REPRESENTATION OF CROSS-DOMAIN DESIGN KNOWLEDGE THROUGH ONTOLOGY BASED FUNCTIONAL MODELS

Dipl.-Ing. Milan Marinov[1], Dipl.-Wi.-Ing. Dan Gutu[1], B.Sc. Janet Todorova[1], Dr. Miklós Szotz[2], András Simonyi[2], Prof. Dr. Dr.-Ing. Dr. h.c. Jivka Ovtcharova[1]

[1]Institute for Information Management in Engineering (IMI),
Karlsruhe Institute of Technology (KIT), Germany
[2]Applied Logic Laboratory (ALL), Budapest, Hungary

ABSTRACT
Domain specific development environments such as MCAD/ECAD systems can partially exchange data based on standard data formats. Due to the complex interdependencies between mechatronical components and the diversity of the related product data, it is not possible to tackle the challenges of cross-domain engineering by means of direct information exchange only. An overarching information backbone, which can be understood by engineers and processed by computers, is necessary.
This paper presents a function oriented, ontology based approach to provide such a backbone. A special functional structure models the functional interdependencies between mechatronical components. A use case of the functional structure as information backbone for representation of interdependencies is introduced. The use of ontology enables the definition of customizable taxonomies for functional modeling dialects which can facilitate even more flexible support for cross-domain engineering collaboration.

Keywords: Mechatronics, cross-domain engineering, re-engineering, functional modeling, ontologies, product information backbone, knowledge representation

INTRODUCTION
The global economy is characterized by rapid innovation, shortened development and product life cycles and rising customer expectations in terms of the performance, quality and price of future products. Product innovations make a decisive contribution to the way in which these products maintain their position in this global economy. Mechatronics – a word made up of mechanics and electronics – represents a potential means of successfully creating future products: the close spatial and functional integration of mechanical engineering, electrical engineering and information technology makes fundamentally new solutions possible, that considerably improve the cost/benefit ratio of currently known products, but can also provide a stimulus for new, as yet unknown products [22].
Highly dedicated applications for each engineering domain are available, e.g. ECAD[1] (Domain: electronics) or MCAD[2] (domain: mechanics) systems, but existing solutions for bridging between domains are still rare and not sufficient.
The complex interdependencies between mechatronic components, the diversity of the related product data and the cross-linkage of the corresponding processes are a major challenge for an effective and efficient information management.
Mechanics, electronics and software engineers use different terminology and even more important, utilize different ways of thinking and of solving problems, which are well established inside of the respective engineering domains. However in cross-domain[3] development there are significant gaps between the engineering domains, which hinder the communication between the actors of the collaboration process.

[1] ECAD: Electronic Computer Aided Design
[2] MCAD: Mechanic Computer Aided Design
[3] "Cross-domain engineering" refers to engineering collaborations which involve at least two engineering domains.

The cost of these gaps is familiar to anyone working in the mechatronics industry; manufacturing or testing teams discover at the last minute, that their electrical designs won't work with their mechanical designs – triggering a frenzy of rework that can delay launches, erode quality, and drive up costs. The same dynamic occurs between embedded software and overall electronics configurations.

Thus one of the major problems of cross-domain engineering is the lack of a unifying information backbone to keep everyone on the same page at the same time. This has to be done in such a way that the work in a specific domain can be carried out in a fashion optimal for this domain. At the same time the backbone has to provide enough information to support the collaboration with engineers from other domains.

The EU-project ImportNET[4], on whose results this paper is based on, had its focus on cross-domain, cross-enterprise and cross-cultural collaborative engineering in a network of SMEs. The main research activity was to provide a framework which builds a base to support cross-domain engineering tasks within an intercultural collaboration.

Functional modeling serves as an information backbone for the cross-domain collaboration. The functional structure contains information about the functional interdependencies of the components of the mechatronical system. The concept of product function is relevant throughout the lifecycle of a product and a function oriented approach can thus be utilized as an important element in capturing, sharing and augmenting of product lifecycle knowledge.

The paper is structured as follows: first, the State of the Art concerning functional modeling, special functional structures and ontologies is presented. Further, it is explained how cross-domain design information is represented with the help of functional structures through a cross-domain collaboration scenario. The ontology support for function-oriented cross-domain engineering is explained. The results of this approach are discussed and concluded, and finally, an outlook is given.

STATE OF THE ART

Functional modelling

On the highest abstract level, several independent approaches exist on functional modelling in the product development process. One of the first approaches to use shared design models – as a predecessor of functional modelling - for designing mechatronic products comes from Cutkosky, Mark, Tenenbaum et al. [3], who proposes with the PACT architecture a distributed agent-based product developing environment to share concepts and terminology for communicating knowledge across disciplines, an interlingua for transferring knowledge among these agents, and a communication and control language that enables the agents to request information and services. Ensuring that the interacting constraints that must be met by the set of components making up the design are fulfilled is done through a design framework described by shared design domain ontologies.

Umeda and Tomiyama proposed in [21] the concept of Function-Behaviour-State (FBS) modelling, where a function is defined as an association of intention and behaviour, and is causally and task-based decomposed into subfunctions. Based on the resulting FBS diagram, a FBS modeller can search for appropriate behaviours for a required function, identify inconsistencies and propose modifications to solve these inconsistencies, or identify and reuse functional redundancies.

Philosophical, physical and technical analysis on technical development postulates that any new invention of technical systems can handle three kinds of functional objects: Material, Energy and Information ([15], [16]). Different taxonomies were developed on this principle, like for example NIST by Szykman and the Functional Basis by Stone and Woods [5]. The Reconciled Functional Basis, resulted from the comparison and combination of these two vocabularies, product functions can be described by a discrete set of Flows (three functional objects) and a functional basis of Functions (discrete set of eight functional verbs).

An abstract product function can be described in a black-box flow model with input function object (Material, Energy, Information), the functional verb (what does the function do?) and a resulting output function object which can of course be input for downstream functions. A functional structure (flow model) can be developed.

Based on the abstract modelling scheme, several taxonomies exist to concretize functions. The functions on the most detailed level are called special functions. A hierarchical three-step-procedure to

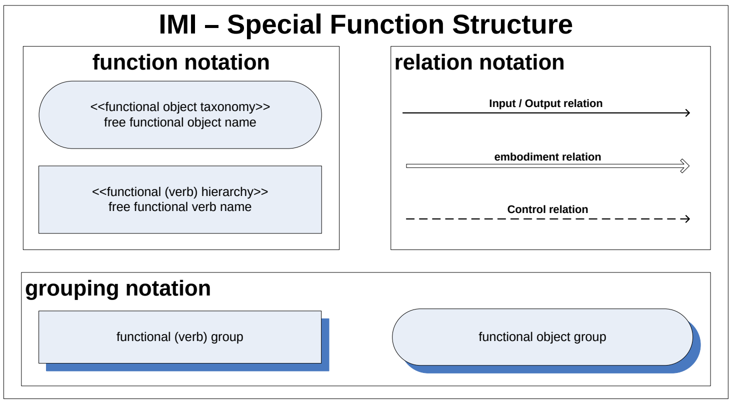
4 FP6-033610

derive the special product function from the abstract product functions under a given taxonomy and vice-versa can be read in [5].

Functional modelling approach: Special Functional Structure

Using the functional basis of verbs and objects as building blocks, function structures can be developed. The so called special functions should formally enable modelling of cross domain products. The process of creating the functional structure depends on how the borders of the system to be modelled are defined and on the intentions of the modeller. For example, a book usually has the function to carry information, but besides information characteristics like data volume and language, it has physical properties like width, height, depth and weight. Thus a book can also be used for example to support a digital projector, provided that it has the right physical properties. Depending on the intentions, emphasis can be put on the information or physical properties.

A special function structure modelling notation taking more than just taxonomy into account was conceived at the IMI. Functional objects consist of Material, Energy and/or Information and furthermore do have material-, energetic- and informative properties that can describe taxonomized state of aggregation, physical principles and informative aspects. The profound modelling notation can be read in [11]. Moreover, by identification tags and additional domain origin information, the special function structure (shortened to SFS) allows the cross domain representation of functional knowledge about a product in a formal way, which can be processed by a computer. The special functional structure modelling notation is shown in Figure 1. A simple example of the modelling notation can be seen in Figure 2.



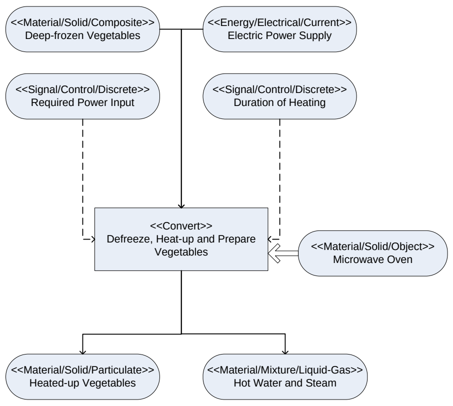

<<Material/Solid/Composite>>
Deep-frozen Vegetables
<<Energy/Electrical/Current>>
Electric Power Supply
<<Signal/Control/Discrete>>
Required Power Input
<<Signal/Control/Discrete>>
Duration of Heating
<<Convert>>
Defreeze, Heat-up and Prepare Vegetables
<<Material/Solid/Object>>
Microwave Oven
<<Material/Solid/Particulate>>
Heated-up Vegetables
<<Material/Mixture/Liquid-Gas>>
Hot Water and Steam

Figure 1: Special functional structure modelling notation

Figure 2: The heating-up of vegetables in a microwave oven modelled using the IMI modelling notation

In the example in Figure 2, the heating up of vegetables in a microwave oven is modelled according to this notation. The frozen vegetables are hereby transformed into heated-up vegetables by addition of heat, with hot water and steam as a by-product. For this, electric energy is transformed into heat by the microwave oven. The oven also receives as (user) input the required power and the duration of the heating. Accordingly, the “Microwave oven” functional object embodies the function “Convert”, as it converts electric energy into heat. The function “Convert” receives as input the material functional object “Deep-frozen Vegetables”, the energy functional object “Electric Power Supply”, as well as the information functional objects Required Power Input” and “Duration of Heating”. The output of the function “Convert” are the material functional objects “Heated-up Vegetables” and “Hot Water and Steam”.

In addition to the special functions, in this paper we also use the concept of customer functions. Such functions represent the customer view on the system. Customer functions can be realized by one or more special functions. A special function can also realize multiple customer functions.

Ontologies

The notion of ontology is widely used in the last decades, but different schools and communities understand it in different ways. In [4] there are listed 7 different interpretations of the term – and we do not consider this to be complete. We quote the definition given by [19] as the most adequate:

An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.

Conceptual adequacy and expressivity is especially important in engineering ontology applications that aim at supporting the creation, analysis or simulation of engineering designs, and this importance has led to a substantial amount of research into the problem of formalizing the central concepts of engineering, e.g. that of the structure, function and behaviour of artefacts. In this area, important work has been done on formalizing Chandrasekan's and Josephson's influential characterization of behaviour and function in the DOLCE upper ontology framework ([1], [2], [13]), and the relationship between devices and their functions [8]. The possibilities of providing an adequate ontological representation of the physical and geometrical characteristics of assemblies have also been intensively researched (see e.g. [7]). The idea of functional ontologies is presented in detail by Kitamura and Mizoguchi in [9].

For the realization of the idea of ontology-supported functional modeling, we shall introduce some of the concepts, which will be used further in this paper.

OWL

OWL is an abbreviation for Web Ontology Language. OWL supports the hierarchical representation of classes, attributes and associations of those classes and their interconnections.

OWL enables the formal representation of domain concepts, which can be used as a knowledge base for further managing and processing. OWL is a semantic mark-up language standardized by the W3C consortium for development, publishing and exchanging of ontologies in the World Wide Web.

SWRL

SWRL is a Semantic Web rules-language, combining sublanguages of the OWL Web Ontology Language with those of the Rule Mark-up Language.

It extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base [6].

Rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

For the purpose of readability and understanding, we will use a Human Readable Syntax instead of the XML Concrete Syntax for the representation of the SWRL rules. In this syntax, a rule has the form:

```
antecedent → consequent
```

where both antecedent and consequent are conjunctions of atoms written $a_1 \wedge ... \wedge a_n$. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., `?x`). Using this syntax, a rule asserting that the composition of parent and brother properties implies the uncle property would be written:

```
parent(?x,?y) ^ brother(?y,?z) → uncle(?x,?z)
```

Built-Ins

Built-Ins are predefined functions which can be used in the context of SWRL rules. For instance, comparison operations such as “less than” or “equal”, or mathematical operations such as subtraction or division are implemented as Built-Ins.

The set of built-ins for SWRL is motivated by a modular approach that allows further extensions in future releases within a (hierarchical) taxonomy.

The following example shows the usage of a Built-In for Comparison:

```
swrlb:lessThan(?x,10)
```

SQWRL

SQWRL (Semantic Query-Enhanced Web Rule Language) is a SWRL-based language for querying OWL ontologies. It provides SQL-like operations to retrieve knowledge from OWL. SQWRL takes a

standard SWRL rule antecedent and effectively treats it as a pattern specification for a query. It replaces the rule consequent with a retrieval specification [14].
The following example shows a SQWRL query:

```
Person(?p) ^ hasAge(?p, ?a) → sqwrl:select(?p, ?a) ^ sqwrl:orderBy(?a)
```

This query will return pairs of individuals and ages with one row for each pair. The results are ordered using the `orderBy` Built-In (`sqwrl:orderBy`).

USING FUNCTIONAL STRUCTURES FOR REPRESENTATION OF CROSS-DOMAIN DESIGN INFORMATION

The functional structure serves as a model-based notation for information relevant for the cross-domain engineering collaboration. Product data, e.g. components, requirements, properties etc., are organized around the functional structure, utilizing it as an information backbone. In this section an example for the representation of cross-domain interdependencies is introduced. An example product illustrates the usage of the functional structure.

Cross-domain engineering collaboration scenario

The mobile robot is a product which can manipulate objects instead of humans in dangerous environments, such as coal mines. The robot has a tracked chassis with tracked legs, which enables it to be used in terrains with variable difficulty. In its basic variant, the robot can perform functions such as movement (e.g. driving in plane, climbing stairs, crossing ditches) and manipulation of objects with a robot arm. The basic variant has a total weight of 73 kg, however, it does not include explosion protection.
Based on this variant, according to customer requirements, two more advanced variants with explosion protection are developed by collaborating SMEs. A passive protection variant includes explosion protection by means of a tougher body with 8 mm thick wall which can withstand an internal explosion. The passive protection variant has a mass of 260 kg. The higher mass of the mechanical component “casing” is the main reason for the increased the weight of the robot. The changes of this component affect parts from other engineering domains. Due to the increased mass, the electrical drives used for robot movement are not able to perform all required operations. The engineers have to check which customer functions can still be performed.
A more complex, active protection variant is developed which includes explosion protection utilizing inert gas to fill the robot body in order to prevent an internal explosion. This variant requires some additional mechanical, electrical and software components, such as a gas cylinder, gas pressure sensor, CH_4-concentration sensor, two valves, as well as a new software module for controlling the inert gas pressure inside the robot. The thickness of the body remains the same as in the basic robot variant (1,5 mm). Consequently, the active explosion protection variant weighs 80 kg.
The collaborative development is performed by the companies SIASUN[5] (China) and CADCAM[6] (Croatia). SIASUN is responsible for the implementation of the basic robot functionality of the robot, such as movement (e.g. climbing stairs) and user interaction with the robot. CADCAM is responsible for the implementation of the explosion protection functionality. An actor in the engineering collaboration can take on different roles, such as: Component responsible, function responsible, mechanics / electronics / software developer etc.
The Croatian mechanics engineer Mr. Kovac, responsible for the customer function “explosion protection” proposes firstly a passive explosion protection design. The Chinese electric/electronics engineer Mr. Ni, responsible for the customer functions “drive in plane” and “climb stairs” finds out in a discussion with the Chinese mechanics engineer Mr. Wang, responsible for the components of the chassis that the higher mass of the passive protection variant leads to some limitations of the movement functions: The function “Climb Stairs” cannot be fulfilled. This information needs to be communicated and stored using the functional structure.

[5] SIASUN Robot & Automation Co., Ltd., Shenyang, China
[6] CADCAM Design Centar d.o.o., Zagreb, Croatia

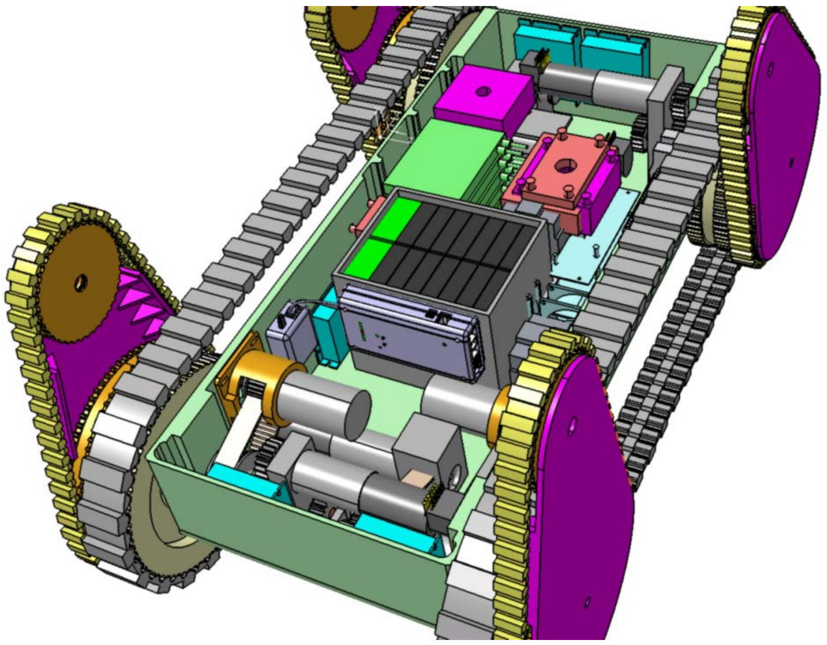


Figure 3: Example product “Mobile Robot”

Obviously, not only the “climb stairs” customer function, but also other customer functions which are realized by the special functions grouped under “transport robot” can be influenced from changes of the robot weight. The engineers decide to model the functional dependencies in a more general way in order to be able to automatically detect similar problems in the future. For the purpose of this example, we take in consideration two of the customer functions: “climb stairs” and “drive in plane”.
Mr. Ni discusses with Mr. Wang what would be the maximum allowed weight of the robot in order to fulfil these functions, without having to perform changes on the chassis or other components. Mr. Wang calculates the following maximum allowed weights: 280 kg for “drive in plane” and 120 kg for “climb stairs” customer function. These values are stored in requirements attached to the customer functions.
The functional dependencies between the “transport” functions and the weight of the robot (and in particular of the casing which caused the initial problem) are stored in rules in the ontology. The rules define that if some of the customer functions which are realized by the “transport” special functions have requirement with a weight limit, the weight of the robot should be checked. If it is greater than allowed, the corresponding customer function cannot be fulfilled and a warning is shown in the console.
The information about functional dependencies can be automatically recalled and used by the MDET tool which recognizes the role of the user and the context in which he is working. For example, if the user has opened a functional structure of the mobile robot where the customer function “climb stairs” is present (see Figure 1) and she wishes to reuse the customer function “explosion protection” in the passive variant, which is realized by the special function “isolate robot internal space”, the rules defining the dependencies are used to check, if the total weight of the robot exceeds the maximum allowed weight. If a problem is discovered, corresponding warnings are shown in the console tab.

ONTOLOGY SUPPORT FOR FUNCTION ORIENTED CROSS-DOMAIN ENGINEERING

The functional structure as well as the additional information about functional interdependencies are stored and managed by ontology. In this section we will present the detailed ontology model of the functional structure as well as the rules which represent the interdependencies. Further, we will show how relevant information can be extracted using ontology queries.

Ontology for functional modeling

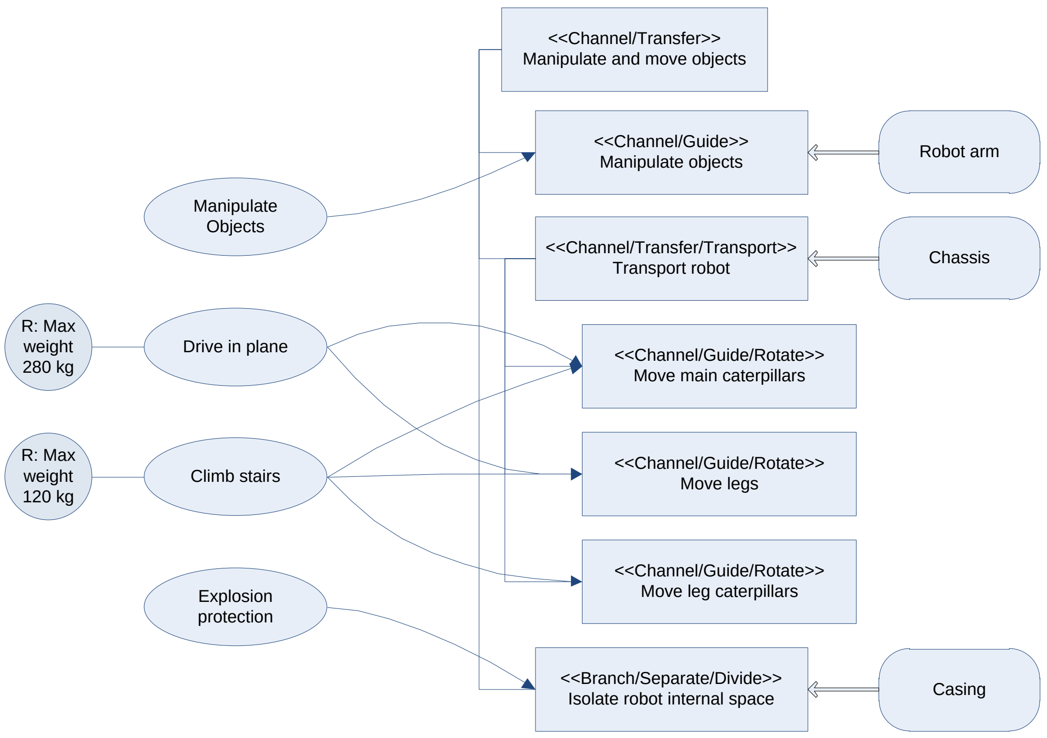
Design ontologies

In ontology modelling it is highly important to identify the individuals. Obviously, the items occurring in a design are not identical to the physical objects that are manufactured according to the design in question, for example a microcontroller figuring in the design of a robot is different from the physical microcontrollers contained in concrete robots. Moreover, an item appearing in a design can occur in several documents about it. The individuals of the design ontology can be neither physical objects nor signs appearing in documents: they must be mental objects corresponding to designs or design components. For all components of a design there is a corresponding class in the manufacturing model that contains the physical objects manufactured according to the design element in question, and also a corresponding class in the documentation model, which contains signs referring to the element (it is advisable to treat the design, manufacturing and documentation models as separate ontologies, and express their connections via ontology-mappings – see [12]). A more detailed discussion of general problems regarding design ontologies can be found in [17].

Ontology for functional design

In order to construct the ontology fragment needed for functional modelling, we have to examine the properties, relations and attributes of the individuals that figure in a functional model. Let us collect what can be said about a function instance, e.g. the special function „*Isolate robot internal space*" occurring in *Figure 4*

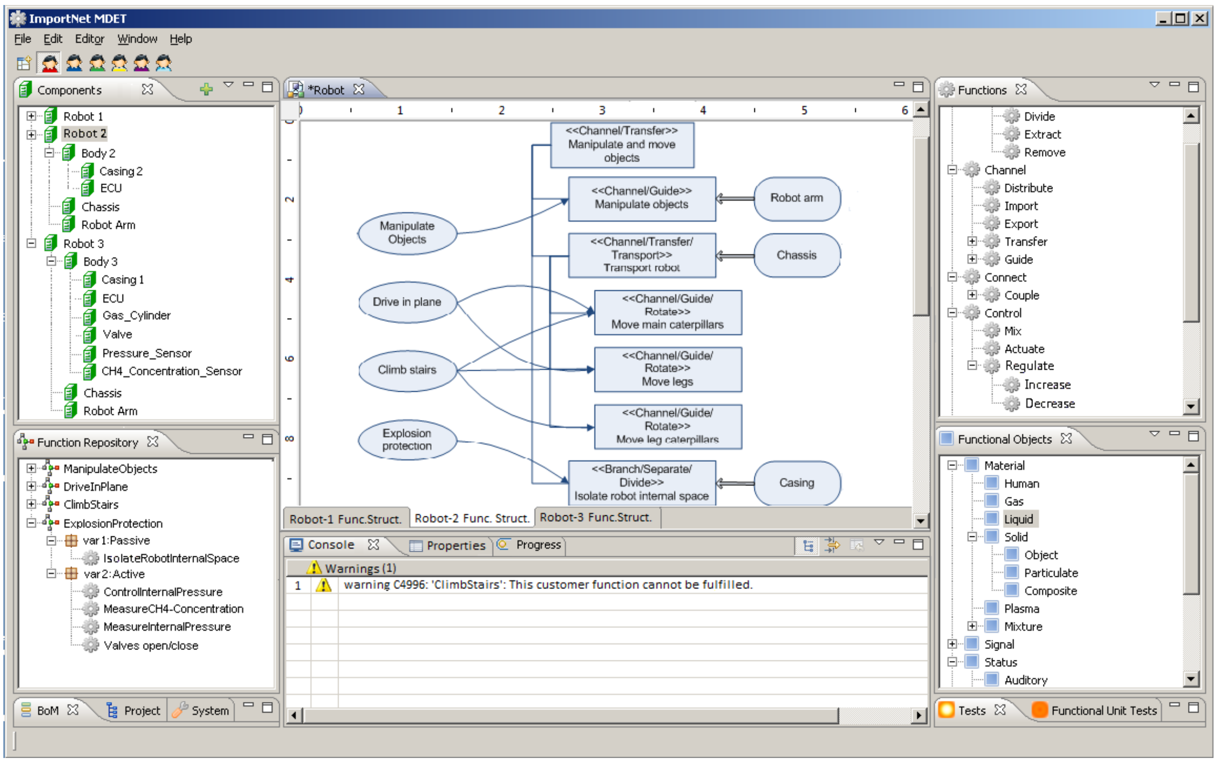
- it belongs to the given functional design;
- it has a name given by the designer;
- it belongs to a class of the functional basis hierarchy (or to a class in an another standardised function hierarchy);
- functional objects can be connected to it by the *input/output* relation (see Figure 1);
- functional objects can be connected to it by the *control* relation;
- functional objects can be connected to it by the *embody* relation;
- functions can be grouped together into a *functional group*;
- customer functions can be connected to special functions with a *isRealizedBy* relation (inverse: *realizes*);



<<Channel/Transfer>> Manipulate and move objects
<<Channel/Guide>> Manipulate objects
Robot arm
Manipulate Objects
<<Channel/Transfer/Transport>> Transport robot
Chassis
R: Max weight 280 kg
Drive in plane
<<Channel/Guide/Rotate>> Move main caterpillars
R: Max weight 120 kg
Climb stairs
<<Channel/Guide/Rotate>> Move legs
<<Channel/Guide/Rotate>> Move leg caterpillars
Explosion protection
<<Branch/Separate/Divide>> Isolate robot internal space
Casing

Figure 4: Functional structure fragment of the mobile robot with passive explosion protection

Clearly function and functional object individuals are needed. Their names (e.g. "Transport robot") do not identify their occurrences, therefore the individuals have to get identifiers, and the names are given as attribute values. The functional basis terminology names classes; the corresponding classification of the individuals can be given by the *instance of* relation (e.g. << Magnitude/Change/Condition >> can be specified by stating that the individual is an instance of the class Condition). We use the *has-input*, *has-output* relations to connect function instances to functional object ones. A relation named *in* connects the individuals to the *functional design* individual in question.



```
Warnings (1)
1   warning C4996: 'ClimbStairs': This customer function cannot be fulfilled.
```

Figure 5: Multi Domain Engineering Tool with functional structure

Further on, the function „*Isolate robot internal space*" in Figure 4 is embodied by a functional object called "Casing". This reflects the fact that the casing provides the encapsulation of the robot internal space from the environment. The ontology fragment in Figure 6 shows a special function F2 which is an instance of the class "Divide" and is embodied by the functional object O2 ("Casing"). The special function F2 realizes the customer function CF3 ("Explosion protection").
Special functions can be organized in a hierarchy, e.g. the functions "Move main caterpillars", "Move legs", "Move leg caterpillars" are sub-functions of "Transport robot". Figure 6 shows for example that the special function F1 ("Transport robot") contains the special functions F1.1, F1.2, F1.3.
Figure 6 shows the ontology model of the functional structure.
Based on the functional structure of the mobile robot (see Figure 4), and on the OWL-ontology in Figure 6, we can define ontology rules and queries which represent the interdependencies between functions and components. The following code shows the rule/query which retrieves the information about all supported customer functions which might be affected by the change of the robot weight:

```
OntologyFM:hasChild(OntologyFM:TransportRobot, ?ef) ^
OntologyFM:realizes(?ef, ?cf) ^
OntologyFM:requirement_maxWeight(?cf, ?max_w) ^
OntologyFM:Assembly(?a)^ OntologyFM:hasPart(?a, ?c) ^ OntologyFM:Component(?c) ^
OntologyFM:hasWeight(?c, ?w) ˚
sqwrl:makeSet(?s, ?w) ˚ sqwrl:sum(?sum, ?s) ^ sqwrl:groupBy(?s, ?a) ^
swrlb:lessThan(?sum, ?max_w) →
sqwrl:select(?a, ?cf)
```

This ontology rule is defined using SWRL rule language and SQWRL query language. The rule consists of an antecedent (left hand side of →) and a consequent part (right hand side of →) and is explained in detail below.

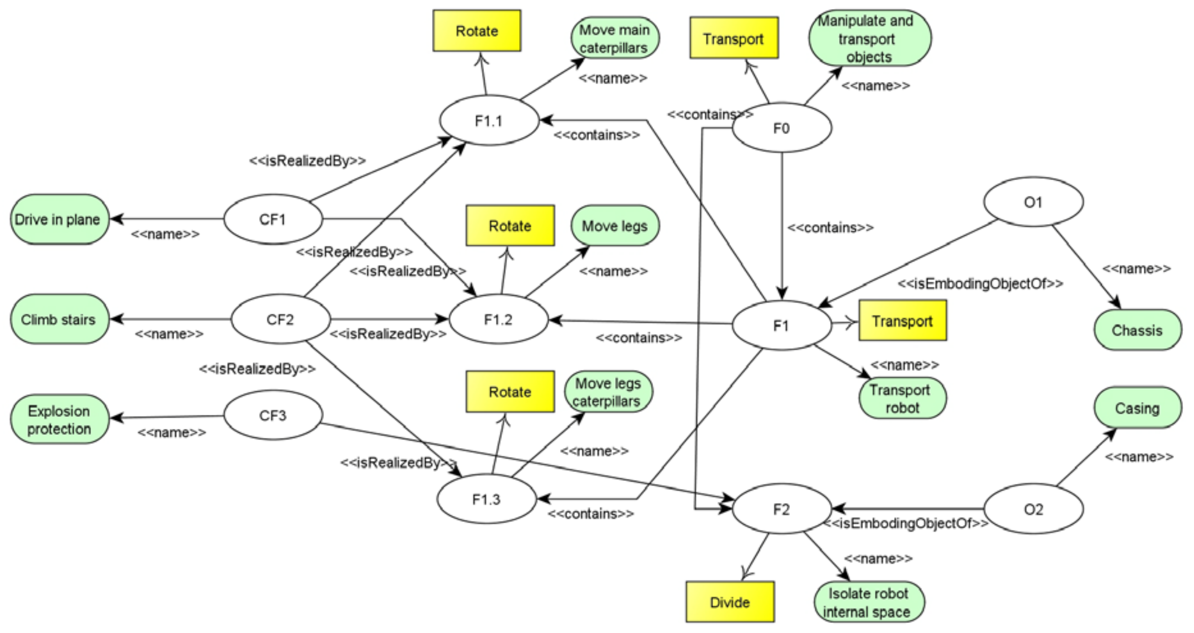


Figure 6: Functional structure modelled in ontology

Rule antecedent:

Rows 1-3: Getting the required values for maximum robot weight, which a customer function can handle.

`(1)` By changing the customer function “Explosion protection”, the engineer is changing the casing of the product which affects the weight of the robot. Because the weight might affect any of the transport functions of the robot, the rule starts with finding all children (sub-functions) of the special function “Transport Robot”. Result: *`Move main caterpillars`*, *`Move legs`* and *`Move leg caterpillars`*.

`(2)` This line retrieves the customer functions which are realized by the special functions found in the previous line. Result: *`Drive in plane`* and *`Climb stairs`*.

`(3)` Gets the requirements for maximal weight, specified earlier by the engineer. Result: in the variable *`?max_w`* now holds the requirements for the functions *`Drive in plane`* (*`max_w = 280kg`*) and *`Climb stairs(max_w=120kg)`*. Those values are used for comparison with the calculated robot weight.

Rows 4-7: Calculating the total weight of each of the robot variants and comparing it with the required maximum weight.

`(4-5)` Finds the weights of the building parts of each of the variants of the robot, so that these values can be used for calculations in the next step.

`(6-7)` Using the SQWRL Built-Ins *`makeSet`*, *`sum`* and *`groupBy`*, this line creates a set of the parts’ weight for each robot variant, and then iterates trough the set and aggregates the elements in the variable *`?sum`*. The SWRL Built-In for comparison `lessThan` compares the total robot weight with the requirements for maximum weight.

Rule consequent:

`(8)` At this line we make a query for retrieving pairs of the type *`(robot version, supported customer function)`*.

```
(Robot1, ClimbStairs)
(Robot1, DriveInPlane)
(Robot2, DriveInPlane)
(Robot3, ClimbStairs)
(Robot3, DriveInPlane)
```

Note: In the shown rule we check only the customer functions, which could be possibly affected by the change. According to the cross-domain engineering collaboration scenario, these customer functions can only be realized by children of the “Transport robot” function. The variants Robot1 and Robot3 are able to fulfil both customer functions “Drive in plane” and “Climb stairs”, because their total weight is less than the required maximum weight in both cases. Robot2 fulfils only the customer function “Drive in plane”, because its’ weight exceeds the required maximum value for “Climb stairs”. The result of the created rule is processed by MDET according to the context. Because in our example the user has selected the context of Robot2, MDET issues a console warning, that the robot variant, which is currently being modelled, cannot fulfil the “Climb stairs” customer function.

Customizable taxonomies for functional modelling dialects

The experiences made in the application of functional modelling in ImportNET showed that it is possible that the functional structure for a particular system can be modelled and interpreted differently by different engineers. The issue is even more evident if cross-enterprise and cross-cultural issues come into play.

One of the causes for this problem is the low granularity of the functional modelling taxonomies for functional verbs and functional objects. The categories defined by these taxonomies are relatively general. In this way the amount of functional verbs and objects is kept relatively low. For a “classical” functional modelling approach, this is a necessity:

1. To enable an efficient functional modelling process, the taxonomies have to be kept compact. In this way, the people creating and using the functional structure have the possibility to get familiar with the functional verbs and objects and can use them fluently.
2. The taxonomy of functional verbs and objects defines the building blocks (the basis) for functional modelling. In order to enable the modelling of a great variety of systems in an uniform way, the elements of the taxonomy have to be on a general level. With an increasing granularity of the taxonomy it becomes difficult to “reach an agreement”, to find elements which apply equally well in different products and enterprises, and at the same time to keep the taxonomy manageable.

The functional modelling methodology of the IMI addresses this issue by enabling the engineer to add a free name (textual description) to functions and functional objects. In this way, the engineer can document his personal interpretation of the function/functional object. This approach is a step forward in reducing the ambiguity of a functional structure, which works well for single engineers or small teams inside an enterprise.

However, if engineers from different domains, enterprises and cultures are involved in the modelling process, the free text names are difficult to comprehend because of their high granularity and high degree of customization.

Thus in the “classical” functional modelling (as without the support of ontology), there is a gap between the high level taxonomies of functional verbs/objects and the free text names of functions and functional objects. This gap could be filled with the definition of enterprise (or project) specific taxonomies for functions and functional objects, which can classify the items by multiple criteria, e.g. previous usage of the function/functional object in combination with MCAD/ECAD-Parts and/or tests, namespace/department in which the taxonomy is valid, purpose of the function etc.

However, the introduction of multiple dimension taxonomies on different abstraction levels raises some considerable problems in the handling of the functional structure: The effort for managing, synchronizing and mapping between the taxonomies explodes [11]. Using ontologies, a more flexible language can be defined: users can create their own system of representation.

The basic hierarchy of the functional basis can be extended by different further taxonomies according to different possible points of view. The parallel taxonomies can live together in the same ontology, and could be used at the same time. The main point is that every working community can build their own ontology, and so their **own dialect of a common functional modelling language**.

We know that engineers do not know how to use ontology editors – and they do not need to know anything about ontologies. The case of databases is similar: information systems using databases are used by users knowing nothing about the SQL language. The MDET functional modeller serves as an ontology editor – of course in a restricted way. This was proved by the Ontology Integration Tool worked out in the ImportNET project [20]. So ontologies give the users liberty to use, modify, refine knowledge and save consistency at the same time.

DISCUSSION RESULTS

The function oriented approach presented here enables a unified view on the mechatronical system, independent of the engineering domain. At the same time, engineers have the freedom to use the tools and data models which are optimized for their respective engineering domains, because the approach utilizes a higher level of modelling, which does not interfere with the existing models, but rather shows their complex interdependencies. The function oriented view on the mechatronic system complements the well known product structure view. It enables the intelligent, IT-based support of tasks, such as representation of cross-domain interdependencies. Through the use of ontology, knowledge about cross-domain interdependencies can be stored and retrieved automatically to intelligently support engineering design tasks.

CONCLUSION AND OUTLOOK

The ontology based, function oriented approach is a step forward to semi-automated, cross-domain engineering applications. It provides a basis on which further research can be performed.
The creation of the functional structure can be automated. One of the promising approaches, with high industry relevance, is the generation of a functional structure from use cases. Another approach is “form follows form” [18]. Such approaches can be utilized to provide a higher level of automation and thus reduce the effort for the user.
Customizable taxonomies enable the definition and usage of dialects of the functional modelling language. The MDET tool is further developed to support the definition and usage of project and enterprise specific taxonomies with automated support for mapping (translating) between different taxonomies.
The MDET can be extended with regard to its visualization capabilities. Virtual reality techniques can be used in order to enable an interactive and immersive interface to MDET. Integrated visualization of mechatronic components and the functional structure, e.g. through so called visualization metaphors, can enhance the user experience with the virtual environment [10].
The application of the functional oriented approach for automation in the virtual validation is of particular interest for the industry. Current research is focused on using the ontology based functional structure for automating the process of virtual validation of product designs.

ACKNOWLEDGEMENTS

The functional basis (taxonomy of functions and functional objects) used in ImportNET and in this paper is described by Hirtz, Stone et. al. in [5]. The example product “Mobile robot” was developed by ImportNET partner SIASUN[7]. The Ontology Integration Tool (OIT) is developed by ImportNET partner Applied Logic Laboratory (ALL[8]). The MDET is developed by the Institute for Information Management in Engineering (IMI[9]).

REFERENCES

[1] Borgo S. et al. *A Formal Ontological Perspective on the Behaviors and Functions of Technical Artifacts.* Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2009, 23(Special Issue 01), 3-21.
[2] Chandrasekaran B. and Josephson J.R. *Function in device representation.* Engineering with Computers, 2000, 16(3), 162-177.
[3] Cutkosky M. R., Engelmore R. S., Fikes R. E., Genesereth M. R., Gruber T. R., Mark W. S., Tenenbaum J. M and Weber J. C. PACT: AN Experiment in Integrating Concurrent Engineering Systems. In: *IEEE Computer Journal, Volume 26*, 1993, pp. 28-37.
[4] Guarino N. and Giaretta P. Ontologies and Knowledge Bases: Towards a Terminological Classification. *Towards Very Large Knowledge Bases*, 1995, pp. 25-32.
[5] Hirtz J., Stone R., McAdams D., Szykman S. and Wood K. A functional basis for engineering design. *Research in engineering design*, 2002, 13, 65-82.
[6] Horrocks, I. et al. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML,* http://www.w3.org/Submission/SWRL/, 2011.

[7] SIASUN Robot & Automation Co., Ltd., Shenyang, China
[8] Alkalmazott Logikai Laboratorium Kutato Fejleszto Szovetkezet, Budapest, Hungary
[9] Institute for Information Management in Engineering, Karlsruhe Institute of Technology, Germany

[7] Kim K.Y., Manley D.G. and Yang H. Ontology-based assembly design and information sharing for collaborative product development. *Computer-Aided Design*, 2006, 38(12), 1233-1250.

[8] Kitamura Y., Koji Y. and Mizoguchi R. An ontological model of device function: industrial deployment and lessons learned. *Applied Ontology*, 2006, 1(3), 237-262.

[9] Kitamura Y. and Mizoguchi R. Ontology-based systematization of functional knowledge. In: *Journal of Engineering Design*, *Volume 15, Issue 4*, 2004, pp. 327-351.

[10] Krappe H. *Extended Virtual Environments for interactive, immersive Usage of Functional Models* (in German), 2009 (Universitätsverlag Karlsruhe).

[11] Langlotz, G. *A Contribution to the Development of Functional Structures for innovative Products* (in German), 2000 (Shaker Verlag)

[12] Maedche A., Motik B., Silva N. and Volz, R. MAFRA - A MApping FRAmework for Distributed Ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, 2002, pp. 235–250 (Springer Verlag).

[13] Masolo C. et al. WonderWeb Deliverable D18. *IST Project 2001-33052 Deliverable, available at: http://www.loa-cnr.it/Papers/ D18.pdf*, 2003.

[14] O'Connor, M. and Das, A. *SQWRL: a query language for OWL.* http://bmir.stanford.edu/file_asset/index.php/1474/BMIR-2009-1395.pdf, 2011

[15] Pahl G. and Beitz W. *Design Theory: Basis for effective Product Development – Methods and Applications* (in German), 6th Edition, 2005 (Springer Verlag, Darmstadt).

[16] Roth K. *Designing with Design Catalogue* (in German), *Volume 1 Design Theory, 3rd Edition*, 2000 (Springer Verlag, Braunschweig)

[17] Simonyi A. and Szőts M. The Need for an Engineering Top Ontology. In *ICCME '09 conference, available at http://importnet.salzburgresearch.at/images/ImportNET_Bilder/Presentations/session2_miklós_szoets.pdf*, 2009.

[18] Stone R. and Bohm M. Employing the Abstraction of Product Functionality to Unify Mechatronic Design. In *ICCME '09 conference, available at http://importnet.salzburgresearch.at/index.php?option=com_content&task=view&id=4&Itemid=5*, 2009.

[19] Studer, R., Benjamins VR., Fensel D. (1998): Knowledge Engineering: Principles and methods IEEE Transactions on Data and Klnowledge Engineering 25(1-2): 161-197

[20] Szőts M. and Schneider F. Generating Ontology User Interface for Naïve Users. In *ICCME '09 conference, available at http://importnet.salzburgresearch.at/images/ImportNET_Bilder/Presentations/session2_miklós_szoets.pdf*, 2009.

[21] Umeda Y. and Tomiyama T. FBS Modeling: Modeling Scheme of Function for Conceptual Design. In *ProceedingsWorking Papers of the 9th Workshop on Qualitative Reasoning about Physical Systems*, 1995, pp. 271-278.

[22] VDI 2206. Design methodology for mechatronic systems. *Verein deutscher Ingenieure, VDI-Gesellschaft Entwicklung Konstruktion Vertrieb*, 2004, pp. 1-120.