# D3.1.1. Requirement-product-process ontology

**Author, company:**
- Akshay Raju Kulkarni, TU Delft

**Version:**
1.5

**Date:**
May 2, 2022

**Status:**
Final / Released

**Confidentiality:**
Public

## Change log

| Revision | Date | Prepared by | Checked by | Description |
|---|---|---|---|---|
| 1.0 | 03/01/2022 | Akshay Raju Kulkarni | | First setup of this deliverable |
| 1.1 | 12/01/2022 | Akshay Raju Kulkarni | | Adding new information to address the feedback from the review of the first iteration |
| 1.2 | 20/01/2022 | Akshay Raju Kulkarni | | Finishing the first draft of the structure + content of this deliverable |
| 1.3 | 25/04/2022 | Akshay Raju Kulkarni | | Finished the final deliverable and submitted for review |
| 1.4 | 02/05/2022 | Max Baan / Ingo Staack | | Received review for the document |
| 1.5 | 02/05/2022 | Akshay Raju Kulkarni | Max Baan | Processed all the comments from the review (Ready for submission) |

# Table of contents

## Acronyms

| Acronym | Definition |
|---|---|
| CAD | Computer Aided Design |
| INCOSE | International Council on Systems Engineering |
| MBSE | Model Based Systems Engineering |
| MML | MOKA Modeling Language |
| MOKA | Methodology and software tools Oriented to Knowledge based engineering Applications |
| SysML | Systems Modeling Language |
| KBE | Knowledge Based Engineering |
| OMG | Object Management Group |
| RFP | Request For Proposal |
| SMEs | Small and Medium-sized Enterprises |
| UML | Unified Modeling Language |
| XML | eXtensible Markup Language |

## Acknowledgements

This research is partly funded by the ITEA 3 Call 6 project DEFAINE of the European Union.

# 1. Introduction

This document is a deliverable of Work Package 3 of DEFAINE project. The goal of this work package is manifold, as it encompasses (1) the development of the engineering services to support the automation of the individual use cases from WP2; (2) the development of new methodologies to support the generation of engineering services based on KBE technology; (3) the enhancement of an existing KBE platform to enable live modification (at execution time, without interrupting computations) of KBE applications.

i. The first objective includes the development, enhancement (in case of existing ones) and wrapping of the engineering services necessary for the various use cases. These services include automation solutions based on KBE technology, as well as other automated design and analysis applications.
ii. The second objective aims at innovation in the KBE application development methodology, with the development of a novel approach, based on MBSE, to model domain knowledge and requirements and to support (semi)-automatic generation of KBE apps through visual editing (rather than standard coding).
iii. The third objective aims at innovation in the core functionalities of existing KBE platforms, to improve their flexibility and usability within large and automated design explorations.

This deliverable is related to the second objective, namely, the development of a novel KBE development methodology, based on MBSE concepts, to bridge the knowledge capturing and formalization phase with the actual code development phase of a KBE application, by automating part of the code generation and rule encoding, while guaranteeing the synchronization and consistency between knowledge base, requirements and KBE application. The methodology will entail:

i. the definition of a generic requirement-product-process ontology, based on an extension/modification of the MOKA methodology and the requirement modelling approach
ii. a KBE language ontology and a strategy (and technical solution) to map this ontology to the product-process one
iii. a KBE “visual editor” (based on the ontologies above) to (partially) automate the generation of KBE applications.

In this deliverable we focus on the first part of the methodology with an emphasis on identifying and developing an appropriate requirement-product-process ontology necessary to (semi-) automatically generate KBE application skeletons to:

i. Speed-up the overall product design process and reduce the lead time
ii. Ensure that the knowledge encapsulated in the application is correct
iii. Improve the trust of domain experts on KBE application (i.e., reduce black-box behaviour) by improving the traceability of requirement compliance by the KBE application.

## 1.1. Intended use and purpose of this deliverable

This deliverable is of type “software + document”. The purpose of this deliverable is to provide a starting point for the technology demonstrator to be delivered in Month 25 of the project. Chapter 2 explains the motivation for improving KBE application process. Chapter 3 presents the current state of KBE application development methodology. Chapter 4 discusses the methodology proposed in

this research work to improve the KBE application development process. Chapter 5 discusses how KBE and MBSE ontologies can be combined. Finally, Chapter 6 provides the conclusion and outlook for future work.

## 1.2. Definitions

This sub-chapter provides some definitions that will be useful to formalize requirements-products-process ontology.

**KBE**
Knowledge Based Engineering (KBE) is a technology based on dedicated software tools called KBE systems, that are able to capture and reuse product and process engineering knowledge. The main objective of KBE is the reduction of time and costs of product development by means of automation of repetitive, non-creative, design tasks and support of multidisciplinary design optimization in all the phases of the design process [1]. In this document, engineering applications developed to automate design tasks using KBE systems are called KBE applications. A KBE system is identified by its ability to guarantee dependency tracking, run-time caching[1] and lazy evaluation[2] [1].

**MBSE**
Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases [2]. Conventional Systems Engineering aims at the same sort of support, but is based on textual documents. MBSE is the formalized application of (digital) modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.

**SYSML**
The Systems Modelling Language (SysML) is a general-purpose modelling language for systems engineering applications extensively used in MBSE projects. SysML is an extension of a subset of the Unified Modelling Language (UML) to support systems engineering activities. It supports the (digital) modelling of specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems.

**Product Development Process**
The product development process is a six-stage plan that involves taking a product from initial concept to final market launch.

- Step 1: Idea Generation – Brainstorming a product concept
- Step 2: Product Definition – Scoping and refining the product concept
- Step 3: Prototyping – Constructing a visual representation
- Step 4: Initial Design – Producing an initial mockup
- Step 5: Validation and Testing – Validating and testing the development strategy
- Step 6: Commercialization – Developing and implementing the product

[1] The result of function evaluation is stored in the memory (cache)
[2] Function and/or attribute evaluation is not triggered until its evaluation is absolutely necessary

Specifically in KBE application development, these 6 steps can be seen as shown in Figure 1.

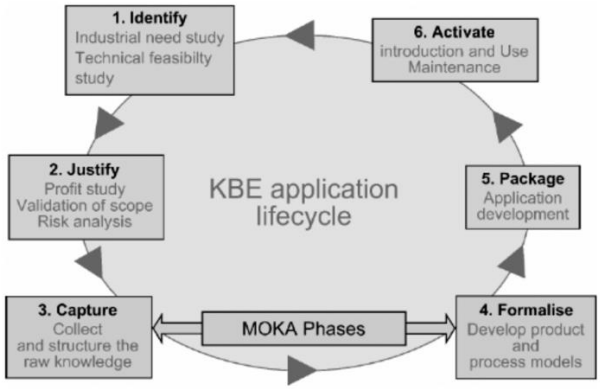


*Figure 1: KBE application life-cycle [3]*

**Domain Expert**

Domain experts are professionals in some domain different from computer science, who need to use computers in their daily work. The domain experts possess the knowledge that is used in the KBE application. They are not directly involved in the development of the KBE application, but their input is essential to build a working KBE application [4].

**Knowledge Engineering**

The field of Knowledge Engineering (KE) gives methods to elicit and formalize domain-specific knowledge [5]. Knowledge engineers are specialists that translate highly technical information, (obtained from domain experts) into models that are understood by the developers that build the actual computer program (such as KBE applications).

**Ontology**

An ontology is an explicit description of concepts in a specific domain, defining a formal domain vocabulary. An ontology defines relations between the concepts in the domain, and adds attributes to further specify a concept, making it a suitable means of capturing and re-use knowledge [6]. An example of an ontology is presented in Figure 2.

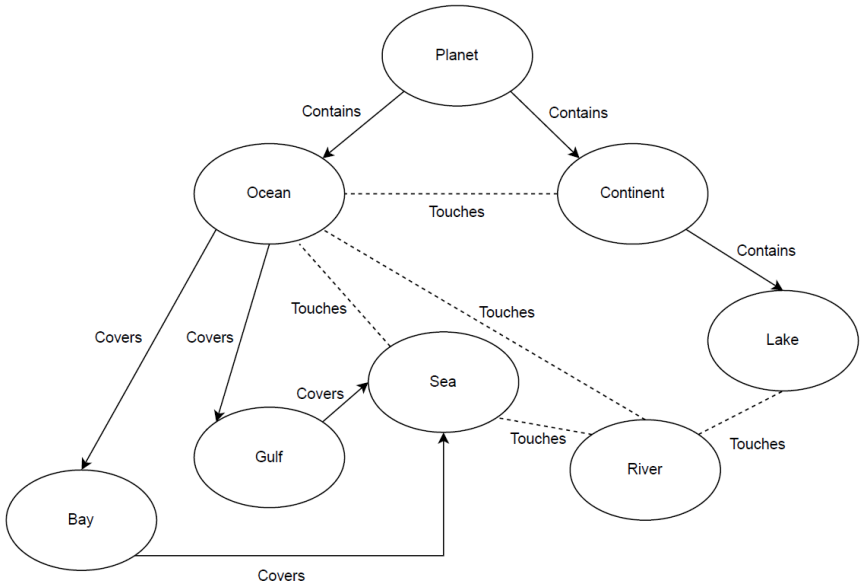


*Figure 2: Example of the beginning of a geographic ontology with spatial relations.*

**Requirement Ontology**

A requirement is a capability or condition that a system must ("shall") satisfy. A functional requirement specifies a function that a system must perform, whereas a non-functional requirement specifies quality criteria that can be used to test the effectiveness of system functions. An example of a requirement ontology is presented in Figure 3.

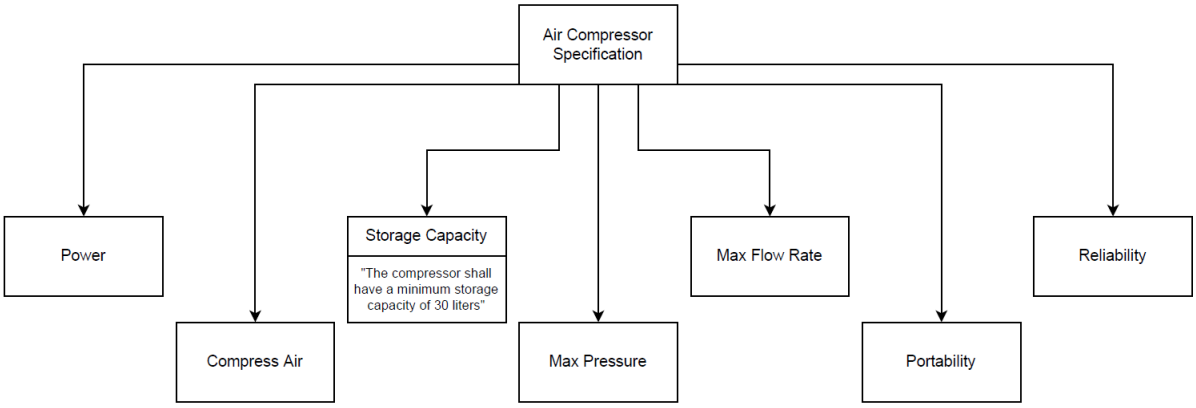


*Figure 3: Example of a requirement ontology for an air compressor specification [7]*

**Product Ontology**

A product ontology is a description of the components of a system and the relations between them. A product ontology specifies system static structures that can be used for control objects, data objects and interface objects. Products represent system components with their encapsulated contents (properties, constraints, interfaces and relationships). Products can be recursively decomposed ("nested") into their atomic parts. Two examples of a product ontology are presented in Figure 4.

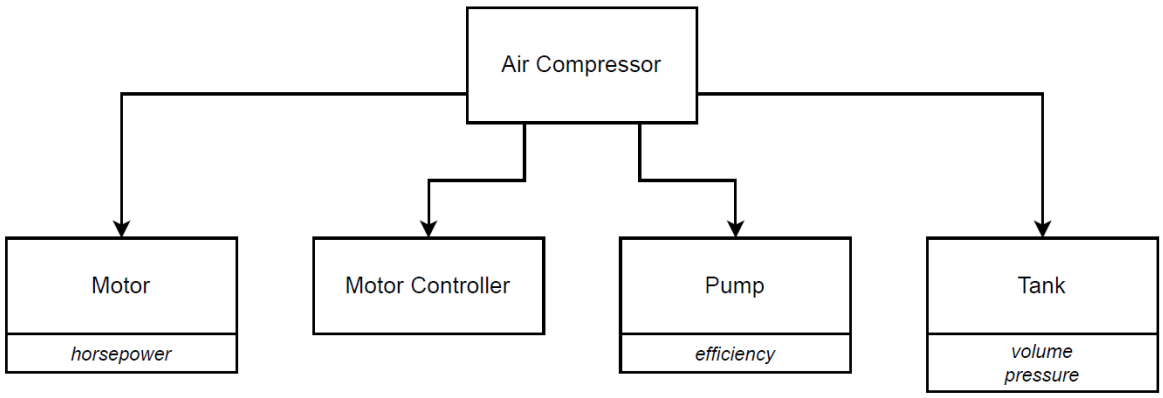


*Figure 4: Example of a product ontology for an air compressor [7]*

**Process Ontology**

A process ontology is a description of a set of steps that must be performed to complete a given activity/task. A process ontology mainly specifies a series of activities taken in order to achieve a particular end. Activities can be recursively decomposed ("nested") into actions (atomic activities which are a primitive executable behavior). An example of a process ontology is presented in Figure 5.

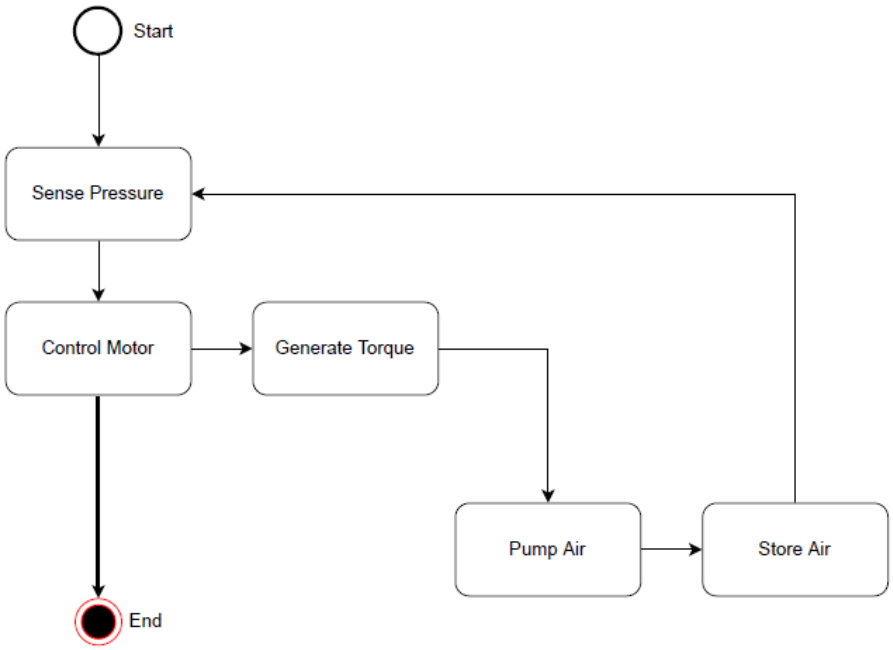


*Figure 5: Example of a process ontology for compressing air [7]*

## 2. Motivation to combine KBE and MBSE systems

Socio-economic challenges and global competition drive high-tech industries to combine their cutting-edge technology with improved time-to-market and cost efficiency. To reduce time-to-market, improvements in product development lead time are necessary, which, in turn requires a combination of a streamlined product development process and systematic design space exploration capabilities. A way forward to reduce lead time, improve cost efficiency and enhance competitiveness is to apply the so-called front-loading product development approach.

Front-loading relies on the company's ability to develop (semi-) automated design systems and knowledge bases (KBs) that store their consolidated knowledge. These design systems and KBs are established before the actual start of a project, so as to enable their quick deployment for the preparation of proposals and to trigger new development processes to further reduce design lead time [8].

Figure 6 qualitatively shows the reduction in development time that can be achieved by applying front-loading in the development of a project instead of concurrent or sequential engineering processes. Concretely, front-loading is a combination of two main tasks, namely:

i. **Project-to-project knowledge transfer:** seeks to transfer information of problems from one project to the next similar project. This saves time needed to solve similar problems. It also allows engineers to work on a problem at an early stage of the project/ before the start of the project [8] [9].
ii. **Rapid problem-solving**: advanced technology and (design automation) methods are used to reduce the time needed for problem-solving. For example, the use of CAE or KBE applications instead of physical prototyping [8].

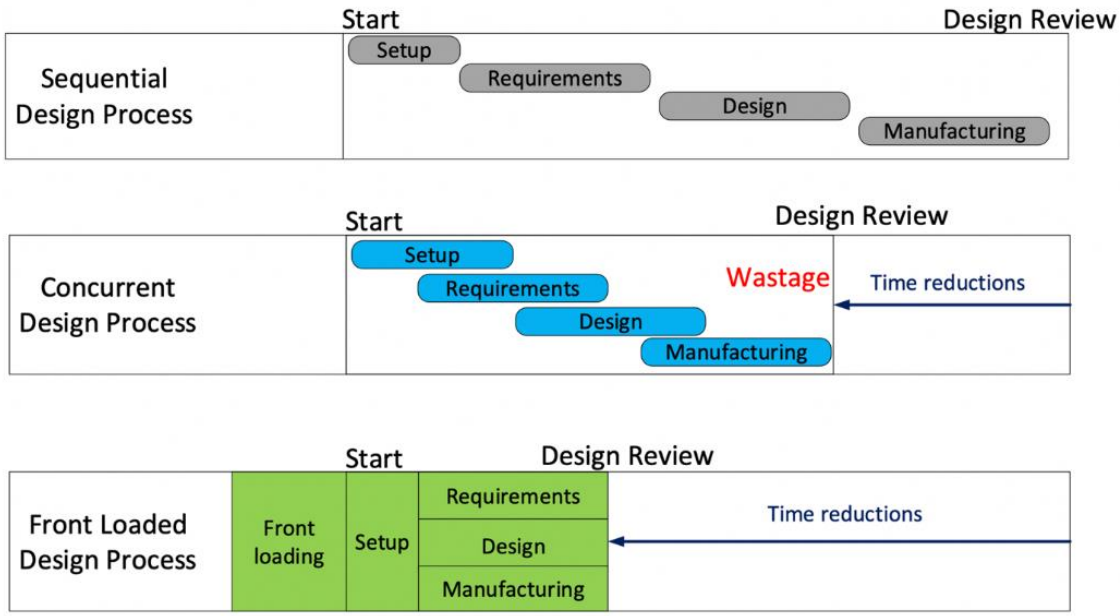


*Figure 6: Effect of front-loading on improving time to market (adapted from [8])*

In project-to-project knowledge transfer, best-practices to solve a design problem are identified and recorded in the form of documents, drawings and digital models or data sets (where possible). Thereafter, rapid problem solution phase starts, where, the best-practices are formalized into a code to achieve (partial) design automation (e.g., KBE applications). This allows engineers to (quickly) create design solution for their inputs (i.e., the ability to perform *"what-if"* studies). However, the full potential of front-loading can only be exploited by efficiently searching the design space for improved designs (i.e., employing Design of Experiments (DoE) or Multidisciplinary Analysis and Optimization (MDAO)) by developing simulation workflows.

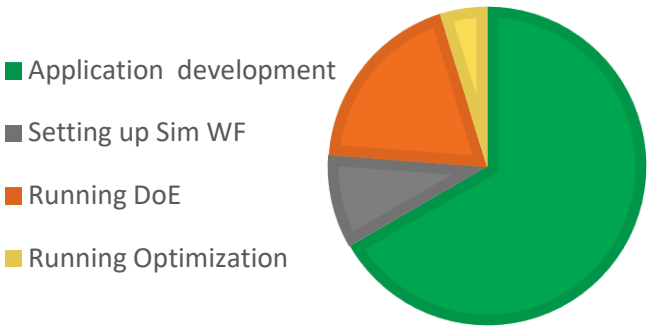


*Figure 7: Qualitative indication of time spent in different activities of current Front-loading process*[3]

From the experience gleaned in the past projects, majority of the time is invested in KBE application (or design automation) development and setting up the simulation workflows (see Figure 7). Consequently, little or no time is available to perform design space exploration (DOE and full-blown MDO), which promises the most benefits in front-loading in terms of improvements in product performance and reduction in costs. The ability to quickly generate KBE applications that are sufficiently flexible, transparent (i.e., requirements and rules can be clearly traced within the application code) and capture knowledge from previous projects is of essence here.

Thus, the development of a methodological approach to speed up KBE development forms the key motivation for this work. As discussed in the following section, we believe that MBSE can aid in speeding up the KBE application development process and in making the knowledge captured within KBE applications explicit and transparent (i.e., the KBE applications will no longer be black-boxes) thereby allowing engineers to utilize a significant portion of the conceptual design time in systematic design space exploration.

[3] While this is a general observation, in practice, the exact time distribution depends on the extent of repetitive tasks in the design process and the available knowledge of the design process from previous projects

# 3. State of the Art in KBE app development

There are several methodologies available in the literature that attempt to formalize and improve KBE application development process, such as Methodology and tools Oriented to Knowledge-based engineering Applications (MOKA) [10] , CommonKADS [11] and KNOMAD [12]. KNOMAD extends on MOKA to account for life cycle management issues, and focusses on the multi-disciplinary aspect of KBE applications. CommonKADS focusses on the organisational aspect of KBS application development and how various agents communicate and relate, and its results are not directly used but its influence can be found back in MOKA. Thus, MOKA appears to be the most accepted methodology for KBE application development.

## 3.1. MOKA

The Methodology and tools Oriented to Knowledge-based engineering Applications, also known as MOKA, is a methodology designed to support knowledge acquisition for KBE systems development. MOKA focuses on capturing and structuring relevant engineering knowledge [13].

The development of MOKA started in 1998 as a European initiative with partners from the aerospace, automotive, IT and academic sectors [10]. The aim was to promote the use of KBE in Europe, since Europe was falling behind the competition from America and East Asia.

The low adoption rate of KBE was most likely due to the lack of well-defined standardized methods. Furthermore, majority of the development costs of a KBE application occurred during the phase where knowledge had to be captured and structured (steps 3 and 4 in Figure 1), which was an ill-defined process before the development of MOKA [14]. Thus, MOKA decided to work on a new methodology for developing knowledge systems in engineering design. The main objectives were to: [14]

- Reduce the lead times and costs of developing KBE applications by 20-25%
- Provide a consistent way of developing and maintaining KBE applications.
- Develop a methodology which will form the basis of an international standard.

The MOKA model is composed of three sub-models as shown in Figure 8. It involves the generation of informal knowledge model (see Section 3.1.1) followed by formulation of formal knowledge model (see Section 3.1.2) which is stored in a neutral language model (Section 3.1.3) [10]. The neutral language knowledge model is then converted to KBE application code that can be used to perform various engineering tasks.

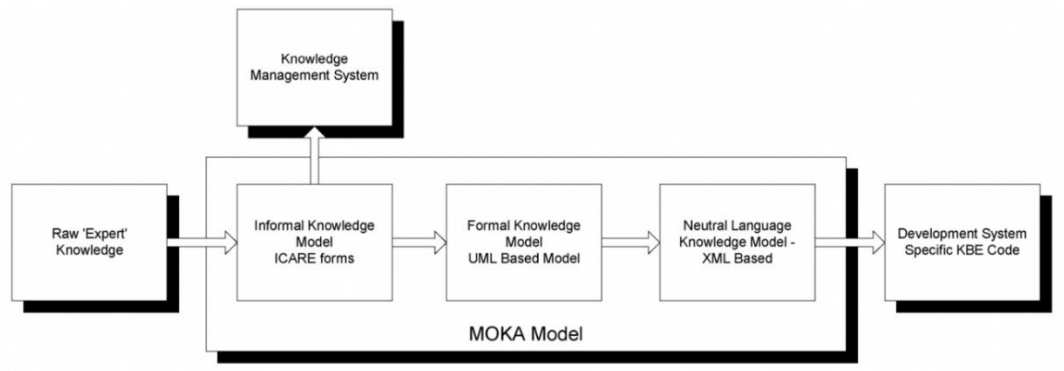


*Figure 8: MOKA Knowledge Models [13]*

### 3.1.1. Informal Knowledge Model

The informal knowledge model is structured upon ICARE forms. This acronym stands for Illustration, Constraints, Activities, Rules and Entities, and is a sub-methodology for the classification and linking of the knowledge gathered during the collection phase (see Figure 9 for an example of one of the forms) [10]. These forms provide a comprehensive and straightforward – if perhaps long-winded – approach to placing structure upon the raw knowledge. This is done by placing the knowledge into the appropriate form depending upon its type, then defining the links between the individual knowledge elements. Whilst they can be completed in paper format, some form of software tool would greatly speed up this process, particularly when checking the validity of the model. MOKA also considers the informal model as being particularly suitable for inclusion into an organisational knowledge management system [13].

The forms can be described as follows: [3]

1- **Illustration forms** can be used to represent all kinds of more generic knowledge, overview descriptions, or comments.
2- **Constraints forms** are used to model interdependencies between entities.
3- **Activity forms** are suitable to describe the various problem-solving steps in the design process.
4- **Rule forms** allow the modelling of control knowledge.
5- **Entities** are used to describe the product object classes (not only the physical ones but also functions, behaviours, etc.).

The set of ICARE-forms describes product and process knowledge at an informal level that is understandable by both the knowledge engineer and the domain expert. This is an important aspect because the expert has to validate the knowledge before it can be processed in the next step [14].

| MOKA ICARE FORMS: | Structural Entity | |
|---|---|---|
| Name | 2 litre bottle assy | |
| Reference | ES5.BOTTLE.ASSY.2L | |
| Related Functions | Containment of liquid<br>EF5/1_LQD.CONTAIN<br>Prevent contamination of contents<br>EF5/2_LQD.CONTAM.PREV<br>Withstand handling and storage<br>EF5/3_ROBUSTNESS | |
| Behaviour | Ease of assembly<br>EB5/1_ASS | |
| Context, info, validity | 2l bottle for non-carbonated soft drinks | |
| Description | The bottle assembly consists of two parts:<br>Main bottle body<br>• Bottle cap<br><br>Main bottle body is blow moulded on-site from recyclable blue or clear PETE depending upon the type of liquid bottle is intended to contain<br><br>Bottle cap is mid-blue in colour and bought in directly from Bericap | |
| Related entities | Parent | |
| | Children | ES5_1.BOTTLE.2L<br>ES5_2.CAP |
| | Undefined | |
| Information origin | Interview with D. Smith from design dept. 11-10-03 | |
| Management | Author | STP |
| | Date | 23-10-03 |
| | Version Number | 1.1 |
| | Status | In Progress |

*Figure 9: Example of MOKA entity form [10]*

### 3.1.2. Formal Knowledge Model

The next stage is to use the informal model (Figure 9) to develop a formal knowledge model, which is made up from two parts, the product model, and the design process model (Figure 10 on the left and on the right, respectively).

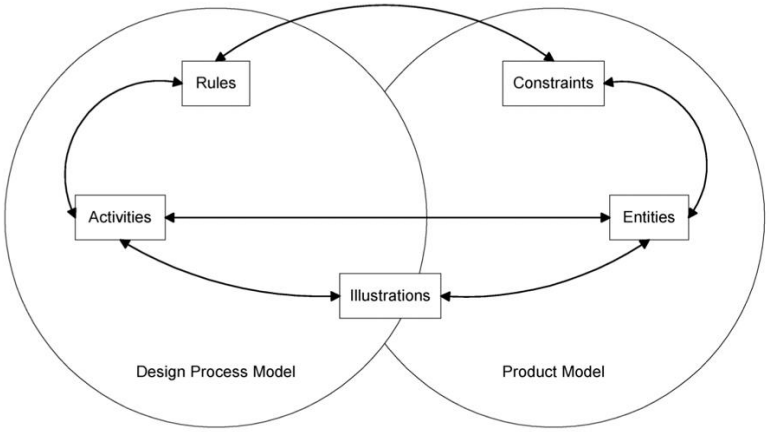


*Figure 10: The MOKA formal model [10]*

A Knowledge Engineer (KE) studies the ICARE forms and manually converts this information into a MOKA Modelling Language (MML) which has been developed in order to notate the formal model; MML being an extension of UML.

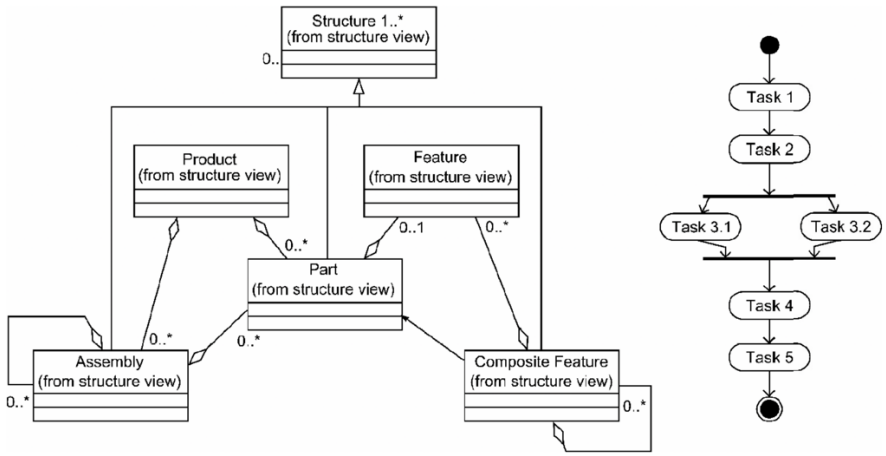


*Figure 11: MML product model and design process model diagrams [3]*

UML is particularly suitable for this task as it has become a recognised standard for the design and analysis of object-oriented software; several KBE systems have been written using (some form of) object-oriented programming language. MML extends on UML primarily through the use of stereotypes. These are generic classes of objects defined by MML to represent the different aspects of the knowledge stored in the informal model. Examples of stereotypes include a product class, an assembly class, a material class, and a manufacturing class. MML also defines product and design process meta-models, displaying the relationships between the stereotypes [13].

### 3.1.3. Neutral Language Knowledge Model (XML)

The formal models are converted to a neutral language knowledge model to provide a link between the UML-based formal knowledge model and the final software code. The purpose is to develop a

model which can readily be translated – perhaps with a high degree of automation – into the KBE development platform source code. MOKA recommends the use of eXtensible Markup Language (XML) to build the neutral model, although it does not provide a detailed methodology as yet for developing such a model, because the direct import of XML-based models is not currently supported by the various KBE development system vendors [10].

## 3.2. Code Generation: State of practice

Since neutral language knowledge model does not exist, developers take on the role of converting knowledge model into source code. In the MOKA approach that is practiced today, this process is completely manual (i.e., no automation scripts or software programs are available to (even partly) generate the code). Thus, the complete architecture of the code, the organization of rules and requirements within the code is at the discretion of the developer. Often, developers can use implicit programming style, which can cause difficulty for other developers and domain experts to comprehend the code. Consequently, guaranteeing that all knowledge rules and requirements are complied with becomes challenging.

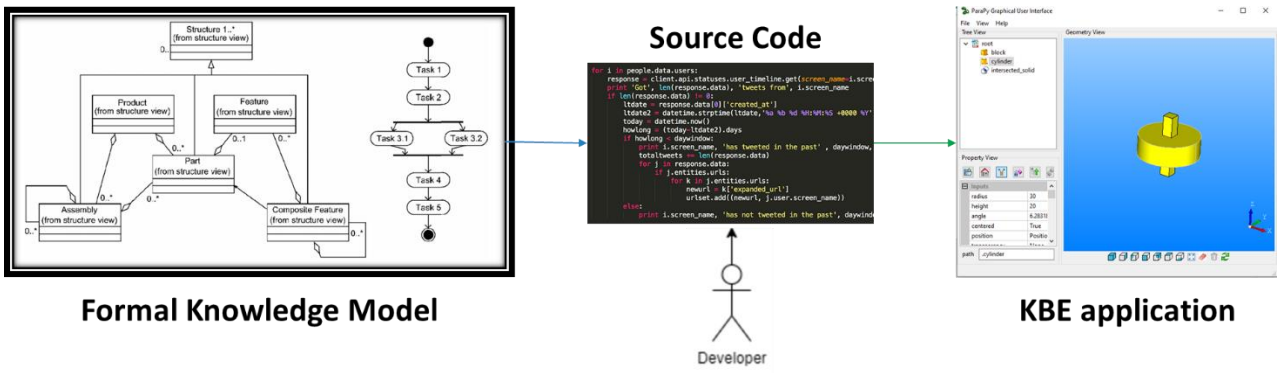


*Figure 12: Role of developer in converting formal knowledge model to KBE application*

Furthermore, for all practical purposes, the project-to-project knowledge transfer happens in this phase. Thus, the developer's comprehension and accuracy of the formal knowledge model plays a critical role in ensuring correct knowledge transfer. In addition, the manual translation of formal model to source code is laborious and time-consuming process that can lead to human errors. All these factors can lead to (significant) knowledge losses. Particularly if the developer of the original source code is not available for consultation during the project where the KBE application must be executed.

## 3.3. Stakeholder's trust in KBE applications

The KBE application development process can be categorized into three main scenarios. Each of these will be discussed in the following paragraphs.

### 3.3.1. Scenario 1

The first scenario is the one described by the MOKA methodology, where the **domain expert** has no coding experience and where there is a **knowledge engineer** available to structure and formalize the knowledge from the domain expert into models, so that it's understandable by the application **developer**. This scenario is schematically represented in Figure 13.

Here, the domain expert is expected to populate a knowledge base to develop an application. However, the domain expert is not able to see or understand how knowledge has been implemented in the KBE application (i.e., the expert is unable to trace where the rules and requirements are placed). This means that there is a lack of trust in the KBE application [15]. Moreover, the lack of transparency in KBE applications gives the domain expert a feeling of exclusion, causing insufficient participation in the development of a KBE application which could hinder adoption of KBE technology [16].

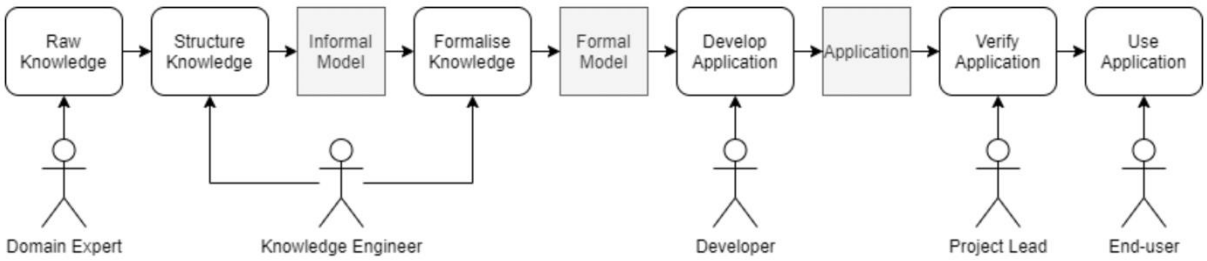


*Figure 13: Current steps of scenario 1 in the KBE app development process*

### 3.3.2. Scenario 2

In conceptual design projects (having small design teams and short lead-time), a single engineer often fulfils multiple roles (e.g. as a domain expert and as a developer), no specialist knowledge engineer is available to structure and formalize the knowledge from the domain expert into models. In this case, the domain expert provides the required knowledge for the development of KBE application to the developer via a set of unstructured informal models / documents. This scenario is schematically represented in Figure 14.

Here again, the domain expert faces the same challenges as in case of scenario 1. Furthermore, the developer has additional challenges as there are no formal models available. The developer is not able to work efficiently as (s)he will not be able to interface easily with other developers as an overarching architectural view is missing. This is especially the case in a collaborative environment that is very common today. A developer will have difficulties understanding the application code of a colleague as most knowledge embedded in application code is represented by formulas and data without clear context [17], making it difficult to process.

Moreover, the black-box perception is aggravated as there are no formal models to compare the code with. Without any formal model, the developer will be prone to making errors in case the comprehension from informal model is not correct.

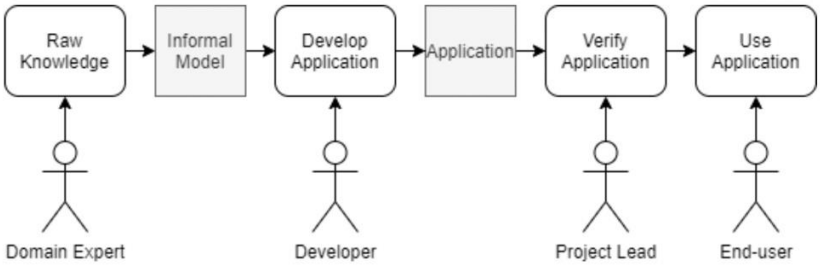


*Figure 14: Current steps of scenario 2 in the KBE app development process*

### 3.3.3. Scenario 3

The third scenario is the one where the domain expert has experience in coding (software engineering) and therefore is also the developer of the KBE application. In this case, the domain expert usually skips the knowledge modeling step and implements the required knowledge directly into the KBE application. This scenario is schematically represented in Figure 15.

Here, when a domain expert leaves the company (or is unavailable for consultation in the project), the knowledge contained in the mind of the expert is lost. This is due to the fact that other domain experts are not able to understand the knowledge that is structured as (implicit) application code. It is also hard to find out where the knowledge is in the vast amount of application source code [4].

This is the most acute problem when one engineer takes on multiple roles of developer, knowledge engineer and domain expert. Furthermore, motivating domain expert/developer to formalize their knowledge model is challenging because they do not find any returns on the time invested in formalization. In stark contrast to scenario 1, guaranteeing proper checks on the final application becomes difficult because the domain expert is far too invested in the application development.

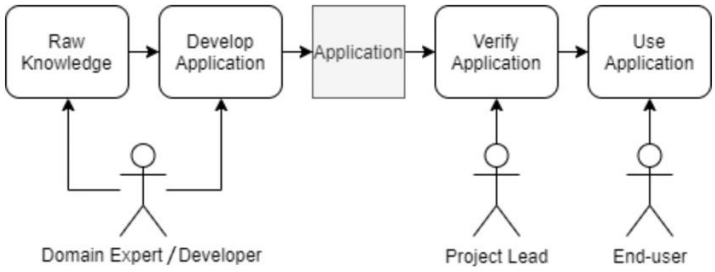


*Figure 15: Current steps of scenario 3 in the KBE app development process*

In all the scenarios mentioned above, the project lead has to verify that the developed application complies with the requirements. Due to the black-box type source code, the project lead is not able to ascertain whether knowledge has been implemented correctly or whether the application is computing an output as required.

The only way a project lead could verify if the application is doing what is required is by running the application and trying out some values. However, the fact that the application came to the correct conclusion does not mean it did so by deriving the correct facts. It is very hard to validate the application, and that makes it difficult to convince the customer that it does what it should [4].

### 3.3.4. Discussion: End-user trust

The end-user is the stakeholder that uses the KBE application to automate (part of) the design process. Often, the domain expert is also the end-user of the KBE application. Thus, depending on the scenario, the end-user's trust on the code might vary. In scenario 1 and 2, the end-user might have a lack of trust in the application as they are not able to ascertain how the application is computing the results. Here, the application just 'magically' computes an output from some inputs [4]. In such cases, adoption of KBE applications in the design process becomes challenging. This is also the case if the end-user is not necessarily the domain expert. In scenario 3, there is no trust-deficit in the developed application. However, the maintainability and scalability of the code in the absence of original developer becomes challenging.

## 3.4. Summary: Limitations of current KBE application development

Based on the current KBE application development process, several limitations can be identified as follows:

i. The encapsulation of knowledge within source code is largely a manual process. As a result, there is excessive reliance on the communication and understanding between the domain expert and the developer. The tedious manual approach can lead to human-errors. Extensive knowledge transfer combined with identifying and rectifying the error leads to ***longer development times***.

ii. Generally, the KBE application source-code tends to be big. When programmed manually, only the developer is conversant with information within the source code. Thus, tracing the errors in the code, extracting the embedded knowledge rules and guaranteeing compliance of requirements by the KBE application becomes challenging if the original developer of the code is unavailable during the project execution (i.e., **the KBE application becomes a black-box**).

iii. In case formal knowledge model is not developed before (manual) code generation, **permanent knowledge loss** can occur if the domain expert and the programmer leave the organization, which can severely hamper project-to-project knowledge transfer.

iv. Till date, industry-accepted **neutral-language knowledge model schema** is not available owing to lack of concerted effort from research fraternity and the industry. This is a major bottleneck in the MOKA application development cycle shown in Figure 8 to quickly generate KBE application from neutral knowledge model.

v. Even if a neutral language knowledge model schema were to be developed, no mechanism exists to **automatically convert the formal knowledge models (UML/MML diagrams) into neutral language knowledge model schema.** To date, this task has been largely performed manually.

vi. Application of MOKA, one of the most used KBE application development approaches, **requires a specialized knowledge engineer** to convert informal knowledge models to formal knowledge models. However, the availability of trained knowledge engineers is scarce, which makes the application of MOKA challenging. Furthermore, MOKA is mainly aligned to support knowledge engineers and not domain experts and developers.

vii. The informal knowledge modelling based on **ICARE schema is not intuitive** to domain experts as they work in terms of requirements, products and processes. In the view of domain experts, the extra effort to convert their knowledge into ICARE format appears to be a pedagogical exercise without significant return on the invested time.

# 4. Moving to faster and transparent KBE application development

To improve the current KBE development process, the challenges summarized in Section 3.4 must be solved. The key solutions to those challenges can be summarized as follows:

i. A knowledge formalization technique which captures the requirements and associates them with appropriate process steps and product features (i.e., Requirement-Product-Process Ontology) as shown in Figure 16 instead of more involved ICARE relationships. This will enhance the involvement of domain experts in the development of KBE applications and thereby enhance its acceptability.
ii. The models should not be in the form of collection of documents but digital models that can be used to dynamically generate natural language knowledge models.
iii. The neutral language knowledge model should be used to automatically generate (a part of) the KBE application. This will justify the efforts invested in the generation of formal knowledge model as the domain experts/developers gain time by automatic generation of KBE application source code.

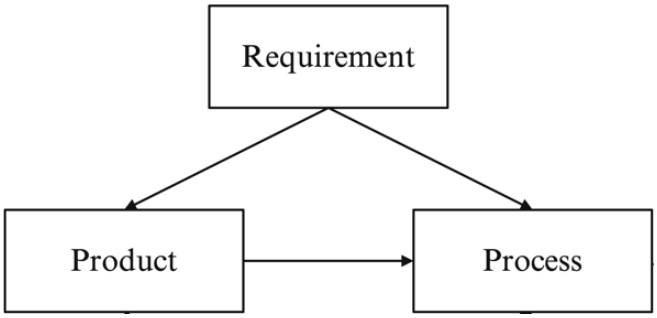


*Figure 16: Relationships among requirement, product and process (adapted from [18])*

In order to incorporate these solutions, we propose the use of a modified application development process. Here, the domain experts directly model their Requirements-Product-Process ontology using an MBSE based systems engineering language called Systems Modelling Language (SysML) (discussed further in Section 5).

MBSE based SysML is proposed because it is an extension of classical UML and is widely used in the industry for a number of systems engineering applications. Furthermore, a number of Commercial Off-the-Shelf (COTS) as well as Free and Open-Source Software (FOSS) SysML-compliant modelling tools for implementing MBSE[4] have already been developed and extensively used. Examples of these COTS tools are:

i. Cameo Systems Modeller (Dassault Systemes)[5]
ii. Rational Rhapsody (IBM)[6]
iii. SCADE (Ansys)[7]
iv. Enterprise Architect (Sparx Systems)[8]

[4] https://sysml.org/sysml-tools/
[5] https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/
[6] https://www.ibm.com/products/systems-design-rhapsody
[7] https://www.ansys.com/products/embedded-software/ansys-scade-suite
[8] https://sparxsystems.com/products/mdg/tech/sysml/index.html

In addition to modelling knowledge, these tools also provide a mechanism to automatically convert diagrammatic knowledge models into neutral language knowledge models. This is an important enabling step in speeding up the KBE application development process which was not possible with classical MML.

The neutral language knowledge model (stored as XML/JSON files) can be used to automatically generate a skeleton code of the KBE application. This includes the definition of the product structure and key attributes in the code that evaluate the product performance. However, this skeleton KBE application does not (yet) include the formulation of programming intensive properties, attributes and functions. After the generation of the skeleton KBE application, developers can use their programming skills to complete the missing code-intensive parts of the KBE application as shown in Figure 17.

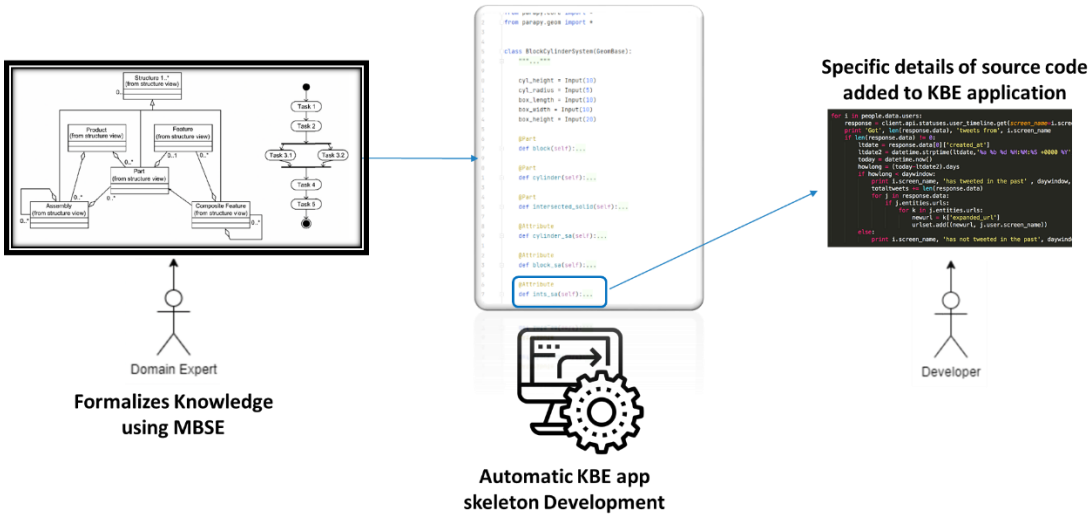


*Figure 17: Proposed methodology to automate the generation of skeleton KBE application supplemented by inputs from developers on programming-intensive parts*

Currently, systematic means to automatically generate the skeleton KBE application is missing. This will be developed during this project. However, the proposed methodology will help in overcoming all the challenges discussed in Section 3.4. Since engineers can automatically generate most of their code by learning knowledge modelling diagrams, their involvement in KBE application development increases.

Another advantage of this methodology is the guaranteed generation of formal knowledge model which cannot be skipped if KBE application skeleton is to be generated automatically. This reduces the excessive reliance on one domain expert or developer and becomes a natural documentation for project-to-project knowledge transfer. Finally, any change in the knowledge model can be **automatically reflected in the skeleton** application. This allows engineers to **quickly trace** the location of different rules, methods and requirements encapsulated in the KBE application and modify them as necessary.

In the remainder of this deliverable, we discuss the MBSE based requirement-product-process ontology modelled using SysML that can be used by domain experts to model their knowledge. Furthermore, we identify a sub-set of SysML called SysML-lite that is sufficient to capture the key elements of KBE application.

# 5. Requirement-Product-Process ontology in MBSE

The need to identify an appropriate requirement-product-process ontology was motivated in the preceding sections. Thus, in the context of this research work, MBSE techniques should be used to produce and control a coherent system model. This system model is then used to specify and design the system. The system model includes system specification, design, analysis, and verification information. The model consists of model elements that represent requirements, design, test cases, design rationale, and their interrelationships.

A number of MBSE modelling languages are available in literature. Some of these languages are:

i. Arcadia
ii. IDEF - Integration DEFinition
iii. OPM - Object Process Methodology
iv. UML - Unified Modeling Language
v. SysML - Systems Modeling Language

In this research, SysML is chosen as the MBSE based graphical modeling language because it is widely used and has a proven track record of meeting the industry standards. In this section we discuss SysML and its relationship with UML further. We also provide an overview of diagrams supported by SysML. Finally, we identify a sub-set of SysML called the SysML-lite which can be used to generate KBE application skeletons.

## 5.1. SysML

SysML, the Systems Modeling Language, is a general-purpose graphical modeling language that supports the analysis, specification, design, verification, and validation of complex systems. The modeled systems may include hardware, software, data, procedures, people, facilities and other elements [19]. It is a subset of UML 2 with extensions needed to satisfy the requirements of the UML for Systems Engineering RFP, as indicated in Figure 18 [20].

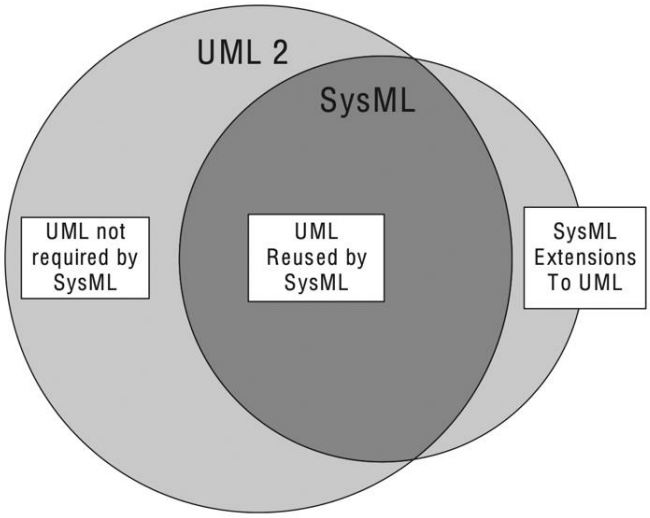


*Figure 18: The relationship between SysML and UML [20]*

The UML for Systems Engineering RFP was developed jointly by the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE) and issued by the OMG in March 2003. The RFP specified the requirements for extending UML to support the needs of the systems engineering community. The SysML Specification was developed in response to these requirements by the diverse group of tool vendors, end users, academia, and government representatives. The Object Management Group announced the adoption in July 2006 and the availability of OMG SysML v1.0 in September 2007. Since then, SysML has been subject to several updates, with the latest RFP for SysML v2 being issued in December 2017. Currently, the most recent specification available is SysML v1.6 published in December 2019 [19].

SysML supports modeling of requirements, structure (product structure), behavior (process information), and parametrics (the so-called “4 pillars of SysML”) in order to provide a complete description of a system, its components, and its environment. The language includes 9 types of diagrams that can be used to describe different views of the system being modeled. The model contains the model elements, which are captured in a model repository. A particular model element may appear on zero, one or multiple diagrams. In addition, a model element often has relationships to other model elements that may appear on the same diagram or other diagrams [21]. This will be discussed further in the following sections.

### 5.1.1. Relation between SysML and UML

SysML is defined as a lightweight dialect (Profile) of UML 2.x (Unified Modeling Language), the industry standard modeling language for software-intensive applications. It's considered a lightweight profile because it makes relatively small modifications to the underlying language, using a small number of simple stereotypes, tagged values and constraints; as opposed to a heavyweight profile that could modify significantly the way that the underlying language is used. Defining SysML as a UML Profile has some advantages, one being that it can reuse the relatively mature notation and semantics of UML 2.x, which is already well established among software engineers and that many modeling tool vendors have already implemented.

In general, UML targets Software Engineers applying Model-Driven Development (MDD) methods, while SysML targets Systems Engineers applying Model-Based Systems Engineering (MBSE) methods. SysML offers systems engineers some advantages over UML for specifying complex systems, namely:

- SysML expresses systems engineering semantics (interpretations of notations) better than UML, reducing UML’s software bias;
- SysML is smaller and easier to learn than UML, since it removes many software-centric and unnecessary constructs, the overall language is smaller as measured in diagram types (9 vs. 13) and total constructs.

### 5.1.2. Diagram Overview

SysML inherits many of the UML diagrams, these are either largely reused without modification or slightly modified with lightweight customizations, and on top of that it adds two new useful diagrams that allow for the modeling of requirements and parametrics. The taxonomy of SysML diagrams is presented in Figure 19.

Each diagram type is summarized here, along with its relationship to UML diagrams [7]:

- Package diagram represents the organization of a model in terms of packages that contain model elements (same as UML package diagram);
- **Requirement diagram** represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability (not in UML);
- **Activity diagram** represents behavior in terms of the order in which actions execute based on the availability of their inputs, outputs, and control, and how the actions transform the inputs to outputs (modification of UML activity diagram);
- **Sequence diagram** represents behavior in terms of a sequence of messages exchanged between systems, or between parts of systems (same as UML sequence diagram);
- **State machine diagram** represents behavior of an entity in terms of its transitions between states triggered by events (same as UML state machine diagram);
- **Use case diagram** represents functionality in terms of how a system is used by external entities (i.e., actors) to accomplish a set of goals (same as UML use case diagram);
- **Block definition diagram** represents structural elements called blocks, and their composition and classification (modification of UML class diagram);
- **Internal block diagram** represents interconnection and interfaces between the parts of a block (modification of UML composite structure diagram);
- **Parametric diagram** represents constraints on property values used to support engineering analysis (not in UML)

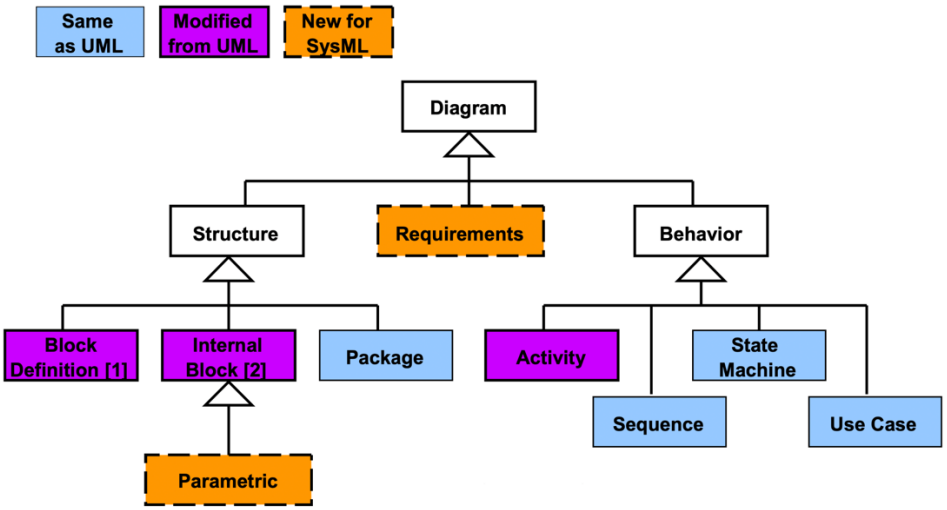


*Figure 19: SysML diagram taxonomy (adapted from [21])*

## 5.2. SysML-Lite for Requirement-product-process ontology of KBE applications

Although SysML is a concise version of UML, it brings with it a number of drawings that are not particularly useful in the generation of KBE application. For example, the sequence diagram and state machine diagrams are not strictly necessary for the development of KBE application. Consequently, when exposed to the diagrams of SysML, domain experts must invest time and effort in understanding and learning diagrams that are not necessary for KBE application skeleton generation.

Thus, to alleviate the learning curve of domain experts, a sub-set of SysML called SysML-lite (originally proposed by Friedenthal et. al. [7]) is proposed as the ontology to support formal knowledge modelling for KBE application development. SysML-lite is composed of 6 main diagrams as shown in Figure 20.

The internal block diagram, parametric diagram and block definition diagrams are used to define the product ontology. The activity diagram is used to define the process ontology and the requirement diagram is used to capture requirement ontology. Each of these diagrams themselves have a number of model elements and connections links used to define the relationship between different model elements. 0provides a detailed overview of all the model elements and their connections necessary to generate a formal knowledge model.

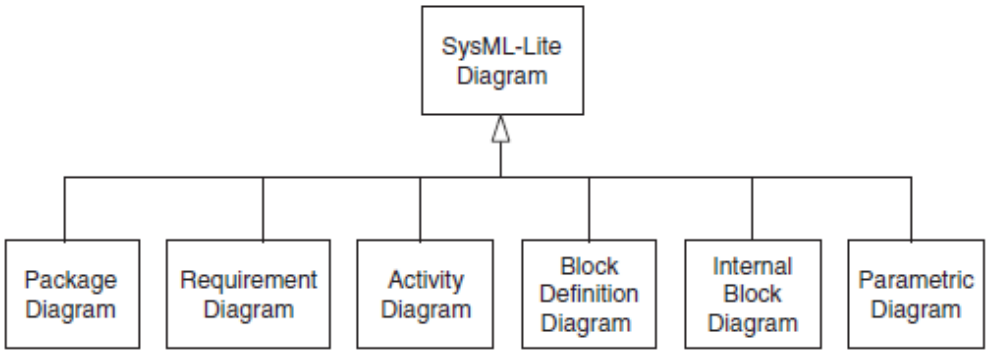


*Figure 20: Taxonomy of SysML-Lite model*

Package diagram is used to combine all the diagrams together and link the requirements to process and product diagrams. Two special matrices are available in SysML, namely:

i. **Satisfaction Matrix**: to indicate product and process diagram elements that satisfy a given requirement
ii. **Allocation Matrix**: to indicate product diagram elements that perform a given task

These matrices help engineers in assessing the compliance of requirements and guaranteeing all the tasks that must be performed by a KBE application are incorporated in the source code.

# 6. Conclusions and Outlook

In order to support Front-Loading in design processes, KBE applications can be effective means. However, the current KBE application development methodology is largely based on manual communication between domain experts and developers (often via knowledge engineers). This leads to long development times. Furthermore, KBE applications generated manually offer formidable challenge in tracing requirements, rules and methods (i.e., implicit knowledge) embedded in the code.

In a bid to move away from implicitly coded KBE applications, we propose the automatic generation of skeleton KBE applications (i.e., part of the application including simple and repetitive syntax) based on formalized knowledge model (provided by domain experts) which the developers can augment by generating the programming-intensive aspects of the application. This requires a definition of a formal knowledge model (i.e., requirement-product-process ontology) that can be automatically converted to neutral language knowledge model.

In this research work, we propose the use of MBSE based SysML language to formalize the knowledge model. SysML is chosen because it provides additional features (compared to UML/MML) such as requirement diagrams and allocation and satisfaction matrices. Furthermore, these models are available as (digital) graphical modelling languages and can be quickly converted neutral language knowledge model using COTS/FOSS modelling software.

In addition, we propose the use of SysML-lite a subset of SysML as it provides adequate means to model all the information necessary to generate skeleton KBE applications. The key diagrams of SysML-lite and an exhaustive list of all the components of each diagram have been discussed in this document. For the first iteration of this deliverable, initial study has been performed by ParaPy in conjunction with their customers, which substantiates the use of MBSE (using SysML-lite) for the knowledge formalization step in the KBE application development process and solving the problem of the disconnect between the knowledge models and the application code. The detailed implementation of the ontology and proof of applicability of MBSE based requirement-product-process ontology description will be provided in the following iterations of the deliverable.

In the current state of the project, the ability to generate KBE application skeleton from the neutral language knowledge model is missing. Furthermore, the neutral language knowledge model provided by COTS/FOSS modelling software are intended to support SysML and not specifically SysML-lite. Thus, these neutral language knowledge models have many more features and details that are typically not useful for KBE application ontology (but adds to the learning curve of domain experts). In the next iteration, we will propose a neutral language knowledge model that specifically caters to SysML-lite and demonstrate its application with case-studies. Furthermore, an attempt will also be made to generate formal knowledge models from pre-existing KBE applications. This will help in documenting large (pre-existing) KBE applications and improve project-to-project knowledge transfer.

# References

[1] G. La Rocca, “Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization,” 2011. Accessed: Jan. 16, 2022. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A45ed17b3-4743-4adc-bd65-65dd203e4a09

[2] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, T. M. Shortell, and International Council on Systems Engineering, Eds., *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th edition. Hoboken, New Jersey: Wiley, 2015.

[3] C. Chapman, S. Preston, M. Pinfold, and G. Smith, “Utilising Enterprise Knowledge with Knowledge-Based Engineering,” *Int. J. Comput. Appl. Technol.*, vol. 28, no. 2/3, pp. 169–179, Apr. 2007, doi: 10.1504/IJCAT.2007.013354.

[4] K. Wheeler, “Methodological Support for Knowledge Based Engineering Application Development: Improving Traceability of Knowledge into Application Code,” 2020. Accessed: Jan. 13, 2022. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A01ee12dd-3941-45e0-9baf-d58ff7b823e0

[5] T. Van den Berg, “Harnessing the potential of Knowledge Based Engineering in manufacturing design,” 2013. Accessed: Jan. 15, 2022. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3Ad44876fe-45dd-44ae-8947-622613eb963c

[6] B. Vermeulen, “Knowledge based method for solving complexity in design problems,” 2007. Accessed: Jan. 16, 2022. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A767e71bd-9156-468e-9563-7e7bff0ee25b

[7] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*, 3rd ed. Boston: Morgan Kaufmann, 2015.

[8] A. Raju Kulkarni, G. La Rocca, T. Berg, and R. Dijk, “A knowledge based engineering tool to support front-loading and multidisciplinary design optimization of the fin-rudder interface,” Oct. 2017.

[9] S. Thomke and T. Fujimoto, “The effect of ‘front-loading’ problem-solving on product development performance,” *J. Prod. Innov. Manag.* vol. 17, no. 2, pp. 128–142, Mar. 2000, doi: 10.1016/S0737-6782(99)00031-4.

[10] MOKA Consortium, *Managing engineering knowledge: MOKA: methodology for knowledge based engineering applications*. London Bury St. Edmunds, 2001.

[11] G. Schreiber, B. Wielinga, R. Hoog, H. Akkermans, and W. Velde, “CommonKADS: A comprehensive methodology for KBS development,” *IEEE Expert*, vol. 9, pp. 28–37, Dec. 1994, doi: 10.1109/64.363263.

[12] R. Curran, W. J. C. Verhagen, M. J. L. van Tooren, and Ton. H. van der Laan, “A multidisciplinary implementation methodology for knowledge based engineering: KNOMAD,” *Expert Syst. Appl.*, vol. 37, no. 11, pp. 7336–7350, Nov. 2010, doi: 10.1016/j.eswa.2010.04.027.

[13] S. Preston, C. Chapman, M. Pinfold, and G. Smith, “Knowledge Acquisition for Knowledge-Based Engineering Systems,” *Int. J. Inf. Technol. Manag.*, vol. 4, no. 1, pp. 1–11, Mar. 2005, doi: 10.1504/IJITM.2005.006401.

[14] P. K. M. Chan, “A New Methodology for the Development of Simulation Workflows: Moving Beyond MOKA,” 2013. Accessed: Jan. 16, 2022. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A51041c5c-66c2-4d47-bf20-91459a538f7d

[15] I.-S. Fan and P. Bermell-Garcia, “International Standard Development for Knowledge Based Engineering Services for Product Lifecycle Management,” *Concurr. Eng.*, vol. 16, no. 4, pp. 271–277, Dec. 2008, doi: 10.1177/1063293X08100027.

[16] R. Curran, W. Verhagen, and M. van Tooren, “The KNOMAD Methodology for Integration of Multidisciplinary Engineering Knowledge within Aerospace Production,” in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, 2010. Doi: 10.2514/6.2010-1315.

[17] W. J. C. Verhagen and R. Curran, “Knowledge-Based Engineering Review: Conceptual Foundations and Research Issues,” in *New World Situation: New Directions in Concurrent Engineering*, London, 2010, pp. 267–276.

[18] B. Yu and H. Zhao, “A Semantic Ontology of Requirement – Product – Process – Resource for Modeling of Product Lifecycle Information,” in *The 19th International Conference on Industrial Engineering and Engineering Management*, Berlin, Heidelberg, 2013, pp. 319–330. Doi: 10.1007/978-3-642-37270-4_31.

[19] Object Management Group, “OMG SysML | OMG Systems Modeling Language.” https://www.omgsysml.org/ (accessed Jan. 17, 2022).

[20] J. Holt and S. Perry, *SysML for Systems Engineering: A Model-Based Approach*, 3d edition. Stevenage: The Institution of Engineering and Technology, 2018.

[21] C. Moreland, “An Introduction to the OMG Systems Modeling Language (OMG SysML™).” ARTiSAN Software Tools, 2008. [Online]. Available: https://www.omg.org/news/meetings/workshops/Real-time_WS_Final_Presentations_2008/Tutorials/00-T2_Moreland.pdf

# Appendix A: Cheat-Sheet SYSML ontology

SysML has 9 different types of drawings. However, not all of them are necessary for Knowledge Acquisition of KBE app development. In the following sections, we discuss the key elements of these drawings necessary for KBE application knowledge formalization. The design of these SysML drawings does not form part of this deliverable and will be discussed in the next iteration.

## A.1: Package Diagram [pkg]

1. **Model** - *A model contains a (hierarchical) set of elements that together describe the physical system being modeled. It can also contain a set of elements that represents the environment of the system, typically Actors together with their interrelationships, such as Associations and Dependencies.*
2. **Containment** - *The Containment relationship represents the containment of elements on the diagram pane. A Containment path replaces a Nesting representation. For example, instead of drawing one package nested into another, you can draw a containment relationship from one package to another.*
3. **Package** - *A package groups classes and other model elements together. All types of SysML model elements can be organized into packages. Each diagram must be owned by one package and the packages themselves can be nested within other packages. Subsystems and models are special kinds of packages.*

## A.2: Requirement Diagram [req]

4. **Physical Requirement** - *A Physical Requirement specifies the physical characteristics and/or physical constraints of a system, or a system part.*
5. **Functional Requirement** - *A Functional Requirement is a requirement that specifies a behavior that a system or part of a system must perform.*
6. **Performance Requirement** - *A Performance Requirement refers to a requirement that quantitatively measures the extent to which a system or a system part satisfy a required capability or condition.*
7. **Containment** - *The Containment relationship represents the containment of elements on the diagram pane. A Containment path replaces a Nesting representation. For example, instead of drawing one requirement nested into another, you can draw a containment relationship from one requirement to another.*
8. **Dependency** - *A dependency is a relationship signifying that one or more model elements require other model elements for their specification or implementation. The complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).*
9. **Derive** - *A 'Derive' relationship is a dependency between two requirements (a derived requirement and a source requirement), where the derived requirement is generated or inferred from the source requirement.*

The arrow points from the dependent model element (client) to the independent model element (supplier). Hence in SysML, the arrow's direction is opposite to that typically used for requirements flows, where the higher-level requirement points to the lower-level requirement.

Satisfy Requirement Matrix: Requirement → Activity

SysML Allocation Matrix: Activity → Block

## A.3: Process: Activity Diagram [act]

1. **Initial Node** - *An Initial node is a control node where flow starts when the Activity is invoked. An Activity may have more than one Initial node.*
2. **Control Flow** - *A Control flow is an edge that starts an Action node after the previous one is finished.*
3. **Action** - *An Action is a named element that is the fundamental unit of an executable functionality. The execution of an Action represents some transformations or processing in the modeled system, be it a computer system or otherwise.*
4. **Fork** - *The fork vertices are used to split an incoming transition into two or more transitions terminating on the orthogonal target vertices.*
5. **Join** - *The join vertices are used to merge several transitions emanating from the source vertices in different orthogonal regions.*
6. **Object Flow** - *Used to define the flow of data between actions.*
7. **Activity Final Node** - *An Activity Final Node is a final node that stops all flows in an activity.*
8. ***Decision Node*** - *Decision Nodes allow you to separate the Activity Edges.*

## A.4: Product

### a. Block Definition Diagram [bdd]

1. **Block** - *Blocks provide a general-purpose capability to describe the architecture of a system, and represent the system hierarchy in terms of systems and subsystems. Blocks describe not only the connectivity relationships within / between a system and its subsystems, but also quantitative values as well as other information about that system (for example, documentation).*
2. **Directed Composition** - *The Direct Composition or Composition relationships convey a structural decomposition of Blocks.*
3. **Generalization (not represented)** - *The Generalization relationship conveys an inheritance between Blocks. It is usually used to create a hierarchy in your system. The notation is a solid line with a hollow, triangular arrowhead on the end.*

### b. Internal Block Diagram [ibd]

1. **Value Property** - *A Value Property is a property that specifies the quantitative property of its containing Block.*
2. **Binding Connector** - *A Binding connector is the only connector that allows to bind Constraint Parameter to the properties of other Block. It is a connector which specifies that the properties at both ends of the connector have equal values.*
3. **Part Property** - *A Part Property is a property that specifies a part with strong ownership and coincidental lifetime of its containing Block. It describes a local usage or a role of the typing Block in the context of the containing Block.*
4. **Constraint Property** - *A Constraint Property is a property that specifies the constraints of other properties in its containing Block.*
5. **Constraint Parameter** - *A Constraint Parameter is the formal term for a variable that appears in a constraint expression.*
6. **Reference Property** - *A Reference Property is a property that specifies a reference of its containing Block to another Block.*

### A.5: Parametrics: Parametric Diagram [par]

Uses the same elements as the Internal Block Diagram but collects all the parametric relationships in one place.

### A.6: Satisfy Requirement Matrix

This matrix is the place where users can link their requirements to different activities, blocks or values. Typically any requirement can be associated to another activity/block/values with two notable exceptions:

1. All performance requirements should be linked to value property
2. All functional requirements should be linked to a task in activity diagram

### A.7: SysML Allocation Matrix

The allocation matrix aims to link the activities to the blocks. This allows users to quickly reach the block (class) where a particular activity is being performed.