylon.js was seamlessly integrated into the existing CAP architecture, without any disruption to the user experience and knowledge of system operation; (3) thereafter, we conducted tests across multiple devices and browsers to ensure compatibility and user experience consistency.

The main benefits of this upgrade are the increased visual quality in the presentation of 3D models and the better interactivity features of this viewer. Babylon.js's provides advanced rendering capabilities that significantly improve the visual quality of our 3D craft artefacts, with realistic textures, lighting, and shadows that provide a more lifelike and engaging experience. In terms of interactivity, users can now interact with the 3D models in more meaningful ways, including zooming, rotating, and exploring different parts of each artefact, which enhances educational and engagement opportunities. Moreover, full-screen viewing and automated rotational animations are provided, as well as integration with 3D and VR viewing are provided.

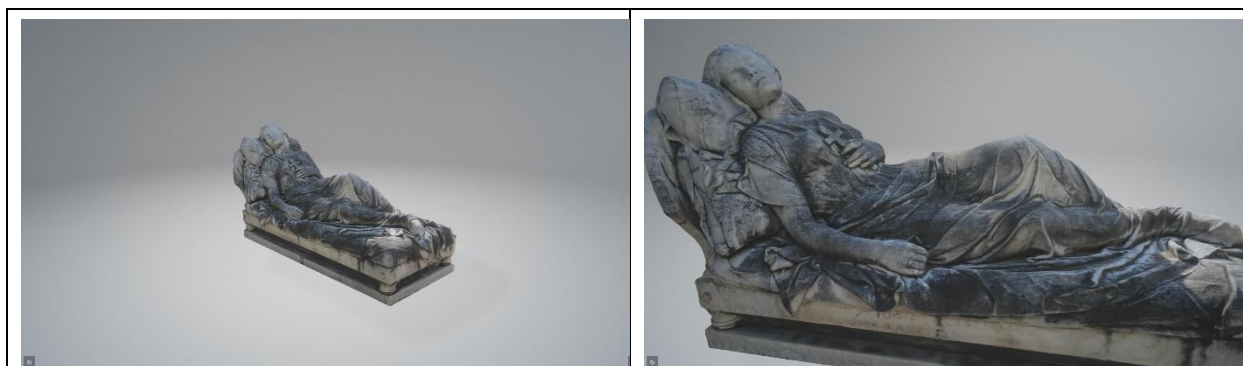


Figure 41. A marble sculpture is shown in the Babylon.js viewer using illumination and shadow features of the viewer.

Technically, Babylon.js has an optimized rendering engine ensuring that 3D models load quickly and run smoothly, even on mobile devices, providing a seamless experience for all users. With Babylon.js, our 3D models are accessible on a wide range of devices and browsers, ensuring that more users can enjoy our digital collections without technical barriers.

In the future, we plan to explore further enhancements, including personalized user experiences, AR and VR integrations, and the use of interactive guides and educational content alongside our 3D models.

3.7.4 Media object types

We have enhanced the types of media objects with two new types.

The "Documents" type is reserved for books, printed works, notebooks, which may be of multiple pages and are in a document format (e.g. PDF, ePub). This way, they can be accessed online and through the web browser, using the plug-in of end-user choice.

The "Data Files" type is a container for formatted data either from simulation or from measurement of physical quantities (e.g. environment conditions).

3.7.5 Mosaics

The surface digitisation of T3.3 produces multiscale, very high-resolution scans. The storage capacity of these scans is, correspondingly, significant. The dimensions of these scans are larger than those of computer monitors. Their access is performed in parts, or tiles, to reduce access time viewer waiting time. This is achieved by loading only the visible tile(s), given the viewpoint and magnification of the GUI. When viewing such data online, this access strategy reduces waiting time further, as loading over the network is slower. This access pattern is widely used in online map viewers which provide magnification.

The datasets produced using the approach of T3.3 exhibit an additional property compared to conventional image mosaics, which are photographed from a single distance. Due to their way of acquisition, these datasets contain additional, high-resolution mosaics that image the surface at different (larger) distances. Moreover, these mosaics are aligned (spatially registered) with the very high-resolution mosaic, which corresponds to the closest distance. These two properties make these datasets ideal for viewing the surface at multiple scales (magnifications). The reason is that at each magnification an authentic photograph is presented to the viewer, without a processed minification of the high-resolution mosaic. Besides providing more accurate and more realistic results, the user experience is more natural and pleasing and has a smaller response time because no image processing is performed. Online map applications retrieve images acquired at different heights (i.e. satellite, aerial) and present them likewise for the same reasons.

To digitally preserve the obtained surface digitisations without multiscale information and for efficient access, we employ a multiscale, tile-oriented mosaic viewer. Correspondingly we use an online viewer ([OpenSeaDragon](#)) to show the image properly. Our goal is to streamline the scanning process and automatically produce such manifest files. We adopted the OpenSeaDragon because it is IIIF compatible. Currently, this viewer is integrated with the CAP [[DEMO](#)]. This is demonstrated in the following diagram:

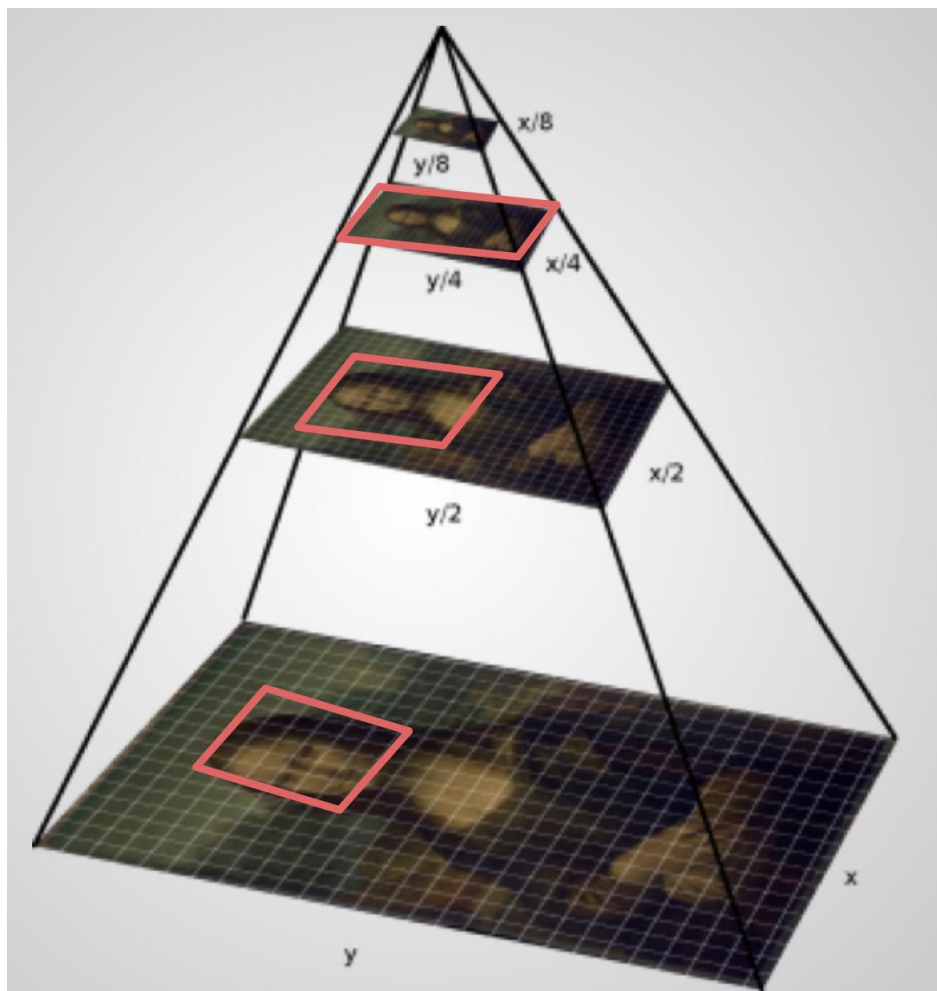


Figure 42. Multiscale image tiling with the OpenSeaDragon viewer.

When the user zooms into an image the viewer only shows you the tiles that are in your view. The view is shown in the red box. This means a very large image can be viewed but the viewer never downloads all of the images. Only the ones that are required. Here is an example of how we use OpenSeaDragon to show this type of content:

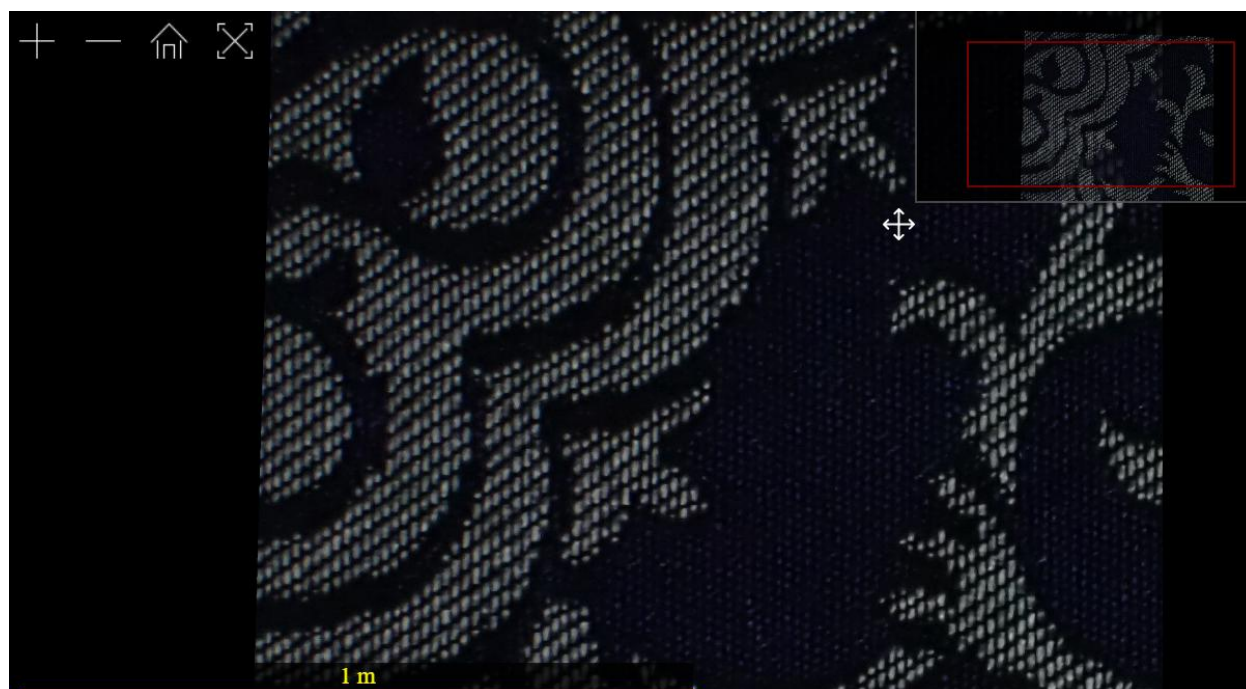


Figure 43. Example usage of the OpenSeaDragon viewer on a textile scan.

The user can use the mouse scroll to zoom and the navigator to move into the image area. HTML files and image files are stored on our server. For testing, we installed locally one of the available [IIIF-compatible image servers](#), [Cantaloupe](#) to generate manifests for image tiles. We uploaded to the server, the structure of the above mosaic. Each image tile is accessible with the IIIF URL. An IIIF JSON manifest is dynamically produced for each image tile. We uploaded to the MOP server a testing mosaic, with all image-tiles references being made with URLs of the local Cantaloupe IIIF image server and referring to images uploaded to it. We also created a script for the dynamic generation-storage of a separate IIIF manifest for each image-tile of the mosaic.

4 Conclusions

This document has presented the CRAEFT Ontology and the CRAEFT authoring Platform.

The CRAEFT Ontology, named CrO, is an application ontology used for representing crafts in the CRAEFT project. For interoperability, the CrO is built on standards. It is an extension of the CIDOC CRM with the concepts needed for representing processes at three different levels:

- The *schema* level, including descriptions of processes and their components, is used for descriptive and prescriptive purposes, as illustrated in Section 2.5.1.1.
- The *virtual* level, including descriptions of the execution of processes and actions performed by digital agents in the virtual world, is used for comparative purposes, as illustrated in Section 2.5.1.2.
- The *physical* level, including descriptions of the execution of processes and actions performed by human agents in the real world, is used for documentation and preservation purposes, as illustrated in Section 2.5.1.3.

The notion of process is inspired by the UML conceptualization of activities, providing primitive categories for processes, their components and the interrelations of these components. At the physical level, processes are also seen as narratives, thereby acquiring narrations and presentations as fundamental aspects, not considered in the UML conceptualisation, but crucial for the achievements of the CRAEFT objectives. The ontology of narratives, previously developed by the authors, is therefore a fundamental component of the CrO.

Finally, the CrO is expressed in the standard language OWL 2 DL, the most expressive representative of the OWL family of languages that retains computational amenability.

The CrO has been applied to represent a few realistic samples of processes and actions, to the end of showing its adequacy preliminarily. A more substantial evaluation of the ontology will be performed in the prosecution of the project, putting the CrO at work on the Representative Craft Instances selected by the project.

The CrO allows linking to the Getty AAT, CONA, TGN, and ULAN dictionaries, as a way to enhance interoperability while enriching the online semantic documentation system. Users can thereby benefit from:

- Expanded vocabulary for more accurate and standardised artefact descriptions.
- Detailed insights into art collections, enhancing cultural exploration.
- Geospatial context for cultural heritage locations, improving contextual understanding.
- Artist information for a deeper appreciation of the creators behind cultural artefacts.

The integration of Getty AAT, CONA, TGN, and ULAN into our online semantic documentation system marks a milestone in advancing the platform's capacity for cultural heritage documentation. This enhancement positions our system as a comprehensive and authoritative resource for users interested in exploring and understanding cultural artefacts, collections, and the individuals behind the creations.



The Craeft Authoring Platform (CAP) is a Web-based, online, multiple-user authoring platform for the documentation of traditional crafts that is under development. It is functional and accessible online, although it is not yet in its final form. The CAP is based on the Mingei Online Platform (MOP) developed in the Mingei Innovation Action.

The implementation of multilingual support has resulted in a more inclusive and user-friendly system. Users can now document traditional crafts in multiple languages, promoting diversity and inclusivity in the knowledge preservation process.

The implementation of multilingual support in the online documentation system is a step towards inclusivity and accessibility. This enhancement ensures that the platform remains a valuable resource for individuals across different linguistic backgrounds, fostering the preservation and exchange of traditional craft knowledge.

The implementation of enhanced entity relationships and garbage collection has resulted in a more cohesive and accurate documentation system. Users benefit from a clearer visualisation of associations, while the garbage collection mechanism ensures the removal or correction of unreferenced entities for improved data quality.

The updates to the online documentation system mark an advancement in both user experience and database cleanliness. The strengthened relationships between knowledge entities and the introduction of garbage collection contribute to the system's overall efficiency and reliability, ensuring a robust platform for the documentation of traditional crafts.

References

- [1] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language structural specification and functional-style syntax (second edition). W3C recommendation, W3C, December 2012. <http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>.
- [2] Martin Doerr. The CIDOC Conceptual Reference Model: An ontological approach to semantic interoperability of metadata. *AI Mag.*, 24(3):75–92, September 2003.
- [3] The CIDOC CRM Special Interest Group. Definition of the CIDOC Conceptual Reference Model. Version 7.2.3, May 2023. Available from https://www.cidoc-crm.org/releases_table
- [4] Technical Committee: ISO/TC 46/SC 4 Technical interoperability. ISO 21127:2014 - Information and documentation — A reference ontology for the interchange of cultural heritage information. <https://www.iso.org/standard/57832.html>
- [5] N. Guarino. Formal ontology in information systems. In *Proceedings of FOIS 98*, pages 3–15. IOS Press, Amsterdam, 1998. Amended version.
- [6] V. Bartalesi, C. Meghini, and D. Metilli. Representing Narratives in Digital Libraries: The Narrative Ontology. *Semantic Web*, 12(2), 241-264.
- [7] Simon Cox, Chris Little. Time Ontology in OWL. W3C Recommendation 19 October 2017. <https://www.w3.org/TR/owl-time/>
- [8] DCMI Metadata Terms. Dublin Core Metadata Initiative. <https://www.dublincore.org/specifications/dublin-core/dcmi-terms/>
- [9] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Second Edition. Addison-Wesley. 2005.
- [10] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 concepts and abstract syntax. W3C Recommendation, WWW Consortium, February 2014. <http://www.w3.org/TR/rdf11-concepts/>.
- [11] Dan Brickley and R.V. Guha. RDF schema 1.1. W3C Recommendation, WWW Consortium, February 2014. <http://www.w3.org/TR/rdf-schema/>
- [12] XML schema definition language (XSD) 1.1 part 2: Datatypes. W3C Recommendation, WWW Consortium, April 2012. <http://www.w3.org/TR/xmlschema11-2/>.
- [13] Hiebel, G., Doerr, M. & Eide, Ø. CRMgeo: A spatiotemporal extension of CIDOC-CRM. *Int J Digit Lib* 18, 271–279 (2017). <https://doi.org/10.1007/s00799-016-0192-4>. Formal specifications available at <https://www.cidoc-crm.org/crmgeo/home-5>
- [14] Chris WELTY and Richard FIKES. A Reusable Ontology for Fluents in OWL. *Proceedings of the 2006 conference on Formal Ontology in Information Systems: (FOIS 2006) Frontiers in Artificial Intelligence and Applications*. Volume 150: Formal Ontology in Information Systems. IO-Press. May 2006. Pages 226–236.
- [15] S. Borgo, R. Ferrario, A. Gangemi, N. Guarino, C. Masolo, D. Porello, E. Sanfilippo, L. Vieu. DOLCE: A descriptive ontology for linguistic and cognitive engineering. *Applied Ontology*, vol. 17, no. 1, pp. 45-69, 2022.
- [16] Cominelli F., 2011. Governing Cultural Commons: The case of traditional craftsmanship in France. In *Biennial Conference. International Association for the Study of the Commons*, Hyderabad, India, 1–27.
- [17] Donkin L., 2001. *Crafts and Conservation*. Report. ICCROM.
- [18] Zabulis X, Partarakis N, Meghini C, Dubois A, Manitsaris S, Hauser H, Magnenat Thalmann N, Ringas C, Panesse L, Cadi N, Baka E, Beisswenger C, Makrygiannis D, Glushkova A, Padilla BEO, Kaplanidi D, Tasiopoulou E, Cuenca C, Carre A-L, Nitti V, Adami I, Zidianakis E, Doulgeraki P, Karouzaki E, Bartalesi V, Metilli D. A Representation Protocol for Traditional Crafts. *Heritage*. 2022; 5(2):716-741. DOI:10.3390/heritage5020040.

- [19] X. Zabulis, N. Partarakis, A. Argyros, A. Tsoi, A. Qammaz, I. Adami, P. Doulgeraki, E. Karuzaki, A. Chatziantoniou, N. Patsiouras, E. Stefanidi, Z. Stefanidi, A. Rigaki, M. Doulgeraki, A. Patakos, S. Manitsaris, A. Glushkova, B. Olivas-Padilla, D. Menychtas, D. Makrygiannis, C. Meghini, V. Bartalesi, D. Metilli, N. Magnenat-Thalmann, E. Bakas, N. Cadi, D. van Dijk, P. de Sterke, M. Wippoo, M. van der Vaart, C. Ringas, M. Fasoula, E. Tasiopoulou, D. Kaplanidi, L. Pannese, V. Nitti, C. Cuenca, A. Carre, A. Dubois, H. Hauser, C. Beisswenger, D. Blatt, I. Neumann, U. Denter. The Mingei Handbook, <https://doi.org/10.5281/zenodo.6580124>
- [20] Meghini C., Bartalesi V., Metilli D., Partarakis N., and Zabulis X.. 2020. Mingei Crafts Ontology. Retrieved from <https://zenodo.org/record/3742829#.Xw1prigZaR>
- [21] Meghini C. and Doerr M.. 2018. A first-order logic expression of the CIDOC conceptual reference model. *Int. J. Metadata Semant. Ontol.* 13, 2 (2018), 131–149.
- [22] Doerr M.. 2003. The CIDOC conceptual reference model: An ontological approach to semantic interoperability of metadata. *AI Mag.* 24, 3 (2003), 75–92.
- [23] Cox S. and Little C. 2017. Time Ontology in OWL, W3C Recommendation. Retrieved from <https://www.w3.org/TR/owl-time/>
- [24] ISO 21127:2014. Information and documentation—A reference ontology for the interchange of cultural heritage information.
- [25] Foundation Open Street Map. 2021. Open Street Map. <https://www.openstreetmap.org/>.