



# Generation of a function-component-parameter multi-domain matrix from structured textual function specifications

T. Wilschut<sup>1</sup> · L. F. P. Etman<sup>1</sup> · J. E. Rooda<sup>1</sup> · J. A. Vogel<sup>2</sup>

Received: 29 May 2017 / Revised: 8 November 2017 / Accepted: 7 February 2018 / Published online: 17 February 2018  
© The Author(s) 2018. This article is an open access publication

## Abstract

This study introduces a method to build a multi-domain matrix (MDM), visualizing the intended architecture of a system within the component, function, and parameter domains. The MDM is generated from textual function specifications that are subject to a specific grammatical structure and vocabulary based upon the functional basis and interaction basis as presented in the literature. Two types of functions are distinguished: functions specifying what functionality a particular component provides to another component, and functions specifying the internal working (transformation of flow) of a particular component. The fixed grammar for the specification of the two types of functions allows for the automated derivation of dependencies between components, between functions of components, and between system parameters. A case study on a navigation lock demonstrates that the system architecture generated from function specifications matches the architecture of the real lock system fairly well. As such the method can be used in the early design phase to reveal the product architecture that is embodied in the function specifications of system components. The method may also support modeling of high-definition DSMs of existing engineering systems.

**Keywords** System architecture · Linguistic function specification · Design structure matrix · DSM · Multi-domain matrix · MDM · Functional basis · Interaction basis

## 1 Introduction

In recent years, many researchers have advocated the importance of system architecture design in the early development phases; see, e.g., Pahl and Beitz (2013), Tilstra et al. (2012), and Eggert (2005). Their rationale is supported by, for example, Simpson (2004) and Jiao et al. (2007), who state that a system's architecture has a profound impact on

its performance and flexibility, i.e., the quality of the delivered functionality and the ease with which the system can be modified to fulfill future requirements. Both properties are important in modern, highly competitive and rapidly changing industries (Alizon et al. 2007).

Ulrich (1995) defined system architecture as the mapping of a system's functions to the physical components within the system, and the dependencies between those components. A well-known concept in modeling and analyzing those component dependencies is the dependency structure matrix (DSM) (Steward 1981), also referred to as design structure matrix (Eppinger and Browning 2012).

A DSM is a binary  $N \times N$  matrix denoting the dependencies among  $N$  system components. Figure 1a shows an example DSM. An off-diagonal-shaded square denotes a dependency between component  $i$  and component  $j$ . A DSM may be symmetric, denoting that dependencies between components  $i$  and  $j$  are undirected, or asymmetric, denoting that dependencies between components  $i$  and  $j$  are directed.

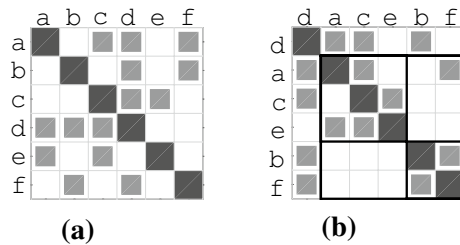
One may distinguish product, organization, and process type of DSMs (Eppinger and Browning 2012). Each type of DSM shows the dependencies between elements from

---

✉ T. Wilschut  
t.wilschut@tue.nl  
L. F. P. Etman  
l.f.p.etman@tue.nl  
J. E. Rooda  
j.e.rooda@tue.nl  
J. A. Vogel  
han.vogel@rws.nl

<sup>1</sup> Department of Mechanical Engineering, Eindhoven University of Technology, 5600 MB, PO Box 513, Eindhoven, The Netherlands

<sup>2</sup> Grote Projecten en Onderhoud, Rijkswaterstaat, Utrecht 3526 LA, The Netherlands

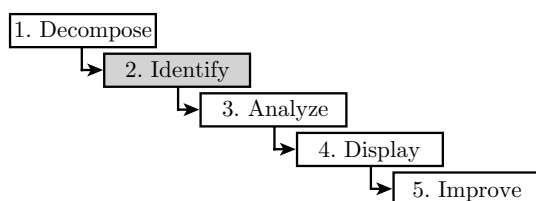


**Fig. 1** **a** Example DSM, unclustered, showing component dependencies. **b** Example DSM, clustered, revealing a bus cluster and two modular clusters (reprinted from Wilschut et al. 2016)

a different domain. These DSMs may be jointly presented along the diagonal of a larger matrix. Such a matrix is called a multi-domain matrix (MDM). Each pair of DSMs is connected via a domain-mapping matrix (DMM). A DMM is an  $N \times M$  matrix, showing the dependencies between the  $N$  elements from one domain and the  $M$  elements from the other domain.

Eppinger and Browning (2012) discuss DSM and MDM applications in many branches of industry, e.g., aerospace, automotive, and the semiconductor industry. DSMs are primarily used due to the compact and analytically advantageous format (Eppinger and Browning 2012). For example, one can highlight the system architecture by permuting the rows and columns of the DSM, as shown in Fig. 1b. That is, the permuted (clustered) matrix reveals an integrative or ‘bus’ module consisting of component d and two relatively independent modules consisting of components {a, c, e} and components {b, f}, respectively.

Eppinger and Browning (2012) introduced a five-step DSM method to architectural modeling and analysis, as depicted in Fig. 2. These five steps can be used to build an MDM as well. In Step 1, the system is decomposed into components. Guidelines for decomposing a system are given in Chiriac et al. (2011) and Tilstra et al. (2012). Next, component dependencies are identified and documented in Step 2. The resulting DSM is analyzed in Step 3, for example, by means of a clustering algorithm. The results are visually inspected in Step 4. Finally, Step 5 concerns the



**Fig. 2** DSM method to architectural modeling and analysis (Eppinger and Browning 2012). This work contributes to the second step: the identification of dependencies

Component (C - C) DSM		
F - C DMM	Function (F - F) DSM	
P--C DMM	P--F DMM	Parameter (P - P) DSM

**Fig. 3** Schematic multi-domain matrix (MDM)

improvement of the DSM by, for example, adding missed component dependencies or manually altering the clustering results.

Recently, Browning (2016) has presented an extensive overview of DSM research. From the overview of papers presented in this survey, it follows that the literature has primarily focused on the third step. The other steps have received less attention. In particular, the second step has received little attention.

In Step 2, interviews with system experts and/or design documentation reviews are often used to identify the dependencies between the various components (Browning 2016). For product architecture DSMs, this often means the identification of spatial, information, energy and material type of dependencies, following the seminal work of Pimmler and Eppinger (1994). However, the identification of dependencies often presents challenges. It appears to be quite labor intensive and time consuming. In the literature, typically efforts of several months are reported. For example, the  $46 \times 46$  NASA pathfinder DSM took 5 months to complete (Brady 2002); the  $64 \times 64$  Pratt and Whitney jet engine DSM 4 months (Sosa et al. 2003), and the  $84 \times 84$  Xerox printing system DSM 3 months (Suh et al. 2010). Second, identification of spatial, energy, information, or material dependencies is focused on form and not on function. Hence, not all identified dependencies may relate to a function. Most DSM studies analyze an existing product relying on design documentation and expert opinions to determine the various spatial, energy, information, and material dependencies. For systems in the early product development stage, an actual product realization or technical drawings may not be available; the DSM modeler has to rely mainly on functional descriptions of components.

This paper presents a method to automatically build a function-component-parameter MDM from functional

specifications of components, as schematically depicted in Fig. 3. The MDM consists of a component DSM, a function DSM, a parameter DSM, and three DMMs representing the mappings between the elements of the three DSMs. The MDM is automatically built from textual function specifications that are specified using a predefined vocabulary and grammatical structure. The functional basis of Stone and Wood (2000), later improved by Hirtz et al. (2002), is used as a basis for the vocabulary. The grammar is based on the well-known verb + noun representation used, for instance, by Stone and Wood (2000) and Pahl and Beitz (2013). Stone and Wood (2000) used the functional basis and verb + noun representation to create solution-free functional models of systems. In our study, the dependencies between components are of particular interest. Therefore, the presented grammar is an extension upon the verb + noun representation that enables the automated derivation of dependencies between components, between component functions, and between parameters that quantify the flows needed to realize functions. The resulting MDM describes the architecture of a system within the component, function, and parameter domains. By applying clustering analysis to the generated DSMs, the component-function-parameter system architecture is highlighted.

The outline of this article is as follows. Section 2 discusses literature regarding DSM model building and function modeling. Section 3 introduces the new method illustrated using an example problem. Section 4 presents a case study in which the presented method is used to derive dependencies between components of a navigation lock. The resulting DSM is compared with a DSM built in a previous study by Dijkstra (2016). Finally, conclusions are presented in Sect. 5.

## 2 Related work

In this section, the literature is discussed which relates to building DSM or MDMs automatically and linguistic specifications for function modeling.

### 2.1 Automated DSM generation

Dong and Whitney (2001) were one of the first to consider automated building of DSMs. In their study, a design matrix (DM) is manually constructed using axiomatic design theory (Suh 1998). The DM is an  $n \times m$  matrix which relates  $n$  functional requirements to  $m$  design parameters. An entry at position  $i, j$  in the DM denotes that functional requirement  $i$  is affected by design parameter  $j$ . Dong and Whitney (2001) used a heuristic to automatically generate a functional requirement DSM as well as a design parameter DSM by permuting the rows and columns of the DM. Their method

requires the DM to be a square matrix, i.e., the number of functional requirements is assumed to be equal to the number of design parameters. However, if one strives for a diagonal DM, the accompanying DSMs also become diagonal.

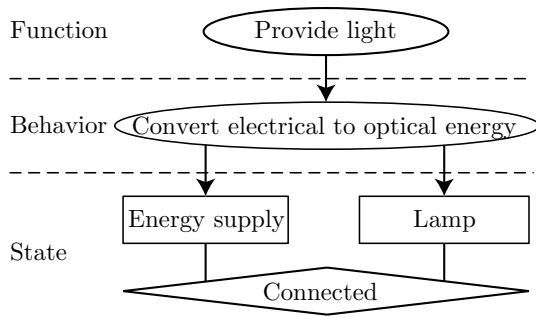
A variation on this approach was introduced by Maurer (2007) who used DMMs and DSMs assembled into a single MDM:

$$M = \begin{bmatrix} C & 0 \\ D & F \end{bmatrix} \quad (1)$$

where  $D$  is an  $n \times m$  DMM that relates  $n$  functions to  $m$  components,  $C$  is an  $m \times m$  component DSM, and  $F$  is an  $n \times n$  function DSM. An entry  $i, j$  in  $D$  denotes that function  $i$  is being (partly) fulfilled by component  $j$ .  $D$  is assumed to be constructed by hand. Subsequently, component DSM  $C$  is computed by  $C = D^T \cdot D$  and function DSM  $F$  is computed by  $F = D \cdot D^T$ . The intuition behind this is that if two components contribute to the fulfillment of the same function, there possibly is a dependency between the two components. Similarly, if two functions are being fulfilled by the same component, there possibly is a dependency between the two functions. The method yields symmetric DSMs. Note that the method yields candidate dependencies; a subset presents the actual dependencies. Manual verification of the candidate component dependencies and candidate function dependencies is required (Maurer 2007). Regarding the construction of matrix  $D$ , it may be difficult to decide whether a component contributes to fulfilling a function. There may be components that do not directly contribute, but that do have an indirect influence. Including or excluding these contributions obviously affects the calculated component and function DSMs.

As an alternative to the aforementioned matrix based methods, graphical models to derive component dependencies were proposed; see, e.g., Eisenbart et al. (2016), Wyatt et al. (2012) and D'Amelio et al. (2011). Of particular interest for this paper are Function-Behavior-State (FBS) models (Umeda et al. 1995) and variants thereof, such as structure-behavior-state models (Goel et al. 2009). Van Beek et al. (2010) used function-behavior-state models to derive DSMs. Figure 4 shows an example FBS model. A function relates to a particular physical phenomenon, i.e., a certain type of desired behavior. The behavior is realized by several entities, i.e., the components of the system. These entities should be in a certain state to realize the behavior. For the example shown in Fig. 4, the energy supply and the lamp should be connected. Though, the practical application of graphical FBS models in industry is low, since written documents are the primary means of communication and documentation in system design (Tomiyama et al. 2013).

Tosserams et al. (2010) observed the limited scalability of graphical models while specifying decomposed



**Fig. 4** Example function-behavior-state model

multi-disciplinary-optimization problems. Therefore, they developed the dedicated specification language  $\Psi$ , which in turn de Borst et al. (2016) used to model a LED-System-in-Package. Based on the  $\Psi$ -language specification, de Borst et al. (2016) automatically generated a DSM displaying the relationships between the approximately 700 response and design parameters ( $\approx 700$ ) characterizing the system.

Dori et al. (2003) developed object-process models which can be described graphically using object-process diagrams and linguistically using the object-process language implemented within the OPCAT tooling. Sharon et al. (2009) used OPCAT to derive DSMs. Their efforts focus on modeling of relationships between system components and activities within the development process. Based on the linguistic and graphical models, dependencies between activities and components are derived.

In summary, in building DSMs linguistic approaches provide better scalability and tractability than graphical approaches (Tosserams et al. 2010). Moreover, written documents are currently the primary communication and documentation means in system design (Tomiya et al. 2013). However, to the authors' knowledge, there is no method to automatically build product DSMs or MDMs based upon textual specifications. In particular, generating DSMs and MDMs from textual specifications of functions would be very powerful since component functions play an important role in system architecture design (Browning 2016). This paper presents exactly such a method. This allows designers to easily switch between text-based and DSM-based representations of the system.

## 2.2 Linguistic function specifications

In function modeling, one can take several perspectives, e.g., a system centered, a process centered or a purpose centered perspective (Erden et al. 2008). Our work aims to derive component dependencies, function dependencies, and parameter dependencies from textual function specifications. This falls into the category of the system centered modeling

perspective, which is common practice in engineering design literature (Eisenbart et al. 2012), see, for instance, Pahl and Beitz (2013) and Hubka and Eder (2012).

In the literature, one can find many definitions of function. Eisenbart et al. (2012) compared 12 different function definitions, and concluded that there are basically two perspectives in defining function. The first perspective emphasizes that a function describes some purpose, goal or requirement. This perspective is consistent with the work of Umeda et al. (1996) and the work of Fernandes and Machado (2016). The second perspective emphasizes that a function describes a transformation of flow. This perspective is used in engineering design literature, such as Pahl and Beitz (2013) and Dieter et al. (2013).

In the engineering design literature, transformation functions are usually represented by a verb and a noun, for example, 'provide power' or 'transfer torque'. These functional descriptions relate to the manipulation of flows of energy, material, and signal through the system and should be accompanied by the physical quantities (Pahl and Beitz 2013). Such a verb–noun representation is advantageous in view of generality, flexibility, and expressiveness, but has limitations with respect to rigorousness and uniqueness (Deng 2002) due to the many synonyms in natural language.

Another disadvantage of the verb–noun representation is that it does not yield complete sentences. The verb–noun phrases typically do not contain a subject. Pahl and Beitz (2013) state that in innovative design one should engage in solution-free function modeling. Therefore, they argue that a function should not contain a subject. On the other hand, solution-free function modeling rarely happens in practice (Tomiya et al. 2013). In fact, many companies tend to reuse design knowledge to decrease product development time and costs (Jiao et al. 2007).

It is not possible to arrive at a single readable and understandable document when using verb–noun phrases without a subject. Nor is it possible to derive component dependencies or function dependencies. That is, the function 'provide power' does not provide any information regarding the dependencies between components or functions. Stone and Wood (2000) use block diagrams to construct verb + noun function chains by hand.

Hirtz et al. (2002) created a functional basis consisting of a unique set of verbs and flows. The verb set can be used to improve the uniqueness of linguistic function specifications. The flow set can be used to characterize the dependencies between components. Tilstra et al. (2012) used the flow set for the creation of an interaction (dependency) basis that they used to (manually) build what they call high-definition DSMs, displaying up to 25 different dependency types. In their approach, they build a separate DSM per interaction type and subsequently assemble those DSMs into a single DSM. Tilstra et al. (2012) noted, however, that manually

building high-definition DSMs of more than ten components requires quite a large amount of work.

Our paper presents an extension upon the verb and noun function representation. The extension uses the functional basis to ensure the uniqueness of the specifications. The interaction basis is used to characterize the resulting dependencies. Our extension results in complete sentences from which dependencies between components, functions, and parameters can automatically be derived. The resulting dependencies are visualized using an MDM.

### 3 Method

This section introduces a method for the automated generation of MDMs from textual function specifications. The method is based on the function specification model depicted in Fig. 5. Function specifications consist of two basic elements: components and parameters. Components denote a specific part of the system. Parameters refer to flows through

the system, e.g., power and information, or to aspects of components, e.g., position and temperature.

In the function specifications, components and parameters are used in together with verbs and prepositions following a fixed format. A sentence specifying a function describes either a goal function or a transformation function. A goal function describes the purpose of a component with respect to the other components of the system. For example, the goal function ‘Component  $x$  provides power  $p$  to component  $y$ ’, denotes that the purpose of component  $x$  is to provide power  $p$  to component  $y$ . A transformation function describes the transformation of flow that occurs within a component. For example, the transformation function ‘Component  $x$  converts power  $p$  into torque  $k$ ’ denotes that an electrical energy flow  $p$  is converted into a mechanical energy flow  $k$  inside component  $x$ .

The method requires design engineers to manually specify the functions following the fixed format and to create a dictionary of component names, verbs, parameter types, parameter names, and prepositions that are allowed in the function specifications. Each word within a specified function is automatically referenced against the dictionary to detect typos and inconsistencies.

A compact fixed format is deliberately chosen to enforce precise and concise function specifications. That is, the quality of the specification is traded against the flexibility of the specification. Albeit the restricted specification flexibility, the proposed fixed format, defined in Fig. 5a, enables automatic processing of the goal and transformation function specifications to generate an MDM consisting of a symmetric component DSM, an asymmetric goal function DSM, an asymmetric parameter DSM, and three DMMs indicating the dependencies between components, goal functions, and parameters. By analyzing these matrices, for instance, by clustering, one can gain insight in the system architecture.

The three main steps of the method are explained in further detail below. Section 3.1 explains how goal functions are specified. Section 3.2 explains how transformation functions are specified. Section 3.3 explains how the MDM is generated from the function specifications.

#### 3.1 Step 1: specifying goal functions

Table 1 shows goal functions of a simple water storage system. Each sentence consists of the following four main elements:

1. The **1st component name** denotes the component which actively contributes to fulfilling the function. The component name must be part of the system decomposition.
2. The **verb** indicates the action that the component performs on the flow. The verbs are constrained to the verbs of the functional basis of Hirtz et al. (2002).

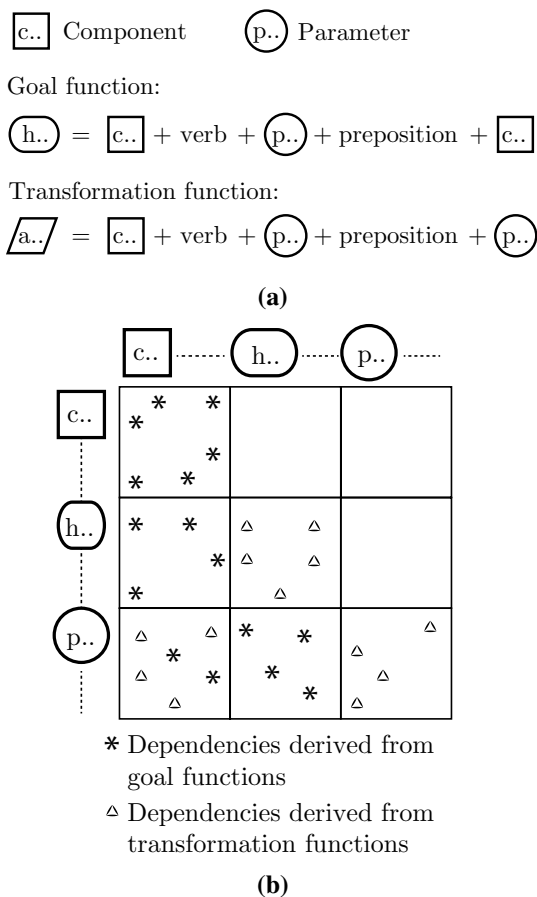


Fig. 5 a Fixed structure of goal and transformation functions. b Multi-domain matrix derived from the goal and transformation component functions



**Table 1** Goal functions of a simple water storage system

ID	1st comp. name	Verb	Parameter name	Prep.	2nd comp. name
h0	Power supply <i>po</i>	provides	power $p_e$	to	electric motor <i>el</i>
h1	Power supply <i>po</i>	provides	power $p_v$	to	control system <i>co</i>
h2	Electric motor <i>el</i>	provides	torque $k_p$	to	pump <i>pu</i>
h3	Electric motor <i>el</i>	signals	status signal $s_e$	to	control system <i>co</i>
h4	Frame <i>fr</i>	secures	position $x_e$	of	electric motor <i>el</i>
h5	Frame <i>fr</i>	secures	position $x_p$	of	pump <i>pu</i>
h6	Frame <i>fr</i>	secures	position $x_s$	of	storage container <i>st</i>
h7	Control system <i>co</i>	signals	control signal $c_e$	to	electric motor <i>el</i>
h8	Sensor <i>se</i>	measures	pressure $w_p$	in	storage container <i>st</i>
h9	Sensor <i>se</i>	signals	status signal $k_s$	to	control system <i>co</i>
h10	Pump <i>pu</i>	provides	flow $q_s$	to	storage container <i>st</i>

**Table 2** Mapping between parameters types and dependency types

Parameter type	Interaction type from dependency basis
Power	Electrical energy
Torque	Mechanical energy (dynamic)
Position	Spatial
Volume	Spatial
Pressure	Hydraulic energy
Flow	Liquid material
Status signal	Status signal
Control signal	Control signal

Note that this grammar does *not* allow the construct: ‘The electric wire *wi* connects power supply *po* with electric motor *el*’, since power supply *po* is not a parameter quantifying a flow, nor is power supply *po* an aspect of electric motor *el*. The sentence describes form and not function. The actual function of electric wire *wi* is to conduct electricity from power supply *po* to electric motor *el*. Following the goal function format, the function of the wire can be captured by the functions: ‘The power supply *po* exports power *pe* to electric wire *wi*’ and ‘Electric wire *wi* conducts power *pe* to electric motor *el*’.

3. The **parameter name** quantifies a flow between the two components (e.g., parameter  $p_e$  in function h0 of Table 1) or alternatively a particular aspect of the 2nd component (e.g., parameter  $x_e$  in function h4 of Table 1). A parameter name consists of a parameter type and an identifier. A parameter type should be an instance of one of the generic dependency types defined in the dependency basis of Tilstra et al. (2012). For example, see Table 2.
4. The **2nd component name** denotes the component that is passively involved in fulfilling the function. The component name must be part of the system decomposition.

### 3.2 Step 2: specifying transformation functions

Table 3 shows transformation functions of the water storage system example. Each sentence consists of the following 4 main elements.

1. The **component name** denotes the component which actively contributes to fulfilling the function. The component must be part of the system decomposition.
2. The **verb** indicates the action that the component performs on the flow(s). The verbs are constrained to the functional basis of Hirtz et al. (2002).

**Table 3** Transformation functions of a simple water storage system

ID	Comp. name	Verb	1st parameter name(s)	Prep.	2nd parameter name(s)
a0	Electric motor <i>el</i>	converts	power $p_p$	into	torque $k_p$
a1	Electric motor <i>el</i>	converts	control signal $c_e$	into	status signal $s_e$
a2	Control system <i>co</i>	processes	status signal $w_s$ and status signal $s_e$	into	control signal $c_e$
a3	Control system <i>co</i>	converts	power $p_e$	into	control signal $c_e$
a3	Control system <i>co</i>	adds	control signal $c_e$	to	log $l_o$
a4	Sensor <i>se</i>	converts	pressure $w_p$	into	status signal $w_s$
a5	Pump <i>pu</i>	converts	torque $k_p$	into	flow $q_s$
a6	Storage container <i>st</i>	accumulates	flow $q_s$	into	volume $w_v$

3. The **1st parameter name(s)** denotes the required input flow(s) for the action performed by the component. An action may require multiple inputs. The inputs are separated by ‘and’.
4. The **2nd parameter name(s)** denotes the resulting output flow(s) of the action performed by the component. An action may yield multiple outputs. The outputs are separated by ‘and’.

The parameter names may be part of the set of parameters resulting from the specification of goal functions. The modeler should take care to ensure consistency in parameter names.

### 3.3 Step 3: automatically building a MDM

The MDM shown in Fig. 6 is automatically built from the goal and transformation function specifications listed in Tables 1 and 3, respectively. The DSMs in the MDM are clustered separately. For the water storage example problem and the case problem presented in Sect. 4, we used Markov clustering (Wilschut et al. 2016, 2017), but any suitable clustering algorithm may be used.

The goal functions represent dependencies between components (C–C) and mapping relations between goal functions and components (F–C), between parameters and components (P–C), and between parameters and goal functions parameters (P–F). In the following it is explained which dependencies and mapping relations are derived from goal functions.

Let  $G$  be the set of all specified goal functions  $g$ , where  $g$  is defined as the tuple:

$$g := (c_{g,1}, v_g, p_g, q_g, c_{g,2}) \tag{2}$$

wherein  $c_{g,1}$  is the first component,  $v_g$  is the verb,  $p_g$  is the parameter,  $q_g$  is the preposition, and  $c_{g,2}$  is the second component. Let  $C = \bigcup_{v_g \in G} c_{g,1}, c_{g,2}$  be the set of all components used in the goal functions. Then, component  $c_i$  and component  $c_j$  are dependent if:

$$\exists g \in G : c_{g,1} = c_i \wedge c_{g,2} = c_j \tag{3}$$

evaluates true. That is, there exists a goal function  $g$  in the set of a goal functions  $G$ , such that the first component  $c_{g,1}$  in  $g$  equals  $c_i$  and the 2nd component  $c_{g,2}$  in  $g$  equals  $c_j$ .

The derived dependency between components  $c_i$  and  $c_j$  is characterized by parameter  $p_g$ , which is defined as the tuple:

$$p_g := (n_{p_g}, t_{p_g}, b_{p_g}) \tag{4}$$

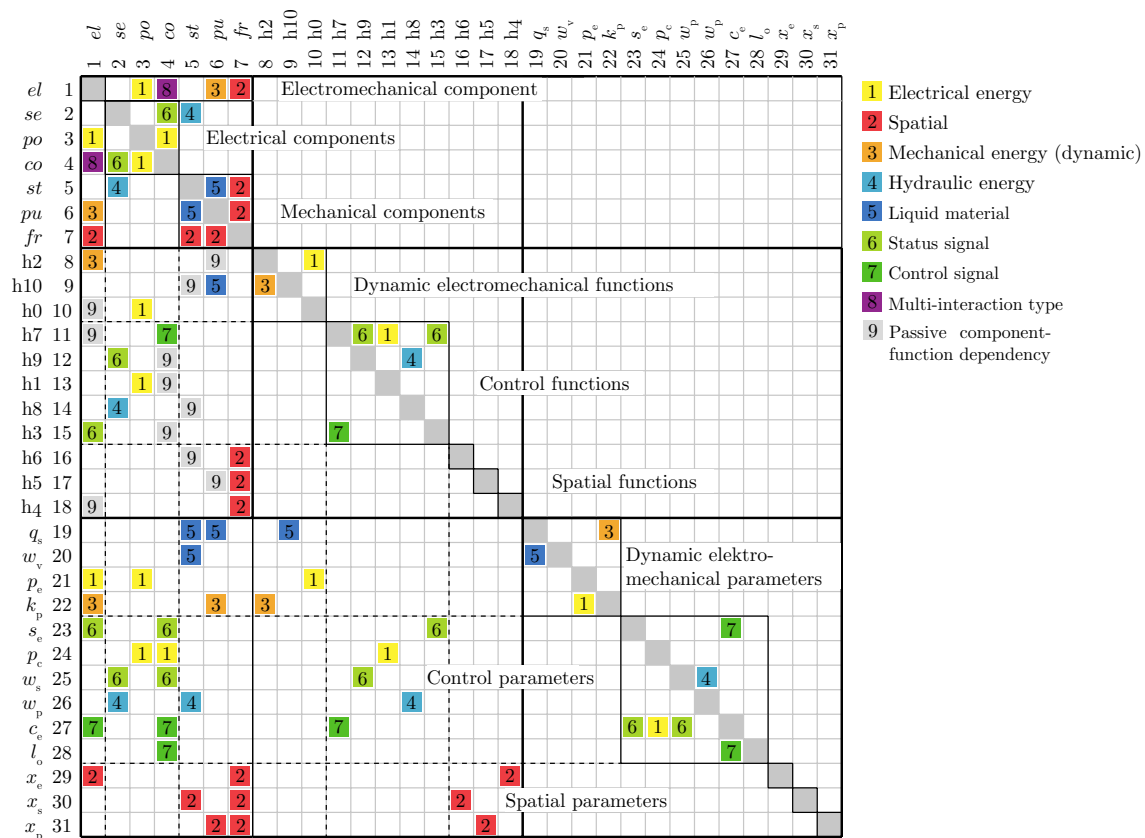


Fig. 6 Multi-domain-matrix generated from the function specifications listed in Tables 1 and 3

wherein  $n_{p_g}$  is the name of parameter  $p_g$ ,  $t_{p_g}$  is the type of parameter  $p_g$ , and  $b_{p_g}$  is the type of the dependency basis of Tilstra et al. (2012) to which type  $t_{p_g}$  belongs. For instance, in the example problem, the parameter type power belongs to the electrical energy type of the dependency basis, as shown in Table 2.

The derived dependency between components  $c_i$  and  $c_j$  is assumed to be bidirectional, as it is common practice to model symmetric product architectures (Browning 2016). That is, if Eq. (3) evaluates true, the dependency entries  $(c_i, c_j)$  and  $(c_j, c_i)$  are placed in a symmetric component–component (C–C) DSM, and are assigned dependency type  $b_{p_g}$ . Each dependency type is represented by a distinct number and color in the DSM.

Furthermore, goal function  $g$  is used to derive mapping relations between goal functions and components. That is, a goal function  $g$  has a mapping relation with component  $c$  if:

$$c_{g,1} = c \vee c_{g,2} = c \quad (5)$$

evaluates true. That is,  $g$  has a mapping relation with  $c$  if the first or second component of  $g$  is equal to component  $c$ . These relations are placed in an asymmetric goal function–component (F–C) DMM, where a relation between  $g$  and its first component  $c_{g,1}$  (entry  $(g, c_{g,1})$ ) is marked as an active relation and assigned type  $b_{p_g}$  and a relation between  $g$  and its second component  $c_{g,2}$  (entry  $(g, c_{g,2})$ ) is marked as being a passive relation indicated by a number 9 and gray color.

Similarly, mapping relations between parameters and components are derived from goal functions. That is, a parameter  $p$  has a mapping relation with component  $c$  if:

$$\exists g \in G : (c_{g,1} = c \vee c_{g,2} = c) \wedge p_g = p \quad (6)$$

evaluates true. That is, parameter  $p$  has a mapping relation with component  $c$  if there exists a goal function  $g$  such the first or second component in  $g$  equals  $c$  and parameter  $p_g$  in  $g$  equals  $p$ . This relation is placed in an asymmetric parameter–component (P–C) DMM (entry  $(p, c)$ ) and is assigned type  $b_{p_g}$ .

Finally, goal functions denote mapping relations between parameters and goal functions. A parameter  $p$  has a mapping relation with goal function  $g$  if:

$$p_g = p \quad (7)$$

evaluates true. Parameter–goal function mapping relations are placed in an asymmetric parameter–goal function DMM and is assigned type  $b_{p_g}$ .

Transformation functions represent dependencies between parameters (P–P), between goal functions (F–F),

and mapping relations between components and parameters (P–C). In the following, it is explained which dependencies and mapping relations are derived from transformation functions.

Let  $T$  be the set of all specified transformation functions  $t$ , where  $t$  is defined as the tuple:

$$t = (c_t, v_t, P_{t,1}, q_t, P_{t,2}) \quad (8)$$

wherein,  $c_t \in C$  is the component within  $t$ ,  $v_t$  is the verb,  $P_{t,1}$  is the first set of parameters,  $q_t$  is the preposition, and  $P_{t,2}$  is the second parameter set. Then, parameter  $p_i$  depends on parameter  $p_j$  if:

$$\exists t \in T : p_i \in P_{t,2} \wedge p_j \in P_{t,1} \quad (9)$$

evaluates true. That is, parameter  $p_i$  depends on parameter  $p_j$  if there exists a transformation function  $t$ , such that  $p_i$  is a member of the second parameter set  $P_{t,2}$  and  $p_j$  is a member of the first parameter set  $P_{t,1}$ .

Parameter dependency entry  $(p_i, p_j)$  is placed in an asymmetric parameter–parameter (P–P) DSM and is assigned the dependency type  $b_{p_j}$  of the column (input) parameter  $p_j$ . Furthermore, transformation functions are used to derive goal function dependencies. That is, goal function  $g_i$  depends on goal function  $g_j$  if:

$$\exists t \in T : p_{g_i} \in P_{t,2} \wedge p_{g_j} \in P_{t,1} \quad (10)$$

evaluates true. That is, goal function  $g_i$  depends on goal function  $g_j$  if there exists a transformation function  $t$ , such that the parameter  $p_{g_i}$  is a member of the second parameter set  $P_{t,2}$  and parameter  $p_{g_j}$  is a member of the first parameter set  $P_{t,1}$ .

Goal function dependency entry  $(g_i, g_j)$  is placed in an asymmetric goal function–goal function DSM and is assigned dependency type  $b_{p_{g_j}}$  of parameter  $p_{g_j}$  of column (input) goal function  $g_j$ .

Finally, parameter–component mapping relations are derived from transformation functions. That is, parameter  $p$  has a mapping relation with component  $c$  if:

$$\exists t \in T : c_t = c \wedge p \in P_{t,1} \cup P_{t,2} \quad (11)$$

evaluates true. That is, parameter  $p$  has a mapping relation with component  $c$  if there exists a transformation function  $t$  in which  $c_t$  equals  $c$  and  $p$  is a member of the union of the first and second parameter sets of  $t$ ,  $P_{t,1}$ , and  $P_{t,2}$ , respectively. Parameter–component mapping relations are placed in an asymmetric parameter–component (P–C) DMM and are assigned dependency type  $b_p$ .

For the water storage system example, the goal functions shown in Table 1 yield: (1) dependencies between components, displayed in the C–C DSM of the MDM shown Fig. 6 (rows 1–7, columns 1–7); (2) between components and goal functions, displayed in the F–C DMM (rows 8–18, columns



1–7); (3) between components and parameters, in the P–C DMM (rows 19–31, columns 1–7); and (4) between goal functions and parameters, shown in the P–F DMM (rows 19–31, columns 8–18). For example, goal function h0: ‘Power supply  $po$  provides electrical power  $p_e$  to electric motor  $el$ ’, represents an electrical energy-type dependency between the power supply  $po$  and the electric motor  $el$ . The dependency is taken to be bidirectional, yielding two symmetrically placed, entries (3, 1) and (1, 3), in the component DSM. The type of the dependency originates from the relation between the parameter type ‘power’ and the dependency basis of Tilstra et al. (2012), as listed in Table 2.

Second, h0 yields two dependencies in the F–C DMM. Power supply  $po$  actively contributes to fulfilling h0 and electric motor  $el$  passively contributes to fulfilling h0: the yellow entry (10, 3) labeled with ‘1’ in Fig. 6 indicates that the power supply  $po$  performs an action on an electrical flow; the gray entry (10, 1) labeled with ‘9’ indicates that electric motor  $el$  passively contributes to fulfilling function h0.

Third, h0 yields two dependencies, (21, 1) and (21, 3), in the P–C DMM, since power supply  $po$  and electric motor  $el$  both relate to parameter  $p_e$ .

Finally, h0 describes a dependency between h0 and parameter  $p_e$ , resulting in a single dependency (21, 10) in the P–F DMM. The label ‘1’ (yellow) indicates that parameter  $p_e$  represents an electric energy flow.

The transformation functions of Table 3 are used to derive: (1) dependencies between parameters, shown in the parameter DSM (rows 19–31, columns 19–31); (2) dependencies between goal functions, shown in the function DSM (rows 8–18, columns 8–18); and (3) dependencies between components and parameters, shown in the P–C DMM (rows 19–31, columns 1–7). For example, function a0: ‘Electric motor  $el$  converts electrical power  $p_e$  into torque  $k_p$ ’, denotes a dependency between electrical power  $p_e$  and torque  $k_p$ . This dependency is taken to be directed, yielding a single dependency, entry (22, 21) in the parameter DSM directed from  $p_e$  to  $k_p$ . Moreover, the directed dependency between parameters  $p_e$  and  $k_p$  implies a directed dependency between goal functions h0 and h2 since power supply  $po$  needs to provide power  $p_e$  to electric motor  $el$ , before electric motor  $el$  can provide torque  $k_p$  to pump  $pu$ . As such, a0 yields a single dependency, entry (8, 10) in the goal function DSM directed from h0 to h2. Additionally, electric motor  $el$  interacts with all parameters used in a0, yielding two entries (21, 1) and (22, 1) in the P–C DMM.

### 3.4 Results and discussion

The component DSM in Fig. 6 shows that the system consists of a cluster of mechanical components, consisting of the frame  $fr$ , pump  $pu$ , and storage container  $st$ , and a cluster of electrical components, consisting of the control

system  $co$ , the power supply  $po$ , and the sensor  $se$ . The two clusters are connected via the ‘bus’ consisting of the electric motor  $el$ .

Note that the generated component DSM only displays intended dependencies between components, i.e., dependencies that follow from function specifications. Unintended dependencies, that may result from physical phenomena such as heat generation, are not displayed by the DSM.

The function and parameter DSMs both show a cluster of dynamic electro-mechanical functions/parameters, a cluster of control functions/parameters, and three independent spatial functions/parameters. Since the function DSM is directed one can identify function chains. For example, one can identify the function chain  $h0 \rightarrow h2 \rightarrow h10$ , which indicates that power supply  $po$  provides power  $p_e$  to electric motor  $el$  (h0), electric motor  $el$  converts power  $p_e$  into torque  $k_p$  and provides torque  $k_p$  to pump  $pu$  (h0), and pump  $pu$  converts torque  $k_p$  into water flow  $q_s$ , which is provided to storage container  $st$  (h10). Similarly, the parameter DSM indicates flow chains. For example, one can identify the chain  $p_e \rightarrow k_p \rightarrow q_s \rightarrow w_v$ , indicating the flow chain electrical energy  $\rightarrow$  mechanical energy  $\rightarrow$  liquid material  $\rightarrow$  liquid material. The function DSM could be analyzed with a sequencing algorithm to find the optimal function cycle within the system. By sequencing the parameter DSM one can find an optimal sequence to set parameter values during the design process.

The three DMMs clearly show the mapping between the component, function, and parameter clusters. Each component cluster fulfills a specific set of functions which are characterized by a specific set of parameters. Moreover, the DMMs indicate which functions and parameters cross the boundary of a component cluster. This is relevant information during the design process. For example, parameter  $w_p$  relates to both the electrical and the mechanical component clusters. The water pressure  $w_p$  depends on the height of the storage container  $st$ . As such, changing the height of storage container  $st$  influences water pressure  $w_p$ . In turn, this may result in the need for a different sensor and/or a different control strategy.

Summarizing, the proposed method aims to improve the uniqueness and clarity of function specifications by using a fixed grammar and vocabulary. The fixed grammar enables the automated construction of an MDM. The MDM can provide valuable and structured information regarding dependencies between system components, functions, and parameters. By relating parameter types to the dependency basis of Tilstra et al. (2012), a variety of distinct dependency types can be derived from the function specifications and visualized using the MDM. This, reduces the required effort to construct high-definition DSMs.

## 4 Case study: navigation lock Sambeek

To validate the proposed method for a larger case problem, a component DSM developed in a previous study by Dijkstra (2016) is compared with a component DSM of the same system generated following the method described in the previous section. The major difference with the study of is that Dijkstra (2016) built the DSM model following the DSM concepts of Pimpler and Eppinger (1994), while our study departs from specifications of goal functions of the system elements to automatically generate the component DSM. Both DSMs aim at identifying and visualizing the dependencies between the components of navigation lock Sambeek, shown in Fig. 7. Through the comparison of the outcomes of the two approaches, we investigate how well the intended dependencies present in the DSM generated from the function specifications match the dependencies in the DSM of the real physical system. In other words, how well specified functions translate to realized form. All the DSMs shown in the remainder of this section were clustered separately and therefore show differences in the order of column and row labels.

### 4.1 Building and generating DSMs of lock Sambeek

Dijkstra (2016) followed the five-step DSM approach of Eppinger and Browning (2012). Dijkstra first reviewed several existing lock decompositions and proposed a decomposition consisting of 51 components for the DSM modeling. To identify interactions, Dijkstra conducted interviews and reviewed design documentation over a period of several

weeks. In this process, Dijkstra considered four types of component dependencies: (1) spatial dependencies which indicate that a dimensional change of one component implies a dimensional change of another component or that two components are physically connected to each other; (2) location dependencies which indicate that two components have a particular alignment but are not physically connected; (3) energy dependencies which indicate that energy is transferred between two components; and (4) information dependencies which indicate that information is transferred between two components. The spatial interaction includes the strain energy and proximity dependency type of interactions in the interaction basis by Tilstra et al. (2012). The location dependency type relates to the alignment type of interaction in the interaction basis. Dijkstra assumed that all dependencies are bidirectional. This resulted in DSM  $H$  (where  $H$  refers to handmade), shown in Fig. 8, displaying spatial, location, energy, and information-type dependencies.

Using the same decomposition as Dijkstra, we specified the goal functions of the various lock components. The method presented in Sect. 3.1 was used to this end. In this process, we have rewritten function descriptions in natural language into function specifications following the format defined in Sect. 3.1. For instance, one of the functions for ship lock Sambeek was formulated as

*‘The filling and emptying system serves to level the water in the chamber, containing one or more vessels, to correspond with the water level in the water way from which the ships are approaching.’ (Glerum and Vrijburcht 2000).*

Fig. 7 Navigation lock Sambeek



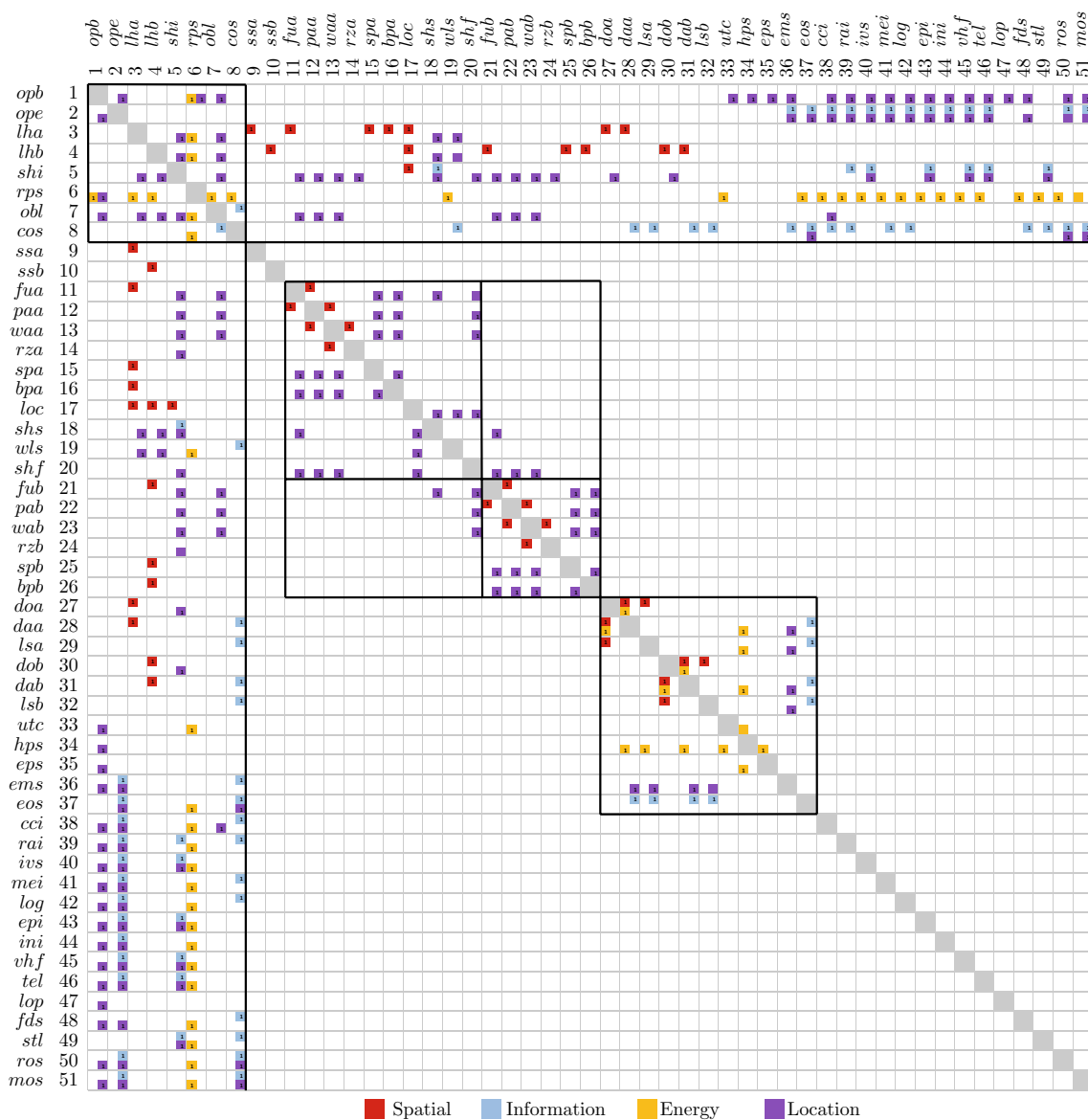


Fig. 8 Handmade DSM H (Dijkstra 2016)

which we have rewritten as the following two functions:

‘Leveling-system *lsa* imports water flow  $Q_{in}$  into lock-chamber *loc*’

‘Leveling-system *lsb* exports water flow  $Q_{out}$  from lock-chamber *loc*’

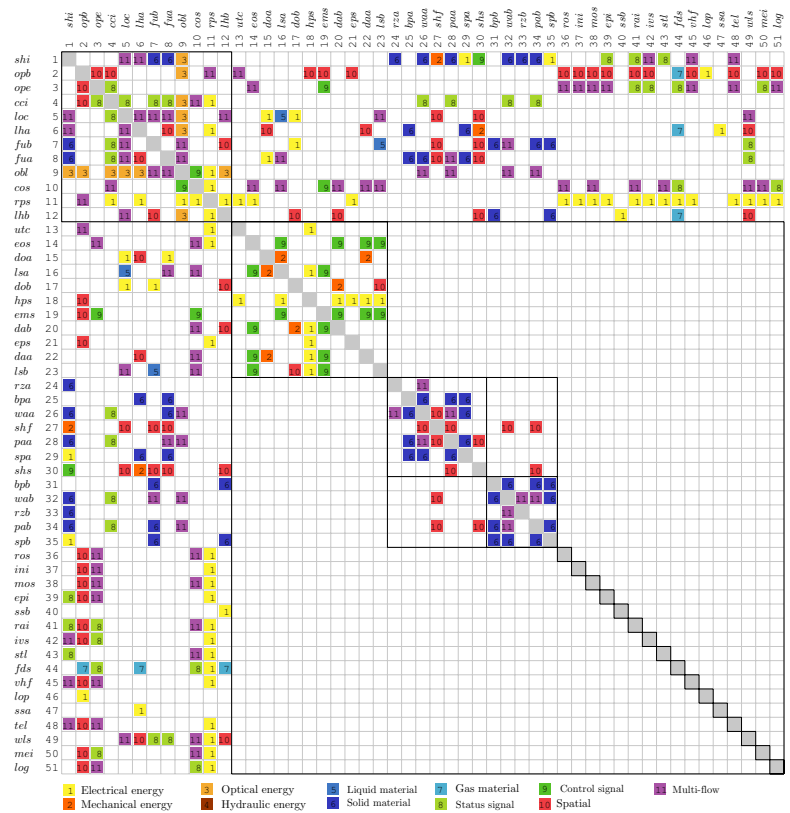
since the import and export of water is performed by two different leveling systems at Lock Sambeek. Note that our fixed sentence structure only allows to specify the actual functions of the leveling systems. Conditions to which a function may be subjected cannot be expressed.

From the function specifications, the component DSM was generated following the logic described in Sect. 3.3.

This resulted in DSM G, shown in Fig. 9, displaying ten different dependencies types. Note that the elements of the two DSMs are presented in different order due to the different clustering outcomes for the two analyses.

It took us approximately one week to complete the full process. However, having the system decomposition and reference DSM of Dijkstra at our disposal, significantly eased the specification process. Moreover, as we were involved in Dijkstra’s study, we gained extensive knowledge on the functioning of lock Sambeek prior to the specification process. Without such prior knowledge, our estimate is that the specification process would have taken several weeks to complete. For example, in the Master’s graduation project of Josten (2017), the presented method was used to describe the architecture of a

Fig. 9 Generated DSM G



bridge control system using a DSM with 45 elements. It took Josten approximately 4–5 weeks to review roughly 800 pages of design documentation (pdf and Word documents) and to manually specify the goal and transformation functions. Some verification with design engineers was needed to resolve ambiguities in the documentation as well.

**4.2 Comparing the two DSMs of lock Sambeek**

Handmade DSM *H* and generated DSM *G* are compared in two steps. The first step consists of comparing the presence of dependencies in *H* and in *G* using the  $\Sigma$ DSM concept introduced by Gorbea et al. (2008). The second step consists of comparing the types of dependencies in *H* and *G*.

In our case DSMs, *H* and *G* contain different dependency types due to the different modeling approaches. As a consequence, *H* and *G* cannot be directly merged into a  $\Sigma$ DSM. To this end, the DSMs are first converted into scalar matrices,  $\bar{H}$  and  $\bar{G}$ , respectively, where  $\bar{H}_{ij} = 1$ , denotes that component  $c_i$  and component  $c_j$  have one or more dependencies in DSM *H*; otherwise,  $\bar{H}_{ij} = 0$ . Similarly,  $\bar{G}_{ij} = 2$ , denotes that component  $c_i$  and component  $c_j$  have one or more dependencies in DSM *G*; otherwise,  $\bar{G}_{ij} = 0$ . The sum of these matrices yields the  $\Sigma$  DSM:

$$S = \bar{H} + \bar{G} \tag{12}$$

where  $S_{ij} = 1$  indicates that a dependency is only present in  $\bar{H}$ ;  $S_{ij} = 2$  indicates that a dependency is only present in  $\bar{G}$ ; and  $S_{ij} = 3$  indicates that a dependency is present in both  $\bar{H}$  and  $\bar{G}$ .

Figure 10 shows  $\Sigma$ DSM *S*. Red squares, marked with a number 1, indicate dependencies that were identified by Dijkstra (2016), but were not derived from the function specifications. Blue squares, marked with a number 2, indicate dependencies that were derived from the functions specifications but were not identified by Dijkstra (2016). Green squares, marked with a number 3, indicate dependencies that were derived from the function specifications and were identified by Dijkstra (2016).

Table 4 lists the number of red, blue, and green marks. All dependencies identified by Dijkstra (2016) could be related to a goal function, resulting in zero red marks. However, 58 (14.7%) dependencies were generated that were not identified by Dijkstra (2016). Most of the blue marks relate to liquid (water) and solid (ships) material flows through the system, which were not considered by Dijkstra (2016). Interesting are the seven blue interactions of the Close Circuit Television (CCTV) installation (cci, row 2). The top of the operating building of lock Sambeek provides a clear line of sight to all areas of the lock complex, as such Dijkstra (2016) only identified a location interaction between the CCTV installation and the operating building. Contrary, from a functional perspective the CCTV installation needs

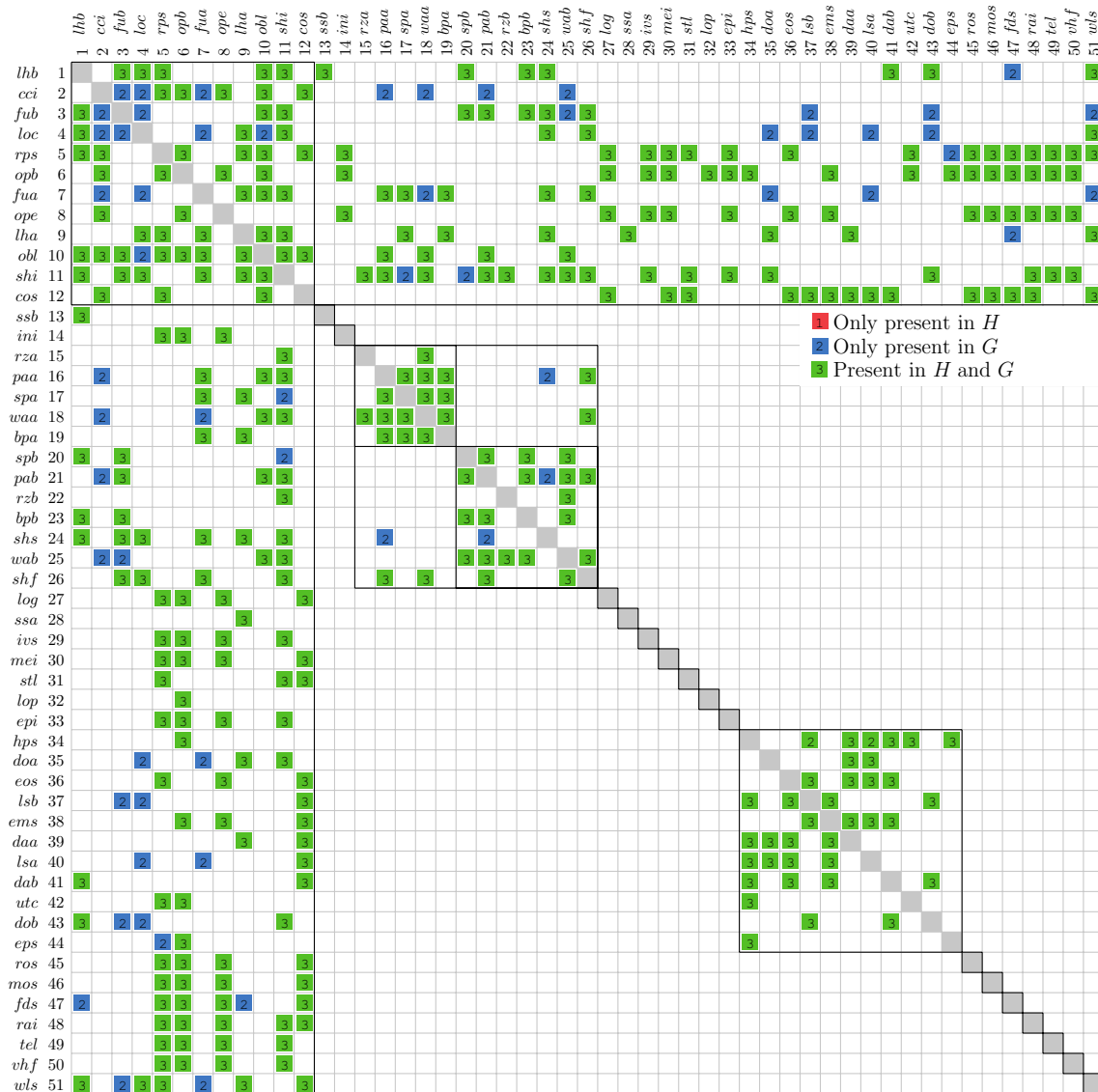


Fig. 10  $\Sigma$ DSM  $S$  showing the differences between handmade DSM  $H$  and generated DSM  $G$

Table 4 Similarity in number of dependencies

$M_H$	$M_G$	$M_T$	Color	#	%
1	0	1	Red	0	0.0
0	2	2	Blue	58	14.7
1	2	3	Green	336	85.3
Total				394	100

to cover all areas of the lock complex. As such, the function specifications yield many interactions throughout the lock complex with the CCTV installation.

Table 5 lists the number of dependencies (entries in the DSM) per dependency type for handmade DSM  $H$  and generated DSM  $G$ . Despite the differences in modeling

approach, the dependency types used in  $H$  and  $G$  do, to some extent, match: the spatial and location types in  $H$  relate to the spatial type in  $G$ ; the information type used in  $H$  relates to the status and control signal types in  $G$ ; and the energy type in  $H$  relates to the electrical, mechanical, optical, and hydraulic energy types in  $G$ ; the solid, liquid, and gaseous material flow-type dependencies in  $G$  do not have an equivalent type in  $H$ .

Note that Table 5 lists 386 dependencies in  $H$  and 494 dependencies in  $G$ , while binary DSMs  $\bar{H}$  and  $\bar{G}$  only contain 336 and 394 dependencies, respectively. In  $H$  25 components, pairs are connected via more than one dependency type, and in  $G$ , 46 component pairs are connected via more than one dependency type. As a result,  $H$  and  $G$  contain more dependencies than  $\bar{H}$  and  $\bar{G}$ , respectively.



**Table 5** Number of dependencies per dependency type in handmade DSM  $H$  and generated DSM  $G$ 

Handmade DSM $H$		Generated DSM $G$	
Type	#	Type	#
Spatial	50	Spatial	100
Location	198		
Information	82	Status signal	102
		Control signal	56
Energy	56	Electrical energy	56
		Mechanical Energy	10
		Optical energy	24
		Hydraulic energy	22
		Solid material	90
		Liquid material	26
		Gaseous material	8
Total	386	Total	494

Furthermore,  $H$  contains 50 spatial dependencies and 198 location dependencies, while  $G$  only contains a 100 spatial dependencies. This is caused by the fact that location relates to form and not to function. For example, in  $H$  object lighting,  $obl$  has multiple location interactions throughout the system, (row seven of Fig. 8), while in  $G$ , object lighting  $obl$  has multiple optical energy relations (row/column 9 of Fig. 9), since the function of object lighting  $obl$  is to provide light at various places of the lock complex. To fulfill this function in the realized system, the object lighting has to be at several locations of the lock complex. Hence, the functional optical energy dependencies turn into physical location dependencies.

Considering information dependencies,  $H$  contains 82 information dependencies, while  $G$  contains 102 status signal dependencies and 56 control signal dependencies. Two components  $i$  and  $j$  may have to interchange multiple status and control signals to fulfill their functions. For example, regular operating system  $ros$  and control system  $cos$  interchange multiple status and control signals. Those signals are exchanged over the same physical connection. Hence,  $G$  contains more information-type dependencies than  $H$ . This example shows that multiple functional dependencies may turn into a single information dependency in the DSM of the as-built (physical) system.

Looking at the energy-type dependencies in Table 5, one can see that there is an exact match between the number of energy-type dependencies in  $H$  and the electrical energy-type dependencies in  $G$ . As such, each functional electrical energy-type dependency has resulted in a physical energy-type dependency in the realized system. The mechanical and optical energy-type dependencies in  $G$  have probably been captured by Dijkstra by spatial- and location-type dependencies in  $H$ .

Dijkstra did not consider material flow-type dependencies, as a result many of the liquid, solid, and gaseous material flow-type dependencies are not included in  $H$ .

To conclude, the method presented in Sect. 3 can be used to generate an intended lock architecture which matches the realized physical architecture fairly well. The generated system architecture provides a reasonable model for the (to be realized) architecture of the physical system. Therefore, the method can be used in the early design phase to determine the functional architecture of a new system, as well as for an existing physical system to identify component and parameter dependencies (most published DSM analyses considered existing systems). On the other hand, the DSM generated from function specifications and the DSM built from data regarding the actual physical embodiment are closely related but at the same time depart from a different modeling perspective: function-based DSM modeling versus form-based DSM modeling.

## 5 Conclusions

Building Dependency Structure matrices (DSMs) are a time consuming and sometimes tedious process. This study aims to introduce a method for the intuitive, easy, and quick specification of goal and transformation functions. The function specifications are constrained to a fixed grammar and vocabulary through which we aim to increase the uniqueness and clarity of function specifications and allow for the automated construction of a multi-domain-matrix (MDM). The MDM consists of a component DSM, a function DSM, a parameter DSM, and three domain-mapping matrices (DMMs) indicating the dependencies between the elements in the three DSMs. The MDM provides valuable and structured information regarding the intended architecture of the system within the component, function, and parameter domains. In addition, by mapping parameter types onto dependencies types, a variety of distinct dependency types can be derived from the function specifications and visualized using the MDM. This reduces the required effort to construct high-definition DSMs.

Case study ‘Navigation lock Sambeek’ showed that the generated intended lock architecture matches the realized physical architecture fairly well. Therefore, the generated system architecture provides a reasonable model for the (to be realized) architecture of the physical system. As such, the method can be applied in the early design phase to gain insight in and reason about the architecture of a new system. The generated MDM provides clear insight into the dependencies inside and across the component, function and parameter domains. The method may also be used to generate a DSM of an existing system, acknowledging that one takes a functional instead of a physical DSM modeling

view. Functional specifications of components may be more straightforward to obtain than identifying spatial, material, information and energy interactions from design documents and interviews. For our lock case and the study of Josten (2017) this appeared to be the case. A reduction of modeling effort was observed.

## 6 Discussion

The method presented in this article requires function specifications to be written in a fixed structured format. Design documents are generally written using far more natural language. This means the design documentation has to be converted in function specifications according to the prescribed format. This may be quite an elaborate process. There is software tooling to automatically process natural language, see, for instance, Bird et al. (2009), which may support the conversion process. However, we have experienced that a significant amount of effort has to do with inconsistencies, errors, and incompleteness of documentation. This may, for instance, relate to the system decomposition, the naming conventions, function descriptions, graphs, and drawings. Automated language processing typically cannot help with this. The conversion process provides a means to encounter these issues an correct for them.

Design documentation usually contains far more information than function descriptions, for example, geometric aspects of a system. Therefore, in our future work we seek to extend our grammar such that non-functional aspects of systems can be described as well. For example, two components may have a spatial relation as they are placed in the same system housing. Such a dependency does not follow from just the specification of the goal and transfer functions of the two components. More information is needed there.

**Acknowledgements** Financial support for this research by Rijkswaterstaat, part of the Dutch Ministry of Infrastructure and Water Management, is gratefully acknowledged. We would like to thank Erik-Jan Houwing, Maria Angenent and Robert de Roos for their feedback and support for this work. We also thank the anonymous reviewers for their comments and suggestions, which have improved the paper.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

- Alizon F, Khadke K, Thevenot HJ, Gershenson JK, Marion TJ, Shooter SB, Simpson TW (2007) Frameworks for product family design and development. *Concurr Eng* 15(2):187–199
- Bird S, Klein E, Loper E (2009) *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media Inc, Newton
- Brady TK (2002) Utilization of dependency structure matrix analysis to assess complex project designs. In: ASME 2002 international design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers, pp 231–240
- Browning TR (2016) Design structure matrix extensions and innovations: a survey and new opportunities. *IEEE Trans Eng Manag* 63(1):27–52
- Chiriac N, Hölttä-Otto K, Lysy D, Suh ES (2011) Three approaches to complex system decomposition. In: Proceedings of the 13th international DSM conference, Cambridge, Massachusetts, USA, September 2011
- D'Amelio V, Chmarra MK, Tomiyama T (2011) Early design interference detection based on qualitative physics. *Res Eng Des* 22(4):223–243
- de Borst ECM, Etman LFP, Gielen AWJ, Hofkamp AT, Rooda JE (2016) Decomposition analysis of the multidisciplinary coupling in LED system-in-package design using a DSM and a specification language. *Struct Multidiscip Optim* 53(6):13951411
- Deng Y-M (2002) Function and behavior representation in conceptual mechanical design. *Artif Intell Eng Des Anal Manuf* 16(05):343–362
- Dieter GE, Schmidt LC, Azarm S (2013) *Engineering design*. McGraw-Hill, New York
- Dijkstra M (2016) RA risico's in een sluizenfamilie en generieke subsystemen. Technical report, Rijkswaterstaat
- Dong Q, Whitney D (2001) Designing a requirement driven product development process. In: Proceedings of the ASME international conference on design theory and methodology, Pittsburgh, PA, USA, September 2001, pp 1–11
- Dori D, Reinhartz-Berger I, Sturm A (2003) OPCAT-A bimodal case tool for object-process based system development. In: Proc. 5th international conference on enterprise information systems, Angers, France, April 2003, pp 286–291
- Eggert R (2005) *Engineering design*. Pearson/Prentice Hall, Upper Saddle River
- Eisenbart B, Blessing L, Gericke K (2012) Functional modelling perspectives across disciplines: a literature review. In: Proceedings of 12th international design conference, Dubrovnik, Croatia, May 2012
- Eisenbart B, Gericke K, Blessing LTM, McAloone TC (2016) A DSM-based framework for integrated function modelling: concept, application and evaluation. *Res Eng Des* 28(1):1–27
- Eppinger SD, Browning TR (2012) *Design structure matrix methods and applications*. MIT Press, Cambridge
- Erden MS, Komoto H, van Beek TJ, D'Amelio V, Echavarría E, Tomiyama T (2008) A review of function modeling: approaches and applications. *Artif Intell Eng Des Anal Manuf* 22(02):147–169
- Fernandes JM, Machado RJ (2016) *Requirements in engineering projects*. Lecture notes in management and industrial engineering. Springer, Cham
- Glerum A, Vrijburcht A (2000) *Design of locks 1*. Rijkswaterstaat, Utrecht
- Goel AK, Rugaber SR, Vattam S (2009) Structure, behavior, and function of complex systems: the structure, behavior, and function modeling language. *Artif Intell Eng Des Anal Manuf* 23(Special Issue 01): 23–35

- Gorbea C, Spielmannleitner T, Lindemann U, Fricke E (2008) Analysis of hybrid vehicle architectures using multiple domain matrices. In: DSM 2008: proceedings of the 10th international DSM conference, Stockholm, Sweden, November 2008
- Hirtz J, Stone RB, McAdams DA, Szykman S, Wood KL (2002) A functional basis for engineering design: reconciling and evolving previous efforts. *Res Eng Des* 13(2):65–82
- Hubka V, Eder WE (2012) Theory of technical systems: a total concept theory for engineering design. Springer, Berlin
- Jiao J, Simpson TW, Siddique Z (2007) Product family design and platform-based product development: a state-of-the-art review. *J Intell Manuf* 18(1):5–29
- Josten T (2017) Dependency Structure Matrix based analysis of the operation, control, and monitoring components of the Wantij bridge. Master's thesis, Eindhoven University of Technology, the Netherlands
- Maurer MS (2007) Structural awareness in complex product design. PhD thesis, Universität München
- Pahl G, Beitz W (2013) Engineering design: a systematic approach. Springer, Berlin
- Pimmler TU, Eppinger SD (1994) Integration analysis of product decompositions. In: Proceedings of the ASME design theory and methodology conference, Minneapolis, MN, USA, September 1994
- Sharon A, Dori D, De Weck O (2009) Model-based design structure matrix: deriving a dsm from an object-process model. In: Proc. second international symposium on engineering systems, Cambridge, Massachusetts, USA, June 2009, pp 1–12
- Simpson TW (2004) Product platform design and customization: status and promise. *Artif Intell Eng Des Anal Manuf* 18(1):3–20
- Sosa ME, Eppinger SD, Rowles CM (2003) Identifying modular and integrative systems and their impact on design team interactions. *J Mech Des* 125(2):240–252
- Steward DV (1981) The design structure system: a method for managing the design of complex systems. *IEEE Trans Eng Manag EM* 28(3):71–74
- Stone RB, Wood KL (2000) Development of a functional basis for design. *J Mech Des* 122(4):359–370
- Suh ES, Furst MR, Mihalyov KJ, de Weck O (2010) Technology infusion for complex systems: a framework and case study. *Syst Eng* 13(2):186–203
- Suh NP (1998) Axiomatic design theory for systems. *Res Eng Des* 10(4):189–209
- Tilstra AH, Seepersad CC, Wood KL (2012) A high-definition design structure matrix (HDDSM) for the quantitative assessment of product architecture. *J Eng Des* 23(10–11):767–789
- Tomiyama T, van Beek TJ, Cabrera AAA, Komoto H, D'Amelio V (2013) Making function modeling practically usable. *Artif Intell Eng Des Anal Manuf* 27(Special Issue 03): 301–309
- Tosserams S, Hofkamp AT, Etman LFP, Rooda JE (2010) A specification language for problem partitioning in decomposition-based design optimization. *Struct Multidiscip Optim* 42(5):707–723
- Ulrich K (1995) The role of product architecture in the manufacturing firm. *Res Policy* 24(3):419–440
- Umeda Y, Ishii M, Yoshioka M, Shimomura Y, Tomiyama T (1996) Supporting conceptual design based on the function-behavior-state modeler. *Artif Intell Eng Des Anal Manuf* 10(04):275–288
- Umeda Y, Tomiyama T, Yoshikawa H (1995) FBS modeling: modeling scheme of function for conceptual design. In: Proc. of the 9th int. workshop on qualitative reasoning, pp 271–8
- Van Beek TJ, Erden MS, Tomiyama T (2010) Modular design of mechatronic systems with function modeling. *Mechatronics* 20(8):850–863
- Wilschut T, Etman LFP, Rooda JE, Adan IJBF (2016) Multi-level flow-based markov clustering for design structure matrices. In: Proceedings of the ASME 2016 international design engineering technical conferences and computers and information in engineering conference, Charlotte, NC, USA, August
- Wilschut T, Etman LFP, Rooda JE, Adan IJBF (2017) Multi-level flow-based markov clustering for design structure matrices. *J Mech Des* 139(12):1–10
- Wyatt DF, Wynn DC, Jarrett JP, Clarkson PJ (2012) Supporting product architecture design using computational design synthesis with network structure constraints. *Res Eng Des* 23(1):17–52