



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
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	105



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”.

It 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 iure* 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 CRAEFT 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.
- 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.
- 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.

- 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 presents the representation of several craft processes and actions in the CRAEFT Ontology, thus offering a preliminary validation of the ontology on realistic examples and at the same time an initial proof of concept of the CRAEFT approach, before implementation and systematic evaluation of the representative craft instances. The considered processes are: inserting a chisel in a piece of wood, creating a pattern of dots on silver, hammering a piece of wood, bending a ply of aluminium and pulling a ply apart.

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
19/1/2024	Carlo Meghini	CNR	First draft
22/1/2024	Xenophon Zabulis	FORTH	Added missing figures.



23/1/2024	Carlo Meghini	CNR	Final versions
26/1/2024	Xenophon Zabulis	FORTH	Edits, formatting.
2/2/2024			Internal Review
16/2/2024	Carlo Meghini	CNR	Final version

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	3
Abbreviations	4
Table of contents	5
1. Introduction	7
2. The CRAEFT Ontology	8
2.1. Notational conventions	10
2.2. Resource identification	10
2.3. The Narrative Ontology	11
2.3.1. Main Classes	11
2.3.2. Main Properties	13
2.3.3. Representing actor roles in events	17
2.3.4. Modelling presentation segments	18
2.3.5. Modelling Time	18
2.3.6. Modelling Space	20
2.3.7. Modelling time-varying properties	23
2.4. Modelling processes and actions	24
2.4.1 Processes	24
2.4.2. Actions	25
2.4.3. Schemas	27
2.4.4. Transitions	29
2.5. An ontology of processes and actions	31
2.5.1. Activities	32
2.5.2. Transitions	45
2.6. Linking to standard dictionaries	57
2.6.1.1. AAT	57
2.6.1.2. CONA	61
2.6.1.3. TGN and Geonames	62
2.6.1.4. ULAN	64
2.6.1.5. Additional dictionaries	65
2.6.1.6. Implementation	68
2.7. Examples	69
2.7.1. Inserting a chisel in a piece of wood	70



2.7.2. Creating a pattern of dots on silver	74
2.7.3. Hammering a piece of wood	77
2.7.4. Bending a ply of aluminium	80
2.7.5. Pulling a ply apart	82
3. The CRAEFT Authoring Platform	85
3.1. Representation of knowledge in the MOP/CAP	86
3.2. User Interface	86
3.3. Progress in Craeft	90
3.4. Multilingualism	90
3.4.1. Objectives	90
3.4.2. Methodology	91
3.4.3. Implementation notes	92
3.5. Cross-references	92
3.5.1. Objectives	93
3.5.2. Methodology	93
3.6. Other enhancements	96
3.6.1. Knowledge element to file	98
3.6.2. Export for Europeana ingestion	98
3.6.3. Zenodo storage	96
3.6.4. Online viewing of 3D models	100
4. Conclusions	102
References	104



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 simulator employed by the project to compute the effects of actions. The linking to standard vocabularies is also presented, and a first, conceptual evaluation of the ontology is given by representing a few sample processes and actions.

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.

A second and final release of the deliverable will be provided in Month 24. This new release is expected to provide: (1) a more systematic evaluation of the CRAEFT Ontology, along with the required amendments to the version of the ontology reported here, and (2) a more complete implementation of the CRAEFT Authoring Platform, providing improvements specifically tailored to the newer part of the 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 CRont 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.rdf >

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 (craft) 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.
- Section 2.7 gives some examples of processes and actions modelled with the CrO.

2.1. Notational conventions

International Resource Identifiers (IRIs, for brevity) are written in the standard prefixed notation, in which the absence of a prefix indicates that the base prefix is used.

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

<i>Ln</i>	Every axiom is given in a row like this, where: <i>Ln</i> is an optional unique identifier and <i>pref</i> is an optional prefix identifying the ontology where the axiom belongs. The axiom is given in the second column 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:



Axioms of the OWL 2 DL languages that are stated in the present document are written using the functional notation and enclosed in table rows with the following graphic format:

<i>OWL 2 DL axiom</i>

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 CRAEFT 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 CRAEFT identifier is an HTTP IRI the de-referenciation of which is under the control of the CRAEFT project. In this way, the project has the opportunity to deliver a CRAEFT web page or an RDF graph for describing the resource to the human or digital users, respectively, who ask for any CRAEFT 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

The main classes and properties of the Narrative Ontology are depicted in Figure 1.

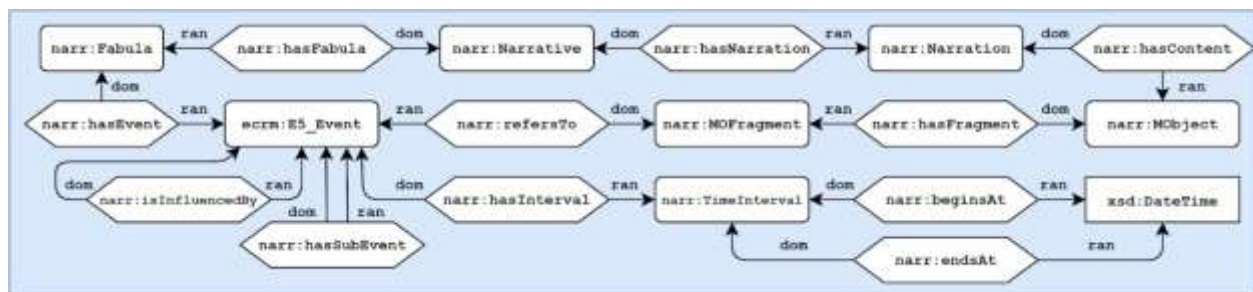


Figure 1 Main classes and properties of the Narrative Ontology.

2.3.1. Main Classes

A narrative is a composite object, including a narration, a fabula and a plot, as defined in [6]. In CrO, 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,

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

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



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
	narr:Narration is a subclass of ecrm:E89_Propositional_Object

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"*.

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

An event is an instance of 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, building of a castle, or battles"*. Events also include actions, which are viewed as happenings with an intention.

Media objects, relevant to CrO since they represent narrations' contents, are instances of class **narr:MObject**, a subclass of **ecrm:E90_Symbolic_Object**, which *"comprises identifiable symbols and any aggregation of symbols, such as characters, identifiers, traffic signs, emblems, texts, data sets, images, musical scores, multimedia objects, computer program code or mathematical formulae that have an objectively recognizable structure and that are documented as single units"*. Types of media objects are instances of class **narr:MObjectType**, a subclass of **ecrm:E55_Type**.

	narr:MObject is a class
	narr:MObject is a subclass of ecrm:E90_Symbolic_Object



	narr:MOType is a class
	narr:MOType is a subclass of ecrm:E55_Type

A fragment of a media object is an instance of class **narr:MOFragment**. Media object fragments are important is the formal representations of narratives since they typically are parts of narrations that present specific events or relevant parts of events. Also, media object fragments are instances of **ecrm:E90_Symbolic_Object**.

	narr:MOFragment is a class
	narr:MOFragment is a subclass of ecrm:E90_Symbolic_Object

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**, is a subclass of **ecrm:E89_Propositional_Object**, similarly to the class **narr:Narration**.

	narr:Presentation is a class
	narr:Presentation is a subclass of ecrm:E89_Propositional_Object

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 semantics relation, in that it connects an information object with a (representation of 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 **Narrative** is a subclass of **ecrm:E89_Propositional_Object**, therefore every narrative is also an instance of **ecrm:E89_Propositional_Object**.

	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

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`.

	<code>narr:hasNarration</code> is a property
	<code>narr:hasNarration</code> is a subproperty of <code>ecrm:E89_Propositional_Object</code>
	The domain of <code>narr:hasNarration</code> is class <code>narr:Narrative</code>
	The range of <code>narr:hasNarration</code> is class <code>narr:Narration</code>

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`.

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

The connection between a Narration and a Presentation is captured by property `narr:hasPresentation`. We consider this connection simply a composition relation thus we make `narr:hasPresentation` a subproperty of `ecrm:P148_has_component`, coherently with the fact that both narrations and presentations are instances of `ecrm:E89_Propositional_Object`.

	<code>narr:hasPresentation</code> is a property
	<code>narr:hasPresentation</code> is a subproperty of <code>ecrm:P148_has_component</code>
	The domain of <code>narr:hasPresentation</code> is class <code>narr:Narration</code>
	The range of <code>narr:hasPresentation</code> is class <code>narr:Presentation</code>

The connection between a media object 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`.

	<code>narr:refersTo</code> is a property
	<code>narr:refersTo</code> is a subproperty of <code>ecrm:P129_is_about</code>



	The domain of narr:refersTo is class narr:MOfragment
	The range of narr:refersTo is class ecrm:E5_Event

The connection between a media object and anyone 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:MObject
	The range of narr:hasFragment is class narr:MOfragment

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:Narration
	The range of narr:hasContent is class narr:MObject

The connection between a presentation and anyone of its segments (see below) is captured by property **narr:hasPresentationSegment**. Also this is a composition property, thus we make it a sub-property of **ecrm:P148_has_component**.



	narr:hasPresentationSegment is a property
	narr:hasPresentationSegment is a subproperty of ecrm:P148_has_component
	The domain of narr:hasPresentationSegment is class narr:Presentation
	The range of narr:hasPresentationSegment is class narr:PresentationSegment

The connection between a presentation segment and a media object fragment is captured by property **narr:refersToMOFragment**. Since the latter is the content of the former, **narr:refersToMOFragment** is a sub-property of **ecrm:P129i_is_subject_of**, similarly to property **narr:hasContent**.

	narr:refersToMOFragment is a property
	narr:refersToMOFragment is a subproperty of ecrm:P129i_is_subject_of
	The domain of narr:refersToMOFragment is class narr:PresentationSegment
	The range of narr:refersToMOFragment is class narr:MOFragment

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**.

	narr:causallyDependsOn is a property
--	---



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

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 CRAFT 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 CRAFT since CRAFT 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 <code>narr:hadParticipant</code> is class <code>narr:ActorWithRole</code>
	<code>narr:hasSubject</code> is an object property
	<code>narr:hasSubject</code> is a subproperty of <code>ecrm:P148_has_component</code>
	The domain of <code>narr:hasSubject</code> is class <code>narr:ActorWithRole</code>
	The range of <code>narr:hasSubject</code> is class <code>ecrm:E39_Actor</code>
	<code>narr:hasRole</code> is an object property
	<code>narr:hasRole</code> is a subproperty of <code>ecrm:P148_has_component</code>
	The domain of <code>narr:hasRole</code> is class <code>narr:ActorWithRole</code>
	The range of <code>narr:hasRole</code> is class <code>narr:Role</code>

2.3.4. Modelling presentation segments

Presentation segments are parts of presentations that have content. They 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”*.

The corresponding axioms are not reported for brevity.

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`.

	<code>narr:TimePoint</code> 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 the 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

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



points:

Table 2 Interval temporal properties from the CRM

	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

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

⁶ <https://www.ogc.org/>

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

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 **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**.

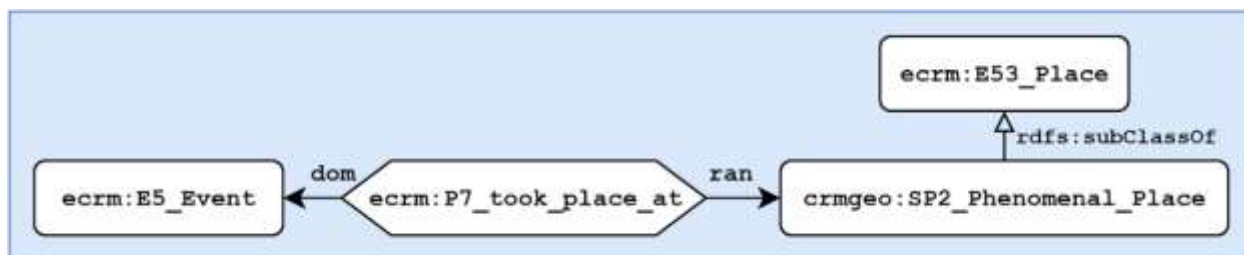


Figure 2 The classes of CRMgeo SP2 Phenomenal Place and its link with CIDOC CRM.

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;

⁸ <https://www.ogc.org/standard/gml/>

- **geosparql:hasGeometry**, which links an instance of class **geosparql:Feature** with an instance of class **geosparql:Geometry** that represents its spatial extent;

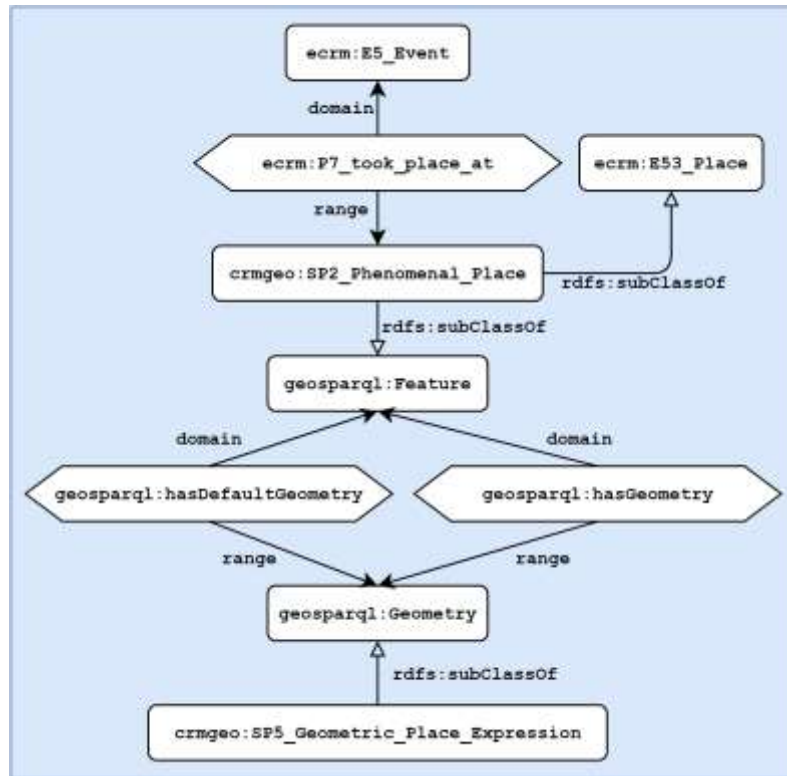


Figure 3 Relation between CRMgeo and Geo SPARQL classes.

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

An instance of `crmgeo:SP4_Spatial_Coordinate_Reference_System` is linked to an instance of `crmgeo:SP3 Reference Space` through property `crmgeo:Q7` describes.

An instance of `crmgeo: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.

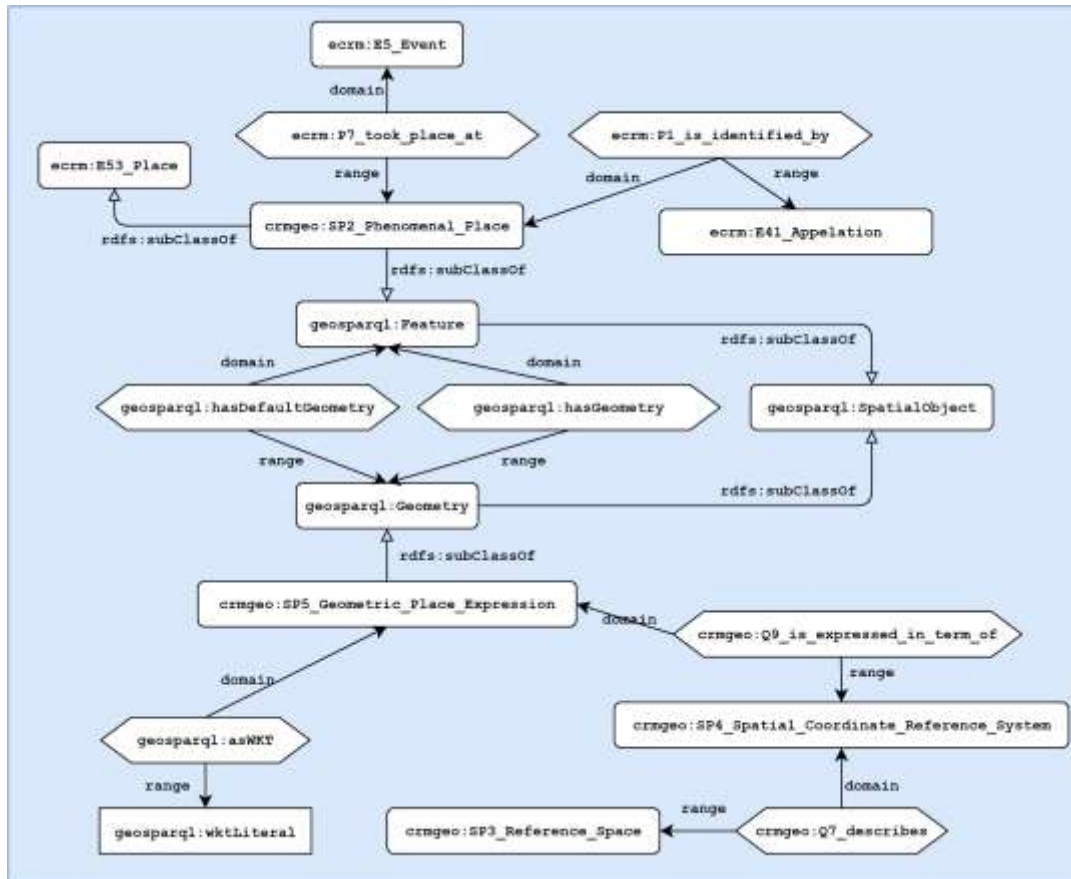


Figure 4 The classes and properties used to represent geospatial knowledge.

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-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).

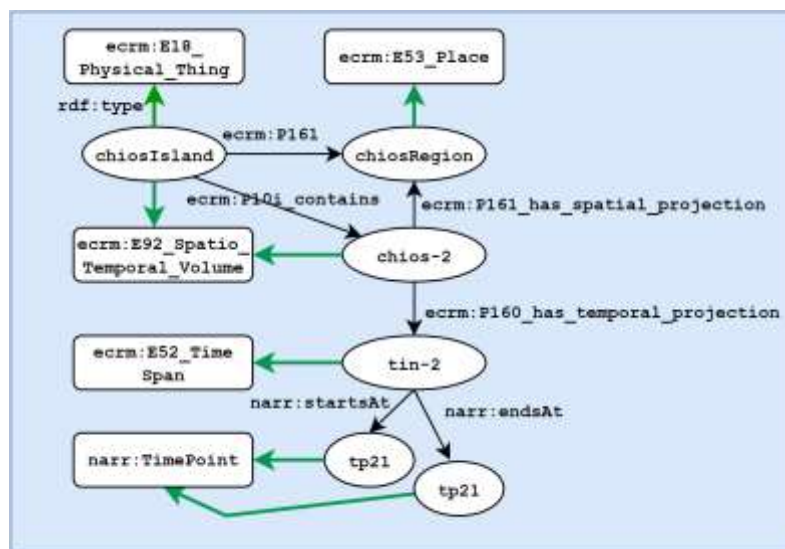


Figure 5 Chios and its time-varying properties.

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 and actions

2.4.1 Processes

A process is composed of interconnected *steps*. A step can be either an *action* performed by one or more persons in the context of a craft, or something that happens without the direct intervention of the persons involved in crafts, such as the growth of a plant or the cooling of a piece of material. We will use the term “process event” for any phenomenon of the latter kind, to distinguish these phenomena from the much



more general and wider phenomena that are meant by the term “event” as it has been used thus far for instance in narratives.

The steps in a process may be interconnected in many different ways: they may be serial or parallel; they may be performed subject to the termination of other steps, or the occurrence of certain conditions, and more. In the Mingei project, craft processes are modelled as (simplified) Unified Modelling Language (UML, for short) activities. As the Mingei model proved adequate, the CRAEFT ontology will be an extension of the Mingei ontology standing on the same basis. Accordingly, we will consider a process structured as a directed hyper-graph whose nodes are called *activity* nodes and whose hyper-edges are called *transitions*.

An activity node represents a piece of the actual work performed to the end of producing the process output. An activity node may be a step, as defined above; or it may be a process in its own right, in fact, a sub-process of the process where the node belongs. As customary, we will use two special action nodes, the *initial* node and the *final* node, to properly represent the beginning and the end of processes. Note that since a process may have sub-processes as parts, initial and final nodes may occur anywhere in a process hypergraph.

A transition is used to manage the flow of execution between activity nodes, using the term “transition tail” (or simply “tail”) for any node where the transition hyperedge originates and “transition head” (or just “head”) for any node where the transition ends

We 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 activity; 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 activity; a join has two or more tails and one head.

Formally, the hypergraph of a process has a unique source (*i.e.*, exactly one node with no incoming hyperedges), a unique sink (*i.e.*, exactly one node with no outgoing hyperedges) and a single component, *i.e.*, every node of the hypergraph is reachable from the source.

Activity nodes and transitions have an internal structure, illustrated in the remaining subsections of this Section. For activity nodes, we will concentrate on actions, as events can be seen as very simple kinds of actions.

2.4.2. Actions

In the Mingei ontology, actions were symbols with no further representation. In contrast, the CRAEFT ontology represents actions in an articulated way, to provide a richer representation of crafts.



Generally, we view an action as an application of forces to some objects, performed by one or more persons possibly with the aid of specific tools. From an ontological point of view, actions are temporal entities called *perdurants* (for more details on perdurants, see for instance [15]); more specifically, actions are events 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. As the representation of incorrect actions is not relevant for the present purposes, they will be disregarded, by using the term "action" as a synonym for "correct action", unless explicitly specified.

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 performed 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. Uncertainty will be captured in action schemas, introduced below.

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*.

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;
- 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.

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 force with which the hammer is driven, 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.

It is important to keep in mind 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.

2.4.3. Schemas

Processes and their components are described by *schemas*. 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. Instances provide 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 CRAFT, 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 aspects of processes that have been identified above. Thus, a process schema is a directed hypergraph composed of activity node schemas and transition schemas, also following the structures identified above. In particular,

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

An action schema describes the types of entities involved in the action, each with its role. In addition, a schema should also give the action function. The schema of a simple action, resulting in no creation but just in a simple transformation, such as heating a piece of glass, is straightforward and its action function can be expressed analytically as a mathematical formula yielding the temperature as a function over time. However, realistic actions imply creations and complex transformations that cannot be expressed analytically but can only be computed via simulation, once the inputs of the action, the causing entities and the initial state of affected entities, are given. For representing complex action schemas, the CrO adopts a twofold strategy:

1. Action schemas are categorised based on the entities they involve and create, and on the action function, expressing the transformation that instance actions produce. That is, all (and only the) actions classified into the same schema have the same involved and created entity types and their transformations are applications of the same action function.
2. The action function is represented *procedurally*, i.e., as a piece of code of some programming language that computes the output on a given input by simulation.

As already pointed out, an *instance* of an action schema is an action that satisfies two conditions:

1. It has caused, created, and affected entities, entities of the type described by the schema.
2. The transformation from the initial to the final states of affected entities is an application of the action function associated with the schema.

Such an instance is called *virtual action* because it is not performed by human beings in a certain place at a certain time, but only an abstraction simulated by a computational agent in abstract space and time. A virtual action can then be *enacted* by an agent that performs it in a real setting, that is, at a given place and at a given time. A virtual action can of course be enacted any number of times, each performed by an agent and situated in its spatio-temporal region, different from the spatio-temporal region of all other actions.

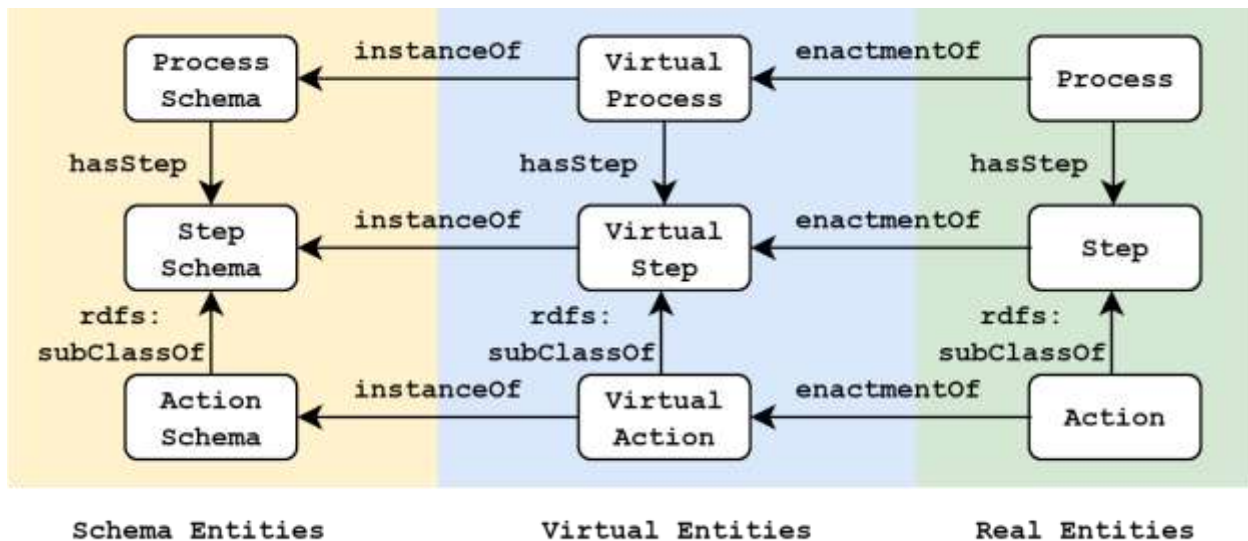
The same instantiation/enactment method can be applied to process schemas: by instantiating all action and transition schemas in a process schema, a *virtual process* is obtained, which can then be enacted giving rise to a process. In sum,

- 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.

Instantiation is a one-to-many relation: a step schema can be instantiated into any number of virtual steps, each determined by a different set of input parameters of the action function. Consequently, a process schema can be instantiated into any number of virtual processes. Similarly, enactment is a one-to-many relation: a virtual step can be enacted into any number of (real) steps, each determined by a different place and time; consequently, a virtual process can be enacted into any number of (real) processes.

The following diagram illustrates the notions introduced so far.



2.4.4. Transitions

We will illustrate transitions by showing their usage in processes, through a running example. Let us assume we wish to model the schema of a football match that is the final match of a competition, therefore the match has to designate a winner. Each process of this kind can be modelled as having three major sub-processes:

- The first sub-process is regular time, consisting of two activities: first-half and second-half.
- In case regular time ends with a tie, the match enters the over-time sub-process, also divided into the first and second half.
- In case also over-time ends with a tie, the match enters a penalty time sub-process, consisting of two activities: two series of five penalties, one per team, followed, in case of a tie, by an unlimited series of one penalty per team, until a winner is determined.
- When a winner is determined, the match ends.

The process schema following this structure has three therefore major (sub)process schemas: regular time schema, over time schema and penalty time schema, enclosed between initial and final activity nodes, and interconnected by transitions that reflect the semantics of the process. Graphically:

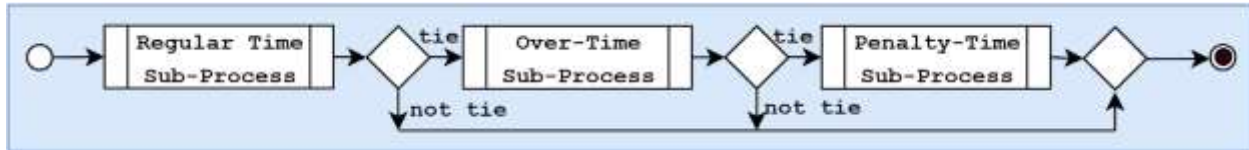


Figure 6 Example process schema.

The transition schema from the initial node to the regular-time sub-process is a simple transition schema, i.e., a hyperedge with just one head node and one tail node. Simple transition schemas are drawn as arrows. To represent the fact that over time only takes place in case regular time ends with a tie, a decision transition schema is used, a hyperedge broken in the middle by a diamond whose incoming arc connects the diamond to the tail node and whose outgoing arcs connect the diamond to the head nodes. Each outgoing arc has a guard condition that is evaluated upon instantiating the schema, and the head node whose guard condition evaluates to true is chosen as the next step of the process. In the example, the decision transition has two head nodes, one guarded by the condition “tie”, and the other guarded by the condition “not tie”. The meaning of transition is evident: if the match is not a tie, it ends; otherwise, the Over Time sub-process, described by the Overt Time Schema, is executed. Upon instantiating the process schema, the transition schema is instantiated as well, except that the instance transition is a simple transition whose only head node corresponds to the guard that evaluates to true. In the example, there are two transition schemas, performing the same test, but at two different stages of the process: the second decision transition schema makes sure that, over time, if the match is not a tie, it ends, otherwise the penalty time sub-process is executed.

Notice that the process schema uses a Merge transition schema to collect all the flows that lead to the final node so that a unique final node is used. A Merge transition schema is drawn exactly as a decision schema, except that it has two or more tails and a single head with no guarding predicate. Upon instantiating the process, the merge transition is instantiated as well. Upon instantiating the process, a Merge transition schema is instantiated by a simple transition, since a single path is always chosen.

The Regular Time and Over Time process Schemas have a very similar and very simple structure: the connections between their activity nodes are just simple transitions, as Figure 7 below shows.

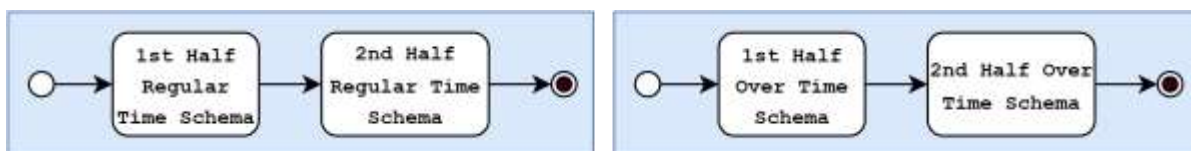


Figure 7 Regular-time and over-time process schemas.

A Penalty Time sub-process Schema has the structure given in Figure 8, whose meaning should be evident:

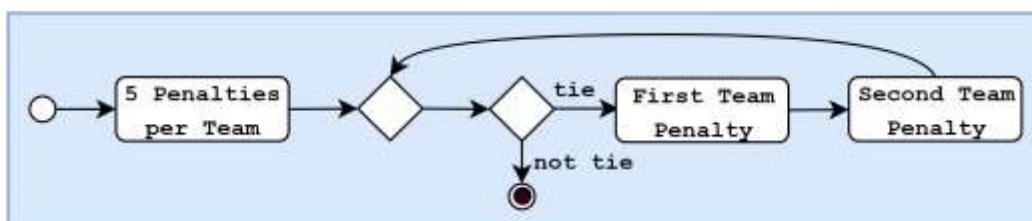


Figure 8 Penalty-time process schema.

Sub-processes connect to the nodes of the schema where they belong in the obvious way: the preceding node connects to the initial node of the sub-process, which is always unique; the following node connects to the final node of the sub-process, which is also always unique.

To specify concurrent flows, synchronisation transition schemas are used. There are two kinds of synchronisations: forks and joins.

- A fork represents the splitting of a single flow into two or more concurrent flows. Consequently, a fork transition schema has always one tail and at least two heads, each of which represents an independent flow. The same structure is preserved in instances since concurrent flows are supposed to be *all* executed simultaneously; hence fork transition schemas are instantiated by fork transitions. Fork transition schemas (and their instances) are drawn as horizontal bars annotated with the word “Fork”: the tail connects to the bar by an arc, while the bar is connected to the heads by arcs (see figure below).
- Conversely, a join represents the synchronisation of two or more concurrent flows. Consequently, a join transition schema has two or more tails, each of which ends an independent flow, and one head node. The same structure is replicated at the instance level, for the reasons pointed out for forks. Join transition schemas are drawn as bars, similar to fork nodes, as shown in Figure 9.

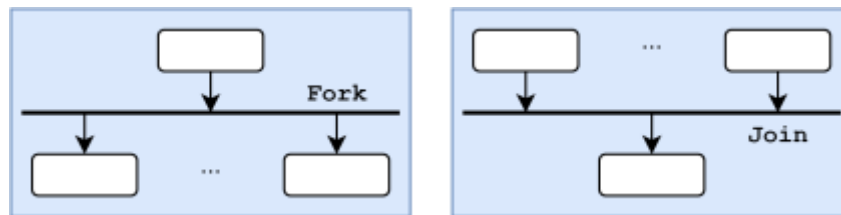


Figure 9 Fork and Join transition schemas.

The following figure presents an instance of the example process schema, a final match between the Italian teams Milan and Inter, ending regular time with a tie and decided over time. Consequently, the “tie” branch of the first decision transition schema is instantiated, leading to the instantiation of the over time sub-process, and the “not tie” branch of the second decision transition schema is instantiated leading to the final node, and the penalty time sub-process is not instantiated. Notice that the final Merge node is instantiated as a simple transition, shown in Figure 10 as a single arrow. For brevity, we do not present the instantiation of the two sub-processes.

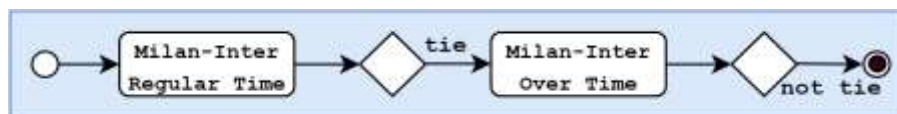


Figure 10 A process instance.

2.5. An ontology of processes and actions

Process hypergraphs are represented in the CrO by two kinds of classes:

- the classes for nodes, holding the knowledge about activities, and



- the classes for transitions, holding the knowledge about connections amongst activities and the flow control of activities.

These classes will be illustrated in the two subsections of this section, each time starting from those relative to schemas and then moving on to those relative to schemas' instances and finally to real ones.

2.5.1. Activities

2.5.1.1. Activity Schemas

Process schemas are instances of the class **craeft:ProcessSchema**, while activity node schemas are instances of the class **craeft:StepSchema**. As explained in the previous Section, an instance of **craeft:StepSchema** can be one of the following:

- event schema, an instance of class **craeft:EventSchema**, a subclass of **craeft:StepSchema**;
- action schema, an instance of class **craeft:ActionSchema**, a subclass of **craeft:StepSchema**;
- process schema, an instance of class **craeft:ProcessSchema**, a subclass of **craeft:StepSchema**.

Notice that **craeft:StepSchema** is the most general class of schemas, having **craeft:ProcessSchema** as a subclass, to enable inclusion of processes as parts of other processes. This implies that process schemas that are *not* part of any other process schema are anyway instances of **craeft:StepSchema**. This poses no problem, as these schemas can be identified by inspecting the process composition property **craeft:hasStep**, introduced below.

Process schemas are kinds of *plans*, that is, descriptions of structured courses of events and actions. The CIDOC CRM offers the class **ecrm:E29_Design_or_Procedure** that “*comprises documented plans for the execution of actions in order 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 **craeft:StepSchema** and its subclasses are subclasses of **ecrm:E29_Design_or_Procedure**.

	craeft:StepSchema is a class
	craeft:StepSchema is a subclass of ecrm:E29_Design_or_Procedure
	craeft:EventSchema is a class
	craeft:EventSchema is a subclass of craeft:StepSchema
	craeft:ActionSchema is a class
	craeft:ActionSchema is a subclass of craeft:StepSchema
	craeft:ProcessSchema is a class
	craeft:ProcessSchema is a subclass of craeft:StepSchema



To connect a process schema to the steps it consists of, and, recursively, a sub-process schema to its sub-steps, we introduce the property **craeft:hasStep** and its inverse **craeft:isStepOf**. For economy and conceptual homogeneity, **craeft:hasStep** is also used for the composition of virtual and real processes, using localisation axioms. This allows us having a single property for multiple, but analogous, uses, even though it prevents direct mapping of **craeft:hasStep** to a property of CRM, as the CRM introduces three properties for part-whole relationships:

- **ecrm:P69_has_association_with**, for part-whole relationships between instances of **ecrm:E29_Design_or_Procedure**;
- **ecrm:P148_has_component** for part-whole relationships between instances of **ecrm:E89_Propositional_Object**; and
- **ecrm:P5_consists_of** for part-whole relationships between instances of **ecrm:E3_Condition_State**.

We should therefore make both **craeft:hasStep** and **craeft:isStepOf** sub-properties of a disjunction of three CRM properties. But this is not supported by OWL 2 DL, which only provides object and data properties as property expressions. In spite of this inconvenience, which applies also to the properties for the composition of transitions (see Section [2.5.2. Transitions](#)), the CrO prefers to use a single property.

	craeft:hasStep is an object property
	craeft:isStepOf is an object property
	craeft:isStepOf is inverse property of craeft:hasStep

The localisation axioms given next capture the fact that a process schema has always at least one schema step. The converse does not hold for process schemas, as not every process schema is a sub-process schema of another process schema, therefore the converse axioms are stated individually for each subclass of **craeft:SchemaStep**.

	An instance of craeft:ProcessSchema is connected by craeft:hasStep only to instances of craeft:StepSchema
	An instance of craeft:ProcessSchema is connected by craeft:hasStep to at least one instance of craeft:StepSchema
	An instance of StepSchema is connected by craeft:isStepOf only to instances of craeft:ProcessSchema
	An instance of craeft:EventSchema is connected by craeft:isStepOf to exactly one instance of craeft:ProcessSchema
	An instance of craeft:ActionSchema is connected by craeft:isStepOf to exactly one instance of craeft:ProcessSchema
	An instance of craeft:ProcessSchema is connected by craeft:isStepOf to at most one instance of craeft:ProcessSchema

2.5.1.2. Virtual activities

The structure of process schemas is replicated for virtual processes, therefore **craeft:VirtualStep** is the class of steps of virtual processes, further specialised in **craeft:VirtualEvent**, **craeft:VirtualAction** and **craeft:VirtualProcess**. All instances of these classes are seen as plans, similar to their corresponding schema, except those virtual entities are more complete plans. As a consequence **VirtualStep** is a subclass of **ecrm:E29_Design_or_Procedure**.

	craeft:VirtualStep is a class
	craeft:VirtualStep is a subclass of ecrm:E29_Design_or_Procedure
	craeft:VirtualEvent is a class
	craeft:VirtualEvent is a subclass of craeft:VirtualStep
	craeft:VirtualAction is a class
	craeft:VirtualAction is a subclass of craeft:VirtualStep
	craeft:VirtualProcess is a class
	craeft:VirtualProcess is a subclass of craeft:VirtualStep

Properties **craeft:hasStep** and **craeft:isStepOf** are re-used for connecting virtual processes to their steps, by localising it via apposite axioms, given next.

	An instance of craeft:VirtualProcess is connected by craeft:hasStep only to instances of craeft:VirtualStep
	An instance of craeft:VirtualProcess is connected by craeft:hasStep to at least one instance of craeft:VirtualStep
	An instance of craeft:VirtualStep is connected by craeft:isStepOf only to instances of craeft:VirtualProcess
	An instance of craeft:VirtualEvent is connected by craeft:isStepOf to exactly one instance of craeft:VirtualProcess
	An instance of craeft:VirtualAction is connected by craeft:isStepOf to exactly one instance of craeft:VirtualProcess
	An instance of craeft:VirtualProcess is connected by craeft:isStepOf to at most one instance of craeft:VirtualProcess

2.5.1.3. Physical activities

Finally, we axiomatise physical processes following the same pattern. As already pointed out, **craeft:Step** is a subclass of **narr:Fabula**.



	craeft:Step is a class
	craeft:Step is a subclass of narr:Fabula
	craeft:ProcessEvent is a class
	craeft:ProcessEvent is a subclass of craeft:Step
	craeft:Action is a class
	craeft:Action is a subclass of craeft:Step
	craeft:Process is a class
	craeft:Process is a subclass of craeft:Step

Axioms for the localisation of **craeft:hasStep** and **craeft:isStepOf** to physical processes are given next.

	An instance of craeft:Process is connected by craeft:hasStep only to instances of craeft:Step
	An instance of craeft:Process is connected by craeft:hasStep to at least one instance of craeft:Step
	An instance of craeft:Step is connected by craeft:isStepOf only to instances of craeft:Process
	An instance of craeft:ProcessEvent is connected by craeft:isStepOf to exactly one instance of craeft:Process
	An instance of craeft:Action is connected by craeft:isStepOf to exactly one instance of craeft:Process
	An instance of craeft:Process is connected by the inverse of craeft:hasStep to at most one instance of craeft:Process

2.5.1.4. Instantiation

We now axiomatize property **craeft:instanceOf**, which represents the instantiation relation between virtual activities and the corresponding schemas. For convenience, we introduce also the inverse property of **craeft:instanceOf**, **craeft:hasInstance**. These properties will also be used for transitions, via the appropriate localisation axioms.

The best approximation that can be found in the CRM for this property, is property **ecrm:P67_refers_to**, which “documents that an instance of E89 Propositional Object makes a statement about an instance of E1 CRM Entity”. Indeed, a virtual step is an instance of **ecrm:E73_Information_Object** and therefore of **ecrm:E89_Propositional_Object**.



	craeft:instanceOf is an object property
	craeft:instanceOf is a sub-property of ecrm:P67_refers_to
	craeft:hasInstance is an object property
	craeft:hasInstance is an inverse property of craeft:instanceOf

The following localisation axioms apply to the instantiation property.

	An instance of craeft:VirtualProcess is connected by craeft:instanceOf only to instances of craeft:ProcessSchema
	An instance of craeft:VirtualProcess is connected by craeft:instanceOf to exactly one instance of craeft:ProcessSchema
	An instance of craeft:ProcessSchema is connected by craeft:hasInstance only to instances of craeft:VirtualProcess
	An instance of craeft:ProcessSchema is connected by craeft:hasInstance to at least one instance of craeft:VirtualProcess
	An instance of craeft:VirtualAction is connected by craeft:instanceOf only to instances of craeft:ActionSchema
	An instance of craeft:VirtualAction is connected by craeft:instanceOf to exactly one instance of craeft:ActionSchema
	An instance of craeft:ActionSchema is connected by craeft:hasInstance only to instances of craeft:VirtualAction
	An instance of craeft:ActionSchema is connected by craeft:hasInstance to at least one instance of craeft:VirtualAction
	An instance of craeft:VirtualEvent is connected by craeft:instanceOf only to instances of craeft:EventSchema
	An instance of craeft:VirtualEvent is connected by craeft:instanceOf to exactly one instance of craeft:EventSchema
	An instance of craeft:EventSchema is connected by craeft:hasInstance only to instances of craeft:VirtualEvent
	An instance of craeft:EventSchema is connected by craeft:hasInstance to at least one instance of craeft:VirtualEvent

The axiom that models 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,



1. *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.*
2. *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 in fact 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:

Let VP be a virtual process and VS be a virtual 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:

	The chain formed by craeft:hasStep and craeft:instanceOf and craeft:isStepOf is a subproperty of craeft:instanceOf
--	--

Unfortunately, this axiom makes **craeft: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 so lost knowledge is recovered procedurally

An important axiom on instantiation concerns 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. In particular, any property declared in an action schema via anyone of **craeft:hasCausingEntityProperties**, **craeft:hasAffectedEntityProperties**, **craeft:hasCeasedEntityProperties** or **craeft:hasCreatedEntityProperties** must be used in any instance of the schema in the appropriate way. However, this axiom cannot be expressed in OWL 2 DL.

2.5.1.5. Enactment

We finally axiomatize enactment by property **craeft:enactmentOf**, which represents the enactment relation between real activities and the corresponding virtual activities. For convenience, we introduce also the inverse property of **craeft:enactmentOf**, **craeft:isEnactedby**. These properties will also be used for transitions, via the appropriate localisation axioms.

The CRM offers property **ecrm:P33_used_specific_technique**, which “*identifies a specific instance of E29 Design or Procedure in order to carry out an instance of E7 Activity or parts of it*”. We therefore make **enactmentOf** a subproperty of **ecrm:P33_used_specific_technique**. Indeed, a virtual step is an instance of **ecrm:E29_Design_or_Procedure**, while a step is an instance of **narr:Fabula** and therefore of **ecrm:E7_Activity**.

	craeft:enactmentOf is an object property
	craeft:enactmentOf is a sub-property of ecrm:P33_used_specific_technique



	craeft:isEnactedby is an object property
	craeft:isEnactedby is an inverse property of craeft: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 localisation axioms apply to the enactment property.

	An instance of craeft:Process is connected by craeft:enactmentOf only to instances of craeft:VirtualProcess
	An instance of craeft:Process is connected by craeft:enactmentOf to exactly one instance of craeft:VirtualProcess
	An instance of craeft:VirtualProcess is connected by craeft:isEnactedby only to instances of craeft:Process
	An instance of craeft:VirtualProcess is connected by craeft:isEnactedby to at least one instance of craeft:Process
	An instance of craeft:Action is connected by craeft:enactmentOf only to instances of craeft:VirtualAction
	An instance of craeft:Action is connected by craeft:enactmentOf to exactly one instance of craeft:VirtualAction
	An instance of craeft:VirtualAction is connected by craeft:isEnactedby only to instances of craeft:Action
	An instance of craeft:VirtualAction is connected by craeft:isEnactedby to at least one instance of craeft:Action
	An instance of craeft:ProcessEvent is connected by craeft:enactmentOf only to instances of craeft:VirtualEvent
	An instance of craeft:ProcessEvent is connected by craeft:enactmentOf to exactly one instance of craeft:VirtualEvent
	An instance of craeft:VirtualEvent is connected by craeft:isEnactedby only to instances of craeft:ProcessEvent
	An instance of craeft:VirtualEvent is connected by craeft:isEnactedby to at least one instance of craeft:ProcessEvent



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,

1. *For every step S that is a step of P , there exists a virtual step VS that is a step of VP , such that S is an enactment of VS .*
2. *Conversely, for every virtual step VS that is a step of VP , there exists a step S that is a step 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:

	The chain formed by craeft:hasStep and craeft:enactmentOf and craeft:isStepOf is a subproperty of craeft:enactmentOf
--	--

Unfortunately, this axiom makes **craeft: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 axiom is not included in the CrO.

2.5.1.6. Objects

Objects play a relevant role in actions, as they are the entities that actions create or transform. In the CRM, class **ecrm:E19_Physical_Object** “*comprises items of a material nature that are units for documentation and have physical boundaries that separate them completely in an objective way from other objects*”. This definition captures the objects involved in crafts, which may be human-made (instances of **ecrm:E22_Human-Made_Object**) or natural, therefore we will reuse it in the CrO to classify any object that is involved in an action. Classes for specific kinds of objects, such as hammers or chisels, will be introduced in due course upon modelling the actions in which these objects are involved. These classes will all be subclasses of **ecrm:E19_Physical_Object**. In addition, we introduce:

- the class **craeft:CompositeObject**, a subclass of **ecrm:E22_Human-Made_Object**, having as instances the objects created in actions by assembling in some way other objects. Even though the way of creating those objects varies from action to action, the objects created are all instances of **craeft:CompositeObject**. Also, the property for associating a composite object to its component is taken directly from the CRM, and is the property **ecrm:P46_is_composed_of**, which “*associates an instance of E18 Physical Thing with another instance of E18 Physical Thing that forms part of it*”. Notice that **ecrm:E22_Human-Made_Object** is a subclass of **ecrm:E18_Physical_Thing**.



	craeft:CompositeObject is a class
	craeft:CompositeObject is a subclass of ecrm:E22_Human-Made_Object

- The class **craeft:RigidObject**, a subclass of **ecrm:E22_Human-Made_Object**, having as instances the objects made of rigid materials, and its complement **craeft:DeformableObject**, also a subclass of **ecrm:E22_Human-Made_Object**, having as instances the objects made of deformable materials. This distinction must be communicated to the simulator, so it is a requirement that the ontology can make it. These two classes are disjoint. The property for associating an object to its material (instance of class **ecrm:E57_Material1**) is taken directly from the CRM and it is the property **ecrm:P45_consists_of**.

	craeft:RigidObject is a class
	craeft:RigidObject is a subclass of ecrm:E22_Human-Made_Object
	craeft:DeformableObject is a class
	craeft:DeformableObject is a subclass of ecrm:E22_Human-Made_Object
	craeft:Class RigidObject is disjoint from class craeft:DeformableObject

The time-independent characteristics of objects that are relevant to the present purposes are material properties, which include several properties, depending on the kind of object they apply to. In addition, shape is a time-independent property of rigid objects, while the same does not apply to deformable objects. To represent the connection between an object and its time-independent characteristics, we introduce two properties, **craeft:hasMaterialProperties** and **craeft:hasShape**, the domain of the former being **ecrm:E18_Physical_Thing**, while that of the latter is **craeft:RigidObject**. The range of these properties is the CRM class **ecrm:E31_Document**, which “*comprises identifiable immaterial items that make propositions about reality. These propositions may be expressed in text, graphics, images, audiograms, videograms or by other similar means*”. The reason for this choice is that time-independent characteristics of objects are exclusively used to be produced or consumed by the simulator, therefore the CrO just provides the machinery to hold documents where these characteristics are held, in a format that is appropriate for the simulator. For this reason, both **craeft:hasMaterialProperties** and **craeft:hasShape** are sub-properties of the CRM 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 any individual, while its range is **ecrm:E31_Document**. For the same reason, both properties are mandatory and functional.

	craeft:hasShape is an object property
	craeft:hasShape is a sub-property of ecrm:P70i_is_documented_in
	The domain of craeft:hasShape is class craeft:RigidObject
	The range of craeft:hasShape is class ecrm:E31_Document



	An instance of craeft:RigidObject is connected by property craeft:hasShape to exactly one instance of ecrm:E31_Document
	craeft:hasMaterialProperties is an object property
	craeft:hasMaterialProperties is a sub-property of ecrm:P70i_is_documented_in
	The domain of craeft:hasMaterialProperties is class ecrm:E18_Physical_Thing
	The range of craeft:hasMaterialProperties is class ecrm:E31_Document
	An instance of ecrm:E18_Physical_Thing is connected by property craeft:hasMaterialProperties to exactly one instance of ecrm:E31_Document

To represent object states, holding the time-dependent characteristics of objects, we introduce the generic class **craeft:State**, regardless of the object's type. **craeft:State** is a subclass of the CRM class **ecrm:E3_Condition_State**, which “*comprises the states of objects characterised by a certain condition over a time-span*”.

	craeft:State is a class
	craeft:State is a subclass of ecrm:E3_Condition_State

Any object is linked to its state through the property **hasState**, a subproperty of **ecrm:P44_has_condition**. The inverse property of **hasState**, **isStateOf**, is also introduced for convenience. An object can have zero, one or many states, while a state is the state of exactly one object.

	craeft:hasState is a subproperty of ecrm:P44_has_condition
	The domain of craeft:hasState is ecrm:E18_Physical_Object
	The range of craeft:hasState is craeft:State
	craeft:isStateOf is an object property
	craeft:isStateOf is the inverse of property craeft:hasState
	An instance of craeft:State is connected by property craeft:isStateOf to exactly one instance of ecrm:E18_Physical_Object

Similarly to time-independent characteristics of objects, object states are no further represented in the CrO, therefore we use the CRM class **ecrm:E31_Document** to hold the states of objects and the CRM property **ecrm:P70i_is_documented_in**, to link a state to the information resource holding an appropriate representation of its value.

2.5.1.7. Actions

An action is typed in three different ways (see Figure 11⁹):

- By a function in a standardised vocabulary of functions; the vocabulary is captured by class **craeft:ActionFunction**, subclass of **ecrm:E55_Type**; these individuals are linked to the action by using the property **craeft: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 **craeft:ActionType** that is a subclass of **ecrm:E55_Type**; these individuals are linked to the action by using the property **craeft: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 **craeft:hasMachineType**. The Archimedean simple machines are individuals, instances of the class **craeft:MachineType** that is a subclass of **ecrm:E55_Type**.

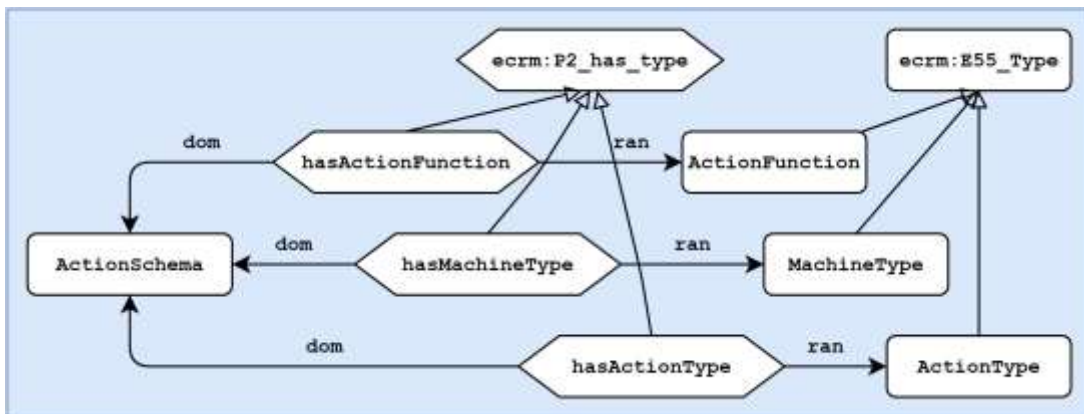


Figure 11 Typing of actions.

These properties are all subproperties of **ecrm:P2_has_type** and the same value for all virtual actions that are instances of the same action schema and, consequently, for all real actions that enact these virtual actions. Therefore, they are associated with a schema action and not replicated in virtual and (real) actions.

	craeft:ActionFunction is a class
	craeft:ActionFunction is a subclass of ecrm:E55_Type
	craeft:eafunction₁ is an instance of the class craeft:ActionFunction
	...
	craeft:eafunction_N is an instance of the class craeft:ActionFunction

⁹ For readability, the prefix **craeft:** is omitted in all Figures.



	craeft:hasActionFunction is an object property
	craeft:hasActionFunction is a subproperty of ecrm:P2_has_type
	The domain of craeft:hasActionFunction is class craeft:ActionSchema
	The range of craeft:hasActionFunction is class craeft:ActionFunction
	craeft:ActionType is a class
	craeft:ActionType is a subclass of ecrm:E55_Type
	craeft:add is an instance of class craeft:ActionType
	craeft:subtract is an instance of class craeft:ActionType
	craeft:interlock is an instance of class craeft:ActionType
	craeft:transform is an instance of class craeft:ActionType
	craeft:hasActionType is an object property
	craeft:hasActionType is a subproperty of ecrm:P2_has_type
	The domain of craeft:hasActionType is class craeft:ActionSchema
	The range of craeft:hasActionType is class craeft:ActionType
	craeft:MachineType is a class
	craeft:MachineType is a subclass of ecrm:E55_Type
	craeft:leverMachineType is an instance of the class craeft:MachineType
	craeft:wedgeMachineType is an instance of the class craeft:MachineType
	craeft:ScrewMachineType is an instance of the class craeft:MachineType
	craeft:InclinedPlanMachineType is an instance of class craeft:MachineType
	craeft:PulleyMachineType is an instance of the class craeft:MachineType
	craeft:WheelAxleMachineType is an instance of the class craeft:MachineType
	craeft:PhysicalAgent is an instance of the class craeft:MachineType
	craeft:ChemicalAgent is an instance of the class craeft:MachineType
	hasMach craeft:ineType is an object property
	craeft:hasMachineType is a subproperty of ecrm:P2_has_type

	The domain of craeft:hasMachineType is class craeft:ActionSchema
	The range of craeft:hasMachineType is class craeft:MachineType

In addition to this characterisation, an action schema gives the entities involved in the action function. Since these entities vary from action to action and are not known *a priori*, action schemas use properties to achieve their goal. Specifically:

- Property **craeft:hasCausingEntityProperties** is used for indicating the properties of any virtual action that instantiates the schema giving the causing entities. For instance, since the action of hitting a chisel with a hammer has just one causing entity, *i.e.*, the force applied to the hammer, then the corresponding action schema has just one property associated with it by the **craeft:hasCausingEntityProperties** property, say **craeft:hasForce**, that associates any virtual action that instantiates the schema with a force. An action has at least one causing entity, so property **craeft:hasCausingEntityProperties** is mandatory.
- Property **craeft:hasAffectedEntityProperties** is used similarly for indicating the properties of any virtual action that instantiates the schema giving the affected entities. For instance, since the action of hitting a chisel with a hammer has three affected entities, *i.e.*, the hammer, the chisel and the piece of wood, then the corresponding action schema has three properties associated with it by the **craeft:hasTransformedEntityProperties** property, say **craeft:hasHammer**, **craeft:hasChisel** and **craeft:hasPieceOfWood**, that associates any virtual action that instantiates the schema with the hammer, the chisel and the piece of wood that are used in the action, respectively. An action may have zero, one or more affected entities.

These properties have **craeft:ActionSchema** as domain and **rdf:Property** as range (see Figure 12).



Figure 12 Domain and range of Action Schema properties.

The association between a virtual action and its newly created entities is captured by a single property, named **produces**, used in all action schemas.

	craeft:hasCausingEntityProperties is an object property
	The domain of craeft:hasCausingEntityProperties is class craeft:ActionSchema
	The range of craeft:hasCausingEntityProperties is class rdf:Property



	An instance of craeft:ActionSchema is connected by craeft:hasCausingEntityProperties to at least one instance of class rdf:Property
	craeft:hasAffectedEntityProperties is an object property
	The domain of craeft:hasAffectedEntityProperties is class craeft:ActionSchema
	The range of craeft:hasAffectedEntityProperties is class rdf:Property
	craeft:produces is an object property
	The domain of craeft:produces is class craeft:VirtualAction
	The range of craeft:produces is class ecrm:E19_Physical_Object

A virtual action is represented using the properties specified in the corresponding action schema. In addition to those, a virtual action is associated with the affected entities' states before and after the action. To this end, properties **craeft:AESBefore** and **craeft:AESAAfter** are introduced in the CrO, both having **craeft:VirtualAction** as domain and **craeft:State** as range. Each state associated with a virtual action via any one of these properties is also associated with the entity it is a state of via property **craeft:isStateOf**, introduced in the previous Section.

	craeft:AESBefore is an object property
	The domain of craeft:AESBefore is class craeft:VirtualAction
	The range of craeft:AESBefore is class craeft:State
	craeft:AESAAfter is an object property
	The domain of craeft:AESAAfter is class craeft:VirtualAction
	The range of craeft:AESAAfter is class craeft:State

The representation of (real) actions does not require the introduction of any additional class or property, since action is an element of the fabula of a narrative, and the ontology of narratives introduced in Section [2.3. The Narrative Ontology](#) provides the necessary machinery for representing actions.

2.5.2. Transitions

Transition schemas are instances of class **craeft:TransitionSchema**. **craeft:TransitionSchema** has the following subclasses, capturing schemas of the kind of transitions introduced in the previous Section:

- **craeft:SimpleTransitionSchema**,
- **craeft:DecisionTransitionSchema**,
- **craeft:MergeTransitionSchema**,
- **craeft:ForkTransitionSchema** and



- **craeft:JoinTransitionSchema**.

Also transitions schemas are plans, therefore **craeft:TransitionSchema** is a subclass of **ecrm:E29_Design_or_Procedure**.

	craeft:TransitionSchema is a class
	craeft:TransitionSchema is a subclass of ecrm:E29_Design_or_Procedure
	craeft:SimpleTransitionSchema is a class
	craeft:SimpleTransitionSchema is a subclass of craeft:TransitionSchema
	craeft:DecisionTransitionSchema is a class
	craeft:DecisionTransitionSchema is a subclass of craeft:TransitionSchema
	craeft:MergeTransitionSchema is a class
	craeft:MergeTransitionSchema is a subclass of craeft:TransitionSchema
	craeft:ForkTransitionSchema is a class
	craeft:ForkTransitionSchema is a subclass of craeft:TransitionSchema
	craeft:JoinTransitionSchema is a class
	craeft:JoinTransitionSchema is a subclass of craeft:TransitionSchema

Similarly, virtual transition classes are introduced to represent the different kinds of virtual transitions that may occur in a virtual process. As pointed out in Section 2, Description and Merge transition schemas are instantiated as simple virtual transitions, therefore there are no classes for Decision virtual transitions and for Merge virtual transitions. For the same reasons given for **craeft:TransitionSchema**, also **craeft:VirtualTransition** is a subclass of **ecrm:E29_Design_or_Procedure**.

	craeft:VirtualTransition is a class
	craeft:VirtualTransition is a subclass of ecrm:E29_Design_or_Procedure
	craeft:SimpleVirtualTransition is a class
	craeft:SimpleVirtualTransition is a subclass of craeft:VirtualTransition
	craeft:ForkVirtualTransition is a class
	craeft:ForkVirtualTransition is a subclass of craeft:VirtualTransition
	craeft:JoinVirtualTransition is a class



	craeft:JoinVirtualTransition is a subclass of craeft:VirtualTransition
--	--

Finally, (real) transition classes are introduced to model transitions occurring in (real) processes. These classes are all subclasses of **ecrm:E5_Event**, since transitions have a more temporally and semantically limited scope than activities: typically, transitions do not extend in time and do not decompose in sub-elements, while processes can do both.

	craeft:Transition is a class
	craeft:Transition is a subclass of ecrm:E5_Event
	craeft:SimpleTransition is a class
	craeft:SimpleTransition is a subclass of craeft:Transition
	craeft:ForkTransition is a class
	craeft:ForkTransition is a subclass of craeft:Transition
	craeft:JoinTransition is a class
	craeft:JoinTransition is a subclass of craeft:Transition

The following axioms localise instantiation properties **craeft:instanceOf** and **craeft:hasInstance** to the various kinds of transitions.

	An instance of craeft:SimpleVirtualTransition is connected by craeft:instanceOf only to instances of craeft:SimpleTransitionSchema or craeft:DecisionTransitionSchema or craeft:MergeTransitionSchema
	An instance of craeft:SimpleVirtualTransition is connected by instanceOf to exactly one instance of craeft:SimpleTransitionSchema or craeft:DecisionTransitionSchema or craeft:MergeTransitionSchema
	An instance of craeft:SimpleTransitionSchema is connected by craeft:hasInstance only to instances of craeft:SimpleVirtualTransition
	An instance of craeft:SimpleTransitionSchema is connected by craeft:hasInstance to at least one instance of craeft:SimpleVirtualTransition
	An instance of craeft:DecisionTransitionSchema is connected by craeft:hasInstance only to instances of craeft:SimpleVirtualTransition
	An instance of craeft:DecisionTransitionSchema is connected by craeft:hasInstance to at least one instance of craeft:SimpleVirtualTransition
	An instance of craeft:MergeTransitionSchema is connected by craeft:hasInstance only to



	instances of craeft:SimpleVirtualTransition
	An instance of craeft:MergeTransitionSchema is connected by craeft:hasInstance to at least one instance of craeft:SimpleVirtualTransition
	An instance of craeft:ForkVirtualTransition is connected by craeft:instanceOf only to instances of craeft:ForkTransitionSchema
	An instance of craeft:ForkVirtualTransition is connected by craeft:instanceOf to exactly one instance of craeft:ForkTransitionSchema
	An instance of craeft:ForkTransitionSchema is connected by craeft:hasInstance only to instances of craeft:ForkVirtualTransition
	An instance of craeft:ForkTransitionSchema is connected by craeft:hasInstance to at least one instance of craeft:ForkVirtualTransition
	An instance of craeft:JoinVirtualTransition is connected by craeft:instanceOf only to instances of craeft:JoinTransitionSchema
	An instance of craeft:JoinVirtualTransition is connected by craeft:instanceOf to exactly one instance of craeft:JoinTransitionSchema
	An instance of craeft:JoinTransitionSchema is connected by craeft:hasInstance only to instances of craeft:JoinVirtualTransition
	An instance of craeft:JoinTransitionSchema is connected by craeft:hasInstance to at least one instance of craeft:JoinVirtualTransition

The following axioms localise enactment properties **craeft:enactmentOf** and **craeft:isEnactedBy** to the various kinds of transitions.

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

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

We now introduce two general properties 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 the properties **craeft:hasTail** and **craeft:hasHead** to represent the nodes that are in the tail or the head of transitions, respectively. For convenience, we also introduce the inverses **craeft:isTailOf** and **craeft:isHeadOf**. Following the modelling style adopted so far, we will reuse these two properties for transition schemas, and for virtual and real transitions, by giving the appropriate localisation axioms. For the same reasons pointed out for property **craeft:hasStep** (see Section [2.5.1.1. Activity Schemas](#)), it is not possible to map these properties to the CRM. Both **craeft:hasTail** and **craeft:hasHead** are properties giving the composition of transitions. However, they apply to transition schemas and virtual transitions, which are instances of **ecrm:E29_Design_or_procedure**, and to transitions, which are instances of **ecrm:E5_Event**. Now, the CRM uses property **ecrm:P106_is_composed_of** for the composition of symbolic objects, including plans, and **ecrm:P5_consists_of** for the composition of temporal entities, including events. Thus, we should relate **craeft:hasTail** and **craeft:hasHead** to a disjunction of CRM properties; unfortunately, this kind of property expressions are not supported by OWL 2 DL.

Notice that a transition may have another transition as tail or head, as illustrated by the football match example given in Section [2.4.4. Transitions](#). 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.

	craeft:hasHead is an object property
	craeft:isHeadOf is an object property



	craeft:isHeadOf is an inverse property of craeft:hasHead
	craeft:hasTail is an object property
	craeft:isTailOf is an object property
	craeft:isTailOf is an inverse property of craeft:hasTail

The classes and properties specific to transition kinds are introduced next.

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 **craeft:SimpleTransitionSchema**, to instances of **craeft:SimpleVirtualTransition** and to instances of **craeft:SimpleTransition**) and can be modelled by appropriately axiomatizing the properties **craeft:hasHead** and **craeft:hasTail** already introduced. In particular:

- A simple transition schema has exactly one tail and one head, which can be a step schema or a transition schema. Conversely, a step schema or a transition schema can be the tail or the head of at most one transition schema, of whatever kind.

	An instance of craeft:SimpleTransitionSchema is connected craeft:by hasTail only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:SimpleTransitionSchema is connected by craeft:hasTail to exactly one instance of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:StepSchema is connected by craeft:isTailOf to at most one instance of class craeft:TransitionSchema
	An instance of craeft:TransitionSchema is connected by craeft:isTailOf to at most one instance of class craeft:TransitionSchema
	An instance of craeft:SimpleTransitionSchema is connected by craeft:hasHead only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:SimpleTransitionSchema is connected by craeft:hasHead to exactly one instance of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:StepSchema is connected by craeft:isHeadOf to at most one instance of class craeft:TransitionSchema
	An instance of craeft:TransitionSchema is connected by craeft:isHeadOf to at most one instance of class craeft:TransitionSchema

- A simple virtual transition has exactly one tail and one head, which can be a virtual step or a virtual transition. Conversely, a virtual step or a virtual transition can be the tail or the head of at most



one virtual transition, of whatever kind.

	An instance of craeft:SimpleVirtualTransition is connected by craeft:hasTail only to instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:SimpleVirtualTransition is connected by craeft:hasTail to exactly one instance of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:VirtualStep is connected by craeft:isTailOf to at most one instance of class craeft:VirtualTransition
	An instance of craeft:VirtualTransition is connected by craeft:isTailOf to at most one instance of class craeft:VirtualTransition
	An instance of craeft:SimpleVirtualTransition is connected by craeft:hasHead only to instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:SimpleVirtualTransition is connected by craeft:hasHead to exactly one instance of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:VirtualStep is connected by craeft:isHeadOf to at most one instance of class craeft:VirtualTransition
	An instance of craeft:VirtualTransition is connected by craeft:isHeadOf to at most one instance of class craeft:VirtualTransition

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

	An instance of craeft:SimpleTransition is connected by craeft:hasTail only to instances of (craeft:Step union craeft:Transition)
	An instance of craeft:SimpleTransition is connected by craeft:hasTail to exactly one instance of (craeft:Step union craeft:Transition)
	An instance of craeft:Step is connected by craeft:isTailOf to at most one instance of class craeft:Transition
	An instance of craeft:Transition is connected by craeft:isTailOf to at most one instance of class craeft:Transition
	An instance of craeft:SimpleTransition is connected by craeft:hasHead only to instances of (craeft:Step union craeft:Transition)
	An instance of craeft:SimpleTransition is connected by craeft:hasHead to exactly one instance of (craeft:Step union craeft:Transition)
	An instance of craeft:Step is connected by craeft:isHeadOf to at most one instance of class craeft:Transition

An instance of craeft:Transition is connected by craeft:isHeadOf to at most one instance of class craeft:Transition
--

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**, we use:

- 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 to each alternative; also **craeft:Predicate** is a subclass of **ecrm:E29_Design_or_Procedure**, as its instances represents parts of plans;
- properties **craeft:leadsTo** and **craeft:hasPredicate** to associate each alternative to a tail of the schema and to 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 on 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 to 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:

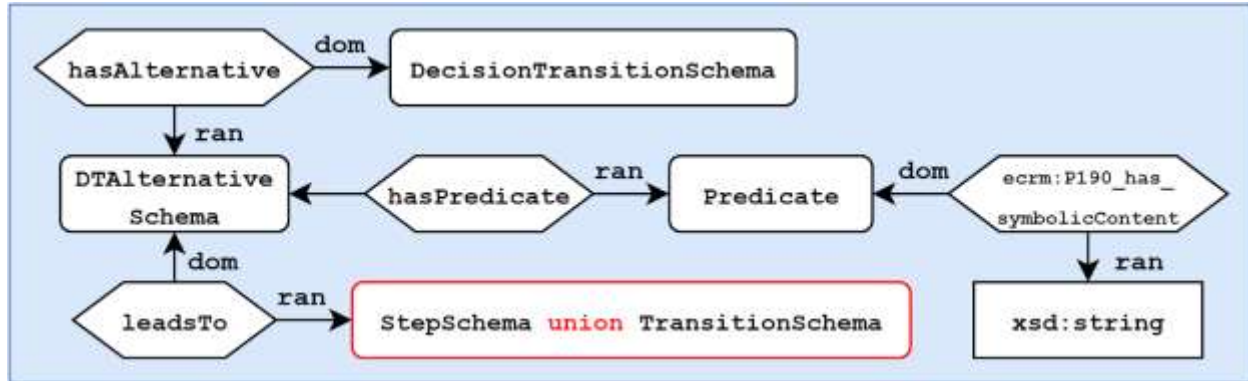


Figure 13 Representation of alternatives in Decision Transition Schemas.

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



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

We now localise **craeft:hasTail** and **craeft:hasHead** to decision transition schemas. A decision transition schema has a single tail, which is either a step schema or a transition schema, and two or more heads, which result from the composition of properties **craeft:hasAlternatives** and **craeft:leadsTo**. Conversely, a step schema or a transition schema is the tail of at most one decision transition schema, and the head of at most one decision transition schema.

	An instance of craeft:DecisionTransitionSchema is connected by craeft:hasTail only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:DecisionTransitionSchema is connected by craeft:hasTail to exactly one instance of (craeft:StepSchema union craeft:TransitionSchema)

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 craeft:MergeTransitionSchema is connected by craeft:hasTail only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:MergeTransitionSchema is connected by craeft:hasTail to at least two instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:MergeTransitionSchema is connected by craeft:hasHead only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:MergeTransitionSchema is connected by craeft:hasHead to exactly one instance of (craeft:StepSchema union craeft:TransitionSchema)

2.5.2.4. Fork transitions

A fork transition has a single tail and two 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 represents 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 craeft:ForkTransitionSchema is connected by craeft:hasTail only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:ForkTransitionSchema is connected by craeft:hasTail to exactly one instance of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:ForkTransitionSchema is connected by craeft:hasHead only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:ForkTransitionSchema is connected by craeft:hasHead to at least two instances of (craeft:StepSchema union craeft:TransitionSchema)

- 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 craeft:ForkVirtualTransition is connected by craeft:hasTail only to instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:ForkVirtualTransition is connected by craeft:hasTail to exactly one instance of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:ForkVirtualTransition is connected by craeft:hasHead only to instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:ForkVirtualTransition is connected by craeft:hasHead to at least two instances of (craeft:VirtualStep union craeft:VirtualTransition)

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

	An instance of craeft:ForkTransition is connected by craeft:hasTail only to instances of (craeft:Step union craeft:Transition)
	An instance of craeft:ForkTransition is connected by craeft:hasTail to exactly one instance of (craeft:Step union craeft:Transition)



	An instance of craeft:ForkTransition is connected by craeft:hasHead only to instances of (craeft:Step union craeft:Transition)
	An instance of craeft:ForkTransition is connected by craeft:hasHead to at least two instances of (craeft:Step union craeft:Transition)

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, similarly 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 craeft:JoinTransitionSchema is connected by craeft:hasTail only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:JoinTransitionSchema is connected by craeft:hasTail to at least two instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:JoinTransitionSchema is connected by craeft:hasHead only to instances of (craeft:StepSchema union craeft:TransitionSchema)
	An instance of craeft:JoinTransitionSchema is connected by craeft:hasHead to exactly one instance of (craeft:StepSchema union craeft:TransitionSchema)

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

	An instance of craeft:JoinVirtualTransition is connected by craeft:hasTail only to instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:JoinVirtualTransition is connected by craeft:hasTail to at least two instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:JoinVirtualTransition is connected by craeft:hasHead only to instances of (craeft:VirtualStep union craeft:VirtualTransition)
	An instance of craeft:JoinVirtualTransition is connected by craeft:hasHead to exactly one instance of (craeft:VirtualStep union craeft:VirtualTransition)

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

	An instance of craeft:JoinTransition is connected by craeft:hasTail only to instances of
--	--

	(craeft:Step union craeft:Transition)
	An instance of craeft:JoinTransition is connected by craeft:hasTail to at least two instances of (craeft:Step union craeft:Transition)
	An instance of craeft:JoinTransition is connected by craeft:hasHead only to instances of (craeft:Step union craeft:Transition)
	An instance of craeft:JoinTransition is connected by craeft:hasHead to exactly one instance of (craeft:Step union craeft:Transition)

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, subclass of **ecrm:E55_Type**, including as instances the terms of the dictionary; the class is named **NNWTerm**, where **NNW** 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 **hasNNWTerm**, where **NNW** 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 **NNWTerm**.

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 these 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).

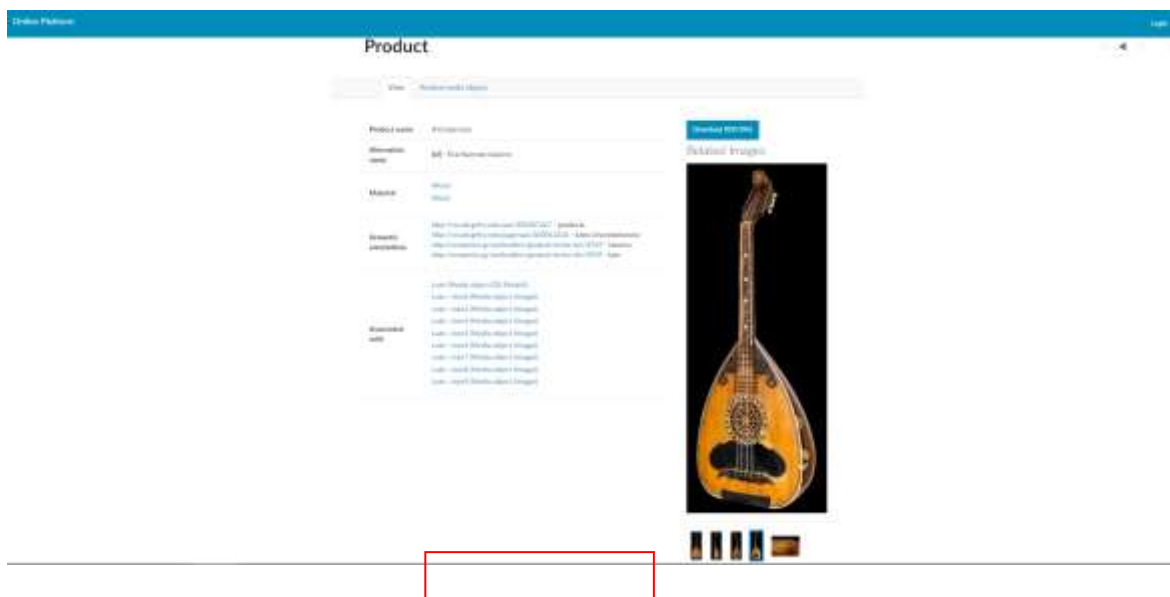


Figure 14. The page for a craft product with semantic annotations.

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.

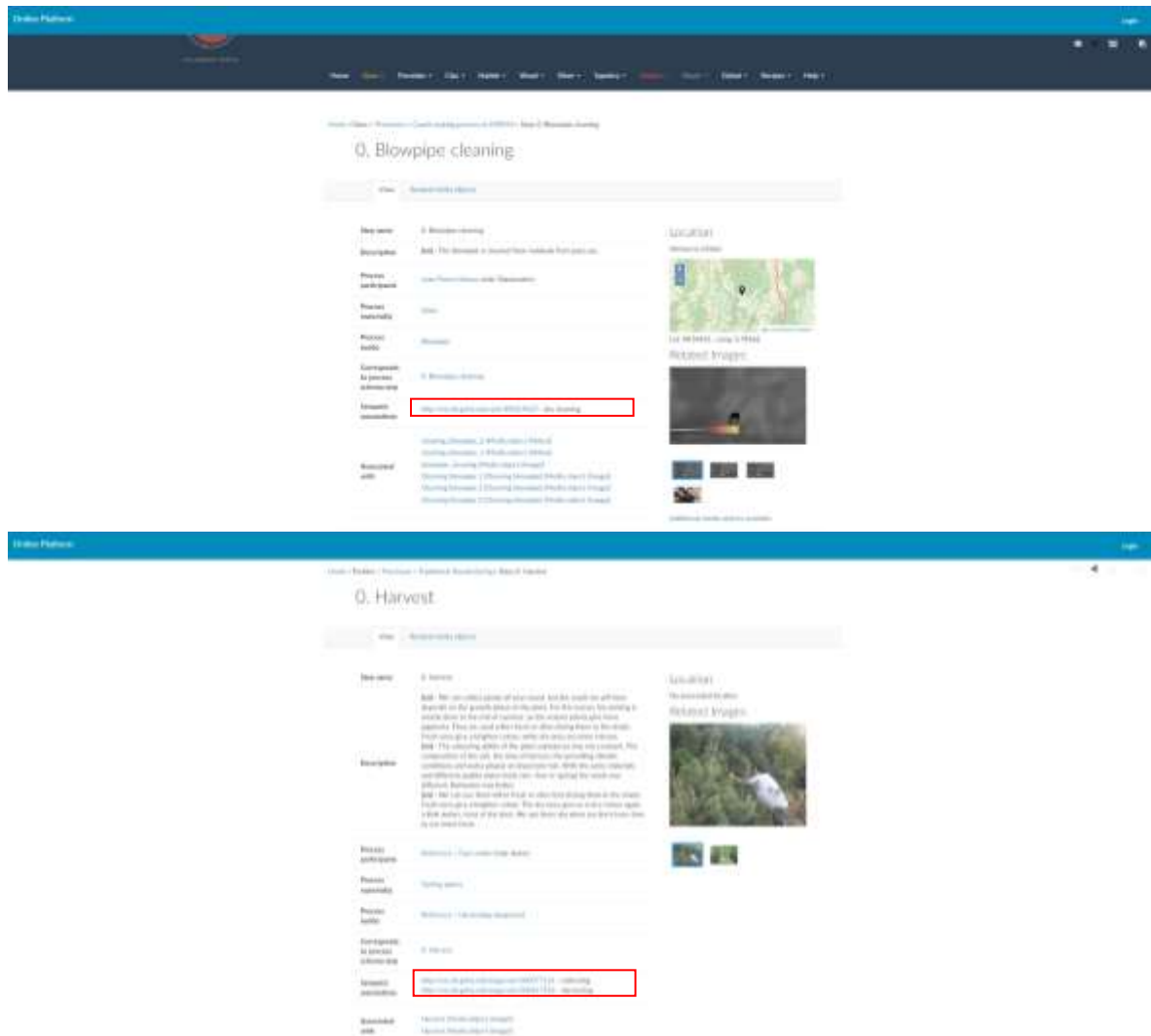


Figure 15. Two pages for craft process steps, with semantic annotations.

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>.

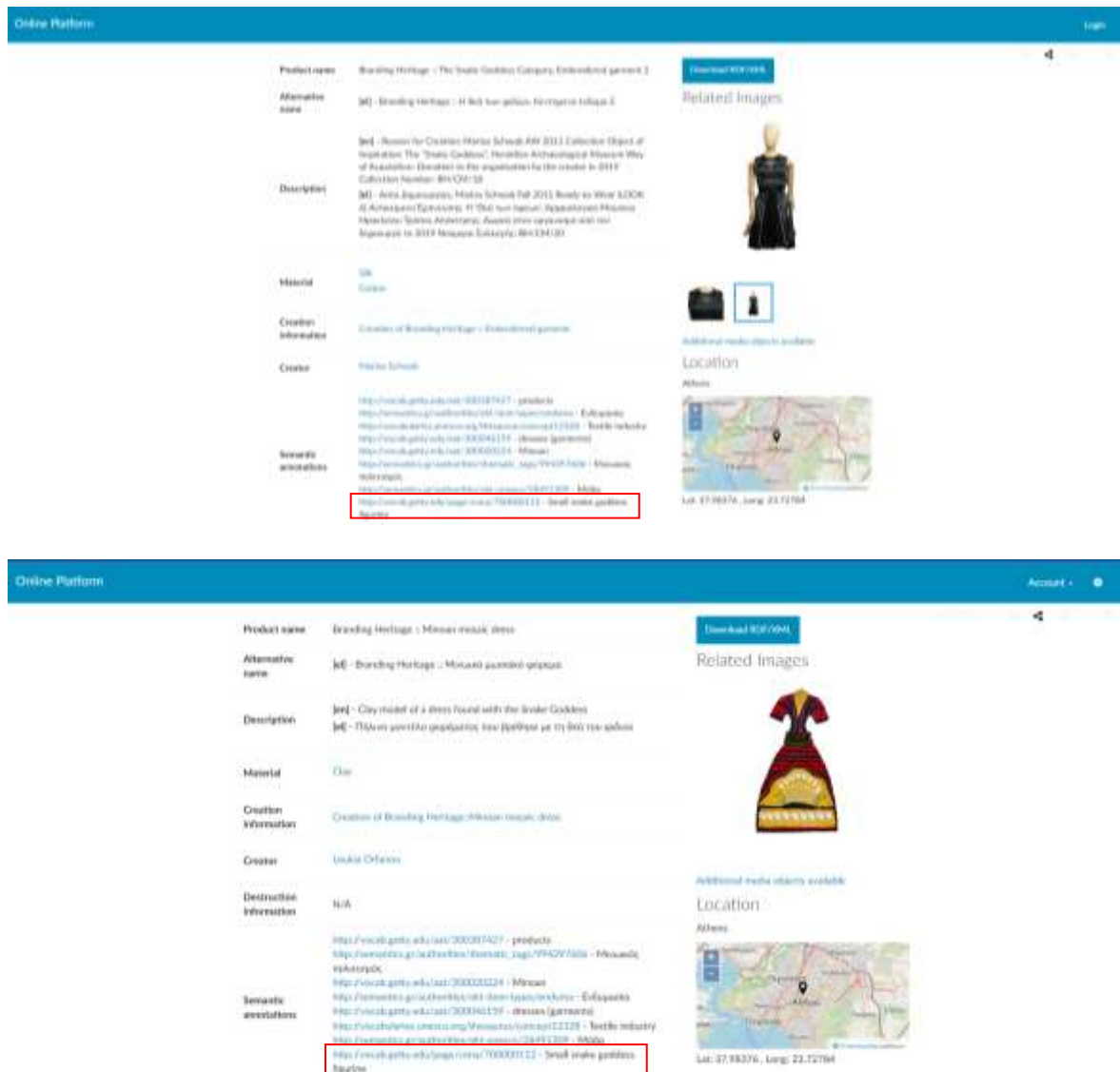


Figure 16. Two pages for two craft artefacts inspired by another artefact that is registered in CONA.

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.

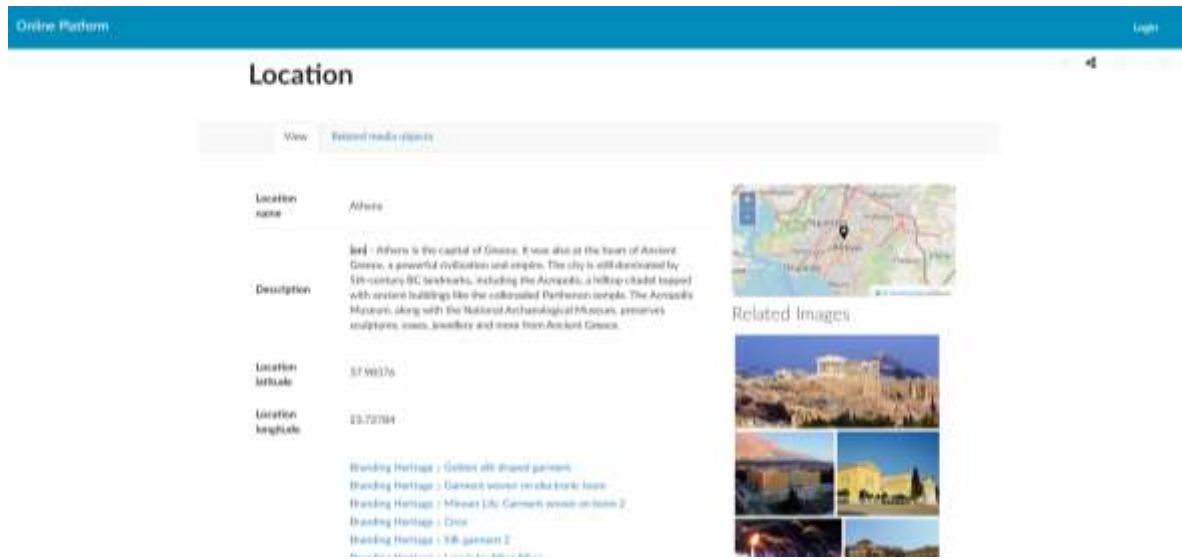


Figure 17. A page for a location knowledge entity, geosemantically annotated with Geonames and TGN labels.

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 <code>hasULANTerm</code> is class <code>ULANTerm</code>

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

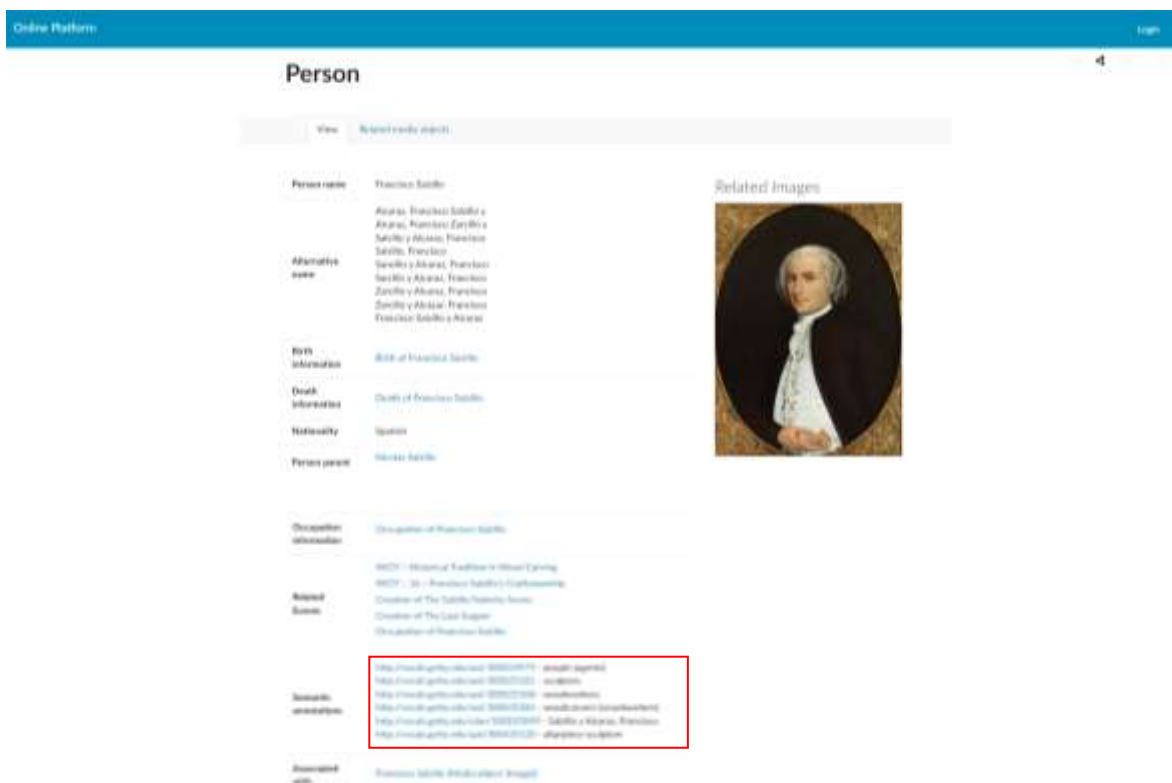


Figure 18. The page for a wood sculptor (person), semantically annotated with a ULAN label.

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
	GNATerm 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”¹⁰ 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”¹¹ 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.

¹⁰ <https://pro.europeana.eu/project/crafted>

¹¹ <http://semantics.gr/authorities/vocabularies/craft-item-types>

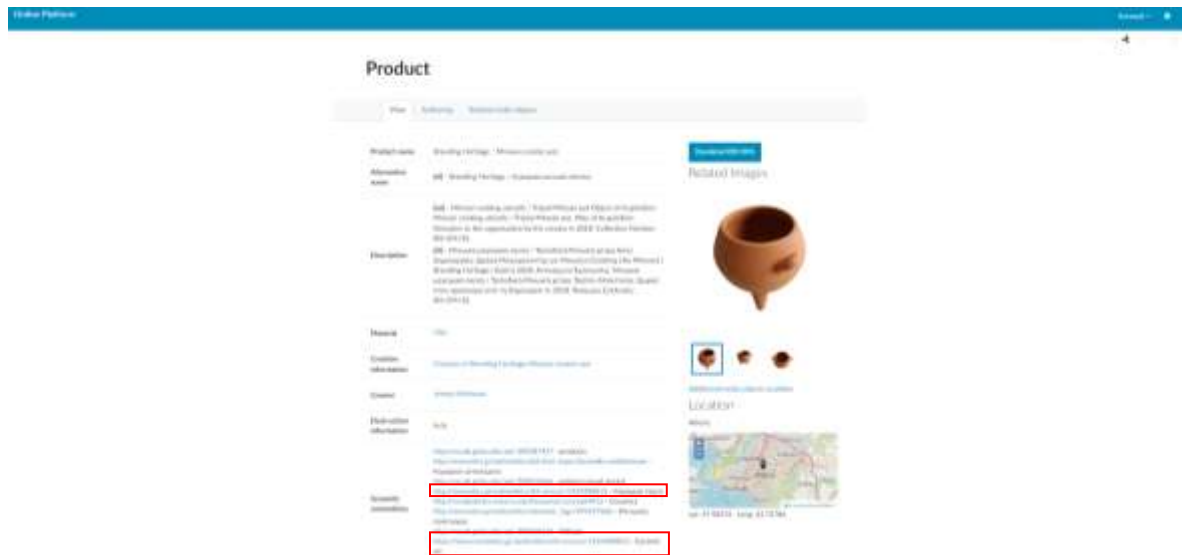


Figure 19. The page for a craft artefact, annotated with English and Greek metadata.

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.

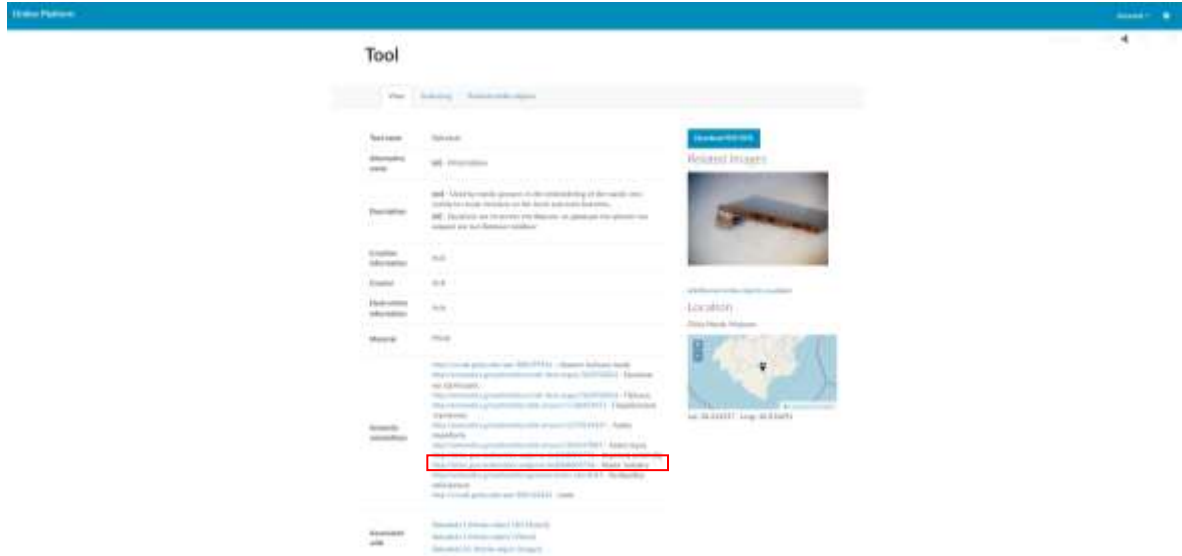


Figure 20. The page for a craft tool, with semantic annotations including annotations from the Library of Congress.

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.

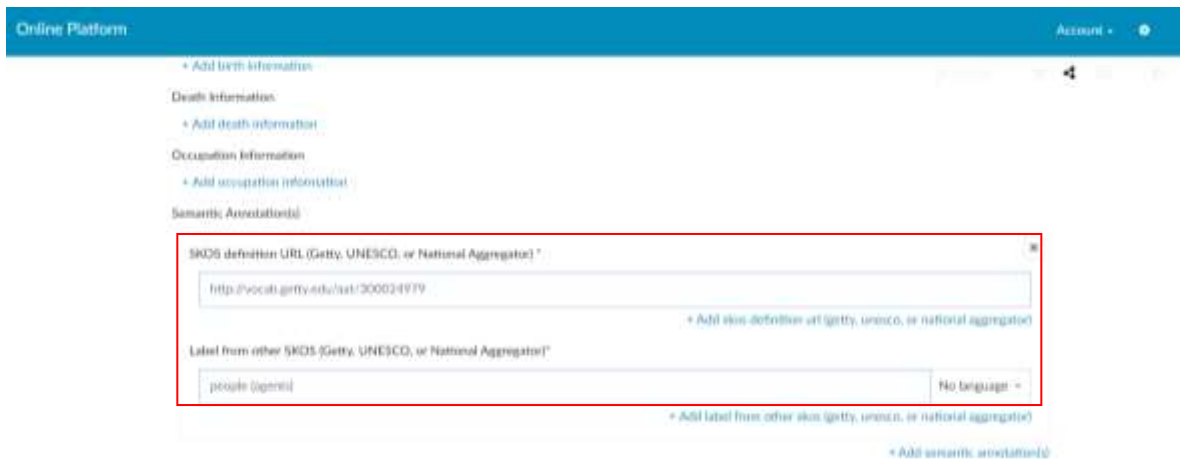


Figure 21. The GUI facility for default semantic annotations.

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.

2.7. Examples

To the end of illustrating the ontology, this Section introduces the axioms to represent a few example actions and processes. These examples will also serve as a preliminary proof of concept in simple real cases, before implementation and systematic evaluation of the representative craft instances.

The first example given concerns the single action of inserting a chisel into a piece of wood by hammering it down. The schema of the action will first be given, followed by an instantiation and an enactment of the action. For completeness, all the OWL 2 DL axioms are given, using the functional notation.

The subsequent examples concern four example processes:

- Creating a pattern of 20 dots in a straight line on a piece of silver
- Hammering a piece of wood until it is split into two (or more) pieces
- Bending a ply of aluminium
- Pulling apart a ply of steel

For the sake of readability, these examples will be illustrated only graphically or using natural language.

2.7.1. Inserting a chisel in a piece of wood

We first introduce the classes that represent the entities in the example.

	Hammer is a class
	Hammer is a subclass of ecrm:E22_Human-Made_Object
	Chisel is a class
	Chisel is a subclass of ecrm:E22_Human-Made_Object
	PieceOfWood is a class
	PieceOfWood is a subclass of ecrm:E22_Human-Made_Object
	Force is a class
	Force is a subclass of ecrm:E77_Persistent_Item

Graphically:

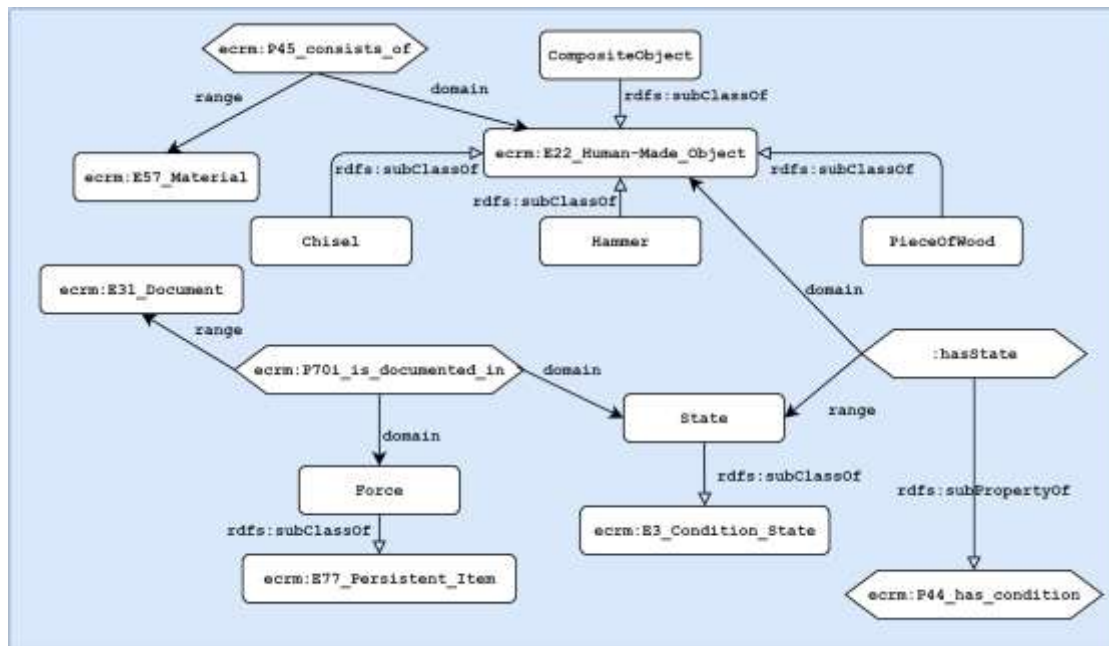


Figure 22 Classes and properties for an example.

Let us now define the action schema (named **ActSch7**) of the example at hand.

```
ClassAssertion( ActSch7 ActionSchema )
```



The following axioms give the characterisation of as in terms of some description properties:

<code>ObjectPropertyAssertion(ActSch7 rdfs:hasLabel "Hitting a chisel with a hammer entering into a piece of wood")</code>
<code>ObjectPropertyAssertion(ActSch7 hasActionFunction http://vocab.getty.edu/page/aat/300053017)</code>
<code>ObjectPropertyAssertion(ActSch7 hasActionType add)</code>
<code>ObjectPropertyAssertion(ActSch7 hasMachineType wedgeMachineType)</code>
<code>ObjectPropertyAssertion(ActSch7 rdfs:comment "The action is performed by a single agent that exercises a force on the chisel by hitting it with a hammer upon its end. As a result, the chisel is inserted into the piece of wood. The action belongs to carpentry.")</code>

The only causing entity of the action is the force applied; the affected entities are the hammer, which is rigid and whose position changes during the execution of the action, the chisel and the piece of wood; the last two objects are deformable and become part of the newly created entity of the action, which is the composite object consisting of the piece of wood with the chisel inserted in it.

<code>ObjectPropertyAssertion(ActSch7 hasCausingEntityProperty hasForce)</code>
<code>ObjectPropertyAssertion(ActSch7 hasAffectedEntityProperty hasHammer)</code>
<code>ObjectPropertyAssertion(ActSch7 hasAffectedEntityProperty hasChisel)</code>
<code>ObjectPropertyAssertion(ActSch7 hasAffectedEntityProperty hasPieceOfWood)</code>

All the above properties have **VirtualAction** as domain since they characterise instances of the **ActSch7** schema. The range of these properties varies depending on the semantics of the property.

<code>ObjectPropertyDomain(hasForce VirtualAction)</code>
<code>ObjectPropertyRange(hasForce Force)</code>
<code>ObjectPropertyDomain(hasHammer VirtualAction)</code>
<code>ObjectPropertyRange(hasHammer Hammer)</code>
<code>ObjectPropertyDomain(hasChisel VirtualAction)</code>
<code>ObjectPropertyRange(hasChisel chisel)</code>
<code>ObjectPropertyDomain(hasPieceOfWood VirtualAction)</code>
<code>ObjectPropertyRange(hasPieceOfWood PieceOfWood)</code>

Now let us introduce a virtual action (**VirtAct32**) that instantiates this schema.

<code>ClassAssertion(VirtAct32 VirtualAction)</code>
<code>ObjectPropertyAssertion(VirtAct32 instanceOf ActSch7)</code>



To represent the action, we use the properties introduced by the action schema. For the sake of readability, in the following axioms the name of each individual is formed by appending a (non-significant) number to the name of the class where the individual belongs.

<code>ObjectPropertyAssertion(VirtAct32 hasForce Force27)</code>
<code>ClassAssertion(Force27 Force)</code>
<code>ObjectPropertyAssertion(VirtAct32 hasHammer Hammer125)</code>
<code>ClassAssertion(Hammer125 Hammer)</code>
<code>ClassAssertion(Hammer125 RigidObject)</code>
<code>ObjectPropertyAssertion(VirtAct32 hasChisel Chisel99)</code>
<code>ClassAssertion(Chisel99 Chisel)</code>
<code>ClassAssertion(Chisel99 DeformableObject)</code>
<code>ObjectPropertyAssertion(VirtAct32 hasPieceOfWood PieceOfWood68)</code>
<code>ClassAssertion(PieceOfWood68 PieceOfWood)</code>
<code>ClassAssertion(PieceOfWood68 DeformableObject)</code>

Let us now provide knowledge about the entities just introduced. Force is described by a sequence of poses giving the motion of the hammer. These poses and their relative timestamps are given in a single document, a web resource identified by IRI **ForceDoc1**. As explained in Section 2, the connection between the force and its value document is represented as follows:

<code>ClassAssertion(ForceDoc1 ecrm:E31_Document)</code>
<code>ObjectPropertyAssertion(ForceDoc1 ecrm:P70_Documents Force27)</code>

The time-independent characteristics of the hammer, are shape and material properties, while those of the chisel and the piece of wood are just the material properties, as these objects are deformable. This knowledge is represented as described in Section 2.5, using the two properties **hasShape** and **hasMaterialProperties** to associate each object to its characteristics, represented as documents.

<code>ClassAssertion(Ham125Sh ecrm:E31_Document)</code>
<code>ObjectPropertyAssertion(Hammer125 hasShape Ham125Sh)</code>
<code>ClassAssertion(Ham125MP ecrm:E31_Document)</code>
<code>ObjectPropertyAssertion(Hammer125 hasMaterialProperties Ham125MP)</code>
<code>ClassAssertion(Ch99MP ecrm:E31_Document)</code>



ObjectPropertyAssertion(Chisel99 hasMaterialProperties Ch99MP)
ClassAssertion(PoW68MP ecrm:E31_Document)
ObjectPropertyAssertion(PieceOfWood68 hasMaterialProperties PoW68MP)

The hammer, the chisel and the piece of wood are the three affected objects of the virtual action, therefore the virtual action must be associated with the state of each of these objects before and after its execution. The difference is that the state of the hammer does not include shape, while those of the chisel and the piece of wood do. In all cases, the state is a document.

ClassAssertion(Ham125State1 ecrm:E31_Document)
ObjectPropertyAssertion(Hammer125 hasState Ham125State1)
ClassAssertion(Ham125State2 ecrm:E31_Document)
ObjectPropertyAssertion(Hammer125 hasState Ham125State2)
ObjectPropertyAssertion(VirtAct32 AESBefore Ham125State1)
ObjectPropertyAssertion(VirtAct32 AESAfter Ham125State1)
ClassAssertion(Chisel99State1 ecrm:E31_Document)
ObjectPropertyAssertion(Chisel99 hasState Chisel99State1)
ClassAssertion(Chisel99State2 ecrm:E31_Document)
ObjectPropertyAssertion(Chisel99 hasState Chisel99State2)
ObjectPropertyAssertion(VirtAct32 AESBefore Chisel99State1)
ObjectPropertyAssertion(VirtAct32 AESAfter Chisel99State2)
ClassAssertion(PoW68State1 ecrm:E31_Document)
ObjectPropertyAssertion(PieceOfWood68 hasState PoW68State1)
ClassAssertion(PoW68State2 ecrm:E31_Document)
ObjectPropertyAssertion(PieceOfWood68 hasState PoW68State2)
ObjectPropertyAssertion(VirtAct32 AESBefore PoW68State1)
ObjectPropertyAssertion(VirtAct32 AESAfter PoW68State2)

In order to complete the representation of the virtual action, we need to assert the object it produces, and the composition of that object.

ClassAssertion(Ch99PoW68 CompositeObject)

ObjectPropertyAssertion(VirtAct32 produces Ch99PoW68)
ObjectPropertyAssertion(Ch99PoW68 ecrm:P46_is_composed_of Chisel99)
ObjectPropertyAssertion(Ch99PoW68 ecrm:P46_is_composed_of PieceOfWood68)

Finally, let us represent a real action (**Action5**) that is an enactment of the virtual action represented so far. We assume that the enactment was performed by agent **Tom**, in **Rome** (which is just an abbreviation for Rome's Geonames IRI: <https://www.geonames.org/3169070/>) on the first of January 2022.

ClassAssertion(Action5 Action)
ObjectPropertyAssertion(Action5 enactmentOf VirtAct32)
ObjectPropertyAssertion(Action5 ecrm:P14_carried_out_by Tom)
ObjectPropertyAssertion(Action5 ecrm:P7_took_place_at Rome)
ObjectPropertyAssertion(Action5 ecrm:P4_has_time_span ts7)
DataPropertyAssertion(ts7 beginsAt "2021-01-01T00:00:00"@xsd:dateTime)
DataPropertyAssertion(ts7 endsAt "2021-01-01T23:59:59"@xsd:dateTime)

2.7.2. Creating a pattern of dots on silver

In this process, an agent uses a hand-driven rod with a hemispherical endpoint to compress silver and create a pattern of 20 dots in a straight line. The dots should be 1.5 cm apart and each one should make a small cusp of 3 mm depth in the material. The material is a 20 x 5 x 100 mm³ piece of silver.

The initial pose of the tool is 2 cm above the material and 1 cm from the middle of its top edge and perpendicular to it. These dimensions are known not to cause any damage or failure to the material, such as breaking or cracking. We are not interested in simulating tool wear, so, we model the tool to be rigid. In contrast, the piece of silver on which the dots are made is considered to be deformable.

The process schema is given graphically in Figure 23. A process schema for creating a pattern of dots on silver.

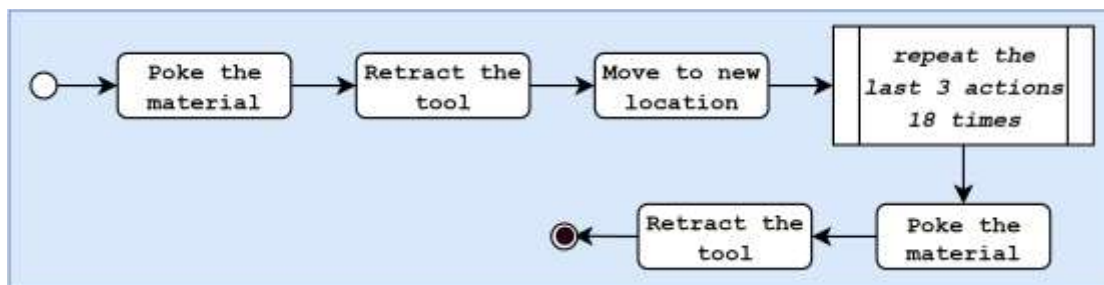


Figure 23 A process schema for creating a pattern of dots on silver.

The process schema is straightforward as it only includes simple transitions. As the ontology does not provide a “loop” transition, the process schema enumerates all the action schemas that compose the process, that is, the “Poke the material” and the “Retract the tool” action schemas have to be repeated 20 times and the “Retract the tool” action schema 19 times. To make the process schema easier to read, in the above Figure we have used the “repeat ...” abbreviation, but this is just a graphic device.

Upon instantiation of the process schema into a virtual process, two actions need to be executed immediately after the start action: (1) Read the input simulation file and (2) initialise the virtual scene. These actions are specific to the simulator and will never be enacted in real processes, therefore they are not represented in the process schema by corresponding action schemas; rather, they are associated with an action schema (the start action schema, in this case) via a special ad-hoc property. Similarly, upon completion of every step, the resultant state of the affected objects and of the created objects must be transferred from the simulator into the KB. This action is not going to be enacted therefore it is not represented in the process schema by a corresponding action schema; rather, it is associated with every action schema other than start and end via the above-mentioned ad-hoc property.

The images in the figure below illustrate the evolution of the simulation and the rendered result of the simulation. Colour encodes material stress.

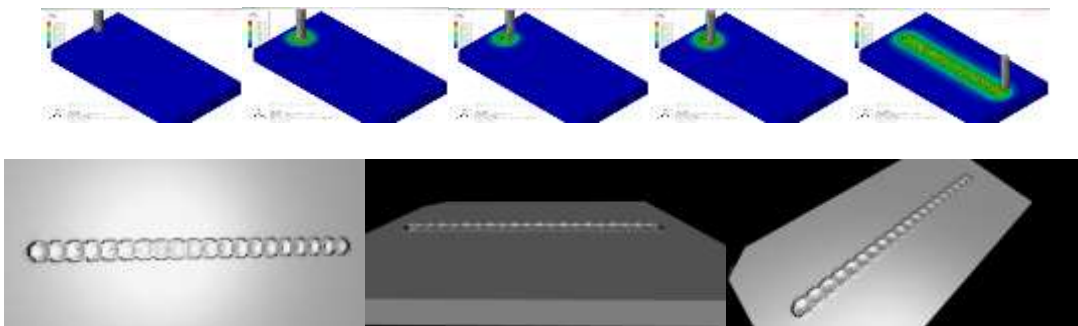


Figure 24 Top (left to right): beginning of first step; end of first step; beginning of second step; end of second step; end of process. Bottom: rendering of the simulation result.

Action Schema “Poke the material”

English name for the action type: “make a 3 mm dot in a piece of silver”

Description of the action type: “The action is performed by a single agent that exercises a force on a hand-driven rod with a hemispherical endpoint to compress silver and create a dot. The dot should make a small cusp of 3 mm depth in the material”.

Function of the action type according to the standardised vocabulary

<http://vocab.getty.edu/page/aat/300053826> embossing (technique)

Type: Transform

Machine Type: wedge



Performer: person

Causing entity: the force exerted by the performer on the rod

Affected objects: 2, the rod (rigid) and the piece of silver (deformable)

The rigid rod is represented as follows:

- **Type:** hand-driven rod with a hemispherical endpoint
- **Documentation of the unchanging properties:** shape and material properties
- **Documentation of the state of the entity before the action:** pose
- **Documentation of the state of the entity after the action:** pose

The deformable piece of silver is represented as follows:

- **Type:** a piece of silver
- **Documentation of the unchanging properties:** material properties
- **Documentation of the state of the entity before the action:** pose, shape
- **Documentation of the state of the entity after the action:** pose, shape

Created objects: none

Action function:

- **Description** the agent drives the tool perpendicularly towards the surface and presses displacing the tool, by 3 mm in the material. The displacement is 23 mm in the vertical direction downwards.
- **Duration** 1 sec.

Action Schema "Retract the tool"

English name for the action type: "retract the rod after poking"

Description of the action type: "The action is performed by a single agent that exercises a force on a hand-driven rod with a hemispherical endpoint to move it vertically from its current position."

Function of the action type according to the standardised vocabulary

<http://vocab.getty.edu/page/aat/300257582> pulling (dismantling)

Type: Transform

Machine Type: N/A

Performer: person

Causing entity: the force exerted by the performer on the rod



Affected objects: the rod

The rod is represented as above.

Created objects: none

Action function:

- **Description** retract the tool perpendicularly and bring it to its initial position. The displacement is *23 mm* in the vertical direction upwards
- **Duration** *1 sec.*

Action Schema “Move to new location”

English name for the action type: “move the rod *1.5 cm* from the current position in a given direction”.

Description of the action type: “The action is performed by a single agent that exercises a force on a hand-driven rod with a hemispherical endpoint to move it *1.5 cm* away from the current position in the direction determined by the line of dots being executed”.

Function of the action type according to the standardised vocabulary

<http://vocab.getty.edu/page/aat/300263062> moving (operational attribute)

Type: Transform

Machine Type: N/A

Performer: person

Causing entity: the transportation exerted by the performer on the rod

Affected objects: the rod

The rod is represented as above.

Created objects: none

Action function:

- **Description** The agent brings the tool to its new position. The displacement is *1.5 cm* in the horizontal direction to the right.
- **Duration** *1 sec.*

2.7.3. Hammering a piece of wood

In this process, an agent hammers a piece of wood until it is split into two or more pieces.

The process schema is given in Figure 26:

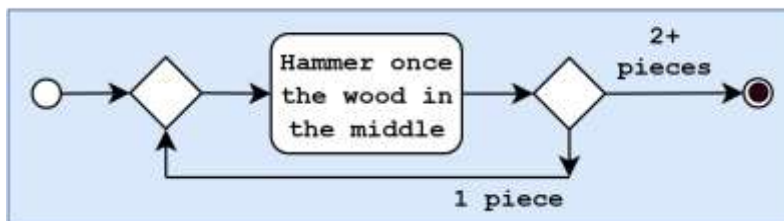


Figure 26 Process schema for hammering a piece of wood.

The above schema includes a main action schema, which captures both cases, i.e., whether the action breaks or does not break the piece of wood, following the principle that the schema of an uncertain action must account for all *potential* outcomes of an action. There are two types of transitions, a decision, needed to evaluate whether the process is over or not, and a merge, needed by those processes that require more than one execution of the main action to achieve their goal of breaking the piece of wood.

When we instantiate the schema, we have to define simulation properties such as the dimensions of the piece of wood and its material properties. Moreover, the shape of the tool plays an important role, as well as the force that the practitioner can apply and the incidence angle of the tool with the material. These parameters determine how many times the practitioner is required to strike the material to be cut into two pieces. In the instantiation of this schema in the simulation illustrated in the figure below, three strokes of the hammer were required.

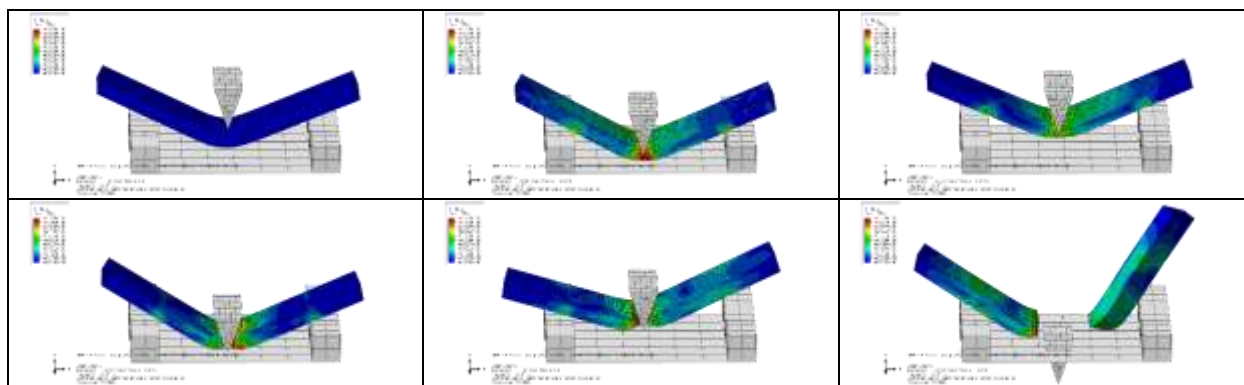




Figure 27 Simulation results for hammering a piece of wood. Top (left to right): initial condition, first strike, and upward retraction of the tool. Middle: second strike, retraction, and third strike. Bottom: rendered 3D model of the simulation result.

In the simulation, the result of friction can be observed: when the hammer is moving upwards the piece of material is stuck with the tool moving upwards. Other details that provide realism to this simulation can be observed such as the slight distortion of the material at the contact locations with its support, as well as its elastic properties that make the material change its shape while moving upwards.

The schema of the main action is given below.

English name for the action type: “hammer a piece of wood to break it into two pieces”

Description of the action type: “The action is performed by a single agent that exercises a force on a hammer to break a piece of wood in two or more pieces”.

Function of the action type according to the standardised vocabulary

<http://vocab.getty.edu/aat/300054098> *hammering*

Type: Transform

Machine Type: wedge

Performer: person

Causing entity: the force exerted by the performer on the hammer

Affected objects: 2, the hammer (rigid) and the piece of wood (deformable)

The rigid hammer is represented as follows:

- **Type:** hand-driven hammer
- **Documentation of the unchanging properties:** shape and material properties
- **Documentation of the state of the entity before the action:** pose
- **Documentation of the state of the entity after the action:** pose

The deformable piece of wood is represented as follows:

- **Type:** piece of wood
- **Documentation of the unchanging properties:** material properties
- **Documentation of the state of the entity before the action:** pose, shape
- **Documentation of the state of the entity after the action:** pose, shape

Created objects: each of the pieces of wood resulting from the action, if any

Action function:

- **Description** the agent drives a strike of the hammer perpendicularly to the middle of the wood. The force vector is $[0 \ 100 \ 0]^T$, measured in Newtons (N).
- **Duration** 1 sec.

When this action schema is instantiated into a virtual action and simulated, one of two things may happen:

1. The piece of wood breaks, in which case the two or more pieces of wood resulting from the action are associated with the action as created entities.
2. The piece of wood does not break, in which case nothing is associated with the action as created entities.

For instance, assuming that the piece of wood breaks immediately, then the virtual process looks like:

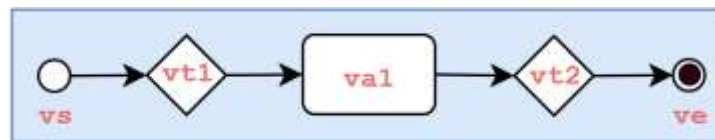


Figure 28 A virtual process.

where:

- vs is a “start” virtual action
- vt1 is a simple virtual transition, an instance of the merge transition schema
- va1 is a virtual action instance of the above action schema, breaking the piece of wood
- vt2 is a simple virtual transition instance of the decision transition schema
- ve is an “end” virtual action

If instead, the piece of wood breaks the second time it is hammered, then the virtual process looks like this:

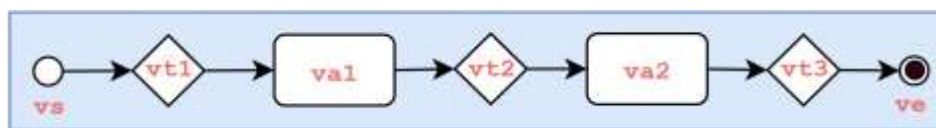


Figure 29 A virtual process.

2.7.4. Bending a ply of aluminium

In this process, a ply of aluminium is bent by using a rigid cylindrical tool. Two additional rigid cylindrical rods are used to stabilise the ply and create the space for the ply to be bent. The initial pose of the tool is in contact with the ply and its middle.

The process schema is given in the figure below:

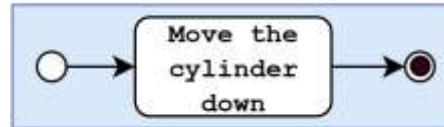


Figure 30 Process schema for bending a ply of aluminium.

The process schema is straightforward, it includes one action schema, which is moving the cylinder down to bend the aluminium ply. When instantiated in a virtual action, executed by simulation, the objects involved in the action evolved as shown in the figure below. The image below also shows the exported 3D result regarding the material.

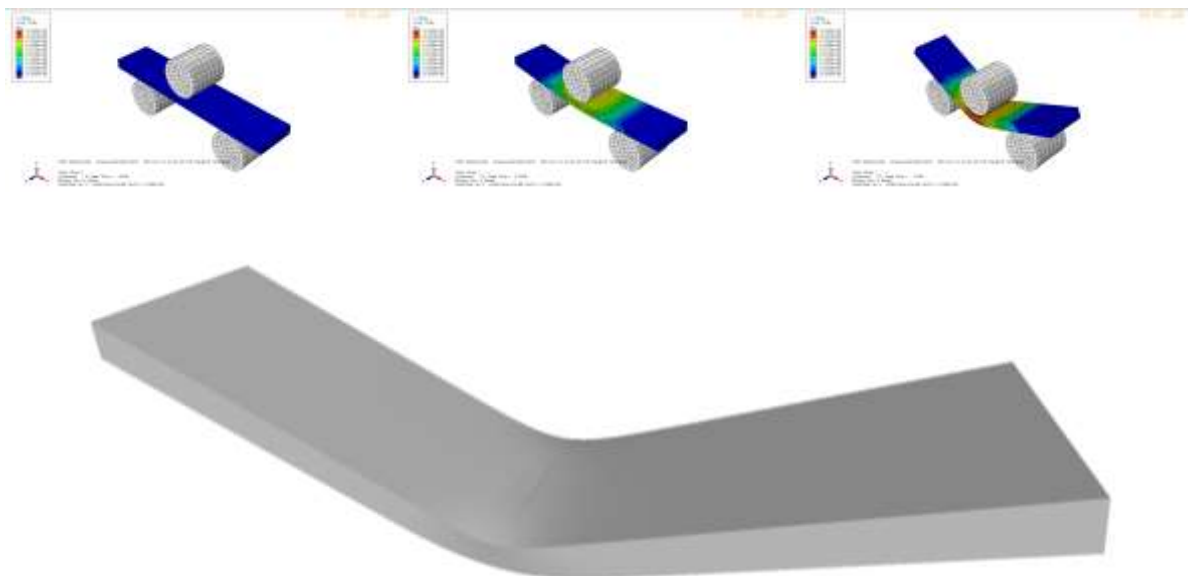


Figure 31 Top(left to right): beginning, middle and end of action. Bottom: rendered 3D model of the simulation result.

The schema of the only action is given below:

English name for the action type: “bend a ply of aluminium by 10 mm”

Description of the action type: “The action is performed by a single agent that exercises a force on a cylindrical tool that distorts the ply”.

Function of the action type according to the standardised vocabulary

<http://vocab.getty.edu/page/aat/300053101> bending

Type: Transform

Machine Type: lever (two levers are formed, one for each side).



Performer: person

Causing entity: the force exerted by the performer on the tool

Affected objects: 4, the tool and the supporting bodies, all rigid, and the ply of aluminium (deformable)

The rigid tool is represented as follows:

- **Type** cylindrical tool
- **Documentation of the unchanging properties** shape (rigid) and material properties
- **Documentation of the state of the entity before the action:** pose
- **Documentation of the state of the entity after the action:** pose

Each of the rigid supporting, cylindrical bodies is represented as follows:

- **Type** supporting element
- **Documentation of the unchanging properties** shape (rigid) and material properties
- **Documentation of the state of the entity before the action:** pose
- **Documentation of the state of the entity after the action:** pose

The deformable ply of aluminium is represented as follows:

- **Type** ply of aluminium
- **Documentation of the unchanging properties** material properties
- **Documentation of the state of the entity before the action:** pose, shape
- **Documentation of the state of the entity after the action:** pose, shape

Created objects: none

Action function:

- **Description** the agent pushes the tool downwards displacing the tool, by *10 mm*. This causes a bending of the ply, whose middle part moves correspondingly by *10 mm* in the same direction.
- **Duration** *1 sec.*
- **Execution** *object = Plastic(object).*

2.7.5. Pulling a ply apart

In this process, a ply of steel is pulled from its two ends to split it into two parts.

The process schema is given in the figure below:

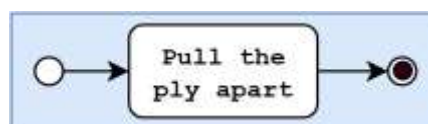


Figure 32 Process schema for pulling a ply apart.

The process schema is straightforward, it includes one action schema, that is pulling the ply apart by 10mm from each part. When instantiated in a virtual action, executed by simulation, the objects involved in the action evolved as shown in the figure below (colour encodes material stress). The image below also shows the exported 3D result regarding the material.

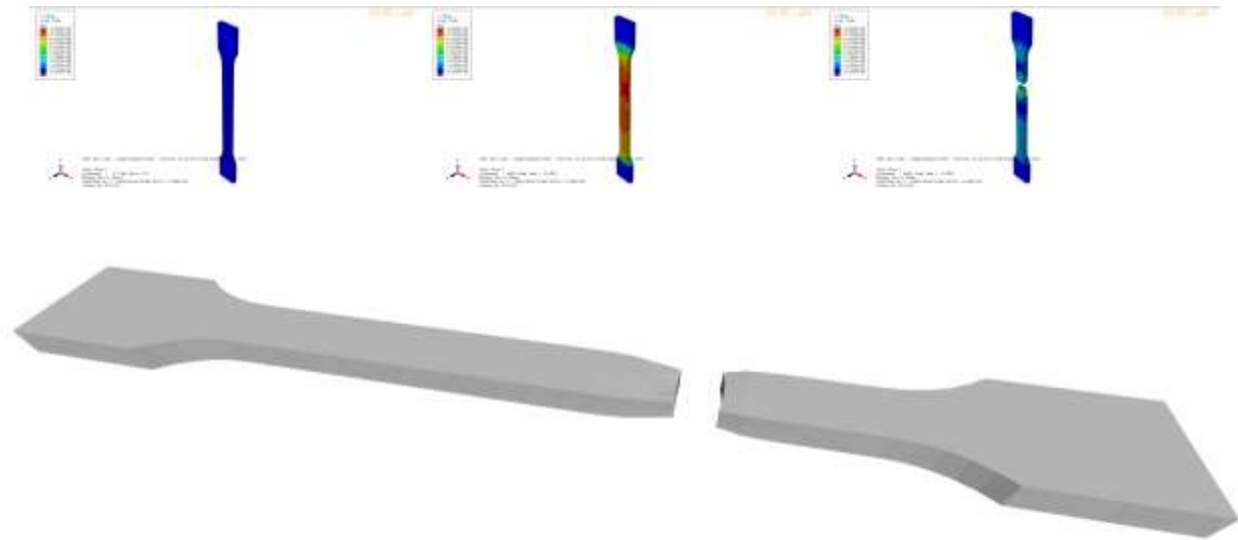


Figure 33 Top (left to right): beginning, middle and end of action. Bottom: rendered 3D model of the simulation result.

The schema of the only action is given below:

English name for the action type: “pull the ply by 10 mm from each end”.

Description of the action type: “The action is performed by a single agent that pulls the ply from its two ends”.

Function of the action type according to the standardised vocabulary

<http://vocab.getty.edu/page/aat/300257582> pulling (dismantling)

Type: Subtract

Machine Type: N/A

Performer: person

Causing entity: the forces exerted by the performer on the ply

Affected objects: 1, the ply (deformable)



D1.1 The deliverable name as in the GA



Created entities: 2 or more, the two pieces in which the ply has been split. Any minute chips of material that may be created from the material damage are ignored.

The deformable ply is represented as follows:

- **Type** ply of steel
- **Documentation of the unchanging properties** material properties
- **Documentation of the state of the entity before the action:** pose, shape
- **Documentation of the state of the entity after the action:** N/A (ceased).

Created objects: 2 ply pieces

The created pieces are represented as follows:

- **Type** piece of steel ply
- **Documentation of the unchanging properties** material properties
- **Documentation of the state of the entity before the action:** pose, shape
- **Documentation of the state of the entity after the action:** pose, shape

Action function:

- **Description** the agent pulls the ply downwards displacing each end of the tool, by *10 mm*. This causes a breaking of the ply in two pieces.
- **Duration** *1 sec*.



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 Space¹² (RS) toolkit, which provides HTML5 semantic components for structuring Web authoring forms, template pages, navigation menus, content panels, and

¹² <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.

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

Tip

- 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 buttons.

Presentation name*

The cultural heritage of mastic

Description

Your 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

Showing order on page

1

Text

According to tradition, the skins trees on Chios owed their being 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 witnessed his father being arrested. He was jailed, tortured and finally beheaded. His martyrdom during transport from Chios to Nicopolis, the beheading and the finding of his body remained indelible in the collective memory of local people, 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 Neoschori, Herakia, Kallianisi, Armodia, Pyrgi, Merits, Isth, Kato, Kioni, Kioni, Agios Georgios Sykouras 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

Narrative

The story of Saint Isidore of Chios

Preview

The cultural heritage of mastic

Patron saint of mastic trees

According to tradition, the skins trees on Chios owed their being 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 witnessed his father being arrested. He was jailed, tortured and finally beheaded. His martyrdom during transport from Chios to Nicopolis, the beheading and the finding of his body remained indelible in the collective memory of local people, 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 Neoschori, Herakia, Kallianisi, Armodia, Pyrgi, Merits, Isth, Kato, Kioni, Kioni, Agios Georgios Sykouras 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.

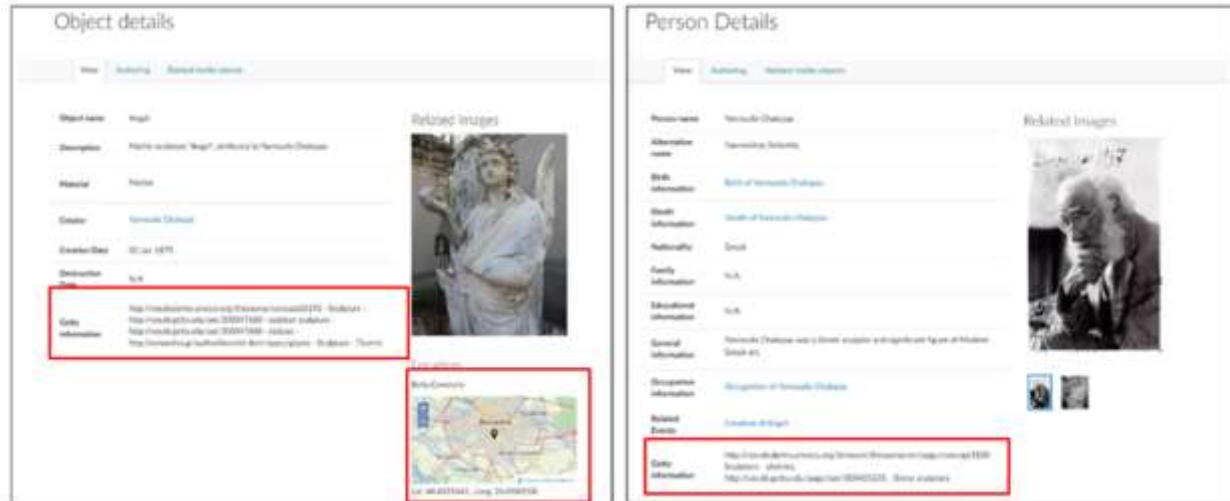


Figure 36. Two examples of semantic annotations using controlled vocabularies, one for a craft product (left) and one for a person (right).

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 are 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.

The screenshot shows the 'Online Platform' interface. At the top, there is a blue header bar with 'Online Platform' on the left and 'Account' on the right. Below the header, there is a search bar with the placeholder text 'Enter information to search'. The main content area displays a form for 'Person name' and 'Alternative name'. The 'Person name' field contains the text 'Alonso Cano'. The 'Alternative name' section contains a list of entries, each with a text field and a language selection dropdown. A red box highlights the language selection dropdown for the first entry, which shows a list of languages: ES, IT, ID, FR, DE, and EN. The dropdown is currently set to 'ES'.

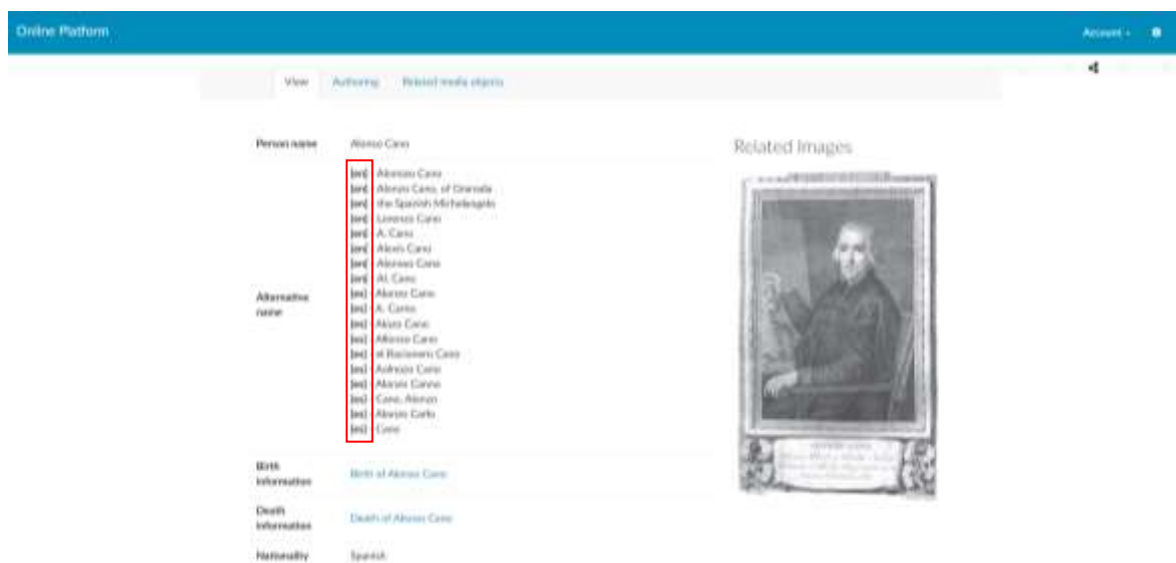


Figure 37. The authoring data entry form (top) and the viewing page (bottom) for a person, illustrate the multilingual capabilities of the platform.

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.

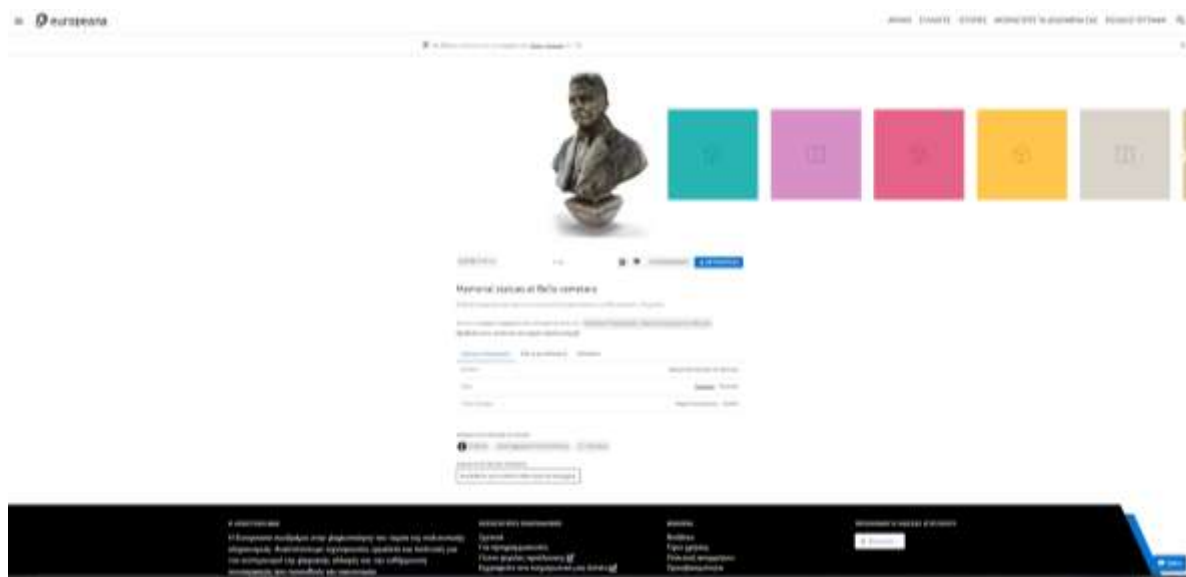


Figure 40. A Web page at Europeana, with a collection of 10 marble sculptures.



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_000155_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.

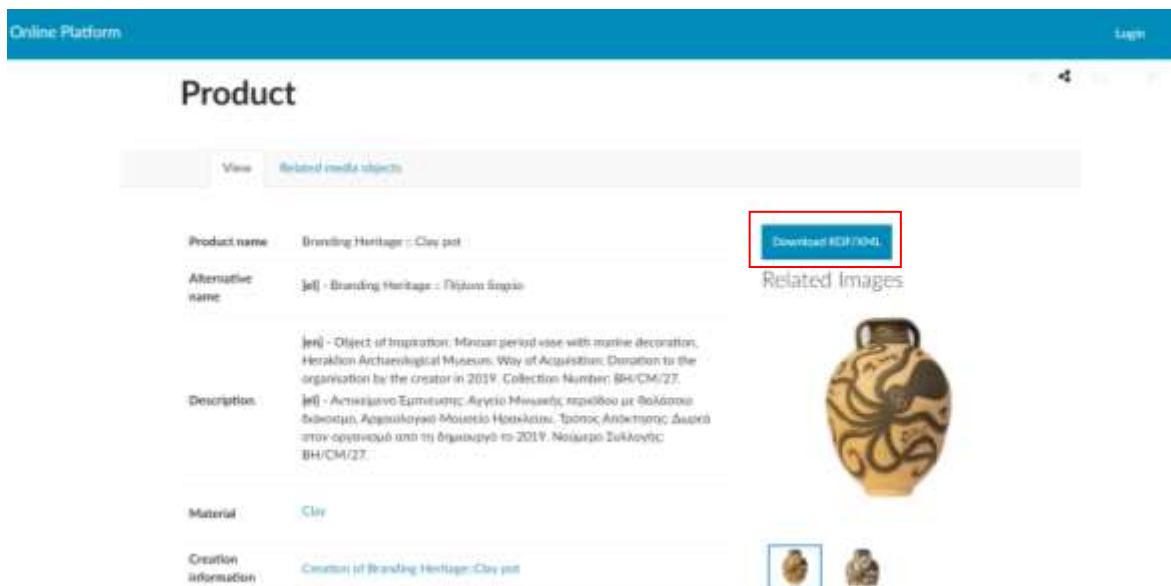


Figure 38. The page for a crafted artefact illustrates the export-to-file capabilities of the platform.

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.

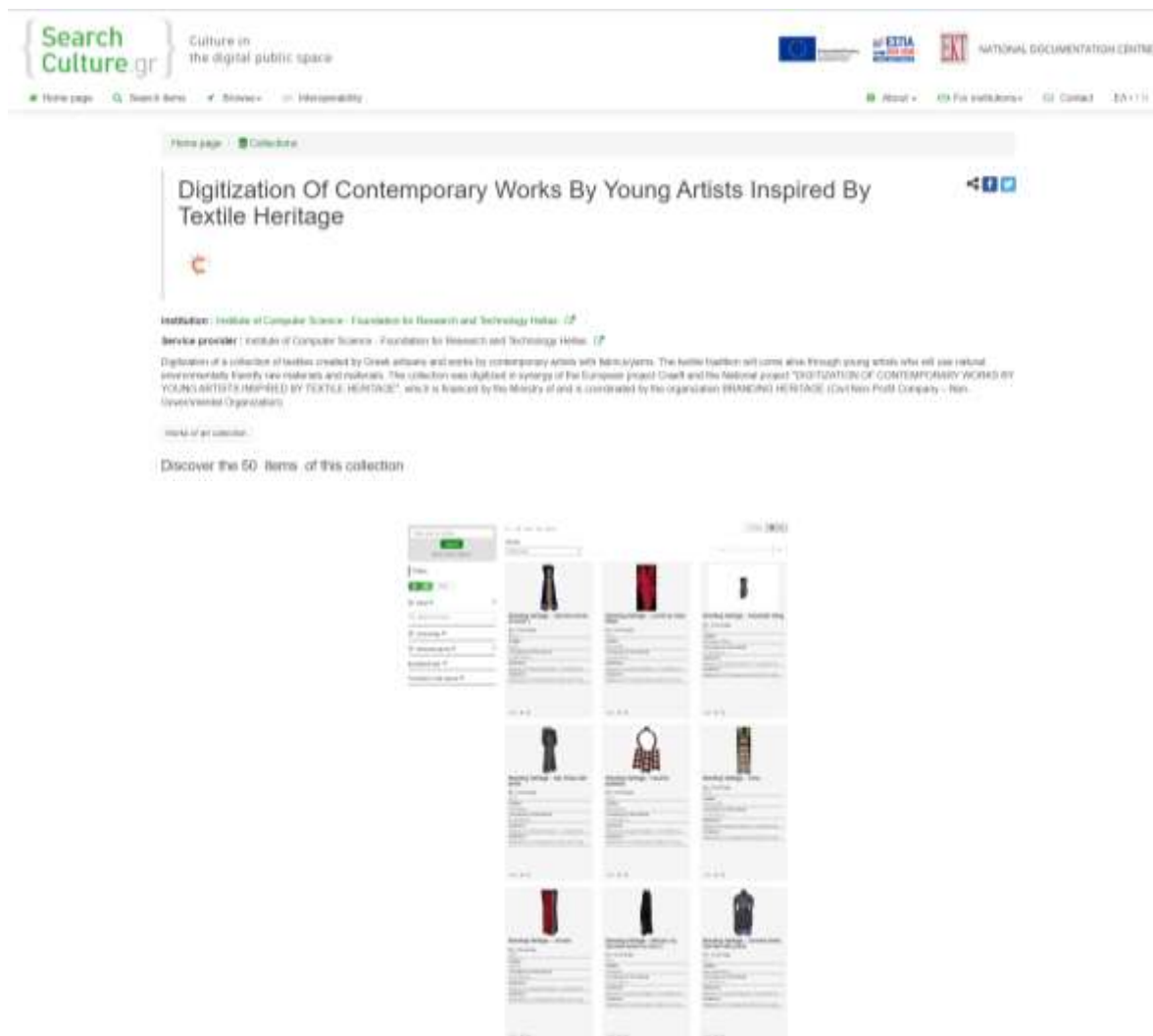


Figure 39. A Web page from the Greek National Aggregator, for a collection of 50 handcrafted garments.

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>

3.7.4. 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.



D1.1 The deliverable name as in the GA



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.

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.



D1.1 The deliverable name as in the GA



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#.Xw1prgzZaR>
- [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/>.