Work Package 1
Multi-agent Architecture

Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

<table>
<thead>
<tr>
<th>Document type</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document version</td>
<td>Final</td>
</tr>
<tr>
<td>Document Preparation Date</td>
<td>Wednesday, February 29, 2012</td>
</tr>
<tr>
<td>Classification</td>
<td>Public</td>
</tr>
<tr>
<td>Author(s)</td>
<td>IPB, Whirlpool, AEA, SIEMENS</td>
</tr>
<tr>
<td>File Name</td>
<td>Deliverable D1-3_1.3.pdf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Ref. Number</th>
<th>246203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Start Date</td>
<td>01/07/2010</td>
</tr>
<tr>
<td>Project Duration</td>
<td>36 months</td>
</tr>
<tr>
<td>Website</td>
<td><a href="http://www.grace-project.org">www.grace-project.org</a></td>
</tr>
</tbody>
</table>

Project founded by the European Commission under the "Seventh Framework Programme" (2007-2013)
Contract n° NMP2-SL-2010-246203
### Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Content</th>
<th>Resp. Partner</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Document structure and preliminary description of the ontology specification</td>
<td>IPB</td>
<td>30/10/2011</td>
</tr>
<tr>
<td>0.2</td>
<td>Definition of the ontology schema and validation</td>
<td>IPB</td>
<td>31/12/2011</td>
</tr>
<tr>
<td>0.3</td>
<td>Improvement of the ontology schema, instantiation with real data, the document grammatical issues</td>
<td>IPB</td>
<td>17/02/2012</td>
</tr>
<tr>
<td>0.4</td>
<td>Revision of the document considering the feedback of the partners</td>
<td>IPB</td>
<td>29/02/2012</td>
</tr>
</tbody>
</table>
Table of Contents

Table of Contents .............................................................................................................. 5
Acronyms .............................................................................................................................. 7
1. Introduction ....................................................................................................................... 9
2. Ontologies for the knowledge representation .............................................................. 11
   2.1. The ontology concept ............................................................................................... 11
   2.2. Methodologies and Languages .............................................................................. 15
   2.3. Existing Ontologies for Manufacturing Systems ................................................... 17
   2.4. Frameworks for Edition and Management of Ontologies .................................... 18
3. Design of the GRACE Meta-model Ontology ............................................................... 21
   3.1. Concepts .................................................................................................................. 22
   3.2. Relations or Predicates .......................................................................................... 25
      3.2.1. comprisesMaterial ......................................................................................... 27
      3.2.2. comprisesOperation ...................................................................................... 27
      3.2.3. createsFunction .............................................................................................. 27
      3.2.4. describesOperation ......................................................................................... 28
      3.2.5. executedBy ...................................................................................................... 28
      3.2.6. hasAppliedRecoveryProcedure .................................................................... 29
      3.2.7. hasCarrier ....................................................................................................... 29
      3.2.8. hasExecuted ..................................................................................................... 30
      3.2.9. hasFailure ........................................................................................................ 30
      3.2.10. hasFailureType ............................................................................................. 31
      3.2.11. hasJournal ..................................................................................................... 31
      3.2.12. hasJournalDetails ........................................................................................ 32
      3.2.13. hasMaterial ................................................................................................... 32
      3.2.14. hasMaterialFamily ....................................................................................... 33
      3.2.15. hasOperation ................................................................................................. 33
      3.2.16. hasOperationPrecedence ............................................................................. 34
      3.2.17. hasPossibleRecoveryProcedures ................................................................. 34
      3.2.18. hasPossibleResources ................................................................................... 35
      3.2.19. hasProcessPlan .............................................................................................. 35
      3.2.20. hasProperty .................................................................................................... 36
      3.2.21. hasSetup ......................................................................................................... 36
      3.2.22. hasTarget ....................................................................................................... 37
      3.2.23. hasTool .......................................................................................................... 37
      3.2.24. isAssociatedWithProduct .......................................................................... 38
      3.2.25. materialUsed .................................................................................................. 38
3.2.26. requiresProperty ........................................................................................................ 39
3.2.27. requiresSetup ............................................................................................................ 39
3.2.28. usesMaterialFamily .................................................................................................. 40
3.3. Attributes ..................................................................................................................... 40
3.4. Restrictions .................................................................................................................... 44
  3.4.1. Operation ................................................................................................................... 44
  3.4.2. Failure ....................................................................................................................... 45
  3.4.3. FailureType ................................................................................................................. 46
  3.4.4. Function .................................................................................................................... 46
  3.4.5. Material .................................................................................................................... 47
  3.4.6. ProcessPlan ............................................................................................................... 47
  3.4.7. Product ..................................................................................................................... 48
  3.4.8. ProductionOrder ....................................................................................................... 49
  3.4.9. Resource ................................................................................................................... 49
  3.4.10. Transporter ............................................................................................................. 50
  3.4.11. Operator .................................................................................................................. 51
  3.4.12. Producer .................................................................................................................. 51
  3.4.13. QualityController ................................................................................................... 52
  3.4.14. Tool ......................................................................................................................... 52
  3.4.15. Journal ..................................................................................................................... 52
  3.4.16. JournalDetails ......................................................................................................... 53
  3.4.17. MaterialFamily ....................................................................................................... 54
4. Validation of the GRACE Ontology .................................................................................. 56

References .......................................................................................................................... 64

Annex A: Formalization of UML to OWL ........................................................................... 67

Annex B: GRACE Meta-model Ontology ........................................................................... 69

Annex C: GRACE Meta-model Ontology in OWL Formalization .................................... 71
  C.1 Classes .......................................................................................................................... 71
  C.2 Object Properties ......................................................................................................... 81
  C.1 Data Type Properties .................................................................................................... 95
## Table of Contents

- Figure 1. Steps in the specification of the GRACE multi-agent system architecture .................. 9
- Figure 2. Need of exchange shared knowledge in distributed systems ................................. 11
- Figure 3. Using ontologies for sharing knowledge ................................................................. 13
- Figure 4. Roles of entities involved in the ontology design ...................................................... 14
- Figure 5. Methodology to build ontologies proposed by Noy and McGuinness ..................... 15
- Figure 6. Evolution of the markup languages ........................................................................ 17
- Figure 7. Screenshot of the OntoEdit editor ......................................................................... 19
- Figure 8. Screenshot of the WebODE editor .......................................................................... 19
- Figure 9. Screenshot of the Protégé editor ........................................................................... 20
- Figure 10. GRACE ontology schema ..................................................................................... 22
- Figure 11. Concepts of the GRACE ontology ......................................................................... 23
- Figure 12. The relation between the Product and ProcessPlan concepts ................................. 25
- Figure 13. Relationships among the GRACE ontology classes ............................................ 26
- Figure 14. Restrictions of predicates associated to the class “Operation” ............................. 44
- Figure 15. Restrictions of predicates associated to the class “Failure” .................................. 45
- Figure 16 Restrictions of predicates associated to the class “Failure Type” ........................... 46
- Figure 17. Restrictions of predicates associated to the class "Function" ................................. 46
- Figure 18. Restrictions of predicates associated to the class “Material” ............................... 47
- Figure 19. Restrictions of predicates associated to the class “ProcessPlan” ......................... 48
- Figure 20. Restrictions of predicates associated to the class “Product” ................................. 48
- Figure 21. Restrictions of predicates associated to the class “Production Order” ............... 49
- Figure 22. Restrictions of predicates associated to the class “Resource” ............................... 50
- Figure 23. Restrictions of predicates associated to the class “Transporter” ........................... 50
- Figure 24. Restrictions of predicates associated to the class “Operator” ............................... 51
- Figure 25. Restrictions of predicates associated to the class “Producer” ............................... 51
- Figure 26. Restrictions of predicates associated to the class “Quality Control” ..................... 52
- Figure 27. Restrictions of predicates associated to the class “Tool” ...................................... 52
- Figure 28. Restrictions of predicates associated to the class “Journal” .................................. 53
- Figure 29. Restrictions of predicates associated to the class “JournalDetails” ..................... 54
- Figure 30. Restrictions of predicates associated to the class “MaterialFamily” ..................... 54
- Figure 31. Consistency Check for the GRACE Ontology using the Pellet tool ..................... 56
Figure 32. OWL 2 Validation Report for the GRACE ontology ................................................................. 57
Figure 33. Representation of class “Product” and its instances ................................................................. 57
Figure 34. Representation of the “Product” and “Material” classes and their instances ........ 58
Figure 35. Representation of the “Resource” class and its instances ......................................................... 59
Figure 36. Representation of the “ProcessPlan”, “Operation” and “Resource” classes and their instances .......................................................................................................................... 60
Figure 37. MaterialFamily-Operation-Function model............................................................................. 61
Figure 38. Representation of the “MaterialFamily”, “Operation” and “Function” classes and their instances ........................................................................................................................................... 61
Figure 39. Representation of the “ProductionOrder”, “Journal” and “JournalDetails” classes and their instances ................................................................................................................................................. 62
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

Acronyms

ACL - Agent Communication Language
ADACOR - ADaptive holonic COntrol aRchitecture for distributed manufacturing systems
AGV - Auto-guided Vehicle
AP - Application Protocol
BOM - Bill of Material
CNC – Computer Numerical Control
DAML - DARPA Agent Markup Language
DOLCE - Descriptive Ontology for Linguistic and Cognitive Engineering
EDI - Electronic Data Interchange
ER - Entity-Relationship
ERP - Enterprise Resource Planning
FIPA - Foundation for Intelligent Physical Agents
FOL - First Order Logic
GRACE - inteGration of pRocess and quAlity Control using multi-agEnt technology
IGES - Initial Graphics Exchange Specification
JADE - Java Agent DEvelopment Framework
KIF - Knowledge Interchange Format
MAS - Multi-Agent System
MASON - MAnufacturing's Semantics Ontology
MES - Manufacturing Execution System
NIST - National Institute of Standards and Technology
OIL - Ontology Inference Layer
OWL - Web Ontology Language
PABADIS’PROMISE - Plant Automation based on Distributed System Product Oriented Manufacturing Systems for Re-Configurable Enterprises
RDF - Resource Description Framework
RDFS - Resource Description Framework Schema
SET - Standard d’Échange et de Transfert
SGML - Standard Generalized Markup Language
STEP - Standard for the Exchange of Product model data
TOVE - Toronto Virtual Enterprise Ontology
UML - Unified Modelling Language
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

URI - Uniform Resource identifier
W3C - World Wide Web Consortium
XML - Extensible Markup Language
1. Introduction

This deliverable contains the outcome of the Task 1.3, entitled “Interoperability and ontology design”, which objective is the design of the ontology to support the knowledge representation that will be used by the multi-agent system integrating process and quality control.

In multi-agent systems (MAS), since each agent has a partial view of the system, the agents need to be able to communicate, in order to achieve a pre-defined goal or solve a problem. The interaction between agents requires that the agents can understand themselves to share knowledge, using interaction protocols, a proper agent communication language and a proper knowledge representation. Since the interaction protocols were addressed in the task 1.2 and the agent-communication language will be the FIPA-ACL (Foundation for Intelligent Physical Agents - Agent Communication Language), specified by the FIPA organization (http://www.fipa.org) (FIPA, 2002), this task will focus on the question of interoperability and knowledge representation. For this purpose, the design of the ontology is crucial to provide a common understanding on the vocabulary used by the intelligent, distributed agents during the exchange and sharing of knowledge, comprising a complementary step to the specification of the GRACE (inteGration of pRocess and quAlity Control using multi-agEnt technology) multi-agent system, elaborated in the Task 1.2 and described in the Deliverable 1.2, as illustrated in Figure 1. Note that the use of domain ontologies is crucial in the development of agent-based control applications, because it is a step toward the integration of heterogeneous agents.

For this purpose, the GRACE ontology will provide the data structure to organize the knowledge that is shared and exchanged between the agents and enable the interoperability between them. In particular, the GRACE ontology formalizes the structure of the knowledge related to:

- The resources and equipments available in the production line.
• The product and process models that describe how to produce the catalogue of products in the production line.

• The description of the production history executed in the production line, including the results from the inspection tests.

The development of the GRACE ontology will comprise two main steps. The first one is related to the design the ontology meta-model (i.e. the ontology schema), which requires the identification of the domain concepts, their attributes, the relations among concepts and the restrictions associated to relations and attributes. The second step is the verification of the correctness of the ontology schema by its instantiation with real production data, creating an ontology model.

The design of the GRACE ontology will use the Protégé ontology editor and knowledge-base framework (http://protege.stanford.edu/), which supports the Web Ontology Language (OWL) (W3C, 2004) and has an easy connection with agent development frameworks, such as JADE (Java Agent DEvelopment Framework).

The document is divided into 4 chapters. After this brief introduction, chapter 2 will provide a contextualization of the need of ontologies to address the knowledge representation and the interoperability in distributed, heterogeneous systems. This chapter also gives an overview about the methodologies and available languages to develop ontologies and the existing ontologies for the manufacturing domain. Chapter 3 is devoted to the design of the GRACE meta-model ontology for production line systems, integrating process and quality control, describing the main concepts, predicates and attributes and restrictions. Finally, Chapter 4 describes the verification of the designed ontology by instantiating the meta-model ontology.
2. Ontologies for the knowledge representation

A collaborative network, enterprise or production system comprises a set of interacting and heterogeneous hardware or software devices or applications. In such collaborative distributed environments, a common understanding of the shared knowledge is required to guarantee their interoperability. In a similar way, in multi-agent systems, characterized to be distributed and heterogeneous systems, each agent representing an organization, factory, cell, device or application, has its own knowledge and needs to communicate to share the knowledge, in order to achieve a pre-defined goal or solve a problem. The interaction between agents, exchanging shared knowledge, requires their understanding, i.e. the understanding of the messages that are used to exchange knowledge, see Figure 2.

![Figure 2. Need of exchange shared knowledge in distributed systems](image)

The exchange of shared knowledge between the distributed agents becomes difficult if each agent has its own knowledge structure. The solution is to use proper mechanisms or techniques that guarantee the common understanding among distributed entities, in analogy with a meeting with attendances coming from different countries and speaking different native languages.

2.1. The ontology concept

The representation and organization of the shared knowledge is not an easy task, as pointed out by the study elaborated by the National Institute of Standards and Technology (NIST) that refers that the automobile sector in United States spends one billion dollars by year to solve interoperability problems (Szykman et al., 2001). In fact, the knowledge sharing may present several problems, namely due to:

- The lack of a common view related to conceptual and terminological terms, leading to confusion and reduced understanding, e.g. the missing common understanding for the meaning of a product or a resource.
- The difficulty to identify the main terms or concepts in a limited domain.
- The inter-operability, reuse and sharing of the knowledge.

The demand for a common understanding for the exchanged knowledge started with the EDI (Electronic Data Interchange) standard that defines a set of rules for the exchange of
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

data between customers and suppliers, each one focusing a particular type of industry, such as TRADNET for the retailing industry, EDIFIC for the electronics industry and ODETTE for the motor manufacturing industry. However, the EDI standard misses the manipulation of technological and engineering information. Other approaches, such as the IGES (Initial Graphics Exchange Specification) in USA and the SET (Standard d’Échange et de Transfert) in France, tried to overcome this problem by addressing the geometrical information, but still missed the technological information.

STEP (Standard for the Exchange of Product model data) is an ISO 10303 standard that addresses the exchange of project data covering all engineering information. STEP defines the object-oriented based EXPRESS language to model the project data in a standard manner. In a similar way to the EDI standard, STEP comprises a set of 26 APs (Application Protocols) to define how to use STEP in particular contexts, for example the AP 203 for the Configuration Controlled 3D Designs of Mechanical Parts and Assemblies and the AP 212 for the Electrotechnical Design and Installation. As examples, Boeing uses STEP to interact with the engine suppliers during the pre-assembly of their aircrafts and General Motors uses STEP to exchange data with its suppliers.

In spite of the potential of the STEP standard, a more general and transparent way to define a common understanding and data semantics is required, as well the capability to reuse and share the knowledge (note that the concept of knowledge is a step further to the concept of information). The concept of ontologies allows to reach this challenge.

The term ontology is vague and not precise. Among the several definitions of ontology that can be found in the literature, the following ones can be pointed out:

- An ontology “defines the basic terms and relations comprising the vocabulary of a topic area as well the rules for combining terms and relations to define extensions to the vocabulary” (Neches et al., 1991).

- An ontology is “a formal, explicit specification of a shared conceptualization” (Gruber, 1995).

- An ontology is “a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world” (Guarino, 1998).

- An ontology “provides meta-information which describes the data semantics, being possible to represent knowledge and use that knowledge to communicate with various types of entities (software agents or humans)” (Fensel, 2004).

- An ontology is “a means of enabling communication and knowledge sharing by capturing a shared understanding of terms that can be used both by humans and programs” (Lai, 2007).

In the second definition, the terms used have the following meaning (Gruber, 1995):

- “Formal” refers to the fact that the ontology should be machine readable.
“Explicit” means that the type of concepts used, and the constraints on their use, is explicitly defined.

“Shared” reflects that ontology should capture consensual knowledge accepted by the communities.

“Conceptualization” refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena.

In spite of all the different definitions, it is consensual that an ontology creates shared understanding, enabling the exchange of knowledge and the capability to reuse that knowledge. In other words, an ontology defines the vocabulary and the semantics that are used in the communication between distributed entities, and the knowledge relating to these terms.

Figure 3 illustrates the use of ontologies to support the interaction among distributed agents, where the agents use the same ontology (but different fragments of the ontology) to express the shared knowledge that is exchanged.

An ontology together with a set of individual instances of classes constitutes a knowledge base. This knowledge includes the definition of the:

- **Concepts or classes**, i.e. the objects of the knowledge domain.
- **Terms**, i.e. the attributes, of each concept, including the meaning of each term (i.e. type of each attribute) and the constraints.
- **Predicates or relations**, i.e. the relationships between the concepts.

Gruber proposed some principles to define ontologies, namely (Gruber, 1993):
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

- **Clarity**: the terms used in the ontology must be clear for those who read and must be readable independently of the social situation or computational situation (implementation independent).

- **Coherence**: the ontology should avoid doubts and misunderstandings about the terms used.

- **Extensibility**: the ontology design should support an easy expansion of the shared vocabulary.

- **Minimal encoding bias**: the design must be conceived in a particular level independent of symbol-level encoding (note that agents sharing knowledge can be implemented in different systems and using different languages).

- **Minimal ontological commitments**: the ontology must demand a minimal ontological compromise in order to support shared activities.

The process for the design of an ontology accounts with the contribution of different entities, each one being expert in its own domain (Knublauch, 2002). The entities involved in the ontology development process, as illustrated in Figure 4, are:

- **System developers**, which are responsible to provide the technological tools to support the development of the ontology.

- **Ontology engineers**, which are responsible for the design and implementation of the ontology.

- **Domain experts**, which are entities that don’t understand the technology to develop the ontology but know reasonably well the knowledge on their domain.

![Figure 4. Roles of entities involved in the ontology design](image)

The ontology architecture is then developed by the ontology engineers according to the suggestions of the domain experts.
A pertinent question is the difference between ontologies and databases. Both ontologies and databases provide a way to store data in a structured way. But ontologies do more, providing the capability for the formal description of data and rules about their context, instead of a static way to store knowledge. As consequence, ontologies allow to deal with incomplete data, to infer answers from current ontology data and to reveal contradictions/inconsistencies. In such way, databases are considered when: i) the schema is small and simple, and ii) all information is available and was not created due to a query. On the other hand, ontologies are considered when: i) the schema is large and complex, and can be created at the query time, ii) it is possible to infer answers (i.e. query answers reflect the schema and the instances/data), and iii) it is necessary to deal with incomplete information.

2.2. Methodologies and Languages

The development of ontologies has a strong derivation from the object-oriented design, presenting some difficulties, namely the manual construction, reuse and definition of concepts. Noy and McGuinness proposes a methodology for the development of ontologies, illustrated in Figure 5, comprising a set of stages to be fulfilled (Noy and McGuinness, 2001).

![Figure 5. Methodology to build ontologies proposed by Noy and McGuinness](image)

The main idea in the development process of an ontology is to verify if existing ontologies can accomplish the proposed requirements, aiming to reuse ontologies. If the requirements are not accomplished, the option is to move to the next phases of the Noy and McGuinness methodology. At this stage, a crucial point is how to represent the knowledge, i.e. the definition of classes, properties and relationships among classes. In the literature it is possible to find diverse forms to represent the knowledge, classified according to different levels of formality (Uschold and Gruninger, 1996):

- Highly informal (expressed in natural language).
- Semi-informal (expressed in a structured form of a natural language).
• Semi-formal (expressed in an artificial and formally defined language).

• Rigorously formal (expressed with precise terms, formal semantic).

Ontologies could be developed by using a wide range of knowledge representation techniques.

KIF (Knowledge Interchange Format) is a language that allows a user to develop ontologies, based on the first-order logic (FOL). It allows the inter-operation of agents with different knowledge bases, through the translation of each knowledge base into the KIF format, which will be shared. When an agent receives a knowledge base in KIF, it converts the data into its own internal form; when the agent needs to communicate with another agent, it maps its internal data structures into KIF.

Ontolingua (Farquhar et al., 1996) is the best-known KIF ontology, intending to provide a common platform in which ontologies developed by different groups can be shared (Wooldridge, 2002). Ontolingua consists on:

• A library of ontologies, expressed in the Ontolingua ontology definition language, which is based on KIF.

• A set of tools for editing and analysing the ontologies.

• A set of translators for converting Ontolingua sources into forms acceptable to implemented knowledge representation systems.

The Resource Description Framework (RDF) (W3C, 2002) is a language used to develop ontologies based on the markup languages, e.g. the Standard Generalized Markup Language (SGML) and the Extensible Markup Language (XML). Since XML is a declarative language, being quite limited, RDF appears to overcome these limitations, e.g. in terms of relations. RDF is used for representing information about resources on the web, thus constituting a basic ontology language. In RDF, the statements used to describe resources are represented as triples, consisting of a subject, predicate and object, i.e. \{S, P, O\}.

The RDF S (Resource Description Framework Schema) is a semantic extension of RDF, namely by extending the RDF vocabulary to allow describing taxonomies of classes and properties, supporting the demand to create a meta-model. It provides mechanisms for describing groups of related resources and the relationships between these resources. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties.

The Web Ontology Language (OWL) (W3C, 2004) is another markup language that semantically extends RDF(S) and is derived from the DAML + OIL (DARPA Agent Markup Language - Ontology Inference Layer) (Horrocks et al., 2001). The basic building blocks in are the classes and relations. OWL has a rich set of modelling constructors, offering improved pre-defined templates, e.g. supporting the inclusion of restrictions in the concepts and predicates. Note that none of the other previous languages offer this kind of feature. Additionally, OWL provides a more expressive manner to represent knowledge, i.e. it is possible to define a model with more and better semantic value. This allows to overcome
the lack in Unified Modelling Language (UML), since in UML it is not possible to represent this as clearly expressive than OWL.

Figure 6 illustrates the evolution of markup languages used to express ontologies, since the SGML to OWL.

![Figure 6. Evolution of the markup languages](image)

Other techniques can be used for the knowledge representation, namely UML used in software engineering and ER (Entity-relationship) diagrams used in databases. These techniques allow a different modelling perspective due to the high abstraction level.

The selection of the proper language to formalize the structure of the knowledge should take into consideration some important issues. The first one is that artificial intelligence based languages (i.e. KIF and markup languages) are better suited to represent and implement ontologies than UML and ER diagrams, allowing to introduce more semantically descriptions. Secondly, languages based on XML are better suited to support the exchange of ontologies between applications. At last, from the set of markup languages, the OWL is the one that provides the most diverse capabilities for description because it has all the characteristics of a markup language and also the reasoning layer that allows representing an ontology in a more expressive manner.

### 2.3. Existing Ontologies for Manufacturing Systems

The domain discussed in this work is the manufacturing field, and particularly the production lines for home appliances. In the literature, several ontologies addressing the manufacturing domain were proposed in the last years by the research community.

The EU FP6 PABADIS’PROMISE (Plant Automation based on Distributed System Product Oriented Manufacturing Systems for Re-Configurable Enterprises) project proposed a reference meta-ontology for manufacturing (Ferrarini et al., 2006). This ontology is very generic where each definition tries to be more abstract, covering a bigger domain. ADACOR (ADAptive holonic CONtrol aRchitecture for distributed manufacturing systems) defines an ontology for manufacturing control domain, which was formalized with the DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) language (Borgo and Leitão, 2004). MASON (Manufacturing’s Semantics Ontology) introduces an ontology with the same objectives, but is expressed with the OWL language in order to unify the ontologies using cognitive architectures, leaving to an implementation of a generic manufacturing ontology (Lemaignan et al., 2006). Other attempts to establish generic manufacturing ontologies are the NIST’s description of shop data model (McLean et al., 2005), the Automation Objects (Lopez and Lasra, 2006), OOONEIDA focusing on the infrastructure of automation components by applying the semantic web technologies (Vyatkin et al., 2005), and TOVE (Toronto Virtual Enterprise Ontology) that describes an ontology for virtual enterprise modelling (Fox, 1992). The ISO 15926 standard (Batres et al., 2007) aims to support the integration of industrial automation systems, being developed by an ontology taking into account diverse variables, including the space and time.
Other ontologies addressing more specific domains in the manufacturing field were proposed, such as the design of ontologies for flexible manufacturing systems (Vrba et al., 2011), for transport systems (Merdan et al., 2008), for assembly lines control (Candido and Barata, 2007), for agent-based reconfiguration of production processes (Al-Safi and Vyatkin, 2007), for rent-a-car business (Andreev et al., 2009) and for supply chain and logistic planning (Andreev et al., 2007). FRISCO is a manufacturing ontology reference that supports the organization of knowledge in automotive supply chains (Hellingrath et al., 2009).

The problem here is to find the ontology that perfectly fits on the pre-requisites established for the GRACE production line domain, since some described ontologies are generic and others focus particular and specific application domains. The idea is to take the insights of several manufacturing ontologies, and particularly from PABADIS’PROMISE and ADACOR, and design a new ontology for the agent-based system integrating process and quality control in production lines, that will be generic enough within the boundaries of the problem specifics. As an example, ADACOR ontology already defines several entities that can be used in the GRACE ontology, but needs to be complemented with new ontological concepts for covering the GRACE particular domain.

2.4. Frameworks for Edition and Management of Ontologies

The development of ontologies is a complex task that requires the support of proper frameworks which assist the creation or manipulation of ontologies and are able to express ontologies in one of many ontology languages.

Examples of relevant criteria for choosing an ontology editor are:

- The degree to which the editor abstracts from the actual ontology representation language used for persistence.
- The visual navigation possibilities within the knowledge model.
- The incorporation of methodologies and languages, in an easy way.
- The ability to import and export foreign knowledge representation languages for ontology matching.
- The licensing costs of the ontology editor.

The use of these tools may lead to an easier ontological learning and also a more productive task in the design of ontologies, supporting the concurrent work of the ontology engineers and the domain experts.

Several frameworks are currently available, namely OntoEdit (Sure et al., 2002), WebODE (Corcho et al., 2002), Protégé (Musen et al., 2003) and Hozo (Kozaki et al., 2002).

OntoEdit, based on CommonKADS (Schreiber et al., 1999), is an ontology editor that has been developed to support the development and maintenance of ontologies through a methodology-guided approach. It provides the capability to develop ontologies with the help of inferencing and be extensible through a plug-in structure. Figure 7 illustrates a screenshot of the OntoEdit tool.
This tool permits the fast development and the possibility to perform different levels of analysis, supported by the graph schemas visualization. OntoEdit is very widely used for reasoning and evaluation phases, and can be used also in online mode.

WebODE is a scalable, extensible and integrated workbench that supports the ontology development process, allowing the design of the ontology by levels, creating different layers that decrease the complexity of the process. The integrated workbench is based on the ontology development methodology METHONTOLOGY. WebODE was one of the first tools for developing ontologies allowing the persistence of the data in Microsoft Access databases. Figure 8 illustrates a screenshot of the WebODE tool.
Protégé is probably the most used tool for the development of ontologies, either the development from the scratch, and the merging, importing, querying and export of ontologies. It is a free, open-source platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. In Protégé it is possible to create ontologies based on different types of expressiveness, being perfect for modelling a knowledge environment by using the Noy and McGuinness methodology (Noy and McGuinness, 2001). Additionally, there are many plugins to be used with Protégé, e.g. to support the validation phase and to export the ontology in different formats. Figure 9 illustrates a screenshot of the Protégé tool.

![Figure 9. Screenshot of the Protégé editor](image)

In this work, Protégé will be used to edit and verify the ontology correctness, since it is a free platform and it provides all necessary characteristics to support a suitable abstraction and technical implementation of the GRACE ontology in a graphical manner.
3. Design of the GRACE Meta-model Ontology

The proposed GRACE ontology aims to represent the knowledge associated to the washing machines production lines domain, which will be used in a multi-agent system application to integrate the production and quality control processes. This ontology considers some insights from PABADIS’PROMISE and ADACOR approaches.

The ontology is based on the definition of a taxonomy of manufacturing components, which contributes to the formalisation and understanding of the problem. The ontology schema defines the vocabulary used by distributed entities, the agents, and indicates the concepts (objects or classes), the predicates (relation between the classes), the terms (attributes of each class), and the meaning of each term (type of each attribute).

The design of an ontology passes by several phases, starting from the conceptualization of the ontology, passing by the specification of the ontology schema and followed by its instantiation. For an easy understanding, the GRACE meta-model ontology has been initially built using the UML class diagram format, and posteriorly edited and instantiated in the Protégé framework using the OWL language.

Figure 10 illustrates the GRACE meta-model ontology using the UML like diagram. In the next sections, the several components of the ontology schema will be deeply analysed. For this purpose, the formal description of the ontological concepts will be performed with the support of the Protégé framework that allows the ontology design in the OWL language.
3.1. Concepts

Concepts are expressions that indicate domain entities with a complex structure that can be defined in terms of classes or objects. Figure 11 illustrates the concepts of the GRACE ontology, as represented in the Protégé editor.
The main concepts defined in the GRACE ontology are described as follows:

- **Failure**: description of an occurred perturbation, including the occurrence date, the applied recover procedure and the recovery time.
- **FailureType**: unexpected event type, like machine failure or delay, which degrades the execution of a production plan.
- **Function**: entity that describes interactions among product components (materials) and/or external environment, e.g. tub contains water and drum move clothes; it is a product function.
- **Journal**: description of the production of a product instance belonging to a production order executed in the production line, including the list of operations performed and the resources that have executed each operation.
• **JournalDetails**: entity that describes the execution of an operation, including the processing time, participants (e.g. type and number of resources), dates and achieved results.

• **Material**: entity used during the production process, e.g. tubs, blocks of steel, bearings, nuts and bolts, according to the BOM.

• **MaterialFamily**: family of the material used during the production process, e.g. bearing or tub (note that each material family could have different materials, e.g. for the family bearing, it is possible to have the bearing A and the bearing B).

• **Operation**: a job executed by one resource like drilling, welding, assembly, inspection and maintenance, that may add value to the product or may measure the value of the product, e.g. the quality control.

• **Operator**: a specialized human resource entity that is responsible for the execution of manual operations, such as an operator connecting the electrical cables.

• **ProcessPlan**: represents the manufacturing process to produce a product, i.e. the description of a sequence of operations (for producing a product) with temporal constraints like precedence of execution.

• **Producer**: a specialized resource entity that is responsible for the execution of producing operations, such as a welding robot or a CNC (Computer Numerical Control) machine.

• **Product**: economic entity (finished or semi-finished), which is produced by the enterprise in a value-adding process (it includes a Bill of Materials (BOM), i.e. the list of materials that are considered as components of a final or intermediate product; it also includes the quantity of each material required).

• **ProductionOrder**: entity obtained by aggregating customer and forecast orders for the production of products, and provided by the ERP (Enterprise Resource Planning)/MES (Manufacturing Execution System) system.

• **Property**: an attribute that characterizes a resource (i.e. a skill) or that a resource should satisfy to execute an operation (i.e. a requirement). It includes a mathematical operator associated to the property value, e.g. a speed equal to 2000 r.p.m..

• **QualityController**: a specialized resource entity that is responsible for the execution of measurement and diagnosis operations, such as a vision control station or a vibration control station.

• **RecoveryProcedure**: entity that describes the procedure to recover from the occurrence of a failure.

• **Resource**: entity that can execute a certain range of operations as long as its capacity is not exceeded. Producer, quality controller, transporter, operator and tool are specializations of resource and inherit its characteristics.

• **Setup**: set of actions that it is necessary to execute in order to prepare a manufacturing resource for the execution of a range of operations.
• **Tool:** a specialized resource entity representing the physical devices used by producer stations and by operators to execute their processing operations, e.g. screwdriver for screwing the counterweights; it may include the physical devices used by transporter resources to execute their handling operations, e.g. grippers.

• **Transporter:** a specialized resource entity that is responsible for the execution of handling/transporting operations, such as an Auto-guided Vehicle (AGV) or a conveyor.

### 3.2. Relations or Predicates

Relations or predicates establish the relationships among the concepts, being characterized by the type of relation and its cardinality.

The most common types of relations are the *association* (representing a relationship between two objects defining its multiplicity), *aggregation* (a specialization of the association relationship, but only in one directional way), *generalization* (representing an inheritance of objects in a form of a “is-a” relationship) and *composition* (a special case of aggregation when the container object has a strong relation with another object; the second cannot exist without the other one). For example, Figure 12 illustrates the *Product* and *Process Plan* concepts that are connected through the relations *hasProcessPlan*, *hasProduct* and *hasProductPrecedence*, all of them are associations.

![Figure 12](image)

*Figure 12. The relation between the Product and ProcessPlan concepts*

Associated to the relation appears the concept of cardinality, which indicates the number of object instances in the association (in case of multiple instances the relation is marked with the symbol “*”). For the example illustrated in Figure 12, the relation *hasProduct* has cardinality 1, which means that a product only have one process plan; on the other hand, the relation *hasProductPrecedence* has cardinality 0..*, which means that a product has several precedences from other products.

Note that the relation *hasProcessPlan* has an inverse relation that is *hasProduct*. This could be useful to represent different paths of knowledge and consequently increase the ontology knowledge.

The main predicates established in the GRACE ontology are illustrated in Figure 13.
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

Figure 13. Relationships among the GRACE ontology classes
The set of predicates defined in the GRACE ontology will be described and briefly formalized in the next points.

### 3.2.1. comprisesMaterial

This predicate establishes the recursive relation between the class “Material”, meaning that a material or component consists of other materials (in a certain quantity) according to the BOM structure, being formally defined as follows:

- \( \text{comprisesMaterial}(x, y) \): material \( x \) uses material \( y \).
- Domain: \( \forall x \exists y(R(x, y) \rightarrow C(x)) \), where \( x \) is the Material.
- Range: \( \forall x, y(R(x, y) \rightarrow C(y)) \), where \( y \) is the Material.

Note that \( R \) and \( C \) intend to represent the Property of the relations.

### 3.2.2. comprisesOperation

This predicate establishes the recursive relation between the class “Operation”, meaning that an operation can be decomposed into several sub-operations, being formally defined as follows:

- \( \text{comprisesOperation}(x, y) \): an operation \( x \) contains the operation \( y \).
- Domain: \( \forall x \exists y(R(x, y) \rightarrow C(x)) \), where \( x \) is the Operation.
- Range: \( \forall x, y(R(x, y) \rightarrow C(y)) \), where \( y \) is the Operation.

### 3.2.3. createsFunction

This predicate establishes the relation between the class “Operation” and the class “Function”, allowing creating a function from the execution of an operation using a material, being formally defined as follows:

- \( \text{createsFunction}(x, y) \): the operation \( x \) creates a function \( y \).
- Domain: \( \forall x \exists y(R(x, y) \rightarrow C(x)) \), where \( x \) is the Operation.
3.2.4. describesOperation

This predicate establishes the relation between the class "JournalDetails" and the class "Operation", meaning that the details about the execution of the operation are reported in the journal details, being formally defined as follows:

- \( \text{describesOperation} (x, y) \): journal details \( x \) describes the execution of the operation \( y \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the JournalDetails.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Operation.

3.2.5. executedBy

This predicate establishes the relation between the class "JournalDetails" and the class "Resource", describing the resource that was executed the operation described in the journal details, being formally defined as follows:

- \( \text{executedBy} (x, y) \): the operation described in the journal details \( x \) was executed by the resource \( y \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the JournalDetails.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Resource.
3.2.6. hasAppliedRecoveryProcedure

This predicate establishes the relation between the class “Failure” and the class “RecoveryProcedure”, describing the recovery procedure applied to solve the occurred failure, being formally defined as follows:

- \( \text{hasAppliedRecoveryProcedure}(x, y) \): the recovery procedure \( y \) was applied to solve the failure \( x \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Failure.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the RecoveryProcedure.

3.2.7. hasCarrier

This predicate establishes the relation between the class “Function” and the class “MaterialFamily”, meaning that the specified material family component is delivering a function, i.e. a material family is designed to provide one or more specific functions to other material family or to external environment, being formally defined as follows:

- \( \text{hasCarrier}(x, y) \): function \( x \) is carried by the material family \( y \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Function.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the MaterialFamily.
3.2.8. hasExecuted

This predicate establishes the relation between the class “Resource” and the class “JournalDetails”, describing the list of operations executed by a specific resource during its production history, being formally defined as follows:

- \( \text{hasExecuted} \ (x, y) \): the resource \( x \) has executed the operation described in the journal details \( y \).
- Domain: \( \forall x \exists y \, (R(x, y) \rightarrow C \, (x)) \), where \( x \) is the Resource.
- Range: \( \forall x, y \, (R(x, y) \rightarrow C \, (y)) \), where \( y \) is the JournalDetails.

3.2.9. hasFailure

This predicate establishes the relation between the class “Resource” and the class “Failure”, meaning that failures can occur during the execution of operations by a resource, being formally defined as follows:

- \( \text{hasFailure} \ (x, y, t) \): a failure \( y \) occurred in resource \( x \) at time \( t \).
- Domain: \( \forall x \exists y \, (R(x, y) \rightarrow C \, (x)) \), where \( x \) is the Resource.
- Range: \( \forall x, y \, (R(x, y) \rightarrow C \, (y)) \), where \( y \) is the Failure.
3.2.10. hasFailureType

This predicate establishes the relation between the class “Failure” and the class “FailureType”, meaning that a failure occurrence is from a specific type of failure, being formally defined as follows:

- \( \text{hasFailureType}(x, y) \): a failure \( x \) belongs to the failure type \( y \).
- Domain: \( \forall x \exists y \left( R(x, y) \rightarrow C(x) \right) \), where \( x \) is the Failure.
- Range: \( \forall x, y \left( R(x, y) \rightarrow C(y) \right) \), where \( y \) is the FailureType.

3.2.11. hasJournal

This predicate establishes the relation between the class “ProductionOrder” and the class “Journal”, meaning that a journal is the description of the execution of a product item defined in the production order, being formally defined as follows:

- \( \text{hasJournal}(x, y) \): a production order \( x \) comprises the production of several product items, each one described by the journal \( y \).
- Domain: \( \forall x \exists y \left( R(x, y) \rightarrow C(x) \right) \), where \( x \) is the ProductionOrder.
- Range: \( \forall x, y \left( R(x, y) \rightarrow C(y) \right) \), where \( y \) is the Journal.
3.2.12. hasJournalDetails

This predicate establishes the relation between the class “Journal” and the class “JournalDetails”, meaning that the journal details describes the execution of the operation belonging to the process plan during the execution of a product item defined in the production order. It is formally defined as follows:

- **hasJournalDetails** (x, y): journal x describes the execution of a product item, comprising the execution of several operations, each one described by journal details y.
- Domain: ∀x∃y(R(x, y) → C(x)), where x is the Journal.
- Range: ∀x, y(R(x, y) → C(y)) , where y is the JournalDetails.

3.2.13. hasMaterial

This predicate establishes the relation between the class “Product” and the class “Material”, meaning that a product consists on a set of materials according to the BOM, being formally defined as follows:

- **hasMaterial** (x, y): product x has the material y.
- Domain: ∀x∃y(R(x, y) → C(x)), where x is the Product.
- Range: ∀x, y(R(x, y) → C(y)) , where y is the Material.
3.2.14. hasMaterialFamily

This predicate establishes the relation between the class “Material” and the class “MaterialFamily”, meaning that a material is from a material family (i.e. a type of material), being formally defined as follows:

- **hasMaterialFamily** \((x, y)\): material \(x\) is from the material family \(y\).
- Domain: \(\forall x \exists y (R(x, y) \rightarrow C(x))\), where \(x\) is Material.
- Range: \(\forall x, y (R(x, y) \rightarrow C(y))\), where \(y\) is MaterialFamily.

3.2.15. hasOperation

This predicate establishes the relation between the class “ProcessPlan” and the class “Operation”, defining the list of operations required to execute a product model, being formally defined as follows:

- **hasOperation** \((x, y)\): a process plan \(x\) contains operation \(y\).
- Domain: \(\forall x \exists y (R(x, y) \rightarrow C(x))\), where \(x\) is the ProcessPlan.
- Range: \(\forall x, y (R(x, y) \rightarrow C(y))\), where \(y\) is the Operation.
3.2.16. hasOperationPrecedence

This predicate establishes the recursive relation between the class “Operation”, defining a precedence to execute an operation, i.e. meaning that the execution of an operation should only be performed after the execution of other operations. It is formally defined as follows:

- \(\text{hasOperationPrecedence}(x, y)\): the execution of operation \(x\) requires the previous execution of operation \(y\).
- Domain: \(\forall x \exists y (R(x, y) \rightarrow C(x))\), where \(x\) is the Operation.
- Range: \(\forall x, y (R(x, y) \rightarrow C(y))\), where \(y\) is the Operation.

3.2.17. hasPossibleRecoveryProcedures

This predicate establishes the relation between the class “FailureType” and the class “RecoveryProcedure”, describing the list of possible recovery procedures that can be applied to solve a failure event type, being formally defined as follows:

- \(\text{hasPossibleRecoveryProcedures}(x, y)\): the failure type \(x\) can be solved by applying the recovery procedure \(y\).
- Domain: \(\forall x \exists y (R(x, y) \rightarrow C(x))\), where \(x\) is the FailureType.
- Range: \(\forall x, y (R(x, y) \rightarrow C(y))\), where \(y\) is the RecoveryProcedure.
3.2.18. hasPossibleResources

This predicate establishes the relation between the class “Operation” and the class “Resource”, meaning that there is a list of potential resources that are able to execute the operation, being formally defined as follows:

- hasPossibleResource \((x, y)\): resource \(y\) is a candidate for the execution of the operation \(x\).
- Domain: \(\forall x \exists y (R(x, y) \rightarrow C(x))\), where \(x\) is the Operation.
- Range: \(\forall x, y (R(x, y) \rightarrow C(y))\), where \(y\) is the Resource.

3.2.19. hasProcessPlan

This predicate establishes the relation between the class “Product” and the class “ProcessPlan”, meaning that the production of a product model requires the execution of a process plan, being formally defined as follows:

- hasProcessPlan \((x, y)\): the production of product \(x\) requires the process plan \(y\).
- Domain: \(\forall x \exists y (R(x, y) \rightarrow C(x))\), where \(x\) is the Product.
- Range: \(\forall x, y (R(x, y) \rightarrow C(y))\), where \(y\) is the ProcessPlan.
3.2.20. hasProperty

This predicate establishes the relation between the class “Resource” and the class “Property” or between the class “Material” and the class “Property”, meaning that a resource has a set of skills that allow it to execute operations or that a material has a set of attributes. It is formally defined as follows:

- $\text{hasProperty}(x, y)$: resource or material $x$ has the property (skill) $y$.
- Domain: $\forall x \exists y (R(x, y) \rightarrow C(x))$, where $x$ is the Resource or Material.
- Range: $\forall x, y (R(x, y) \rightarrow C(y))$, where $y$ is the Property.

3.2.21. hasSetup

This predicate establishes the relation between the class “Resource” and the class “Setup”, meaning that a resource may have different setups that will allow the execution of different operations, being formally defined as follows:

- $\text{hasSetup}(x, y)$: resource $x$ has the setup $y$.
- Domain: $\forall x \exists y (R(x, y) \rightarrow C(x))$, where $x$ is the Resource.
- Range: $\forall x, y (R(x, y) \rightarrow C(y))$, where $y$ is the Setup.
3.2.22. hasTarget

This predicate establishes the relation between the class “Function” and the class “MaterialFamily”, meaning that one material family component is receiving a function. It is formally defined as follows:

- \( \text{hasTarget}(x, y) \): function \( x \) has target of material family \( y \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Function.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the MaterialFamily.

3.2.23. hasTool

This predicate establishes the relation between the class “Producer” and the class “Tool”, meaning that a producer has a set of tools to execute processing operations, being formally defined as follows:

- \( \text{hasTool}(x, y, t) \): producer (a specialization from the resource class) \( x \) has the tool \( y \) available in its internal magazine at time \( t \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Producer.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Tool.
3.2.24. isAssociatedWithProduct

This predicate establishes the relation between the class “ProductionOrder” and the class “Product”, meaning that a production order is related to the production of a certain quantity of an available product model, being formally defined as follows:

- \( \text{isAssociatedWithProduct}(x, y) \): a production order \( x \) is associated to the product \( y \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the ProductionOrder.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Product.

3.2.25. materialUsed

This predicate establishes the relation between the class “JournalDetails” and the class “Material”, meaning that a journal details describes the material used to execute an operation, being formally defined as follows:

- \( \text{materialUsed}(x, y) \): journal details \( x \) describes that the material \( y \) was used to execute the operation.
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the JournalDetails.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Material.
3.2.26. requiresProperty

This predicate establishes the relation between the class “Operation” class and the class “Property”, meaning that the execution of the operation requires the fulfilment of a set of requirements by potential resources, being formally defined as follows:

- \( \text{requiresProperty}(x, y) \): operation \( x \) requires the property \( y \) to be executed.
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Operation.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Property.

3.2.27. requiresSetup

This predicate establishes the relation between the class “Operation” and the class “Setup”, meaning that the execution of the operation requires the existence of a proper setup in the resource, being formally defined as follows:

- \( \text{requiresSetup}(x, y) \): operation \( x \) needs the setup \( y \) to be executed.
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Operation.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the Setup.
3.2.28. usesMaterialFamily

This predicate establishes the relation between the class “Operation” and the class “MaterialFamily”, meaning that the execution of the operation uses a material from a proper family. It is formally defined as follows:

- \( \text{usesMaterialFamily}(x, y) \): operation \( x \) uses a material family \( y \).
- Domain: \( \forall x \exists y (R(x, y) \rightarrow C(x)) \), where \( x \) is the Operation.
- Range: \( \forall x, y (R(x, y) \rightarrow C(y)) \), where \( y \) is the MaterialFamily.

At this stage, the relations defined in the GRACE meta-model ontology are identified and described. Later on, in the section related to the restrictions, the description of the relations will be completed by the definition of the type of relation (connection) and the cardinality associated.

3.3. Attributes

Attributes are values relative to properties of concepts. These values could be DataTypes (e.g. string, integer) or PropertyTypes (i.e. ontology concepts, e.g. resource and operation, established as relations). These attributes work like restrictions to the concepts (other types of restrictions will be analysed in the next section).

The following Table 1 defines the main attributes established in the GRACE ontology. In this definition, the item Domain refers to the concept that holds the attribute and the item Range refers to the type of that attribute, e.g. “XSD.Integer” in case of Integer or “XSD.string” in case of String.
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A statement describing something, e.g. a product or a setup.</td>
<td>Setup, Product, FailureType, Material, Operation, MaterialFamily and RecoveryProcedure</td>
<td>XSD.string</td>
</tr>
<tr>
<td>detailedResult</td>
<td>The detailed description of the results obtained in a measurement/testing operation.</td>
<td>JournalDetails</td>
<td>XSD.string</td>
</tr>
<tr>
<td>dueDate</td>
<td>The date on which an obligation, e.g. the production of a product, must be accomplished.</td>
<td>ProductionOrder</td>
<td>XSD.date</td>
</tr>
<tr>
<td>earliestDate</td>
<td>The date before which an activity or event cannot start.</td>
<td>ProductionOrder</td>
<td>XSD.date</td>
</tr>
<tr>
<td>endTime</td>
<td>The date describing the end of the execution of an activity.</td>
<td>Journal, Journal Details</td>
<td>XSD.date</td>
</tr>
<tr>
<td>expectedDetailedResult</td>
<td>The description of the expected (ideal) result for a function created by the execution of an operation.</td>
<td>Function</td>
<td>XSD.string</td>
</tr>
<tr>
<td>failureID</td>
<td>A non-negative integer number that provides the unique identification of the failure.</td>
<td>Failure</td>
<td>XSD.int</td>
</tr>
<tr>
<td>functionType</td>
<td>The type of the failure that can occur during the production execution.</td>
<td>Function</td>
<td>XSD.string</td>
</tr>
<tr>
<td>journalDetailsId</td>
<td>A non-negative integer number that provides the unique identification of the journal details.</td>
<td>JournalDetails</td>
<td>XSD.int</td>
</tr>
<tr>
<td>journalID</td>
<td>A non-negative integer number that provides the unique identification of the journal (e.g. could be the serial number identified the produced product item).</td>
<td>Journal</td>
<td>XSD.int</td>
</tr>
<tr>
<td>location</td>
<td>A place where something, e.g. a resource, is located.</td>
<td>Resource</td>
<td>XSD.string</td>
</tr>
<tr>
<td>materialFamilyID</td>
<td>A label that provides the identification of the material family.</td>
<td>MaterialFamily</td>
<td>XSD.string</td>
</tr>
<tr>
<td>materialID</td>
<td>A non-negative integer number that provides the unique identification of the material.</td>
<td>Material</td>
<td>XSD.int</td>
</tr>
<tr>
<td>materialType</td>
<td>The type of material to be used in the execution of an operation during the production of a product.</td>
<td>Material</td>
<td>XSD.string</td>
</tr>
<tr>
<td>mathOperator</td>
<td>The mathematical operator that can establish a comparison in the value of a property.</td>
<td>Property</td>
<td>XSD.string</td>
</tr>
<tr>
<td>name</td>
<td>The designation of a thing, e.g. a resource, a property or an operation.</td>
<td>Resource, Property, Operation, Product</td>
<td>XSD.string</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td>occurrenceDate</td>
<td>The date when a failure occurred.</td>
<td>Failure XSD.date</td>
<td></td>
</tr>
<tr>
<td>operationID</td>
<td>A non-negative integer number that provides the unique identification of the operation.</td>
<td>Operation XSD.int</td>
<td></td>
</tr>
<tr>
<td>operationType</td>
<td>The type of operation to be executed, e.g. drilling, painting and welding.</td>
<td>Operation XSD.string</td>
<td></td>
</tr>
<tr>
<td>overallResult</td>
<td>The overall result obtained in a measurement/testing operation, for example OK or KO.</td>
<td>Journal Details XSD.string</td>
<td></td>
</tr>
<tr>
<td>procedureID</td>
<td>A non-negative integer number that provides the unique identification of the recovery procedure.</td>
<td>RecoveryProcedure XSD.int</td>
<td></td>
</tr>
<tr>
<td>processPlanID</td>
<td>A non-negative integer number that provides the unique identification of the process plan.</td>
<td>ProcessPlan XSD.int</td>
<td></td>
</tr>
<tr>
<td>productID</td>
<td>A non-negative integer number that provides the unique identification of the product.</td>
<td>Product XSD.int</td>
<td></td>
</tr>
<tr>
<td>productionOrderID</td>
<td>A non-negative integer number that provides the unique identification of the production order.</td>
<td>ProductionOrder XSD.int</td>
<td></td>
</tr>
<tr>
<td>quantity</td>
<td>A positive rational number that defines the specific amount of things to be produced.</td>
<td>ProductionOrder, Material XSD.int</td>
<td></td>
</tr>
<tr>
<td>recoveryTime</td>
<td>A positive rational number, it gives the indication of the recovery time after a failure (expressed in seconds).</td>
<td>Failure XSD.int</td>
<td></td>
</tr>
<tr>
<td>setOfSymptoms</td>
<td>List of symptoms that may lead to the occurrence of a failure (important to forecast failures and to support the diagnosis).</td>
<td>FailureType XSD.string</td>
<td></td>
</tr>
<tr>
<td>setupID</td>
<td>A non-negative integer number that provides the unique identification of the setup.</td>
<td>Setup XSD.int</td>
<td></td>
</tr>
<tr>
<td>startDate</td>
<td>The date describing the start of the execution of an activity.</td>
<td>ProductionOrder, Journal, Journal Details XSD.Date</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>The current state of the resource, e.g. waiting, running and broken.</td>
<td>Resource, Journal XSD.string</td>
<td></td>
</tr>
<tr>
<td>thisID</td>
<td>A non-negative integer number that provides the unique identification of the resource.</td>
<td>Resource XSD.int</td>
<td></td>
</tr>
</tbody>
</table>
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

<table>
<thead>
<tr>
<th>toolType</th>
<th>The type of tool used to perform a processing or handling operation.</th>
<th>Tool</th>
<th>XSD.string</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The designation of a type, e.g. a description. For example, a quality controller could be a vision station or a vibration control station.</td>
<td>Resource, Operation, FailureType, Material, Function</td>
<td>XSD.string</td>
</tr>
<tr>
<td>unit</td>
<td>The description of the units used to represent the value in the property.</td>
<td>Property</td>
<td>XSD.string</td>
</tr>
<tr>
<td>value</td>
<td>A specific amount related to a property type.</td>
<td>Property</td>
<td>XSD.string</td>
</tr>
</tbody>
</table>

From the analysis of the list of attributes, the concept Property requires a special attention. The tuple {name, value, unit, mathOperator}, associated to the datatype property, represents the properties and characteristics exhibited by a resource or required by an operation to be performed. Examples of properties are:

- **LifeTime**: a positive rational number that defines the life time of a tool (expressed in seconds).
- **Axes**: a non-negative integer, e.g. the number of axes of a machine.
- **ProcessingType**: a type of processing e.g. turning, milling, or drilling.
- **Repeatability**: a non-negative float, it gives an indication about the degree to which repeated measurements under unchanged conditions show the same results (expressed in mm).
- **FeedRate**: a positive rational number, it gives the feed rate of a specific axis (expressed in mm/rot).
- **SpindleSpeed**: a range of non-negative integers, it gives the spindle speed in the form [min, max] (expressed in rpm).
- **Tailstock**: a range of non-negative integers, it gives the size in form [min,max] of pieces that the machine can process (expressed in mm).
- **Payload**: positive integer, it gives the maximum load of the robot that guarantees the repeatability (expressed in kg).
- **MaxReachability**: positive integer, it gives the work volume of the robot (expressed in mm).
- **MagazineCapacity**: non-negative integer, it gives the number of tools or grippers that the magazine of a machine or robot can store.
- **CuttingSpeed**: a positive rational number, it gives the cutting speed (expressed in mm/s).
- **Wear**: a positive number defining the wear of a cutting tool (expressed in mm).
- **CycleTime**: a positive number defining the length of the performed quality control task (expressed in seconds).
• **PercentOfScraps**: a positive number defining the percentage of scraps identified (referred to the total number of inspected items).

### 3.4. Restrictions

The design of an ontology may consider the restrictions associated to the ontological concepts, restricting the values and the cardinality associated to the identified predicates. The restrictions related to the values are created to implement the obligation of having the relation among concepts, and the restrictions about the cardinality are related to create the obligation in terms of number in those relations. Here, it is also important to consider the restriction in the attributes associated to the allowed values.

This section defines the predicates restrictions, by directly mapping the UML classes into the OWL concepts restrictions (Kiko and Atkinson, 2008) (see Annex A for more details). In this description, it is analysed separately each concept and identified the restrictions of the existing predicates associated to the concept. The fulfilment of the identified restrictions is crucial to preserve the consistency of the ontology.

#### 3.4.1. Operation

The class “Operation” has several attributes as illustrated in Figure 14. Some of them are relations with the classes, namely requiresProperty (with the class “Property”), requiresSetup (with the class “Setup”), createsFunction (with the class “Function”), hasPossibleResources (with the class “Resource”), usesMaterialFamily (with the class “MaterialFamily”), and comprisesOperation and hasOperationPrecedence (with itself). Additionally, it has DataType properties, such as duration, name, operationID and type.

![Figure 14. Restrictions of predicates associated to the class “Operation”](image-url)
The defined relations have specific restrictions. In terms of the type of connection, all of them are associations among the classes. In terms of cardinality, the predicates createsFunction, requiresSetup and usesMaterialFamily have the restriction \( \textit{minCardinality} = 1 \), i.e. the cardinality of these relations is equal to 1, or in a more formal manner,

\[
\forall x (A(x) \rightarrow \{ y \mid R(x, y) \} = N), \text{ where } N = 1
\]

The predicates requiresProperty, hasPossibleResources, comprisesOperation, and hasOperationPrecedence don’t present any restriction in terms of cardinality.

### 3.4.2. Failure

The class “Failure” has several attributes as illustrated in Figure 15. Some of them are relations with the classes, namely hasFailureType (with the class “FailureType”) and hasAppliedRecoveryProcedure (with the class “RecoveryProcedure”). Additionally, it has DataType properties, such as failureID, occurrenceDate and recoveryTime.

![Figure 15. Restrictions of predicates associated to the class “Failure”](image-url)

The defined relations have specific restrictions. In terms of the type of connection, all of them are associations among the classes. In terms of cardinality, the predicate hasFailureType has the restriction \( \textit{minCardinality} = 1 \), i.e. the cardinality of this relation is equal to 1, or in a more formal manner,

\[
\forall x (A(x) \rightarrow \{ y \mid R(x, y) \} = N), \text{ where } N = 1
\]

The predicate hasAppliedRecoveryProcedure don’t present any restriction in terms of cardinality.
3.4.3. FailureType

The class “FailureType” has several attributes as illustrated in Figure 16. One is the relation hasPossibleRecoveryProcedures (with the class “RecoveryProcedure”), and others are DataType properties, such as name, description, setOfSymptoms and type.

![Figure 16](image1)

**Figure 16** Restrictions of predicates associated to the class “Failure Type”

The predicate has restrictions due to the association among the classes. In terms of cardinality, the predicate hasPossibleRecoveryProcedures don’t present any restriction.

3.4.4. Function

The class “Function” has several attributes as illustrated in Figure 17. One is the relation hasTarget (with the class “MaterialFamily”), and others are DataType properties, such as name, expectedDetailedResult and type.

![Figure 17](image2)

**Figure 17.** Restrictions of predicates associated to the class “Function”

The defined relation has a specific restriction: in terms of cardinality, the cardinality of this relation is equal to 1, i.e. minCardinality = 1, or in a more formal manner,
∀x(A(x) → | {y | R(x,y)}| = N), where N is 1

3.4.5. Material

The class “Material” has several attributes as illustrated in Figure 18. Some of them are relations with the classes, namely hasMaterialFamily (with the class “MaterialFamily”), hasProperty (with the class “Property”) and comprisesMaterial (with itself). Additionally, it has DataType properties, such as description, name, materialID, quantity and type.

![Figure 18. Restrictions of predicates associated to the class “Material”](image)

The defined relations have specific restrictions. In terms of the type of connection, all of them are associations among the classes. In terms of cardinality, the predicate hasMaterialFamily has the restriction minCardinality = 1, i.e. the cardinality of this relation is equal to 1, or in a more formal manner,

∀x(A(x) → | {y | R(x,y)}| = N), where N is 1

The predicates comprisesMaterial and hasProperty don’t present any restriction in terms of cardinality.

3.4.6. ProcessPlan

The class “ProcessPlan” has several attributes as illustrated in Figure 19. One is the relation hasOperation (with the class “Operation”), and others are DataType properties, such as processPlanID and name.
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

Figure 19. Restrictions of predicates associated to the class “ProcessPlan”

The defined relation is an aggregation and has a specific restriction: in terms of cardinality, the cardinality of this relation is more than 1, i.e. \( \text{minCardinality} \geq 1 \), or in a more formal manner,

\[
\forall x(A(x) \rightarrow \{ y | R(x, y) \}) \geq N, \text{ where } N \text{ is } 1
\]

3.4.7. Product

The class “Product” has several attributes as illustrated in Figure 20. Some of them are relations with the classes, namely hasMaterial (with the class “MaterialFamily”) and hasProcessPlan (with the class “ProcessPlan”). Additionally, it has DataType properties, such as description, productId and name.

Figure 20. Restrictions of predicates associated to the class “Product”

The defined relations have specific restrictions. In terms of the type of connection, all of them are associations among the classes. In terms of cardinality, the predicate hasProcessPlan has the restriction \( \text{minCardinality} = 1 \), i.e. the cardinality of this relation is equal to 1, or in a more formal manner,

\[
\forall x(A(x) \rightarrow \{ y | R(x, y) \}) = N, \text{ where } N \text{ is } 1
\]

The predicate hasMaterial doesn’t present any restriction in terms of cardinality.
3.4.8. ProductionOrder

The class “ProductionOrder” has several attributes as illustrated in Figure 21. Some of them are relations with the classes, namely isAssociatedWithProduct (with the class “Product”) and hasJournal (with the class “Journal”). Additionally, it has DataType properties, such as productionOrderID, quantity, startDate, earliestDate and dueDate.

<table>
<thead>
<tr>
<th>ProductionOrder</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>isAssociatedWithProduct</td>
<td>Instance</td>
<td>Product</td>
</tr>
<tr>
<td>earliestDate</td>
<td>String*</td>
<td></td>
</tr>
<tr>
<td>dueDate</td>
<td>String*</td>
<td></td>
</tr>
<tr>
<td>productionOrderID</td>
<td>Integer*</td>
<td></td>
</tr>
<tr>
<td>hasJournal</td>
<td>Instance*</td>
<td>Journal</td>
</tr>
<tr>
<td>startDate</td>
<td>String*</td>
<td></td>
</tr>
<tr>
<td>quantity</td>
<td>String*</td>
<td></td>
</tr>
</tbody>
</table>

Figure 21. Restrictions of predicates associated to the class “Production Order”

The defined relations have specific restrictions. The predicate isAssociatedWithProduct is an association that has the restriction minCardinality = 1, i.e. the cardinality of this relation is equal to 1, or in a more formal manner,

$$\forall x (A(x) \rightarrow | \{ y | R(x, y) \}| = N)$$, where $N$ is 1

The predicate hasJournal is a composition that has the restriction minCardinality $\geq 1$, i.e. the cardinality of this relation is more than 1, or in a more formal manner,

$$\forall x (A(x) \rightarrow | \{ y | R(x, y) \}| \geq N)$$, where $N$ is 1

3.4.9. Resource

The class “Resource” has several attributes as illustrated in Figure 22. Some of them are relations with the classes, namely hasFailure (with the class “Failure”), hasProperty (with the class “Property”) and hasSetup (with the class “Setup”). Additionally, it has DataType properties, such as name, thisID, type, state and location.
The defined relations have specific restrictions. In terms of the type of connection, all of them are associations among the classes. In terms of cardinality, the predicates don’t present any restriction in terms of cardinality.

A special remark on the attribute \textit{thisID}: Since the inherited classes will received the same kind of attributes of the Resource class, it does not make any sense to create a unique attribute for the identification of each one, so a generic label attribute is created to be derived lately by each inherited class.

### 3.4.10. Transporter

The class “\textit{Transporter}” has several attributes as illustrated in Figure 23, all of them are inherited from the class “\textit{Resource}”, since the class “\textit{Transporter}” is a specialization of the class “\textit{Resource}”. In terms of restrictions, they are also inherited from the parent restrictions.
3.4.11. Operator

The class “Operator” has several attributes as illustrated in Figure 24, all of them are inherited from the class “Resource”, since the class “Operator” is a specialization of the class “Resource”. In terms of restrictions, they are also inherited from the parent restrictions.

![Figure 24. Restrictions of predicates associated to the class “Operator”](image)

3.4.12. Producer

The class “Producer” has several attributes as illustrated in Figure 25, most of them are inherited from the class “Resource”, since the class “Producer” is a specialization of the class “Resource”. In terms of restrictions, they are also inherited from the parent restrictions.

![Figure 25. Restrictions of predicates associated to the class “Producer”](image)

Besides the inherited attributes, this class has the relation hasTool (with the class “Tool”), that in an association between the two classes. In terms of cardinality, the predicate doesn’t have any restriction.
3.4.13. QualityController

The class “QualityController” has several attributes as illustrated in Figure 26, all of them are inherited from the class “Resource”, since the class “QualityController” is a specialization of the class “Resource”. In terms of restrictions, they are also inherited from the parent restrictions.

![Figure 26. Restrictions of predicates associated to the class “Quality Control”](image)

3.4.14. Tool

The class “Tool” has several attributes as illustrated in Figure 27, most of them are inherited from the class “Resource”, since the class Tool is a specialization of the class “Resource”. In terms of restrictions, they are also inherited from the parent restrictions.

![Figure 27. Restrictions of predicates associated to the class “Tool”](image)

Besides the inherited attributes, this class has the attribute toolType as DataType properties.

3.4.15. Journal

The class “Journal” has several attributes as illustrated in Figure 28. One is the relation hasJournalDetails (with the class “JournalDetails”), and others are DataType properties, such as journalID, state, startDate and endDate.
The defined association relation is a composition and has a specific restriction: in terms of cardinality, the cardinality of this relation is more than 1, i.e. minCardinality $\geq 1$, or in a more formal manner,

$$\forall x (A(x) \rightarrow | \{ y \mid R(x,y) \}| \geq N)$$, where $N$ is 1

### 3.4.16. JournalDetails

The class “JournalDetails” has several attributes as illustrated in Figure 29. Some of them are relations with the classes, namely describesOperation (with the class “Operation”), executedBy (with the class “Resource”) and materialUsed (with the class “Material”). Additionally, it has DataType properties, such as journalDetailsID, name, overallResult, detailedResult, startDate and endDate.
The defined relations have specific restrictions. In terms of the type of connection, all of them are associations among the classes. In terms of cardinality, the predicates `describesOperation`, `materialUsed` and `executedBy` have the restriction $\text{minCardinality} = 1$, i.e. the cardinality of this relation is equal to 1, or in a more formal manner,

$$\forall x(A(x) \rightarrow |\{y \mid R(x, y)\}| = N), \text{ where } N \text{ is } 1$$

### 3.4.17. MaterialFamily

The class “MaterialFamily” has several attributes as illustrated in Figure 30. One is the relation `hasCarrier` (with the class “Function”) and the others are DataType properties, such as `materialFamilyID` and `description`.

---

**Figure 29. Restrictions of predicates associated to the class “JournalDetails”**

**Figure 30. Restrictions of predicates associated to the class “MaterialFamily”**
The $\text{hasCarrier}$ is an association and in terms of cardinality has the restriction $\text{minCardinality} = 1$, i.e. the cardinality of this relation is equal to 1, or in a more formal manner,

$$\forall x (A(x) \rightarrow | \{ y \mid R(x, y) \} | = N),$$

where $N$ is 1.
4. Validation of the GRACE Ontology

At this stage, the ontology for production lines integrating quality and process control, was designed (and edited in the Protégé framework). An important step before its implementation and usage is the verification of its correctness and the adjustment of some ontological entities. For this purpose, this chapter describes the validation of the designed ontology by using the Protégé framework.

A first verification can be performed by using the Java framework for building Semantic Web (JENA\(^1\)) that provides the environment to handle the RDF, RDFS and OWL languages. It provides several reasoning tools, like Pellet (http://pellet.owldl.com), to check the consistency of the ontology and all the characteristics of an ontology. In this work, this validation test was performed by using the Pellet tool provided by the Protégé framework, which allows to verify the consistency of the ontology based on four checking tests: the subsumption checking, the equivalence checking, the consistency checking and the instantiation checking. Figure 31 shows that the GRACE ontology has passed with success the set of checking tests.

Figure 31. Consistency Check for the GRACE Ontology using the Pellet tool

A second test can be performed by submitting the ontology to an OWL Validator\(^2\), to check the ontology compliance with the W3C standard.

---

\(^1\) Jena is an open source Semantic Web framework for Java, that can be accessed at http://jena.sourceforge.net/.

\(^2\) The OWL validator accepts ontologies written in RDF/XML, OWL/XML, OWL Functional Syntax, Manchester OWL Syntax; it can be accessed at http://owl.cs.manchester.ac.uk/validator/.
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

A third validation can be performed in a manual way, by instantiating the ontology concepts for a particular case study, to support the verification of the ontology correctness and the detection of missing or misunderstanding ontological entities. In fact, the manual verification allows illustrating the relevance of the proposed concepts, relations, and to improve, add, modify or remove some of the proposed concepts, relations, attributes or restrictions.

This section describes the manual verification performed by instantiation for a case study derived from a washing machine production line. The representation of the GRACE ontology classes, their relations and their own instances for this case study is illustrated in the Annex B. In order to achieve a better understanding, the instantiation will be presented by analysing separately different parts of the ontology model, i.e. different fragments.

Figure 33 illustrates two different instances of the class “Product”, i.e. “_859201049010_0000” and “_859201049011_1111”, representing two different product models that can be produced in the production line.

Figure 32. OWL 2 Validation Report for the GRACE ontology

![Figure 32](image-url)

Figure 33. Representation of class “Product” and its instances

<table>
<thead>
<tr>
<th>Product</th>
<th>hasMaterial</th>
<th>Instance*</th>
<th>Material</th>
<th>hasProcessPlan</th>
<th>Instance</th>
<th>ProcessPlan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>_859201049011_1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hasMaterial = _46197408451</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hasProcessPlan = FrontLoader</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>productId = 85920104901111</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>description = LA-1200-WP-PL-E8+IL-K64-A22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>name = AWQER9120</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>hasMaterial</th>
<th>Instance*</th>
<th>Material</th>
<th>hasProcessPlan</th>
<th>Instance</th>
<th>ProcessPlan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>_859201049010_0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hasMaterial = _46197408451</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hasProcessPlan = FrontLoader</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>productId = 859201049010000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>description = LA-1200-WP-PL-E8+IL-K64-A11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>name = AWQER9120</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note that the relation “Io”, represented between the class and the object, means “instance of”. The two instances of the product class are filled with the real data in their attributes, e.g. in the case of the instance “_859201049010_0000”, the DataType attributes name and description are respectively filled with “AWOE9120” and “LA-1200-WP-PL-E8+IL-K64-A11”, and the ObjectProperty attribute hasProcessPlan is filled with the link to the “ProcessPlan” class “FrontLoader”.

In a progressive manner, it is possible to analyse more parts of the ontology. Figure 34 represents the several instances of the “MaterialFamily” class, namely “RearTub”, “Hub”, “ABearing”, “BBearing”, “ShaftSeal” and “CrossPiece”. The relation hasMaterial between the “Product” and “MaterialFamily” classes is also defined for the different instances of the “Product” classes. After instantiating the different types of materials, the relationship between the classes was automatically reflected among the objects; e.g. the product “_859201049010_0000” has the following materials: “_461971408451”, “_461971422013” and “_461971422011”. This exercise is followed to the rest of the instances.

![Figure 34](image.png)

Figure 34. Representation of the “Product” and “Material” classes and their instances

Figure 35 shows the class “Resource” and the six instances considered in this model: “Seal_Insertion1”, “Seal_Insertion2”, “Bearing_Insertion1”, “Bearing_Insertion2”, “Marriage1” and “Marriage2”. Each one of these instances is filled with the details about its characteristics.
Figure 35. Representation of the “Resource” class and its instances

Figure 36 illustrates the validation of the fragment of the ontology comprising the “ProcessPlan”, “Operation” and “Resource” concepts. Here, it is possible to verify that the process plan “FrontLoader”, that defines the process to execute the product “_859201049010_0000”, comprises the execution of three operations:

- “BearingInsertion-Program1”, which uses components from the “ABearing”, “BBearing” and “RearTub” material families.

- “SealInsertion-Program1”, which should only be executed after the execution of the operation “BearingInsertion-Program1”, and uses components from the “RearTub” and “ShaftSeal” material families.

- “Marriage-RearTub-Drum-Program1”, which should only be executed after the execution of the operation “SealInsertion-Program1”, and uses components from the “ABearing”, “BBearing” and “CrossPiece” material families.
Figure 36. Representation of the “ProcessPlan”, “Operation” and “Resource” classes and their instances

Also in this fragment, it is possible to verify the indication of the possible resources that can execute each operation. In this way:

- The operation “BearingInsertion-Program1” can be executed by the resources “Bearing_Insertion1” and “Bearing_Insertion2”,
- The operation “SealInsertion-Program1” can be executed by the resources “Seal_Insertion1” and “Seal_Insertion2”,
- The operation “Marriage-RearTub-Drum-Program1” can be executed by the resources “Marriage1” and “Marriage2”.

Another important fragment to be analysed, is the static data model related to the “MaterialFamily”, “Operation” and “Function” model, illustrated in Figure 37, which is an innovative feature of the GRACE ontology.
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

Figure 37. MaterialFamily-Operation-Function model

Figure 38 illustrates the instantiation for the MaterialFamily-Operation-Function model. Here, it is possible to verify that the several operations described in the ontology model will create, each one, a function.

Examples of the functions created are the “BBearing_Hold_CrossPiece”, “ABearing_Hold_CrossPiece”, “RearTub_Hold_ABearing”, “RearTub_Hold_BBearing” and “RearTub_Hold_ShaftSeal”.

The relations between the class “Function” and the class “MaterialFamily”, i.e. hasCarrier and hasTarget, allow indicating which components are involved in the operation that creates
the function, defined through the relation between the “Function” and “Operation” classes. For example, the function “ABearing_Hold_CrossPiece” is completely defined by the following relations:

- hasCarrier: “ABearing”
- hasTarget: “CrossPiece”
- createsFunction: “Marriage-RearTub-Drum-Program1”

Another perspective of the ontology is related to the classes that describe the dynamic production data related to the execution of production orders in the production line, i.e. the classes “ProductionOrder”, “Journal” and “JournalDetails”, as illustrated in Figure 39.

Figure 39. Representation of the “ProductionOrder”, “Journal” and “JournalDetails” classes and their instances

Here, it is considered a production order to produce a batch of 2 items of the product “_859201049010_0000”. The order leads to the production of the two machines described
by the instances “Journal_411142011153” and “Journal_411142011154” from the “Journal” class. Since the production of this product model requires the execution of three operations, as described in the process plan, each one of the instances “Journal_411142011153” and “Journal_411142011154” has three instances of the “JournalDetails” class, related to the description of the execution of each operation. For example, for the machine produced in the production line and described with “Journal_411142011153”, the details are:

- “Journal_Details1”: the operation “BearingInsertion-Program1” was performed by the resource “Bearing_Insertion1” with an overall result of OK.

- “Journal_Details2”: the operation “SealInsertion-Program1” was performed by the resource “Seal_Insertion1” with an overall result of OK.

- “Journal_Details3”: the operation “Marriage-RearTub-Drum-Program1” was performed by the resource “Marriage1” with an overall result of OK.

The manual validation of the meta-model ontology allowed a better understanding of the considered domain and the correction of some misunderstanding issues in the design of the ontological concepts, predicates, attributes and restrictions.

Other tools and methods may also be used for evaluating the functionality of the ontology, e.g. OntoMetric (Lozano-Tello and Gomez-Perez, 2004) and OntoClean (Guarino and Welty, 2002) that are used to detect both formal and semantic inconsistencies. In this work, the validation was performed using the capability provided by the Protégé framework, as previously described.

At this stage, the designed ontology is ready to be used, i.e. implemented to be integrated in the multi-agent system specified in the deliverable 1.2. The implementation, and its integration within the GRACE multi-agent system, will be discussed in the deliverable 1.4.
Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control

References


Deliverable D1.3

Document defining the ontology for line-production system, integrating process and quality control


Annex A: Formalization of UML to OWL

The ontology modelling phase is easier if the abstraction of the modelling tool is simple and abstract, where any person can understand and develop. This top-down approach starts with the UML diagram and finishing with the ontology implementation based on OWL.

As discussed earlier, the formalization of UML and OWL have few differences, and consequently the ontology engineer should be aware of these differences where there is no direct mapping between these methods of modelling and knowledge representation (Schreiber, 2005). The three major differences are:

- Inside the domain of the classes defined in OWL, when it is create a triple, it always has a direction, which in the case of UML is justified with the attributes and a special case are the associations. Thus OWL properties provide the property $\text{InverseOF}$, which represents associations with bi-directionality.

- The relationships constructed through the properties in OWL are always binary, i.e. the predicate always exists between two concepts, thereby making it an advantage in terms of simplification, but the ontology engineer should be aware of this limitation, to get around this problem, and create relationships $n$-ary. In UML modelling, an association with multiple relationships is more intuitive (W3C, 2004).

- Classes in OWL are identified by a name, which should be unique in the ontology and known as the Uniform Resource identifier (URI). In UML modelling, it is possible to create multiple classes with the same name, without breaking modelling rules (Kiko and Atkinson, 2008).

Table 2 summarises the mapping of UML into OWL concepts.
Table 2. Mapping UML and OWL Concepts

<table>
<thead>
<tr>
<th>UML</th>
<th>OWL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Class</td>
<td>Represents a concept.</td>
</tr>
<tr>
<td>Attribute</td>
<td>DataTypeProperty</td>
<td>If it is a specific data type, it will have a direct mapping.</td>
</tr>
<tr>
<td>Association</td>
<td>Object Property</td>
<td>If it is a specific association, it will have a direct mapping.</td>
</tr>
<tr>
<td>AssociationClass</td>
<td>Class and ObjectProperty</td>
<td>Represents a class generalization.</td>
</tr>
<tr>
<td>Generalization between Classes</td>
<td>subClassOf</td>
<td>Represents the generalization between two classes.</td>
</tr>
<tr>
<td>Generalization between Associations</td>
<td>subPropertyOf</td>
<td>Represents the multiplicity that is used in UML, and has a direct relation with OWL cardinality.</td>
</tr>
<tr>
<td>Multiplicities</td>
<td>Cardinalities</td>
<td>Represents the reunion between classes.</td>
</tr>
<tr>
<td>(Generalization)</td>
<td>UnionOf</td>
<td>Represents the intersection between classes.</td>
</tr>
<tr>
<td>Generalization</td>
<td>IntersectionOf</td>
<td>Represents for intersection between classes.</td>
</tr>
<tr>
<td>-</td>
<td>ComplementOf</td>
<td>UML is not capable of specifying a class as the complement of another class.</td>
</tr>
<tr>
<td>Object</td>
<td>OneOf</td>
<td>In UML the term Instance represents the individual OWL.</td>
</tr>
</tbody>
</table>

The validation performed in the Protégé framework has a strong derivation of mathematical expressions. Table 3 describes the formal validation of the OWL restrictions.

Table 3. OWL Symbol of Formalization (http://www.w3.org/TR/owl-features/)

<table>
<thead>
<tr>
<th>OWL element</th>
<th>Symbol</th>
<th>Analogy on UML</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allValuesFrom</td>
<td>∀</td>
<td>*</td>
<td>Property Restrictions</td>
<td>Intends to represent all the values of a specific domain.</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>∃</td>
<td>?</td>
<td>Property Restrictions</td>
<td>Intends to represent some values of a specific domain.</td>
</tr>
<tr>
<td>hasValue</td>
<td>⊳</td>
<td>$</td>
<td>Filler Information</td>
<td>Intends to represent a specific value of a domain.</td>
</tr>
<tr>
<td>minCardinality</td>
<td>≥</td>
<td>&gt;</td>
<td>Restricted Cardinality</td>
<td>Represents a minimal cardinality.</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>≤</td>
<td>&lt;</td>
<td>Restricted Cardinality</td>
<td>Represents a maximal cardinality.</td>
</tr>
<tr>
<td>cardinality</td>
<td>=</td>
<td>=</td>
<td>Restricted Cardinality</td>
<td>Represents a class cardinality of a value.</td>
</tr>
<tr>
<td>intersectionOf</td>
<td>∩</td>
<td>&amp;</td>
<td>Boolean Combinations of Class Expressions</td>
<td>Represents the intersection between classes.</td>
</tr>
<tr>
<td>unionOf</td>
<td>∪</td>
<td></td>
<td></td>
<td>Boolean Combinations of Class Expressions</td>
</tr>
<tr>
<td>complementOf</td>
<td>∼</td>
<td>!</td>
<td>Boolean Combinations of Class Expressions</td>
<td>Complement of a class.</td>
</tr>
</tbody>
</table>
Annex B: GRACE Meta-model Ontology

This section presents the GRACE meta-model ontology in a graphical manner and also the model ontology for the manual instantiation.
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control
Annex C: GRACE Meta-model Ontology in OWL Formalization

This annex provides the summary of the OWL formalization of the GRACE meta-model ontology.

C.1 Classes

Class: Failure

Description of an occurred perturbation, including the occurrence date, applied recover procedure, and recovery time.

- owl:Thing
  - Failure

Super Classes

hasFailureType EQ 1 owl:Thing
owl:Thing

Abstract Syntax

Class(Failure partial restriction(hasFailureType cardinality(1)) owl:Thing)

Usage

Domain of
description, failureID, hasAppliedRecoveryProcedure, hasFailureType, occurrenceDate, recoveryTime

Class: FailureType

Unexpected event type, like machine failure or delay, which degrades the execution of a production plan.

- owl:Thing
  - FailureType

Super Classes

owl:Thing

Abstract Syntax

Class(FailureType partial owl:Thing)

Usage

Domain of
description, hasPossibleRecoveryProcedure, name, setOfSymptoms, type
Class: Function

Entity that describes interactions among product components (materials) and/or external environment, e.g. tub contains water and drum move clothes; it is a product function.

• owl:Thing
  • Function

Super Classes

hasTarget EQ 1 owl:Thing

owl:Thing

Abstract Syntax

Class(Function partial restriction(hasTarget cardinality(1)) owl:Thing)

Usage

Domain of

expectedDetailedResult, hasTarget, name, type

Class: Journal

Description of the production of a product instance belonging to a production order executed in the production line, including the list of operations performed and the resources that have executed each operation.

• owl:Thing
  • Journal

Super Classes

hasJournalDetails MIN 1 owl:Thing

owl:Thing

Abstract Syntax

Class(Journal partial restriction(hasJournalDetails minCardinality(1)) owl#Thing)

Usage

Domain of

endDate, hasJournalDetails, journalID, name, startDate, state
Class: JournalDetails

Entity that describes the execution of an operation, including the processing time, participants (e.g. type and number of resources), dates and achieved results.

• owl:Thing
  • JournalDetails

Super Classes

describesOperation EQ 1 owl:Thing
owl:Thing
executedBy EQ 1 owl:Thing
materialUsed EQ 1 owl:Thing

Abstract Syntax

Class(JournalDetails partial restriction(describesOperation cardinality(1))
  restriction(executedBy cardinality(1))
  restriction(materialUsed cardinality(1))
  owl#Thing)

Usage

Domain of
describesOperation, detailedResult, endDate, executedBy, journalDetailsID,
materialUsed, name, overallResult, startDate

Class: Material

Entity used during the production process, e.g. tubs, blocks of steel, bearings, nuts and bolts, according to the BOM.

• owl:Thing
  • Material

Super Classes

hasMaterialFamily EQ 1 owl:Thing
owl:Thing

Abstract Syntax

Class(Material partial restriction(hasMaterialFamily cardinality(1)) owl#Thing)

Usage

Domain of
comprisesMaterial, description, hasMaterialFamily, hasProperty, materialID, name, quantity, type

**Class: MaterialFamily**

Family of the material used during the production process, e.g. bearing or tub (note that each material family could have different materials, e.g. for the family bearing, it is possible to have the bearing A and the bearing B).

- owl:Thing
  - MaterialFamily

**Super Classes**

hasCarrier MIN 0 owl:Thing

owl:Thing

**Abstract Syntax**

Class(MaterialFamily partial restriction(hasCarrier minCardinality(0)) owl#Thing)

**Usage**

Domain of
description, hasCarrier, materialFamilyID, name

**Class: Operation**

A job executed by one resource like drilling, welding, assembly, inspection and maintenance, that may add value to the product or may measure the value of the product, e.g. the quality control.

- owl:Thing
  - Operation

**Super Classes**

requiresSetup EQ 1 owl:Thing

usesMaterialFamily EQ 1 owl:Thing

createsFunction EQ 1 owl:Thing

owl:Thing

**Disjoint Classes**

Resource

**Abstract Syntax**

Class(Operation partial restriction(requiresSetup cardinality(1))
Class: Operator

A specialized human resource entity that is responsible for the execution of manual operations, such as an operator connecting the electrical cables.

• owl:Thing
  • Resource
    • Operator

Super Classes
  Resource

Disjoint Classes
  Transporter, Producer, QualityController, Tool

Abstract Syntax
  Class(Operator partial Resource)
  DisjointClasses(Operator Transporter Producer QualityController Tool)

Class: ProcessPlan

Represents the manufacturing process to produce a product, i.e. the description of a sequence of operations (for producing a product) with temporal constraints like precedence of execution.

• owl:Thing
  • ProcessPlan

Super Classes
  owl:Thing
  hasOperation MIN 1 owl:Thing
Abstract Syntax
Class(ProcessPlan partial owl#Thing

  restriction(hasOperation minCardinality(1)))

Usage
Domain of
  hasOperation, name, processPlanID

Class: Producer
A specialized resource entity that is responsible for the execution of producing operations, such as a welding robot or a CNC (Computer Numerical Control) machine.

•owl:Thing
  •Resource
    •Producer

Super Classes
Resource

Disjoint Classes
  Transporter, Operator, QualityController, Tool

Abstract Syntax
Class(Producer partial Resource)

DisjointClasses(Producer, Transporter, Operator, QualityController, Tool)

Usage
Domain of
  hasTool

Class: Product
Economic entity (finished or semi-finished), which is produced by the enterprise in a value-adding process (it includes a Bill of Materials (BOM), i.e. the list of materials that are considered as components of a final or intermediate product; it also includes the quantity of each material required).

•owl:Thing
  •Product

Super Classes
  hasProcessPlan EQ 1 owl:Thing
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

owl:Thing

Abstract Syntax
Class(Product partial restriction(hasProcessPlan cardinality(1))
    owl#Thing)

Usage
Domain of
description, hasMaterial, hasProcessPlan, name, productId

Class: ProductionOrder
Entity obtained by aggregating customer and forecast orders for the production of products, and provided by the ERP (Enterprise Resource Planning)/MES (Manufacturing Execution System) system.
•owl:Thing
  •ProductionOrder

Super Classes
isAssociatedWithProduct EQ 1 owl:Thing
hasJournal MIN 1 owl:Thing
owl:Thing

Abstract Syntax
Class(ProductionOrder partial restriction(isAssociatedWithProduct cardinality(1))
    restriction(hasJournal minCardinality(1))
    owl#Thing)

Usage
Domain of
dueDate, earliestDate, hasJournal, isAssociatedWithProduct, productionOrderID, quantity, startDate

Class: Property
An attribute that characterizes a resource (i.e. a skill) or that a resource should satisfy to execute an operation (i.e. a requirement). It includes a mathematical operator associated to the property value, e.g. a speed equal to 2000 r.p.m.
•owl:Thing
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

•Property

Super Classes
owl:Thing

Abstract Syntax
Class(Property partial owl#Thing)

Usage
Domain of
mathOperator, name, unit, value

Class: QualityController
A specialized resource entity that is responsible for the execution of measurement and diagnosis operations, such as a vision control station or a vibration control station.

•owl:Thing
  •Resource
    •QualityController

Super Classes
Resource

Disjoint Classes
Transporter, Operator, Producer, Tool

Abstract Syntax
Class(QualityController partial Resource)
DisjointClasses(QualityController Transporter Operator Producer Tool)

Class: RecoveryProcedure
Entity that describes the procedure to recover from the occurrence of a failure.

•owl:Thing
  •RecoveryProcedure

Super Classes
owl:Thing

Abstract Syntax
Class(RecoveryProcedure partial owl#Thing)

Usage
Domain of 
  description, name, procedureID

**Class: Resource**

Entity that can execute a certain range of operations as long as its capacity is not exceeded. Producer, quality controller, transporter, operator and tool are specializations of resource and inherit its characteristics.

- `owl:Thing`
  - `Resource`

**Super Classes**

`owl:Thing`

**Disjoint Classes**

`Operation`

**Abstract Syntax**

- `Class(Resource partial owl#Thing)`
- `DisjointClasses(Resource Operation)`

**Usage**

Operator, Producer, QualityController, Tool, Transporter

Domain of 
  hasExecuted, hasFailure, hasProperty, hasSetup, location, name, state, thisID, type

**Class: Setup**

Set of actions that it is necessary to execute in order to prepare a manufacturing resource for the execution of a range of operations.

- `owl:Thing`
  - `Setup`

**Super Classes**

`owl:Thing`

**Abstract Syntax**

- `Class(Setup partial owl#Thing)`

**Usage**

Domain of 
  description, setupID
Class: Tool

A specialized resource entity representing the physical devices used by producer stations and by operators to execute their processing operations, e.g. screw driver for screwing the counterweights; it may include the physical devices used by transporter resources to execute their handling operations, e.g. grippers.

-owl:Thing
  -Resource
    -Tool

Super Classes
Resource

Disjoint Classes
Transporter, Operator, Producer, QualityController

Abstract Syntax
Class(Tool partial Resource)
DisjointClasses(Tool Transporter Operator Producer QualityController)

Usage
Domain of
toolType

Class: Transporter

A specialized resource entity that is responsible for the execution of handling/transporting operations, such as an Auto-guided Vehicle (AGV) or a conveyor.

-owl:Thing
  -Resource
    -Transporter

Super Classes
Resource

Disjoint Classes
Operator, Producer, QualityController, Tool

Abstract Syntax
Class(Transporter partial Resource)
C.2 Object Properties

Property: comprisesMaterial

Establishes the recursive relation between the class “Material”, meaning that a material or component consists of other materials (in a certain quantity) according to the BOM structure.

Types
owl:ObjectProperty

Domain
Material

Range
Material

Abstract Syntax

ObjectProperty(comprisesMaterial annotation(rdf-schema#comment "Establishes the recursive relation between the class Material, meaning that a material or component consists of other materials (in a certain quantity) according to the BOM structure")

domain(Material)

range(Material))

Property: comprisesOperation

Establishes the recursive relation between the class Operation, meaning that an operation can be decomposed into several sub-operations.

Types
owl:ObjectProperty

Domain
Operation

Range
Operation

Abstract Syntax

ObjectProperty(comprisesOperation annotation(rdf-schema#comment "Establishes the recursive relation between the class Operation, meaning that an operation can be decomposed into several sub-operations")

domain(Operation))
Property: createsFunction

Establishes the relation between the class “Operation” and the class “Function”, allowing creating a function from the execution of an operation using a material.

Types
owl:ObjectProperty

Domain
Operation

Range
Function

Abstract Syntax
ObjectProperty(createsFunction annotation(rdf-schema#comment "Establishes the relation between the class Operation and the class Function, allowing creating a function from the execution of an operation using a material.")

domain(Operation)
range(Function))

Property: describesOperation

Establishes the relation between the class “JournalDetails” and the class “Operation”, meaning that the details about the execution of the operation are reported in the journal details.

Types
owl:ObjectProperty

Domain
Journal_Details

Range
Operation

Abstract Syntax
ObjectProperty(describesOperation annotation(rdf-schema#comment "Establishes the relation between the class JournalDetails and the class Operation, meaning that the details about the execution of the operation are reported in the journal details.")

domain(Journal_Details)
range(Operation))
Property: executedBy

Establishes the relation between the class “JournalDetails” and the class “Resource”, describing the resource that was executed the operation described in the journal details.

Types
owl:ObjectProperty

Domain
JournalDetails

Range
Resource

Abstract Syntax
ObjectProperty(executedBy annotation(rdf-schema#comment "Establishes the relation between the class JournalDetails and the class Resource, describing the resource that was executed the operation described in the journal details.")

    domain(JournalDetails)
    range(Resource))

Property: hasAppliedRecoveryProcedure

Establishes the relation between the class “Failure” and the class “RecoveryProcedure”, describing the recovery procedure applied to solve the occurred failure.

Types
owl:ObjectProperty

Domain
Failure

Range
RecoveryProcedure

Abstract Syntax
ObjectProperty(hasAppliedRecoveryProcedure annotation(rdf-schema#comment "Establishes the relation between the class Failure and the class RecoveryProcedure, describing the recovery procedure applied to solve the occurred failure.")

    domain(Failure)
    range(RecoveryProcedure))
Property: hasCarrier

Establishes the relation between the class “Function” and the class “MaterialFamily”, meaning that the specified material family component is delivering a function, i.e. a material family is designed to provide one or more specific functions to other material family or to external environment.

**Types**
- owl:ObjectProperty

**Domain**
- MaterialFamily

**Range**
- Function

**Abstract Syntax**

```
ObjectProperty(hasCarrier annotation(rdf-schema#comment "Establishes the relation between the class Function and the class MaterialFamily, meaning that the specified material family component is delivering a function, i.e. a material family is designed to provide one or more specific functions to other material family or to external environment.")

domain(MaterialFamily)
range(Function))
```

Property: hasExecuted

Establishes the relation between the class “Resource” and the class “JournalDetails”, describing the list of operations executed by a specific resource during its production history.

**Types**
- owl:ObjectProperty

**Domain**
- Resource

**Range**
- Journal_Details

**Abstract Syntax**

```
ObjectProperty(hasExecuted annotation(rdf-schema#comment "Establishes the relation between the class Resource and the class JournalDetails, describing the list of operations executed by a specific resource during its production history.")

domain( Resource)
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

Property: hasFailure
Establishes the relation between the class “Resource” and the class “Failure”, meaning that failures can occur during the execution of operations by a resource.

Types
owl:ObjectProperty

Domain
Resource

Range
Failure

Abstract Syntax
ObjectProperty(hasFailure annotation(rdf-schema#comment "Establishes the relation between the class Resource and the class Failure, meaning that failures can occur during the execution of operations by a resource.")
  domain(Resource)
  range(Failure))

Property: hasFailureType
Establishes the relation between the class “Failure” and the class “FailureType”, meaning that a failure occurrence is from a specific type of failure.

Types
owl:ObjectProperty

Domain
Failure

Range
FailureType

Abstract Syntax
ObjectProperty(hasFailureType annotation(rdf-schema#comment "Establishes the relation between the class Failure and the class FailureType, meaning that a failure occurrence is from a specific type of failure.")
  domain(Failure)
  range(FailureType))
Property: hasJournal
Establishes the relation between the class “ProductionOrder” and the class “Journal”, meaning that a journal is the description of the execution of a product item defined in the production order.

Types
owl:InverseFunctionalProperty
owl:ObjectProperty

Domain
ProductionOrder

Range
Journal

Characteristics
Inverse Functional

Abstract Syntax
ObjectProperty(hasJournal annotation(rdf-schema#comment "Establishes the relation between the class ProductionOrder and the class Journal, meaning that a journal is the description of the execution of a product item defined in the production order.")
InverseFunctional
domain(ProductionOrder)
range(Journal))

Property: hasJournalDetails
Establishes the relation between the class “Journal” and the class “JournalDetails”, meaning that the journal details describes the execution of the operation belonging to the process plan during the execution of a product item defined in the production order.

Types
owl:ObjectProperty

Domain
Journal

Range
JournalDetails

Abstract Syntax
ObjectProperty(hasJournalDetails annotation(rdf-schema#comment "Establishes the relation between the class Journal and the class JournalDetails, meaning that the journal details describes the execution of the operation belonging to the process plan during the execution of a product item defined in the production order.")
  
domain(Journal)
  range(JournalDetails))

Property: hasMaterial

Establishes the relation between the class “Product” and the class “Material”, meaning that a product consists on a set of materials according to the BOM.

Types
  owl:ObjectProperty

Domain
  Product

Range
  Material

Abstract Syntax

ObjectProperty(hasMaterial annotation(rdf-schema#comment "Establishes the relation between the class Product and the class Material, meaning that a product consists on a set of materials according to the BOM.")
  
domain(Product)
  range(Material))

Property: hasMaterialFamily

Establishes the relation between the class “Material” and the class “MaterialFamily”, meaning that a material is from a material family (i.e. a type of material).

Types
  owl:ObjectProperty

Domain
  Material

Range
  MaterialFamily

Abstract Syntax
Deliverable D1.3
Document defining the ontology for line-production system, integrating process
and quality control

ObjectProperty(hasMaterialFamily annotation(rdf-schema#comment "Establishes the
relation between the class Material and the class MaterialFamily, meaning that a
material is from a material family (i.e. a type of material).")

domain(Material)
range(MaterialFamily))

Property: hasOperation

Establishes the relation between the class “ProcessPlan” and the class “Operation”,
defining the list of operations required to execute a product model.

Types
owl:ObjectProperty

Domain
ProcessPlan

Range
Operation

Abstract Syntax

ObjectProperty(hasOperation annotation(rdf-schema#comment " Establishes the
relation between the class ProcessPlan and the class Operation, defining the list of
operations required to execute a product model.")

domain(ProcessPlan)
range(Operation))

Property: hasOperationPrecedence

Establishes the recursive relation between the class “Operation”, defining a
precedence to execute an operation, i.e. meaning that the execution of an operation
should only be performed after the execution of other operations.

Types
owl:ObjectProperty

Domain
Operation

Range
Operation

Abstract Syntax
ObjectProperty(hasOperationPrecedence annotation(rdf-schema#comment "Establishes the recursive relation between the class Operation, defining a precedence to execute an operation, i.e. meaning that the execution of an operation should only be performed after the execution of other operations.")

domain(Operation)
range(Operation))

**Property: hasPossibleResources**

Establishes the relation between the class “Operation” and the class “Resource”, meaning that there is a list of potential resources that are able to execute the operation.

**Types**
owl:ObjectProperty

**Domain**
Operation

**Range**
Resource

**Abstract Syntax**

ObjectProperty(hasPossibleResources annotation(rdf-schema#comment "Establishes the relation between the class Operation and the class Resource, meaning that there is a list of potential resources that are able to execute the operation.")

domain(Operation)
range(Resource))

**Property: hasPossibleRecoveryProcedure**

Establishes the relation between the class “FailureType” and the class “RecoveryProcedure”, describing the list of possible recovery procedures that can be applied to solve a failure event type.

**Types**
owl:ObjectProperty

**Domain**
FailureType

**Range**
RecoveryProcedure
Abstract Syntax
ObjectProperty(hasPossibleRecoveryProcedure annotation(rdf-schema#comment "Establishes the relation between the class FailureType and the class RecoveryProcedure, describing the list of possible recovery procedures that can be applied to solve a failure event type.")
  domain(FailureType)
  range(RecoveryProcedure))

Property: hasProcessPlan
Establishes the relation between the class “Product” and the class “ProcessPlan”, meaning that the production of a product model requires the execution of a process plan.
Types
  owl:ObjectProperty
Domain
  Product
Range
  ProcessPlan
Abstract Syntax
ObjectProperty(hasProcessPlan annotation(rdf-schema#comment "Establishes the relation between the class Product and the class ProcessPlan, meaning that the production of a product model requires the execution of a process plan.")
  domain(Product)
  range(ProcessPlan))

Property: hasProperty
Establishes the relation between the class “Resource” and the class “Property” or between the class “Material” and the class “Property”, meaning that a resource has a set of skills that allow it to execute operations or that a material has a set of attributes.
Types
  owl:ObjectProperty
Domain
  Resource
  Material
Range
Property

Abstract Syntax

ObjectProperty(hasProperty annotation(rdf-schema#comment "Establishes the relation between the class Resource and the class Property or between the class Material and the class Property, meaning that a resource has a set of skills that allow it to execute operations or that a material has a set of attributes.")
  domain(unionOf(Resource Material))
  range(Property))

Property: hasSetup

Establishes the relation between the class “Resource” and the class “Setup”, meaning that a resource may have different setups that will allow the execution of different operations.

Types
  owl:ObjectProperty

Domain
  Resource

Range
  Setup

Abstract Syntax

ObjectProperty(hasSetup annotation(rdf-schema#comment "Establishes the relation between the class Resource and the class Setup, meaning that a resource may have different setups that will allow the execution of different operations.")
  domain(Resource)
  range(Setup))

Property: hasTarget

Establishes the relation between the class “Function” and the class “MaterialFamily”, meaning that one material family component is receiving a function.

Types
  owl:ObjectProperty

Domain
  Function
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

Property: hasTarget

Establishes the relation between the class Function and the class MaterialFamily, meaning that one material family component is receiving a function.

Types
owl:ObjectProperty

Domain
Function

Range
MaterialFamily

Abstract Syntax

ObjectProperty(hasTarget annotation(rdf-schema#comment "Establishes the relation between the class Function and the class MaterialFamily, meaning that one material family component is receiving a function.")

domain(Function)
range(MaterialFamily))

Property: hasTool

Establishes the relation between the class “Producer” and the class “Tool”, meaning that a producer has a set of tools to execute processing operations.

Types
owl:ObjectProperty

Domain
Producer

Range
Tool

Abstract Syntax

ObjectProperty(hasTool annotation(rdf-schema#comment "Establishes the relation between the class Producer and the class Tool, meaning that a producer has a set of tools to execute processing operations.")

domain(Producer)
range(Tool))

Property: isAssociatedWithProduct

Establishes the relation between the class “ProductionOrder” and the class “Product”, meaning that a production order is related to the production of a certain quantity of an available product model.

Types
owl:ObjectProperty

Domain
ProductionOrder

Range
Product

Abstract Syntax

ObjectProperty(isAssociatedWithProduct annotation(rdf-schema#comment "Establishes the relation between the class ProductionOrder and the class Product, meaning that a production order is related to the production of a certain quantity of an available product model.")

domain(ProductionOrder)

range(Product))

Property: materialUsed

Establishes the relation between the class “JournalDetails” and the class “Material”, meaning that a journal details describes the material used to execute an operation.

Types

owl:ObjectProperty

Domain

JournalDetails

Range

Material

Abstract Syntax

ObjectProperty(materialUsed annotation(rdf-schema#comment "Establishes the relation between the class JournalDetails and the class Material, meaning that a journal details describes the material used to execute an operation.")

domain(JournalDetails)

range(Material))

Property: requiresProperty

Establishes the relation between the class “Operation” class and the class “Property”, meaning that the execution of the operation requires the fulfilment of a set of requirements by potential resources.

Types

owl:ObjectProperty

Domain

Operation

Range
Property

Abstract Syntax

ObjectProperty(requiresProperty annotation(rdf-schema#comment "Establishes the relation between the class Operation class and the class Property, meaning that the execution of the operation requires the fulfilment of a set of requirements by potential resources.")

domain(Operation)
range(Property))

Property: usesMaterialFamily

Establishes the relation between the class “Operation” and the class “MaterialFamily”, meaning that the execution of the operation uses a material from a proper family.

Types
owl:ObjectProperty

Domain
Operation

Range
MaterialFamily

Abstract Syntax

ObjectProperty(usesMaterialFamily annotation(rdf-schema#comment "Establishes the relation between the class Operation and the class MaterialFamily, meaning that the execution of the operation uses a material from a proper family.")

domain(Operation)
range(MaterialFamily))

Property: requiresSetup

Establishes the relation between the class “Operation” and the class “Setup”, meaning that the execution of the operation requires the existence of a proper setup in the resource.

Types
owl:ObjectProperty

Domain
Operation

Range
Setup

Abstract Syntax

ObjectProperty(requiresSetup annotation(rdf-schema#comment "Establishes the relation between the class Operation and the class Setup, meaning that the execution of the operation requires the existence of a proper setup in the resource.")

domain(Operation)
range(Setup))

C.1 Data Type Properties

Property: description

A statement describing something, e.g. a product or a setup.

Types

owl:DatatypeProperty

Domain

Setup
Product
Failure
Material
MaterialFamily
RecoveryProcedure
Operation
FailureType

Range

XMLSchema#string

Abstract Syntax

ObjectProperty(description annotation(rdf-schema#comment "A statement describing something, e.g. a product or a setup.")

domain(unionOf(Setup Product Failure Material MaterialFamily RecoveryProcedure Operation FailureType))
range(XMLSchema#string))
**Property: detailedResult**

The detailed description of the results obtained in a measurement/testing operation.

Types

owl:DatatypeProperty

Domain

JournalDetails

Range

XMLSchema#string

Abstract Syntax

ObjectProperty(detailedResult annotation(rdf-schema#comment "The detailed description of the results obtained in a measurement/testing operation.")

domain(JournalDetails)

range(XMLSchema#string))

**Property: dueDate**

The date on which an obligation, e.g. the production of a product, must be accomplished.

Types

owl:DatatypeProperty

Domain

ProductionOrder

Range

XMLSchema#date

Abstract Syntax

ObjectProperty(dueDate annotation(rdf-schema#comment "The date on which an obligation, e.g. the production of a product, must be accomplished.")

domain(ProductionOrder)

range(XMLSchema#date))

**Property: earliestDate**

The date before which an activity or event cannot start.

Types

owl:DatatypeProperty
Domain
  ProductionOrder

Range
  XMLSchema#date

Abstract Syntax
  ObjectProperty(earliestDate annotation(rdf-schema#comment "The date before which an activity or event cannot start."))
    domain(ProductionOrder)
    range(XMLSchema#date))

Property: endDate
  The date describing the end of the execution of an activity.

Types
  owl:DatatypeProperty

Domain
  Journal
  JournalDetails

Range
  XMLSchema#dateTime

Abstract Syntax
  ObjectProperty(endDate annotation(rdf-schema#comment "The date describing the end of the execution of an activity.")
    domain(unionOf(Journal JournalDetails))
    range(XMLSchema#dateTime))

Property: expectedDetailedResult
  The description of the expected (ideal) result for a function created by the execution of an operation.

Types
  owl:DatatypeProperty

Domain
  Function

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(expectedDetailedResult annotation(rdf-schema#comment "The description of the expected (ideal) result for a function created by the execution of an operation.")
  domain(Function)
  range(XMLSchema#string))

Property: failureID
A non-negative integer number that provides the unique identification of the failure.

Types
  owl:DatatypeProperty

Domain
  Failure

Range
  XMLSchema#int

Abstract Syntax
ObjectProperty(failureID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the failure.")
  domain(Failure)
  range(XMLSchema#int))

Property: journalDetailsID
A non-negative integer number that provides the unique identification of the journal details.

Types
  owl:DatatypeProperty

Domain
  JournalDetails

Range
  XMLSchema#string

Abstract Syntax
ObjectProperty(journalDetailsID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the journal details.")

domain(JournalDetails)
range(XMLSchema#string))

**Property: journalID**

A non-negative integer number that provides the unique identification of the journal (e.g. could be the serial number identified the produced product item).

**Types**

owl:DatatypeProperty

**Domain**

Journal

**Range**

XMLSchema#string

**Abstract Syntax**

ObjectProperty(journalID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the journal (e.g. could be the serial number identified the produced product item).")

domain(Journal)
range(XMLSchema#string))

**Property: location**

A place where something, e.g. a resource, is located.

**Types**

owl:DatatypeProperty

**Domain**

Resource

**Range**

XMLSchema#string

**Abstract Syntax**

ObjectProperty(location annotation(rdf-schema#comment "A place where something, e.g. a resource, is located.")

domain(Resource)
Property: materialFamilyID
A label that provides the identification of the material family.

Types
owl:DatatypeProperty

Domain
MaterialFamily

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(materialFamilyID annotation(rdf-schema#comment "A label that provides the identification of the material family.")

domain(MaterialFamily)
range(XMLSchema#string))

Property: mathOperator
The mathematical operator that can establish a comparison in the value of a property.

Types
owl:DatatypeProperty

Domain
Property

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(mathOperator annotation(rdf-schema#comment "The mathematical operator that can establish a comparison in the value of a property.")

domain(Property)
range(XMLSchema#string))

Property: name
The designation of a thing, e.g. a resource, a property or an operation.
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

**Types**

owl:DatatypeProperty

**Domain**

Resource
Property
Operation
Product
MaterialFamily
JournalDetails
Journal
FailureType
Function
ProcessPlan
Material
RecoveryProcedure

**Range**

XMLSchema#string

**Abstract Syntax**

ObjectProperty(name annotation(rdf-schema#comment "The designation of a thing, e.g. a resource, a property or an operation."))

domain(unionOf(Resource Property Operation Product MaterialFamily JournalDetails Journal FailureType Function ProcessPlan Material RecoveryProcedure))

range(XMLSchema#string))

**Property: occurrenceDate**

The date when a failure occurred.

**Types**

owl:DatatypeProperty

**Domain**

Failure

**Range**

XMLSchema#dateTime
Abstract Syntax

ObjectProperty(occurrenceDate annotation(rdf-schema#comment "The date when a failure occurred.")
  domain(Failure)
  range(XMLSchema#dateTime))

Property: operationID

A non-negative integer number that provides the unique identification of the operation.

Types
  owl:DatatypeProperty

Domain
  Operation

Range
  XMLSchema#int

Abstract Syntax

ObjectProperty(operationID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the operation.")
  domain(Operation)
  range(XMLSchema#int))

Property: overallResult

The overall result obtained in a measurement/testing operation, for example OK or KO.

Types
  owl:DatatypeProperty

Domain
  JournalDetails

Range
  {"OK"^^XMLSchema#string "Not OK"^^XMLSchema#string}

Abstract Syntax

ObjectProperty(overallResult annotation(rdf-schema#comment "The overall result obtained in a measurement/testing operation, for example OK or KO.")
Property: materialID
A non-negative integer number that provides the unique identification of the material.

Types
owl:DatatypeProperty

Domain
Material

Range
XMLSchema#int

Abstract Syntax
ObjectProperty(materialID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the material.")
  domain(Material)
  range(XMLSchema#int))

Property: procedureID
A non-negative integer number that provides the unique identification of the recovery procedure.

Types
owl:DatatypeProperty

Domain
RecoveryProcedure

Range
XMLSchema#int

Abstract Syntax
ObjectProperty(procedureID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the recovery procedure.")
  domain(RecoveryProcedure)
  range(XMLSchema#int))
Property: processPlanID

A non-negative integer number that provides the unique identification of the process plan.

Types
owl:DatatypeProperty

Domain
ProcessPlan

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(processPlanID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the process plan.")
-domain(ProcessPlan)
-range(XMLSchema#string))

Property: productID

A non-negative integer number that provides the unique identification of the product.

Types
owl:DatatypeProperty

Domain
Product

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(productID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the product.")
-domain(Product)
-range(XMLSchema#string))

Property: productionOrderID

A non-negative integer number that provides the unique identification of the production order.

Types
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

owl:DatatypeProperty

Domain
ProductionOrder

Range
XMLSchema#int

Abstract Syntax
ObjectProperty(productionOrderID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the production order."))

  domain(ProductionOrder)
  range(XMLSchema#int))

Property: quantity

A positive rational number that defines the specific amount of things to be produced.

Types

owl:DatatypeProperty

Domain
ProductionOrder
Material

Range
XMLSchema#int

Abstract Syntax
ObjectProperty(quantity annotation(rdf-schema#comment "A positive rational number that defines the specific amount of things to be produced."))

  domain(unionOf(ProductionOrder Material))
  range(XMLSchema#int))

Property: recoveryTime

A positive rational number, it gives the indication of the recovery time after a failure (expressed in seconds).

Types

owl:DatatypeProperty

Domain
Failure

Range
XMLSchema#int

Abstract Syntax
ObjectProperty(recoveryTime annotation(rdf-schema#comment "A positive rational number, it gives the indication of the recovery time after a failure (expressed in seconds).")

domain(Failure)
range(XMLSchema#int))

Property: setOfSymptoms
List of symptoms that may lead to the occurrence of a failure (important to forecast failures and to support the diagnosis).

Types
owl:DatatypeProperty

Domain
FailureType

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(setOfSymptoms annotation(rdf-schema#comment "List of symptoms that may lead to the occurrence of a failure (important to forecast failures and to support the diagnosis).")

domain(FailureType)
range(XMLSchema#string))

Property: setupID
A non-negative integer number that provides the unique identification of the setup.

Types
owl:DatatypeProperty

Domain
Setup

Range
Deliverable D1.3
Document defining the ontology for line-production system, integrating process and quality control

**XMLSchema#int**

**Abstract Syntax**

ObjectProperty(setupID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the setup."))
  
  domain(Setup)
  
  range(XMLSchema#int))

**Property: startDate**

The date describing the start of the execution of an activity.

**Types**

owl:DatatypeProperty

**Domain**

ProductionOrder

Journal

JournalDetails

**Range**

XMLSchema#dateTime

**Abstract Syntax**

ObjectProperty(startDate annotation(rdf-schema#comment "The date describing the start of the execution of an activity."))

  domain(unionOf(ProductionOrder Journal JournalDetails))

  range(XMLSchema#dateTime))

**Property: state**

The current state of the resource, e.g. waiting, running and broken.

**Types**

owl:DatatypeProperty

**Domain**

Journal

Resource

**Range**

XMLSchema#string
Abstract Syntax

ObjectProperty(state annotation(rdf-schema#comment "The current state of the resource, e.g. waiting, running and broken.")
   domain(unionOf(Journal Resource))
   range(XMLSchema#string))

Property: thisID

A non-negative integer number that provides the unique identification of the resource.

Types
   owl:DatatypeProperty

Domain
   Resource

Range
   XMLSchema#int

Abstract Syntax

ObjectProperty(thisID annotation(rdf-schema#comment "A non-negative integer number that provides the unique identification of the resource.")
   domain(Resource)
   range(XMLSchema#int))

Property: toolType

The type of tool used to perform a processing or handling operation.

Types
   owl:DatatypeProperty

Domain
   Tool

Range
   XMLSchema#string

Abstract Syntax

ObjectProperty(toolType annotation(rdf-schema#comment "The type of tool used to perform a processing or handling operation.")
   domain(Tool)
   range(XMLSchema#string))
Property: type

The designation of a type, e.g. a description. For example, a quality controller could be a vision station or a vibration control station.

Types
owl:DatatypeProperty

Domain
Resource
FailureType
Operation
Material
Function

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(type annotation(rdf-schema#comment "The designation of a type, e.g. a description. For example, a quality controller could be a vision station or a vibration control station.")

domain(unionOf(Resource FailureType Operation Material Function))
range(XMLSchema#string))

Property: unit

The description of the units used to represent the value in the property.

Types
owl:DatatypeProperty

Domain
Property

Range
XMLSchema#string

Abstract Syntax
ObjectProperty(unit annotation(rdf-schema#comment "The description of the units used to represent the value in the property.")

domain(Property)
Property: value

A specific amount related to a property type.

Types

owl:DatatypeProperty

Domain

Property

Range

XMLSchema#string

Abstract Syntax

ObjectProperty(value annotation(rdf-schema#comment "A specific amount related to a property type."

domain(#Property))

range(XMLSchema#string))