

Exploitation Method for Functional Product Requirements - An Integrated Function Oriented Approach

Daniel P. Politze
Daimler AG
Böblingen, Germany
Email: daniel.politze@daimler.com

Jens Bathelt
inspire AG
Zürich, Switzerland
Email: bathelt@inspire.ethz.ch

Abstract—The development of mechatronic products - in particular across the domain borders - is challenging. Possible interdisciplinary improvements are neglected and design inconsistencies are neither prevented nor identified efficiently. Moreover, the complexity of mechatronic products is growing continuously due to an ongoing impact of software on product functions. In addition, the quality of a product is often judged by the quality of its functions. Thus it becomes hard to track the functions of a specific product and how they are realized and the quality of the product functions cannot be assured.

Daimler is facing this problem by extending the traditional requirement list with functional requirements in the early design stages. This Function Oriented Product Description (FOPD) is leading to a mature product specification, because it is able to grow and adapt while designing the current mechatronic product and the following product generations of one product family. This work presents a novel approach to gain a benefit directly from the FOPD for the succeeding design process. Therefore the authors provide directions for deriving the so-called Extended Function Structure (EFS) from a FOPD. The EFS is enhancing the traditional function structure by considering sensors, actors and the control logic explicitly. This approach will be exemplified in a case study.

Index Terms—Functional requirements, Function Oriented Product Description, Extended Function Structure, Functional Modeling, Design Methodology

I. INTRODUCTION

The growth of system complexity in automotive industry is mainly driven by the fact that product functions are more and more realized by the combination of mechanical, electric/electronic, and software components.

We assume that customers want product functions and they think in functions, because when a customer has a problem with his car he normally complains about defect functionality and not about certain components. In the same way he demands reliable functions, when he is looking for a new car.

Therefore, we infer that the quality of a product is also perceived by its functionality. That means, in order to provide high quality product functions, it is not sufficient to bring mechanical, electric/electronic and software components to perfection separately. Hence, the component's interactions and their contributions to product functions become important and should be considered very early in the design process. For

this reason product functions shall be captured and described, resulting in an explicit function oriented product description (FOPD), which allows reuse and improvement of existing function specifications and thus evolves to a mature function oriented product specification. In order to provide tool support for easy FOPD creation and maintenance, a formal model for representing a FOPD is needed. Such a model has been presented in [1] and is sketched in the next section of this paper.

Another approach presents an extended function structure (EFS) [2], which describes a mechatronic system from a technical point of view focusing on the seamless integration into the system and domain-specific design.

FOPD and EFS are suited to co-operate in the development of mechatronic products. Thus we identified a way how the information from a FOPD can be transferred into a EFS, which enables further development as described in [2].

The remainder of this paper is structured as follows: In the next section II a formal model is presented, that fulfils special modeling demands and hence is suitable for the representation of function oriented product descriptions. Subsequently the idea and basic principles of the extended function structure is given in section III. Based on the foundations from the former sections, an approach on how to exploit an FOPD in order to derive an EFS is presented in section IV. Both approaches are embedded in the common design methodology VDI 2206 [3] for the development of mechatronic products. Furthermore five steps are provided and explained that enable the derivation of the EFS based on the FOPD. The application of the steps is exemplified in section V before section VI concludes.

II. FUNCTION ORIENTED PRODUCT DESCRIPTIONS

In this section we present a formal function model that is appropriate for describing customer-related functions of high complex mechatronic products. The model presented here allows traditional functional modeling and provides a solution for the modeling demands that have been identified in [1]. In short, there is a need to express some sort of activities between functions, which means in some cases it is necessary to say that a function starts, pauses or stops another function. It is

also demanded to define sequences between those activities. Furthermore a way to define preconditions for the availability of functions as well as for the activities between them is needed. Finally it was pointed out in [4], that it is advantageous to have variability information integrated in a FOPD.

Because a customer-related function may be realized by a composition of subfunctions, functional decomposition must also be supported by a FOPD model, like in most functional models. Therefore the paradigm of Pahl and Beitz [5] is followed where a hierarchy of functions is working on flows. Based on that a function is assumed to have inputs and outputs, which are defined as explicit parts of the model. In this way we extend the traditional distinction between function and flow by describing their interrelation with corresponding input descriptors and output descriptors.

For every outgoing flow, there exists an output descriptor, represented by a small white ellipse. Similar to that there is an input descriptor for every incoming flow. Depending on the type of associated activity, an input descriptor is either represented as a small coloured box. In this example white and black are used, which means a flow starts or stops a function respectively. Additionally there is a chronological order defined between two input descriptors. Thus, it can be represented that a "function 1" first stops "function 2" and then starts "function 3".

At this point it should be clear how activities and sequences may be represented, which addresses half of the modeling demand from above. Additionally, the solution for preconditions and variability information requires special attributes on functions and descriptors. Thus, preconditions and variability information are simply added by the use of a character string. Strings have been successfully applied for this purpose in bill of materials (BOM) systems in the automotive industry. Also strings allow the formulation of variability information and preconditions in natural language and therefore ensures enough expressiveness.

In Fig.1 the models for the representation of functions, flows and descriptors are given. Each has an identifier and its definition is surrounded by braces.

According to the given definition every function must have a single *name*. We encourage to use a pragmatic solution as suggested in the work of Eisenhut [6]. In order to be consistent with existing functional modeling approaches, the *type* of a function can be specified according to a function class taxonomy, such as the functional basis [7]. *var* and *cond* are the attributes for providing information about variability and preconditions. *sub* is to refer to other functions in the sense of functional decomposition. These functions are considered as subfunctions. *in* and *out* are references to descriptors that are intended to describe the input and the output of a function. Our model of a function is also conform with the idea of having at least one input and one output.

Since descriptors are unique there is no need to name them. However it is required to specify its *type*, which may be either input descriptor or output descriptor. Also necessary is the reference to a single *flow*, the descriptor is meant for. The next

```

Function{
    name      String      1
    type      FunctionType 1
    var       String      0..1
    cond      String      0..*
    sub       Function    0..*
    in        Descriptor   0..*
    out       Descriptor   0..*
    props    Property     0..*
}

Descriptor{
    type      DescriptorType 1
    flow      Flow          1
    var       String        0..1
    cond      String        0..*
    act       ActivityType   0..1
    before    Descriptor     0..1
    after     Descriptor     0..1
}

Flow{
    name      String      1
    type      FlowType    1
}

```

Fig. 1. FOPD representation scheme

two items are *var* and *cond*, which are already known from the scheme for functions and serve the same purpose, which is to provide information about variability and preconditions.

The remaining items are *act*, *before* and *after*, which are intended for describing activities and sequences. Those have to be specified for input descriptors only. The item *act* is for specifying the type of activity that happens when the referenced flow is detected. From our current experience we propose "starts", "stops", "pauses" and "resumes" as adequate types of activities. *before* and *after* point to other input descriptors and thereby define a chronological order between them.

As in the case of functions, each flow must have a unique *name*. One reason for that is that it helps to check for consistency. Another reason is that it allows to express particular events. Such events can be interpreted as binary information signals and labelled with pragmatic names. The last item of the flow scheme is meant for the *type* of flow. For the same reasons as they apply for functions this type can be described according to an existing flow taxonomy, such as the functional basis [7].

III. EXTENDED FUNCTION STRUCTURE

The EFS extends the traditional function structure (FS), which is traditionally used in mechanics towards mechatronics. The traditional FS as described by Pahl and Beitz [5] provides the following elements: Functions, Material, Energy and Information flows between the functions and a hierarchy between the functions. In addition, the EFS provides [2], [8]:

- Transition conditions on the information flows

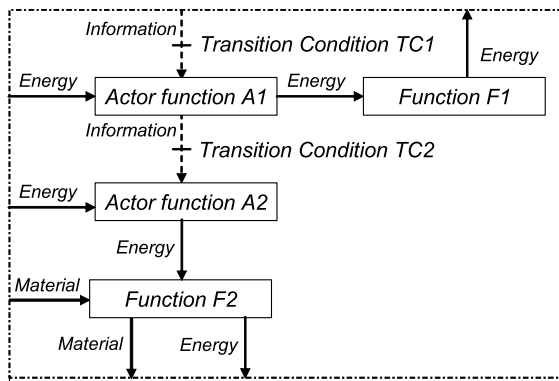


Fig. 2. Fragment of a generalized EFS

- Actor and sensor definitions
- Defined input and output variables

The input and output variables are collected in the I/O-list and do carry the information from the sensors and for the actors. Those global variables are used in the control software to read the information provided by the sensor via the input variables and to force the actor via the output variables. A transition condition on an information flow contains a Boolean expression build up on the input variables. Some functions in the EFS are so called actor functions, if an actor is assigned to that function.

A generalized EFS illustrating the actor functions and transition conditions is depicted in Fig.2. The transition conditions - carrying the information whether the next function is exploited - are always connected to an information flow. The actor functions ("A1" or "A2" in Fig.2) are receiving supply energy and information such as "start" or "stop". The output of actor functions consists of energy acting on the basic system via the following function ("F1" or "F2" in Fig.2). Since hierarchy is supported, a superfunction may contain the fragment shown in Fig.2 and "F1" and "F2" can be further detailed by subfunctions. The actor functions "A1" and "A2" are usually not further detailed, since the corresponding actors are typically purchased items when designing a car or other mechatronic systems.

Thanks to the extensions realized in the EFS, the main control sequence of the mechatronic system can be extracted from the EFS as described in [8].

The software ELVAN¹ supports the EFS approach as described above. It is a add-on extending MS Visio and uses MS Access in the background to manage the EFS data. Specific ELVAN shapes are provided to enable the definition of the EFS. The engineer starts by defining the overall function. Sub functions are collected on a separate sheet. Flows are propagated automatically throughout the hierarchy. The appearance of a function depends on the function type (function with or without sub function and/or function with or without an actor definition). The corresponding hierarchy view is build automatically based on the functionstructure modeled on the

¹See <http://www.nova-innovation.com>, ELVAN-Homepage

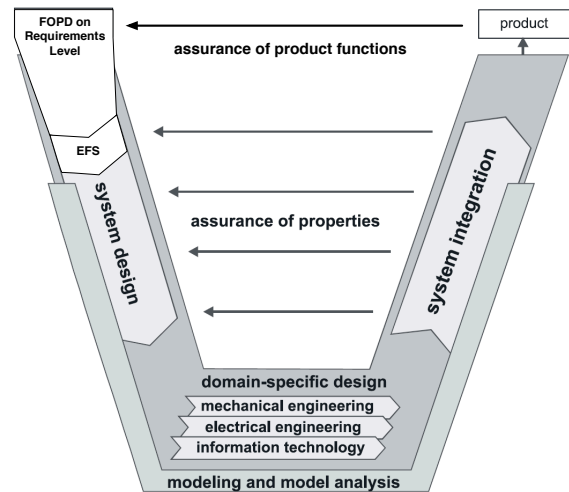


Fig. 3. FOPD and EFS in the context of VDI2206

Visio sheets and displays additional information concerning the used actors, sensors and transition conditions.

In [2] the EFS and ELVAN is embedded in the V-model of the VDI guideline 2206 [3]. The common mechatronic data base, which is maintained and defined in ELVAN, ensures a common interdisciplinary functional description (EFS) of the mechatronic system. Furthermore, ELVAN is able to derive the programming language SFC (Sequential Function Chart) [9] automatically. This allows the software engineers to start the software design with a compilable specification of the main control logic i.e. in the PLC programming environment SIMATIC². Moreover, the physical design is supported by the automatic derivation of a morphologic table [3], [5].

IV. DERIVATION OF THE EFS EXPLOITING THE FOPD

The FOPD is describing and managing all requirements and product variants from the customer's point of view in terms of product functions in a holistic way. Whereas the EFS is describing the mechatronic system from a technical point of view and focuses on the seamless integration into the system and domain-specific design.

As shown in Fig.3, FOPD and EFS are suited to cooperate in the development of mechatronic products and can be integrated in the beginning of the design process according to VDI2206 [3]. A FOPD aims at early capturing and describing the overall product functions from a requirements perspective and the EFS enables the transition to a technical point of view.

In the following, the authors present five steps that enable a straightforward transition from the FOPD to an EFS, which has two major advantages. Firstly, all functional requirements and product functions of all product variants that are desired by the customers can be managed and maintained with an FOPD [1], [4]. Secondly, the convenient derivation of the EFS allows the detailed description of software and hardware functions of a mechatronic system from a technical viewpoint that serves as a basis for the domain-specific design.

²See <http://automation.siemens.com/simatic>

TABLE I
STEPS FOR EXPLOITING THE FOPD

No.	Step Description	Result
1	A new subfunction under the overall function of the FOPD is created in the FOPD. It has the same name as the overall function beside the underscore in addition as a first character. Mainly, this collector function will be exploited in the EFS when detailing the technical functionalities of the system.	General FOPD including the collector function for the EFS.
2	Specify a product variant and eliminate variability information	Specific FOPD describing a single product variant
3	All functions and flows of the FOPD resulting from stage 2 forms the initial EFS	Initial EFS
4	Every input descriptor in the FOPD is a subfunction of the corresponding function in the EFS. The name of this new subfunction uses the corresponding activity type as a prefix followed by the name of the corresponding function. These new subfunctions are actor functions in the EFS.	EFS enhanced with actor functions
5	Three cases are exploited in step 5: (5.1) A single information flow connected to an input descriptor (5.2) A function only connected to an output descriptor which again only is connected to an information flow (5.3) In addition to 5.2, the function has an energy flow as an input. The corresponding function is a sensor function pointing to a sensor providing specific information for the control in the mechatronic system. Case 5.1 implies a sensor function accordingly. The corresponding transition condition in the EFS is combining the <i>cond</i> -attribute of the sensor function and the information flow in a Boolean expression.	EFS enhanced with transition conditions

These corresponding steps are summarized in table I and represent a predefined process module for handling recurrent working steps as introduced in the VDI guideline 2206 [3]. For the proper application of these steps a consistent FOPD must be given as a prerequisite. That means, all incoming and outgoing flows must be handled by subfunctions and vice versa.

In step 1, a collector subfunction is introduced, that has not been described initially in [1] but is conform with the modeling. This collector subfunction is marked by a leading underscore and holds all not yet known subfunctions to maintain completeness and consistency. Thus the sum of all subfunctions always yield the complete superfunction. Often this collector function holds functions of the later control logic, but in any case it is intended that this function will be refined in later development stages.

As intended, the steps presented in table I allow the derivation of an EFS from a FOPD. Therefore a specific product variant must be selected in step 2, because a FOPD is able and meant to manage functional requirements across multiple products or product variants, whereas an EFS is specific for a single product variant. Thus the decision regarding product configuration, product structure and design units must be made based upon the FOPD. When a product variant has been selected, the information in the FOPD must be filtered according to that. This will result in a specific FOPD that does not contain variability information anymore.

In step 3 an initial EFS is created by all functions and flows contained in the current FOPD from step 2. It is important to note that the flow information contained in a FOPD must be seen as a minimum, which means for each flow in a FOPD there exist at least one flow in an EFS, but there can be more due to further refinement. So flows in a FOPD should be differentiated from real existing flows, which are more likely to be captured in an EFS.

Step 4 aims at the identification of actors from the FOPD. All input descriptors with a certain activity assigned are interpreted this way and the initial EFS is enhanced by defining actor functions correspondingly. Therefore the input descriptors from a FOPD have to be translated into additional subfunctions in the EFS, that capture the activity, e.g. an input descriptor that "starts" a certain functionality e.g. "press can" is translated into the actor function "start press can".

The final step 5 is completing the EFS by deriving transition conditions. This is done by identifying sensor functions from the FOPD and by exploiting the *cond*-attribute of the corresponding functions and flows. In a FOPD functions that have at least an energy flow as an input and an information flow as an output can be interpreted as sensor functions (case 5.3). This is due to the fact that actors transform information (from the control) into energy (acting on the basic system) and sensors transform energy into information [3], [10]. Besides a single information flow that is connected to an input descriptor (case 5.1) and a function that has no input but an information flow as output (case 5.2), can also be seen as sensor functions; it is just not yet fully described in the FOPD³ or beyond consideration. The contents of *cond*-attribute of the input descriptor and if existing of the emitting function are combined to a single logic expression, which defines a transition condition in the EFS.

V. EXAMPLE

The following example describes an automatic can squeezer and is meant to illustrate the approach and the application of the steps presented in this work. The can squeezer is a mechatronic system that is able to squeeze a manually provided can automatically. The example assumes three different product variants of the can squeezer. The "standard" system

³This is conform with the idea and modeling of a FOPD, since it aims at capturing functional specifications/requirements. Thus a FOPD must only be consistent, but not necessarily complete.

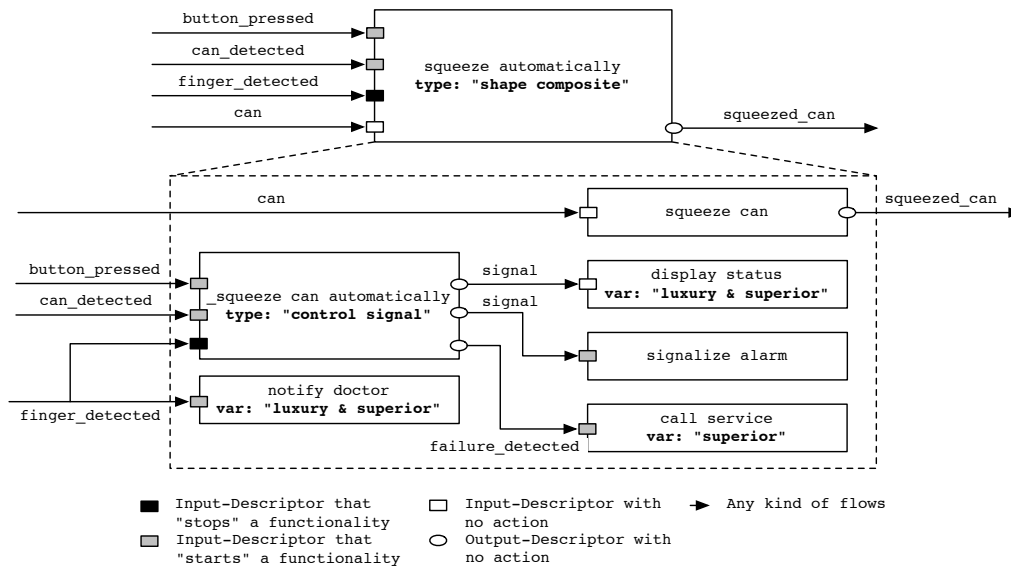


Fig. 4. After step 1: the FOPD of the can squeezer with its product variants and a collector function

starts with the automatically squeezing as soon as either a can has been detected or a designated button has been pressed. In the case when a finger is detected it will stop the squeezing and an alarm will sound. There is also a "luxury" variant of the system that shows the status of the system and notifies a doctor whenever a finger is detected. Furthermore there is a "superior" variant that includes all product functions of the other variants and additionally makes a service call in the case of a failure.

The information given in the previous paragraph can be modeled in a FOPD as presented in [1] containing the product functions, its subfunctions, variability information and further requirements of the automatic can squeezer and its product variants. Therefore several information flows are assumed that start or stop the overall function respectively and since the abstract function in the sense of [5], [7] is "shape composite", an additional material flow has been added. Those flows and their corresponding descriptors are then assigned to the five subfunctions in Fig.4 that have been inferred from the given system description above. Finally a collector subfunction "_squeeze can automatically" has been introduced as described in the initial step and all flows that could not be assigned so far have been assigned to it. This results in a FOPD as shown in Fig.4.

In step 2 the variability information contained in the FOPD will be processed and evaluated. Since an EFS is meant to be specific for a single product variant a configuration decision must be made at this point. Based on that the information in the FOPD is filtered leading to a subset that is specific for a certain product variant and does not contain variability information anymore. In this example the "luxury" variant has been chosen. Thus the subfunction "call service" is omitted.

According to step 3, all functions and flows of the FOPD as shown in Fig.4 are composing the initial EFS. Fig.5 is depicting the corresponding EFS for the automatic can squeezer

captured in the software tool ELVAN [8]. The hierarchy of the functions is maintained in the upper left corner of the window. Every function containing subfunctions has a dedicated sheet displaying the subfunctions, all incoming and outgoing flows of that function and all flows between the subfunctions. A bidirectional navigation browsing and editing the EFS exploiting both the hierarchy and flow view is possible in ELVAN.

Step 4 is enhancing the initial EFS with actor functions by exploiting the input descriptors of the FOPD. The collector function "_squeeze can automatically" has two input descriptors starting the corresponding functionality and one input descriptor stopping that function (see Fig.4). Those actions are implying actor functions in the EFS accordingly. In this example two new actor functions "Start pressing" and "Stop pressing" are created as subfunctions from the collector function, as shown in the hierarchy window in Fig.5. As actor functions, they contain output variables exploited in the control forcing the corresponding actor. In the case of the actor function "Start pressing", the digital output variable "do_press_start" is assigned. Moreover, the value *TRUE* is assigned to this variable in order to trigger the corresponding signal to the actor.

The final step enhances the EFS with the transition conditions by extracting and processing the cond-attribute of the sensor functions and their corresponding information flows. For instance, the information flow "can detected" is implying a sensor function (case 5.3) detecting the can. This leads to the transition condition "recognize can", which is also depicted in Fig.5. The transition condition is only fulfilled if the can is detected *AND* the preconditions such as a proper machine state are *TRUE*. In the sheet of the collector function "_squeeze can automatically" (not shown) is the corresponding information flow connected to the actor function "Start pressing".

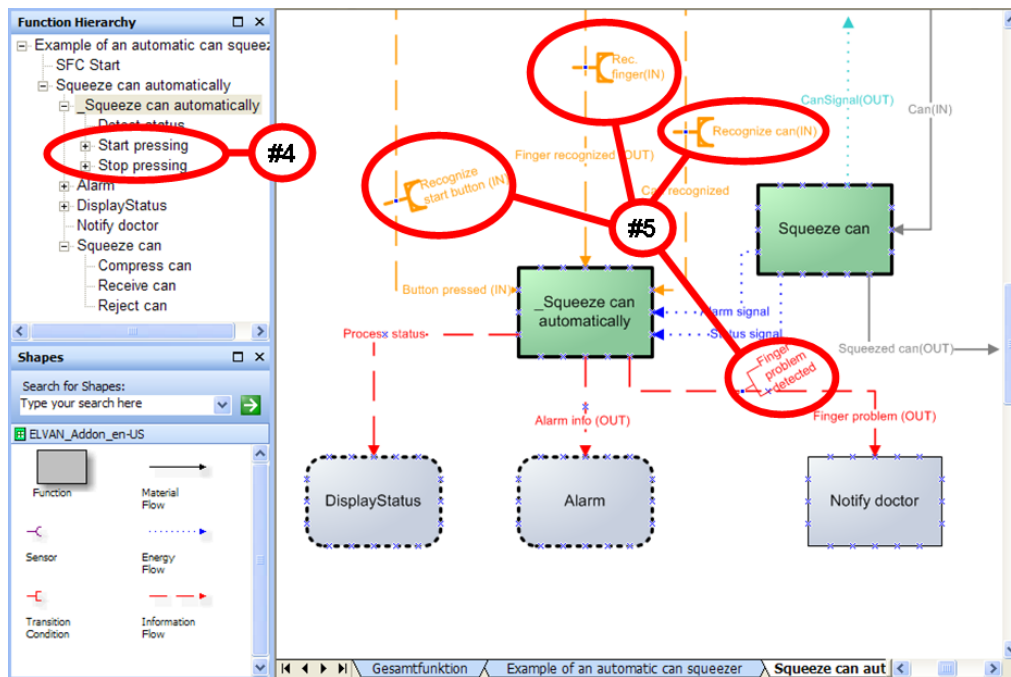


Fig. 5. After step 5: an EFS modeled in ELVAN

Finally, the EFS of the automatic can squeezer has been derived from the FOPD supporting the further system and domain-specific design according to VDI 2206 [3] as documented in [2], [8]. This implies the automatic derivation of the SFC by ELVAN supporting the further refinement of the control software by the software engineers. Furthermore, ELVAN is collecting all unspecified functions in a morphologic table supporting the elaboration of suitable solutions. In addition, the heuristic rules of Stone [11], [12] can be applied on the EFS in order to derive physical modules of the electro-mechanical subsystem (without software).

VI. CONCLUSION

This paper presents a predefined process module with five steps that embeds a FOPD in the VDI guideline 2206 and demonstrates the cooperation between FOPD and EFS. Requirements, variants and functions demanded by the customer that are contained in a FOPD are serving as a base when applying the steps and thus enables the derivation of an EFS that describes the mechatronic functions from a technical perspective. The EFS has already been embedded in the V-model of the VDI 2206 as documented in [13].

The benefit of the work presented in this paper is the straightforward transition from a FOPD into an EFS. Thus all functional requirements of all product variants are connected to the final product functions and their further refinement performed in the domain-specific design. In addition, the applicability of the new process module has been exemplified.

REFERENCES

[1] D. P. Politze and S. Dierssen. *A functional model for the function oriented description of customer-related functions of high variant products*. In Proceedings of NordDesign'08, Tallinn, Estonia, August 2008.

[2] J. Bathelt. *Entwicklungsmethodik für SPS-gesteuerte mechatronische Systeme*. PhD thesis, ETH Zürich, 2006.

[3] Verein Deutscher Ingenieure (VDI). *Entwicklungs-Methodik für mechatronische Systeme, VDI 2206*. Beuth Verlag, Berlin, 2004.

[4] D. P. Politze. *Experiences on Including Variability Information in a Function Oriented Product Documentation*. In J. Gausemeier, F. J. Rammig, and W. Schäfer, editors, *Self-optimizing Mechatronic Systems: Design the Future*, HNI-Verlagsschriftenreihe, Paderborn, pages 53–63. Heinz Nixdorf Institut, Heinz Nixdorf Institut, Feb. 2008.

[5] G. Pahl and W. Beitz. *Konstruktionslehre: Methoden und Anwendung*. Springer, 7th edition, 2007.

[6] A. Eisenhut. *Service Driven Design: Konzepte und Hilfsmittel zur informationstechnischen Kopplung von Service und Entwicklung auf der Basis moderner Kommunikations-Technologien*. PhD thesis, ETH Zürich, 1999.

[7] J. Hirtz, R. Stone, D. McAdams, S. Szykman, and K. Wood. *A functional basis for engineering design: Reconciling and evolving previous efforts*. *Research in Engineering Design*, 13(2):65–82, 2002.

[8] J. Bathelt and J. Meile. *Computer Aided Methods Supporting Concurrent Engineering when designing Mechatronic Systems Controlled by a PLC*. In Proceedings of International Conference on Manufacturing Automation (ICMA07), Singapore, May 2007.

[9] International Electrotechnical Commission (IEC). *Programmable controllers - Part 3: Programming languages, IEC 61131-3*. Beuth Verlag, Berlin, 2003.

[10] J. Gausemeier and J. Lückel. *Entwicklungsumgebungen Mechatronik - Methoden und Werkzeuge zur Entwicklung mechatronischer Systeme*. HNI-Verlagsschriftenreihe, Paderborn, Band 80, Heinz Nixdorf Institut, Universität Paderborn 2000.

[11] R. Stone. *Towards a Theory of Modular Design*. PhD thesis, University of Texas at Austin, 1997.

[12] R. Stone and K. Wood and R. Crawford. *A Heuristic Method to Identify Modules from a Functional Description of a Product*. In Proceedings of the 1998 ASME Design Engineering Technical Conferences, Atlanta, September 1998.

[13] J. Bathelt and A. Jönsson and C. Bacs and S. Dierssen and M. Meier. *Applying the new VDI Design Guideline 2206 on Mechatronic Systems Controlled by a PLC*. In Proceedings of International Conference on Engineerin Design (ICED), Melbourne, Australia, August 2005.