INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED O3 STOCKHOLM, AUGUST 19-21, 2003

GENERATING PRODUCT ARCHITECTURES USING PARTITIONING ALGORITHMS

Berislav Gaso and Kevin Otto

Abstract

Modularity is a critical determination for effective product architecture. We present here a computational approach that implements the dominant flow and branching flow heuristics developed for subjectively partitioning a functional diagram into modules. We partition by representing the flow connectivity, and compute the matrix that contains only the functions that are connected by a particular flow. That becomes a candidate for a dominant flow heuristic.

Keywords: Modularization, product architecture, function structure, partitioning algorithm

1 Introduction

Determining product architecture is one of the key activities of any industrial product development activity, and can substantially simplify the delivery of product derivatives and updates, thereby reducing time to market. The goal is to define the product architecture as a working set of sub-systems, properly partitioned for maximum value – complete with robust interfaces capable of the variety necessary to support multiple product variants and their evolution over time. Every product line has a product architecture, but whether the architecture is well planned for the supported markets, supply and delivery chains, and the development process is not so clear. Often an argument presented against modular design is the increase in cost and size that can occur. However, focusing on a product's part cost will not maximize value in the long term, especially when considering ability to offer multiple product variants. Also, a modular design need not be bulky if designed well; laptop PCs testify to this fact.

The ease with which a new core technology can be introduced into a product line as a revolutionary or evolutionary change is directly a function of the product architecture. The ease of product development depends upon how the systems were designed to evolve as a portfolio of products. A well-defined architecture permits rapid incorporation of new technologies, styling changes, or size and capacity changes with trends. Subsets of the product need only be changed, decreasing development cycles. A poorly understood architecture back-propagates design changes throughout the product forcing extensive efforts and delays. This is compounded into changes the production, distribution, sales and service functions, further degrading the customer's experience and extending delays.

This is not to say businesses do not manage their product portfolios. Product derivatives are often explored with business cases and user interviews and the like. Pipeline management tools are used to select a proper mix of projects on risk-value charts. Such portfolio management tools, however, remain distinct and separate from the technical decision making

on modularity, interfaces, capacities and size ranges of subsystems, and upgradability. Poor technical architectural decisions cause severe problems, often attributed to other causes.

[1] noted the most often cited cause of product failures was inadequate *detailed* market studies, though quite interestingly, the same surveys showed that preliminary assessments of the markets had been done. In other words, the marketing plans were done, but the fundamental detailed systems architecture was not compared against the market. [2] also reported a study demonstrating the disconnect between business strategy and daily operations of new product development. Their surveys showed that engineers doing the product development often do not understand how to leverage a company's available technologies into their projects, and that strategists often make misguided decisions on what to develop. Again, this is a demonstration of poor systems architecting – failures to understand how to evolve a product line to adopt new technologies and failures to understand how to match evolving customer requirements with the product line strategic plans. Clausing's 10 cash drains [3] of new product development include technology push, disregard of the customer, and the eureka concept, which are all failures due to no real portfolio architecting being done.

These failures have many causes. But the fundamental cause we see is an incorrect focus on purely technical design issues to meet the communicated product requirements when forming the system architecture. The result is products that, when technically when examined alone and not against the future evolving competition or the future evolving market, function well as an engineered system with the given components. The problem is they just don't stack up in the evolved market against competitive technology, the architecture was too rigid to accommodate necessary changes to keep pace.

A successful way that many industries have exploited to offer this needed variety while reducing the need for resources is to launch *product families* based on a common *platform*. To do this effectively, a well determined product architecture is required – a means to partition the product functionality and systems into separate and distinct modules. Yet, few computational approaches exist to help in this endeavor [4]. In this paper, we develop and demonstrate a computational structure to calculate a partitioning basis for a function structure that is consistent with modularization rules, and maximizes a partitioning metric.

1.1 Related work

Most related to the work here are efforts done by researchers using the design structure matrix (DSM) technique. [5] and [6] have analyzed the sequence of and the technical relationships among many design tasks in complex design projects in order to identify design parameter groupings that must be solved iteratively. [7] focus on using decomposition to structure tasks and parameters in the detail design stage. [8] also use binary interactions represented in a digraph to develop physical design layouts. [9] try to use the DSM for product decompositions to identify reasonable modules. [10] also use DSM techniques to modularize product architectures. All these techniques build on Steward's design structure matrix [5] and Warfield's early article on binary matrices in system modeling [11].

1.2 Background – EML Structures and Modularization Rules

In order to combine the concept of modularization metrics with the concept of function structures, more modeling structure than simple functions are required. Functions in the structure must be equipped with added variables, and further mathematics to combine these individual values into grouped aggregates. This paper will use a formal and very generic modeling language that was posed by [12]. This modeling language – in the following also referred to as the *Element Modeling Language* (EML) – is based on system-theoretic

fundamentals and is made of only a few formal components that can be adapted flexibly. The basis of the formal modeling language is the system definition as depicted in Figure 1. The system definition is structured into the following four statements: (1) A system consists of elements, (2) the elements have attributes (properties and functions), (3) elements are interacting by means of relations and (4) an element can be a system. From this basic definition, the components of the modeling language (formal components) can be derived [12], [13].

Modularization rules were first proposed by Stone [14] and further developed by [15] and [16]. The dominant will be analytically computed in this work. The dominant flow rule examines each non-branching flow of a function structure and groups the sub-functions the flow travels through until it exits the system or is transformed into another flow. The branching flow rule first requires an identification of flows that split into parallel branches. Each branch of the flow defines a potential module.

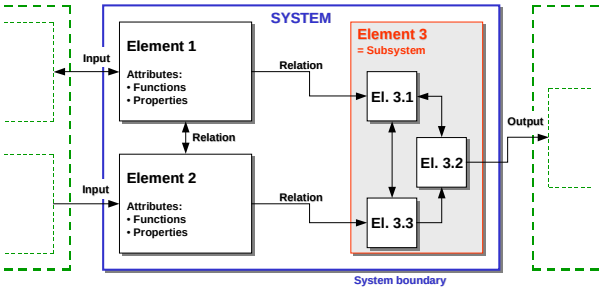


Figure 1. System Definition and Formal Components of EML (Igenbergs, 1993)

2 Matrix representation

This section introduces a system of binary and non-binary matrices that is capable of unequivocally representing an EML structure. First, the binary square *Element-Element-Matrix* (**EEM**) captures all general connections between two elements. The **EEM** does not take into account by how many flows two elements are connected. It simply notices that there is some sort of connection between two elements. This **EEM** is defined as follows:

$$\mathbf{EEM} = \left(a_{mn}\right)_{\mathrm{i}\times\mathrm{j}}, \text{ where } a_{mn} = \begin{cases} 1 & \text{if linked and } \mathrm{m} \neq \mathrm{n} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The *i* rows and *j* columns represent all elements $i = j$ from an EML-structure. For example, a mark or "1" in row 2, column 1 means that element 2 is connected through some at this point unknown number of flows with element 1 (element two provides information to element one – in other words, the arrowhead is at element one).

The binary *Element-Relation-Matrix* (**ERM**) unequivocally identifies the number of flows by which two elements are connected. To translate an EML-structure in a reasonable way into a system of matrices all the relations need to be numbered, whereby a branching flow is counted only as one relation. The *i* rows of the **ERM** represent the *i* elements and the *k* columns the *k* numbered relations from an EML-structure. Hence, the **ERM** can be stated as:

$$\mathbf{ERM} = \left(b_{mn}\right)_{\mathrm{i}\times\mathrm{k}}, \text{ where } b_{mn} = \begin{cases} 1 \text{ if linked} \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

subject to

$$\sum_{m=1}^{i} b_{mn} \leq 2 \quad \wedge \quad \left\{n \middle| n \in (1,\ldots,k)\right\} \quad \text{for non - branching flows,} \tag{3a}$$

or

$$\sum_{m=1}^{i} b_{mn} > 2 \quad \wedge \quad \left\{n \middle| n \in (1,\ldots,k)\right\} \quad \text{for branching flows.} \tag{3b}$$

Since the **ERM** matrix does neither capture the input nor the output flows through the system, two additional transposed vectors, which will provide this information, are introduced. The *k* components of these vectors equal exactly the total number of relations *k* within the EML-structure. Thus, the input vector is stated as

$$\vec{in}^{T} = (i_{11}, \quad i_{12}, \quad \cdots \quad i_{1k}), \text{ where } i_{1k} = \begin{cases} 1 \text{ if flow is input into system} \\ 0 \text{ otherwise.} \end{cases} \tag{4}$$

and the output vector is defined as:

$$\overrightarrow{out}^{T} = (o_{11}, \quad o_{12}, \quad \cdots \quad o_{1k}), \text{ where } o_{1k} = \begin{cases} 1 \text{ if flow is output from system} \\ 0 \text{ otherwise.} \end{cases} \tag{5}$$

The **ERM** together with the transposed input and output vectors does not provide information about which flow types connect two elements; rather it shows by how many flows two elements are connected. The next translation step clearly identifies the flow type of each numbered relation. The *Relationtype-Relation-Matrix* (**RRM**) identifies the flow types and consists of *l* rows, representing all *l* occurring flow types within an EML-structure, and of *k* columns, representing the total number *k* of relations within an EML-structure.

$$\mathbf{RRM} = \left(c_{mn}\right)_{\mathrm{l}\times\mathrm{k}}, \text{ where } c_{mn} = \begin{cases} 1 \text{ if linked} \\ 0 \text{ otherwise.} \end{cases} \tag{6}$$

Since each relation consists only of one flow type, the **RRM** is subject to

$$\sum_{n=1}^{k} c_{mn} = 1 \quad \forall \quad m = 1,\ldots,l \tag{7}$$

Lastly, the non-binary *Element-Property-Matrix* (**EPM**) introduces a representation of element properties or respectively metrics is. The **EPM** exhibits values of metrics or any other defined characteristics of an element:

$$\mathbf{EPM} = \left(\nabla_{mn}\right)_{i\times o} \tag{8}$$

Figure 2 summarizes the entire system of matrices as a representation, and its isomorphic relation to an EML-structure (which recall is a function structure equipped with added variables).

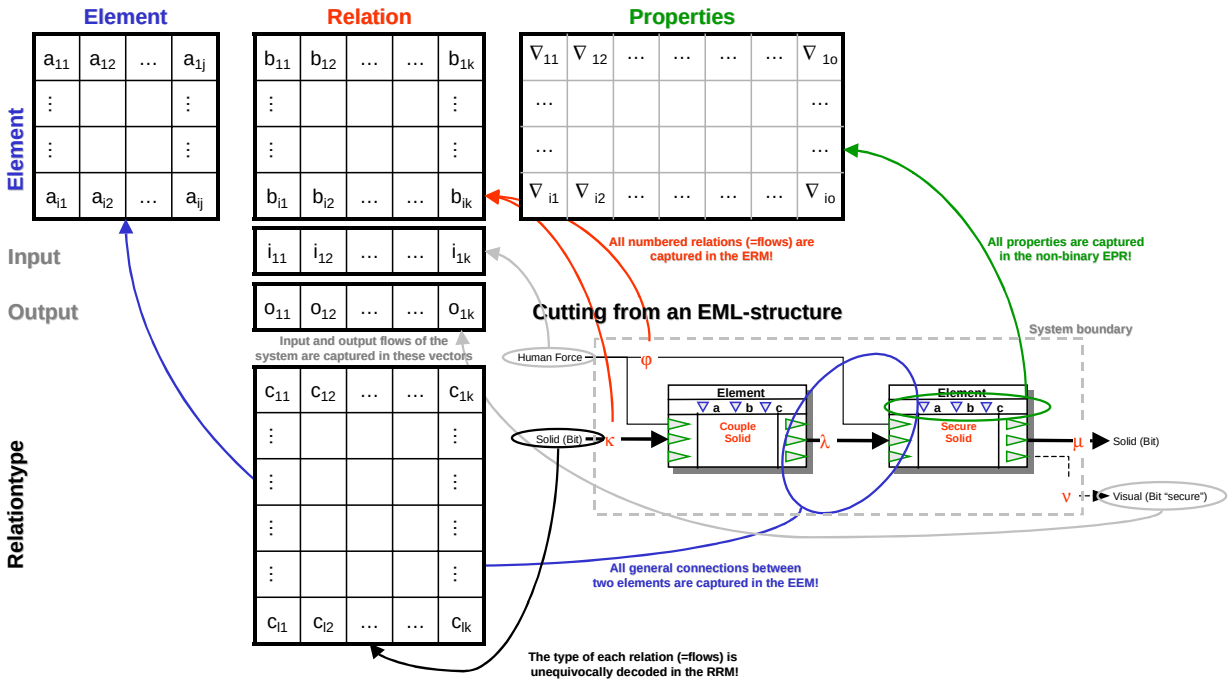


Figure 2. Matrix Modularization System

3 Algorithms

This section will describe the dominant flow partitioning algorithm for the *Matrix Modularization System* (MMS) which is based on the modularization heuristic [14]. For a detailed description of the branching flow algorithm the authors refer to [17]. Then module indices will be introduced as fulfilment-goodness indicators for the prior defined metrics. They will later be used for trade-off studies between different modular architecture candidates.

3.1 Generating Candidate Modules – Dominant Flow Algorithm

The dominant flow rule examines each non-branching flow of an EML-structure and groups the sub-functions that the flow travels through until it exits the system or is transformed into another flow. The identified set of sub-functions defines a module that deals with the flow traced through the system. The following algorithms allows the precise identification of such dominant flow modules within MMS.

Step 1. Since this section is dealing with non-ranked flow algorithms, the following procedure must be applied to every flow within a MMS. Start by selecting the first row from the **RRM**, which represents one of the flows. This selected row is called the filter vector $\vec{\mathbf{v}}_{\text{filter, k}}$. Note that it sometimes makes sense to trace several flows, when considering grouping, for example, all material or liquid flows together in one module. In this case, the filter vector $\vec{\mathbf{v}}_{\text{filter}}$ is constructed by summing up all the desired rows to an overall filter vector:

$$\vec{\mathbf{v}}^{\,T}_{\text{overall–filter, k}} = \vec{\mathbf{v}}^{\,T}_{m_1 n} + \vec{\mathbf{v}}^{\,T}_{m_2 n} + \ldots + \vec{\mathbf{v}}^{\,T}_{m_p n} \quad \forall \quad n = 1, \ldots, k \quad \wedge \quad \{m \,|\, m \in (1, \ldots, l)\}. \tag{9}$$

Step 2. Before continuing, one needs to check whether the dominant flow rule is applicable to the selected flow, depending upon whether a flow is non-branching. The check is done by calculating the vector $\vec{\mathbf{v}}_{\Sigma(\text{Columns ERM}),\,k}$ as follows:

$$\mathbf{ERM}^{T} \cdot \vec{\mathbf{v}}_{\text{all elements},\,k} = \vec{\mathbf{v}}_{\Sigma(\text{Columns ERM}),\,k}, \text{ where } \vec{\mathbf{v}}_{\text{all elements},\,k} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \tag{10}$$

Thereby, the vector $\vec{\mathbf{v}}_{\Sigma(\text{Columns ERM}),\,k}$ contains the sum of each column 1…*k* in **ERM**. With this, one needs to check which sum-components of $\vec{\mathbf{v}}_{\Sigma(\text{Columns ERM}),\,k}$ are caused by the selected filter vector $\vec{\mathbf{v}}_{\text{filter},\,k}$. Therefore, the purged column vector $\vec{\mathbf{v}}_{\text{purged }\Sigma(\text{Columns ERM}),\,k}$ is calculated by using Boolean multiplication as follows:

$$\vec{\mathbf{v}}_{\text{purged }\Sigma(\text{Columns ERM}),\,k} = \vec{\mathbf{v}}_{\text{filter},\,k} \otimes \vec{\mathbf{v}}_{\Sigma(\text{Columns ERM}),\,k} \tag{11}$$

Once this is done, one only needs to check the remaining sum-components sc_k of $\vec{\mathbf{v}}_{\text{purged }\Sigma(\text{Columns ERM}),\,k}$ according to inequality (3b), which in this case can be restated as:

$$sc_n > 2 \quad \wedge \quad \{n \mid n \in (1,\ldots,k)\} \quad \text{for branching flows.} \tag{12}$$

If (12) is true for one of the vector-components sc_k of

$$\vec{\mathbf{v}}_{\text{purged }\Sigma(\text{Columns ERM}),\,k} = \begin{pmatrix} sc_1 \\ \vdots \\ sc_k \end{pmatrix} \tag{13}$$

the dominant flow algorithm is not applicable on the selected (branching) flow. On the other hand, if the sum-components sc_k of the vector satisfy inequality (3a), which in this case can be stated as

$$sc_n \le 2 \quad \wedge \quad \{n \mid n \in (1,\ldots,k)\} \quad \text{for non - branching flows,} \tag{14}$$

then the dominant flow rule can be applied on the selected flow and one can continue with Step 3.

Step 3. Here one identifies all elements in the EML-structure which are affected by the selected filter vector. By carrying out the operation:

$$\mathbf{ERM} \otimes \vec{\mathbf{v}}_{\text{filter}} = \vec{\mathbf{v}}_{\text{elements}} = \begin{pmatrix} e_1 \\ \vdots \\ e_i \end{pmatrix}, \text{ where } e_i = \begin{cases} 1 & \text{if flow goes through element} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

using Boolean multiplication and addition, one obtains as result the element vector $\vec{\mathbf{v}}_{\text{elements}}$. This vector contains all elements which are influenced by the chosen filter option. Note that the vector $\vec{\mathbf{v}}_{\text{elements}}$ does not provide information whether all identified elements are connected through (only) one or several separated flows. This is examined in Step 5.

Step 4. This step purges the **EEM** of entries that are not relevant for further considerations. First rows and columns of the Element-Element-Matrix (**EEM**) are summed up as follows:

Find all $m = 1 \ldots i$ where $\sum_{n=1}^{j} a_{mn} > 1$. For these rows m, consider each of the indices n for $a_{mn} =$ 1 of the corresponding row m and continue as follows: Take the **ERM** and check for each n whether there is really a connection between two elements as follows: If

$$\sum_{o=1}^{k} (b_{mo} + b_{no}) = 2 \quad \wedge \quad c_{lo} = 1, \text{ where1 corresponds to the chosen } \mathbf{RRM} \text{ filter–vector row} \tag{17}$$

is true, then a connection exists and the entry in the **EEM** is correct, otherwise it is wrong and a_{mn} is set to 0.

Next, find all $n = 1 \ldots j$ where $\sum_{m=1}^{i} a_{mn} > 1$. For these columns n, consider each of the indices m for a_{mn}=1 and continue as follows: Take the **ERM** and verify for each m according to equation (17) whether there is really a connection between two elements. If (17) is true a connection exists and the entry in the **EEM** is correct, otherwise a_{mn} is set equal 0.

Step 5. Once the **EEM** is purged, the rationalizing procedure (adapted from [11] and [12]) presented in this step allows determination of how many separate dominant flow modules are extractable from a given EML-structure.

We define the top-level elements T:={t_i} as the rows of **EEM** that are filled only with zeros. The elements represented by the top-level are the last elements that are part of the respective dominant flow searched. We define the bottom-level elements B:={b_j} as the columns of **EEM** that are filled only with zeros. The elements represented by the bottom-level are the first elements that are part of the respective dominant flow searched.

An element is called isolated $I := \{t_i = b_j\}$ if both the *s*th row (*i*=*s*) and the *s*th column (*j*=*s*) are only filled with zeros. An isolated element can be understood as the trivial solution of the dominant flow algorithm. It's simply only one element with (at least) one input and/or output that was chosen by the filter vector. The solution vector for one isolated element equals to:

$$S^{I,DF} = \begin{pmatrix} e_1 \\ \vdots \\ e_i \end{pmatrix}, \text{ where } e_i = \begin{cases} 1 & \text{if element is isolated} \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

If we define for a given flow *l* (filter vector) the # of isolated elements, which are extractable from an EML-structure as *s*, the solution space of all isolated elements can be stated as:

$$SS^{I,DF} = \left\lfloor S_{1,s}^{I,DF}; \ldots; S_{l,s}^{I,DF} \right\rfloor \quad \wedge \quad s \le i \tag{19}$$

These isolated elements have to be deleted from the **EEM**.

The next step identifies the (separate) dominant flow modules represented by the **EEM**. Since the purged **EEM** consists of only regular rows and columns (they contains only a single 1), the general procedure for forming each dominant flow will be to eliminate such regular rows and columns from **EEM**, while simultaneously making additions to the element structure that represents each dominant flow.

To extract a dominant flow, one has to investigate the bottom-level elements b_j of **EEM** successively. This is done by taking the first bottom-level element b_{j*} and reconstructing its connection to the successor as follows: Suppose first, that there is at least one regular row,

row *i=j**, and its lone 1 shows subordination to column *j*. Form a new column vector for the *j*th column as follows: The new column vector is

$$C_j^* = C_j \cdot \overline{C_i} \tag{20}$$

where C_j is the old column vector for the *j*th column and $\overline{C_i}$ is the complement of C_i. Insert the new column vector in the matrix where column *j* was. (In the case that there is no column *i*, let $C_j^* = C_j$.) Delete the *i*th row from the matrix and delete any row or column that is filled with zeros. Place the element *i* on the forming EML-structure and connect it to the bottom-level element b_j. If the element *i* is a top-level element t_i then a module is identified and the procedure can be repeated for the next bottom-level element. If the element *i* is not a top-level element continue with the elimination of the (next) row *i*=j*. Hence, the solution vector for one dominant flow module can be stated as follows:

$$S^{DF} = \begin{pmatrix} e_{1,u} \\ \vdots \\ e_{i,u} \end{pmatrix}, \text{where } e_i = \begin{cases} 1 & \text{if element is on the dominant flow} \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

and the index *u* denotes the sequence-number of an element within a dominant flow module. If we define for a given flow *l* (filter vector) the number of dominant flow modules which are extractable from an EML-structure as *t*, then the solution space of all dominant flow modules can be stated as:

$$SS^{DF} = \left[S_{1,t}^{DF}; \ldots; S_{l,t}^{DF} \right] \tag{22}$$

Hence, the dominant flow overall solution space SSDF consists of the following partial solutions:

$$SS_{DF} = SS^{I,DF} + SS^{DF} = \left[S_{1,s}^{I,DF}; \ldots; S_{l,s}^{I,DF}; S_{1,t}^{DF}; \ldots; S_{l,t}^{DF} \right] \quad \wedge \quad s \le i \tag{23}$$

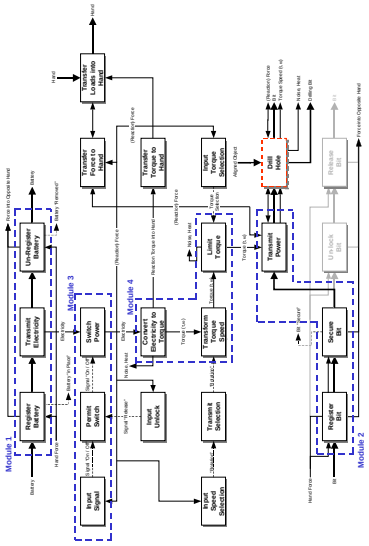
4 Sorting Modules

Customer preferences present an opportunity for application of the algorithm above. One can map the customer needs to the flows that provide them, and thereby prioritize which flows to modularize first. That means, that elements (functions), which were included e.g. in the module(s) for the highest ranked flow cannot be part of a module for lower-ranked flows, etc.

Once a flow ranking is established, the solution space for the product architecture is partitioned and reduced. This is done by first applying the dominant flow algorithm to the most important flow on the given EML-structure. Thereby elements, which were included e.g. in the module(s) for the highest ranked flow cannot be part of a module for lower-ranked flows, etc. This can be done iteratively, with subsequent algorithm applications applied to the remaining un-modularized product functions, until a comprehensive modularization is completed. Table 1 shows rankings for the function structure of a Versapak Cordless Drill found in [17] and [18]. Notice the most important flow, the batteries which power the drill, is modularized first, followed by the bit flows, followed by the electrical flow. The resulting modules are consistent with the actual Versapak product architecture.

Table 1: Modules defined by iterative application of the algorithm to a cordless drill

Module	Flow	Functions
Module 1	Battery flow	Register battery, transmit electricity, unregister battery
Module 2	Bit Flow	Register bit, secure bit, transmit power
Module 3	Electrical Flow	Input signal, permit switch, switch power
Module 4	Torque Flow	Convert electricity to torque, transform torque speed, limit torque



5 Conclusion and Outlook

A more advanced method of applying the ranking algorithms would be to rank the flows in different ways in order to establish different partitionings of the same product architecture. Such architectures could later be compared against each other in order to find the one, which best fits the given customer requirements. This means that, for example, customers could first rank the flows based on their specific needs and then according to specific objectives, which should be satisfied by the product-to-be. For example, such a ranking could mainly focus on user-functions or on reliability. In order to enable a comparison and a tradeoff between those different partitionings, one needs module metrics, which can be used as an underlying decision basis.

In this paper, we presented a computational approach that implements the dominant flow and branching flow heuristics developed for subjectively partitioning a functional diagram into modules. We partitioned by representing the flow connectivity, and computed the matrix that contained only functions that were connected by a particular flow. That became a candidate for a dominant flow heuristic.

6 Acknowledgements

The research reported in this document was made possible in part by the MIT Center for Innovation in Product Development under NSF Cooperative Agreement Number EEC-9529140. Any opinions, findings, or recommendations are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Cooper Robert (1993):

[2] PDC/Lucent, "Benchmarking Study", Best Practices Report, Vol. 6 (10), 1999.

[3] Clausing, D., "Total Quality Development: A Step-By-Step Guide to World-Class Concurrent Engineering", ASME Press, 1994.

[4] Gaso B. et al., "A Generic Methodology for Partitioning Product Architectures", 12th International Symposium of INCOSE, Las Vegas, Nevada, 2002.

[5] Steward D.V., "The Design Structure System: A Method for managing the Design of Complex Systems", IEEE Transaction on Engineering Management, Vol. EM-28, No. 3, 1981, pp. 71-75.

[6] Eppinger S.D. et al., "A Model-Based Method for Organizing Tasks in Product Development", Research in Engineering Design Vol. 6, 1994, pp.1-13.

[7] Kusiak A. and Wang J., "Decomposition of the Design Process", Journal of Mechanical Design **115,** 1993, pp. 687-695.

[8] Kusiak A. and Szczerbicki E., "Transformation from conceptual to Embodiment Design", IIE Transactions, Vol. 25, No. 25, 1993, pp. 6-12.

[9] Pimmler T. and Eppinger S., "Integration Analysis of Product Decompositions", Proceedings of the ASME 6th Int. Conference on DTM, 1994, Minneapolis, MN.

[10] Baldwin C.Y. and Clark K.B., "Design Rules – The Power of Modularity", MIT Press, Cambridge, Massachusetts, 2000.

[11] Warfield J. N., "Binary Matrices in System Modeling", IEEE Transaction on System, Man, and Cybernetics, Vol. SMC-3, No. 5, 1973, pp. 441-449.

[12] Igenbergs E., "Systems Engineering", Course Material, TU Munich, 1993.

[13] Negele H.; Fricke E. and Igenbergs E., "ZOPH – A systematic Approach to the modeling of Product Development Systems", 7th International Symposium of INCOSE, Los Angeles, California, 1997.

[14] Stone R.B., "Towards a Theory of Modular Design", PhD thesis, The University of Texas at Austin, 1997.

[15] McAdams D.A.; Stone R.B. and Wood K.L., "Understanding Product Similiarity Using Customer Needs", 1998 ASME Design Engineering Technical Conferences, DTM-5660.

[16] Zamirowski E. and Otto K.N., "Identifying Product Portfolio Architecture Modularity Using Function and Variety Heuristics", 1999 ASME Design Engineering Technical Conferences, DTM-8760.

[17] Gaso B., "Modular Product Architectures – A Method for Partitioning Platforms", MIT Working Paper, 2001.

[18] Otto K.N. and Wood K.L., "Product Design", Prentice Hall, Englewood Cliffs, 2000.

Corresponding Author:
Berislav Gaso
University of St. Gallen
Institute of Technology Management
Unterstrasse 22
CH-9000 St. Gallen
Switzerland
Tel: +41-71-2282475
Fax: +41-71-228-2455
E-mail: berislav.gaso@unisg.ch