Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Information schema constructs for instantiation and composition of system manifestation features

Shahab POURTALEBI‡, Imre HORVÁTH

(*Faculty of Industrial Design Engineering, Delft University of Technology, Landbergstraat 15, 2628 CE Delft, Zuid Holland, the Netherlands*)

E-mail: shahab60p@gmail.com; i.horvath@tudelft.nl

Received May 8, 2016; Revision accepted Oct. 7, 2016; Crosschecked Sept. 25, 2017

Abstract: Complementing our previous publications, this paper presents the information schema constructs (ISCs) that underpin the programming of specific system manifestation feature (SMF) orientated information management and composing system models. First, we briefly present (1) the general process of pre-embodiment design with SMFs, (2) the procedures of creating genotypes and phenotypes of SMFs, (3) the specific procedure of instantiation of phenotypes of SMFs, and (4) the procedure of system model management and processing. Then, the chunks of information needed for instantiation of phenotypes of SMFs are discussed, and the ISCs designed for instantiation presented. Afterwards, the information management aspects of system modeling are addressed. Methodologically, system modeling involves (1) placement of phenotypes of SMF in the modeling space, (2) combining them towards the desired architecture and operation, (3) assigning values to the parameters and checking the satisfaction of constraints, and (4) storing the system model in the SMFs-based warehouse database. The final objective of the reported research is to develop an SMFs-based toolbox to support modeling of cyber-physical systems (CPSs).

Key words: System manifestation features (SMFs); Information schema constructs; Database schemata; SMF genotypes; SMF phenotypes; SMF instances; Software tool box; System-level design; Cyber-physical systems

https://doi.org/10.1631/FITEE.1601235 **CLC number:** TP391; TP311

1 Introduction

The content of this paper complements what has been reported in our previous paper (Pourtalebi and Horváth, 2016c). This paper introduces information schema constructs for instantiation and composition of system manifestation features (SMFs) to create models of cyber-physical systems (CPSs). It was argued in the previous paper that modeling of CPSs is a great challenge due to unique characteristics (e.g., heterogeneity, complexity, dynamics, and adaptation) of these systems (Derler *et al.*, 2012). The traditional logical and analytical modeling tools have several limitations in handling these characteristics (Broman *et al.*, 2012; Zhou *et al.*, 2013). They use simplifications and abstractions at various extents and levels, respectively (Lee, 2015). They typically focus on modeling and designing one specific kind of artifact, e.g., analogue hardware, digital hardware, control software, or information structures. Consequently, they provide only sub-optimal solutions when transdisciplinary (multi-concern) modeling of CPSs is needed (Simko *et al.*, 2014). It is also argued that real progress in modeling such systems can be expected only from novel approaches that provide semantics-driven-system-level support for conceptualization and design of CPSs (Frevert *et al.*, 2005; Macal and North, 2006).

Our current research is done to contribute to system-level modeling of CPSs with underpinning theories, computational approaches, and design methodologies (Erbas *et al.*, 2007). The first milestone

‡ Corresponding author
ORCID: Shahab POURTALEBI, http://orcid.org/0000-0003-3482-5492; Imre HORVÁTH, http://orcid.org/0000-0002-6008-0570

result of our effort was the elaboration of the Mereo-Operandi theory (MOT) that provides a robust theoretical framework for the development of modeling tools (Horváth and Pourtalebi, 2015). It was followed by the complementary theory of system-level features, which was intended to extend the feature-based design paradigm to system level, and towards this end, to introduce SMFs and SMFs-based system composition as a possible approach (Pourtalebi and Horváth, 2016a). The information structures represented by SMFs have been converted into computational constructs, and the procedures of processing genotypes (GTs) and phenotypes (PTs) and deriving instances of SMFs have been discussed (Pourtalebi and Horváth, 2016b). Considering all these the basis, this paper presents the approach of SMF instance generation and system model creation.

According to the literature, the most general interpretation of constructs is that they are abstraction-based structural formations used to describe, represent, and examine phenomena of theoretical or procedural interest (Edwards and Bagozzi, 2000). In the context of digital computation, constructs encapsulate variables and their relationships to implement particular computational concepts (Lee *et al.*, 2007). They may capture input, outcome, structural, transformational, logical, knowledge, storage, interaction, interoperation, etc. concepts. These information constructs can be both static and dynamic, and determine the structure—(de)composition—of a computational or informational model (Richter, 1981).

In the context of SMFs-based modeling of CPSs, constructs have been employed to assist organization of complex information structures (Pourtalebi and Horváth, 2016c) and to support the implementation of complex database schemata. This is why we named them information schema constructs (ISCs). They are neither objects, nor concepts, but they can be a model of both. ISCs have been defined based on a semantic framework, which specifies the chunks of information needed to represent modeling concepts and their 'functional' relationships in a formal manner.

We used ISCs in our research as cognitive enablers to design the external schemata of the warehouse databases and the data input/transformation procedures. The advantages of using constructs for complex data scheme specification and computation organization tasks are that they: (1) can be derived easily from the specified unit functionalities, (2) enable handling of information in a logically structured and consistent-in-time manner (Gavrilescu *et al.*, 2010), (3) can be reused in multiple contexts, (4) provide upward compatibility, and (5) make validation of large-scale semantic information structures more transparent. We can differentiate construct transformations performed (1) within the context of a single construct (i.e., construct versioning), (2) on two or more semantically matching constructs (i.e., construct integration), and (3) on two or more semantically mismatched constructs (i.e., construct adaptation).

As semantically arranged structures of identifiers, parameters, keys, values, descriptors, and functional relations, ISCs are based on one or more physical/logical concepts (theoretical definitions). They also specify the data elements to be processed, their relationships, and the pertaining processing commands and procedures. A construct can be implemented in various programming languages. Eventually, an ISC enables the implementation of tailor-made elementary functions by defining executable code-level components or selecting off-the-self components that realize a function. Structurally, the model warehouse comprises: (1) a meta-level database unit, (2) a database management unit, and (3) a relational database unit. The abovementioned enabling potential has been used in the implementation of the warehouse database schemes of GTs and PTs. In this study, it is exploited in the realization of the meta-level database and the relational database of the model warehouse.

2 Workflow of SMF-based system modeling

2.1 Pre-embodiment design methodology

SMF-based modeling is a computational implementation of our pre-embodiment design methodology. It includes six major activities of model composition: (1) deciding on the components of the model, (2) defining the spatial positions and orientations of the components, (3) determining the functional roles of the components, (4) architectural connection of components, (5) operational connection of components, and (6) validation of the composition by checking the constraints. This seems to be a sequential workflow, but activities (2) and (3) and activities (4) and (5) are respectively done concurrently and in an interrelated manner (Fig. 1).

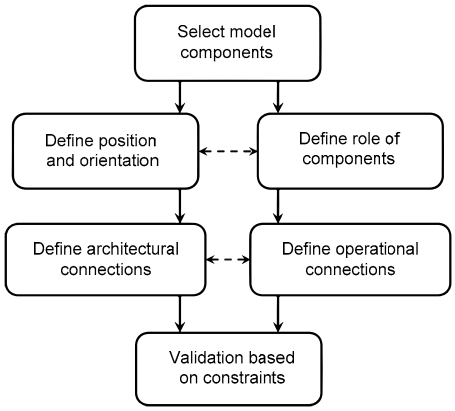

Fig. 1 Generic workflow of pre-embodiment design

The methodology also supports the mapping of pre-embodiment design activities onto formal computational procedures. In the mapping process, (1) the involved human and software actors, (2) the entry format of the necessary chunks of information, and (3) the organization scheme of the chunks of information in the warehouse are considered. Based on these factors, an all-embracing workflow of SMF-based model composition (an arranged set of computational procedures) is proposed (Fig. 2).

SMFs-based pre-embodiment design can be best characterized by the term 'cognigeering', which expresses that its intent is to simultaneously 'cognize' (obtain information/knowledge about the constituents and the whole of the planned system) and 'engineer' (plan and execute a systematic realization of the model of the whole system). SMFs determine the way of availing constituents-related information for designers, and the way of generating a system model. In the cognigeering process, designers synthesize the architecture and functionality of a system in terms of SMFs, and create a model of the system as a composition of multi-disciplinary SMFs. The SMFs-based modeling process has been decomposed to intermittent and terminal process elements. First of all, the process is divided into two sub-processes, which are called 'defining SMF entities' and 'composing CPS models'. The former sub-process includes the procedure of creating GT and the procedure of deriving PTs. The procedure of creating GT includes several cycles of activities, such as (1) basic definition of GTs, (2) operational definition of GTs, and (3) connectivity definitions of the units of operation of GTs. The latter sub-process includes the procedure of composition and instantiation of PTs and the procedure of model management. As the name implies, every cycle of activities consists of specific modeling activities, which are further decomposed to specific actions.

2.2 Workflow of pre-embodiment design

Since composition and instantiation of the elements of the system model occur at the same time,

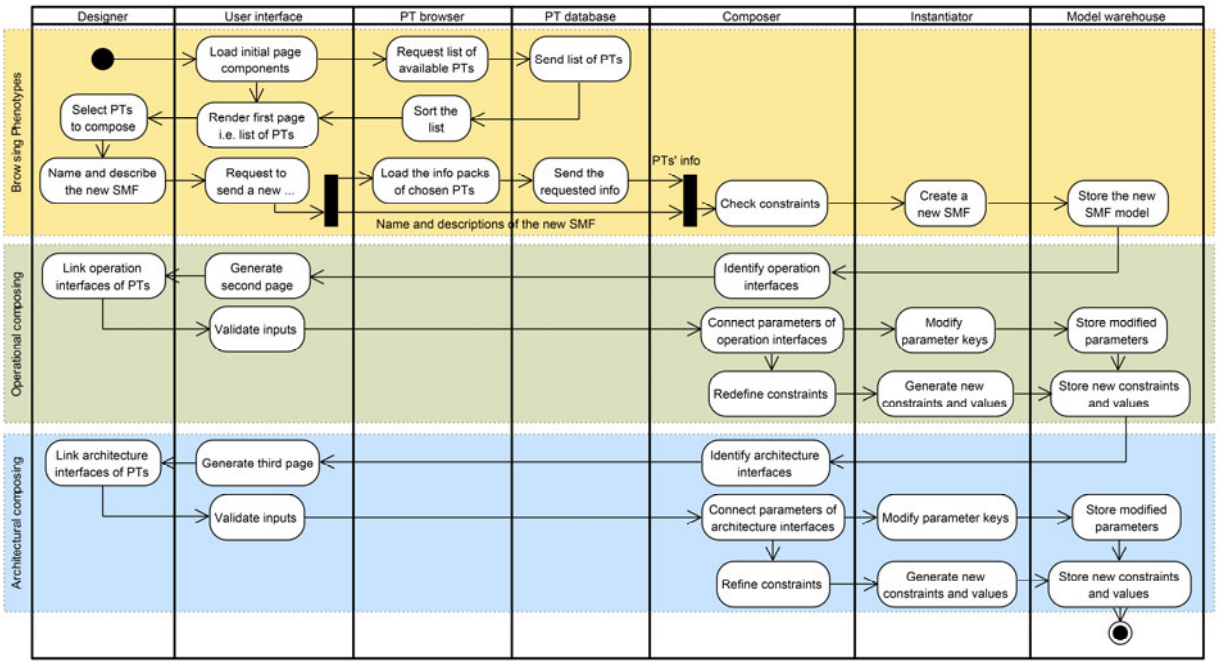

Fig. 2 Overall workflow of SMF-based model composition

there are some confined interactions between composer and instantiator modules in this process. As shown in Fig. 2, the composer module works as a front-end during the model composition process, while the instantiator module works in the background and supports the composer modules with data processing actions. The three horizontal blocks shown in Fig. 2 include the three activity cycles of model composition. The first activity cycle is concerned with browsing PTs from the PT database and storing them into the model warehouse. The second activity cycle is for establishing operational couplings among the chosen PTs. The third activity cycle is for establishing architectural couplings among the chosen PTs.

Designers of CPSs can create system models based on the information provided by knowledge engineers, who create and maintain the warehouses of the modeling entities (SMFs) of SMF-TB (Petnga and Austin, 2016). As explained above, composition of system models by SMFs starts with selection of relevant GTs and deriving PTs by the model creation tools of the platform part of SMF-TB. Genotypes are either created by the GT editor tool, or retrieved from the GT warehouse by the warehouse manager tool. PFs are either derived by the PT editor tool, or retrieved from the PT warehouse by the warehouse manager tool. Instantiation of SMFs is supported by the tools of the workbench part of SMF-TB, namely, by the instantiator, composer, and model warehouse manager modules.

2.3 Computational implementation of modeling in pre-embodiment design

The whole modeling process is physicality driven. Systematization of the modeling process is facilitated by the concepts of GTs and PTs. GTs manage the structural variations of SMFs, and PTs manage the morphological, attributive, and operational variations of SMFs. By definition, genotypes are type-level computational objects that capture the possible structural variants of SMFs in a parameterized form. They include concrete structural parameters and attributes, which are evaluated to get to concrete PTs. While GTs are structurally parameterized implements, PTs are implements that describe the morphological and attributive parameters of possible manifestations of SMFs. They are partially pre-programmed computational entities, whose final content is defined by the user selections and inputs. Rather than imposing any higher level of abstraction, GTs and PTs represent physically feasible phase models. Methodologically, a multi-stage mapping is implemented that involves: (1) creation or retrieval of SMF genotypes, (2) deriving PTs from GTs or retrieving relevant ones, (3) specification and storing their contents in libraries, (4) deriving specific instances of SMFs based on PTs, and (5) structural and operational modeling and simulation of system models of CPSs.

The procedure of composition and instantiation of PTs establishes spatial, morphological, and operational relationships between selected PTs in a pairwise manner. The composition is made possible by the interface specification of the concerned PTs. The interface and internal parameters are evaluated to make physical composition possible. The modeling software provides the mechanism by means of which PTs can be combined and matching values can be generated for the parameter variables, with the consideration of the specified constraints. The values of the instantiated PTs are stored in the model database. The interfaces of the PTs have a decisive role in terms of the possibility of architectural and operational coupling. Architecture interfaces are exemplified by contracts and morphological ports (Bhave *et al*., 2010). Operation interfaces provide specifications over external streams, and the related states and events. In addition, parameter constraints are specified by both interfaces, depending on the types of the parameters. Instances of SMFs take over some chunks of information and their relationships from the relational data tables of phenotypes. Instances share parts of the model database.

3 ICSs for structuring the model database

In our previous publication (Pourtalebi and Horváth, 2016c), the ISCs needed to map genotype and phenotype information into relational database schema have been explained. In this paper we present those additional constructs that are needed for mapping the results of composition and instantiation of phenotypes in the system model. It is worth mentioning that: (1) a strictly physical view is enforced in SMFs-based modeling, (2) the highest aggregation level of SMFs is actually the system model, and (3) aggregation concerns both the architecture and the operations of the system.

Fig. 3 presents an overview of the external schema of the model warehouse. The PTs of SMFs

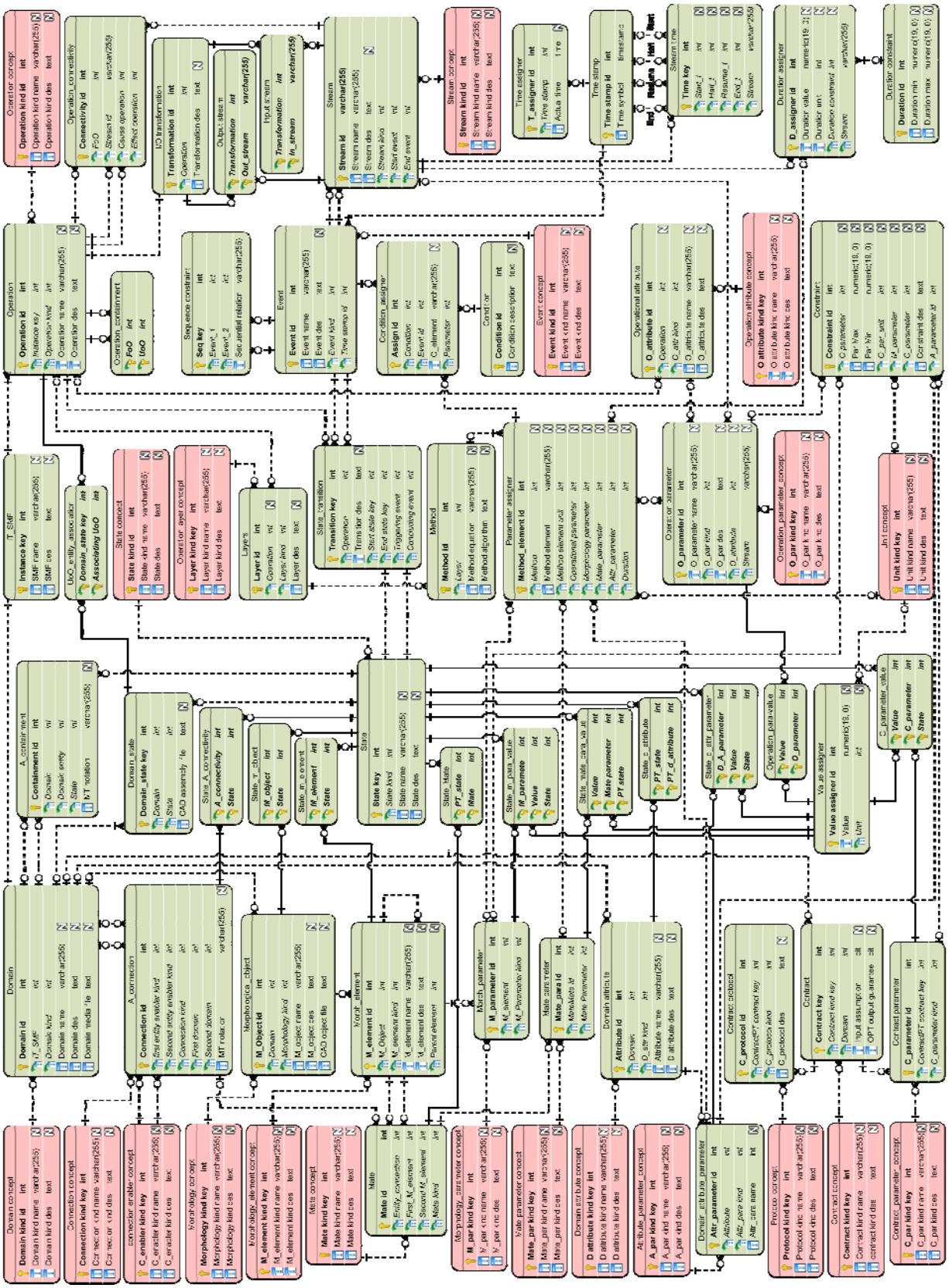

Fig. 3 Overview of the external schema of the model warehouse

imported to the model space are coupled and instantiated in the model space. Accordingly, their architectural and operational relationships are changed. Actually, these relationships should be captured in the database of the model warehouse during coupling and instantiation of PTs. From a different viewpoint, aggregation leads to formation of compound PTs of SMFs. Together with the simple SMFs, the instances of compound SMFs are stored as such in the database of the model warehouse. The aggregation process, i.e., coupling and instantiation of PTs of SMFs, can be continued without any theoretical limitation. On the other hand, 'multi-granularity' of the built SMFs-based models lends itself to an effective composition and decomposition process. Below we describe the ISCs that have been designed for the management of information about operational and architectural connectivity and containment relations in the database of the model warehouse.

3.1 ISCs for representation of operation and architecture relations

When their instances are derived, PTs are placed into architectural and operational relationships. The ISCs discussed below should capture all established relationships (which do not belong to any of the concerned instances individually, but do belong to both related PTs). Coupling of PTs means that the parameters related to their interfaces will be coupled and the assigned values will be shared by both. Consequently, values are assigned to all related parameter variables of the concerned PTs. In the composition process, the interfaces of PTs play two important roles, since they are both operation interfaces and architecture interfaces. Operation interfaces handle external streams and the associated events and states. Architecture interfaces handle contracts and respective morphological elements (ports). Depending on the type of parameters, constraints on parameter values can be considered in both interfaces.

3.1.1 ISC for operation containment and connectivity

The multi-granularity of SMFs makes it necessary to capture aggregation of operations of coupled PTs. Based on the data stored in the model database, multiple units of operation (UoOs) can be represented as a flow of operation (FoO) of a higher level operation aggregate. Though several levels of aggregation should be taken into consideration, we decided to include only one table, which is however able to store all operations without considering their aggregation levels (Fig. 4). The 'operation_containment' table consists of two columns representing FoO and UoO. Each of the rows of this table captures one containment relation. There are two one-to-many relations established between 'operation' and 'operation_containment' tables. One of these relations concerns an FoO, and the other indicates one of its UoOs. Operations are physically interrelated by the streams that they manipulate. That is the reason why the table 'stream' is included in this construct.

Connectivity of operations is captured by the 'operation_connectivity' table. This table includes five columns: (1) 'connectivity_id', which is the primary key, (2) FoO, which is formed by the connections of UoOs, (3) 'stream_id', which is a foreign key imported from the 'stream' table, (4) 'cause operation', which is a foreign key imported from the 'operation' table, and (5) 'effect operation', which is another foreign key imported from the 'operation' table. The 'cause operation' refers to the UoO that is a sender of a given stream. The 'effect operation' refers to the UoO

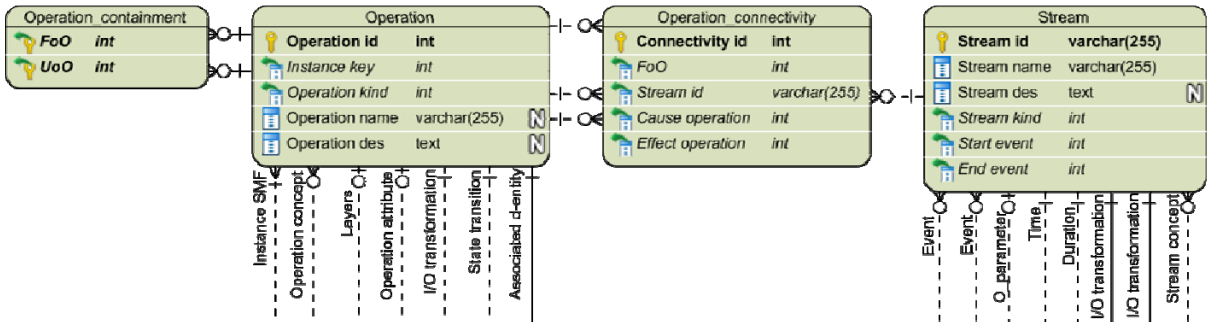

Fig. 4 Construct for arranging relational tables for capturing operation containment and connectivity relationships of SMF instances

that is a receiver of a given stream. There are three links established among 'operation' and 'operation connectivity' tables. One link allows referring to the 'FoO' key, and two links refer to the keys of UoOs.

3.1.2 ISC for architecture containment and connectivity

In a Euclidean space, the spatial metric of an instance of an SMF is represented by the concerned architectural domain. In the case of a compound instance, the domain is an aggregate of sub-domains (specific domain elements defined in the PT of an SMF). When a system model of a CPS is developed, domains can be subjects of multiple aggregation or de-aggregation. The results of this compositional process can be captured in the database as a sequence of containment relations. Consequently, for the purpose of capturing a list of domains, the 'domain' table has been introduced. For capturing the list of containment relations among domains, the 'A_containment' table has been specified. They are necessary, but together are also sufficient, to describe the architectural aggregation and de-aggregation hierarchy of a system (Fig. 5). The 'A_containment' table includes five columns: (1) 'containment_id', which is the primary key, (2) 'domain', which is the foreign key imported from the 'domain' table, (3) 'domain_entity', which is another foreign key from the 'domain' table (referring to one of the sub-domains of the chosen domain), (4) 'state', which defines the state of operation, in which a containment relation exists, and (5) 'MT_notation', which represents the mereotopological definition of the captured relation.

Due to the exclusion of self-containment, which is imposed by the applied strictly physical view, a domain cannot be concurrently included in one row of this table as both domain and domain entity. Moreover, an instance of a component cannot be part of two separate components at the same time. However, it is possible and allowed to be part of several components in various states (time sections). The 'state' column has been introduced because of the need to capture this situation. It specifies if a 'domain entity' is part of a 'domain' in a particular state, and if it is a part of another 'domain' in another state. If two domains are architecturally connected, they should be registered as 'first domain' and 'second domain' in one row of the 'A_connection' table. This table is similar to the table included in the architecture containment and connectivity construct of PTs.

3.2 ICSs for assigning values to parameter variables

The major difference between an instance of an SMF and its PT is in the explicit and dynamic value specification of instances. In the case of a PT, it is sufficient to specify the value range of variables (i.e., the maximum and minimum values). Technically, there are two ways of specifying values of architecture and operation parameters (1) by their constraints (highest and lowest values) and (2) by their precise values. Typically, constraints are therefore used in specification of PTs, and mostly precise values are used for specification of instances. The values of parameter variables may be of two kinds: constant values and variable values. The parameters with constant values cannot be changed by input and output operations, while values of some variable parameters may change continuously according to physical variations of the related parameters.

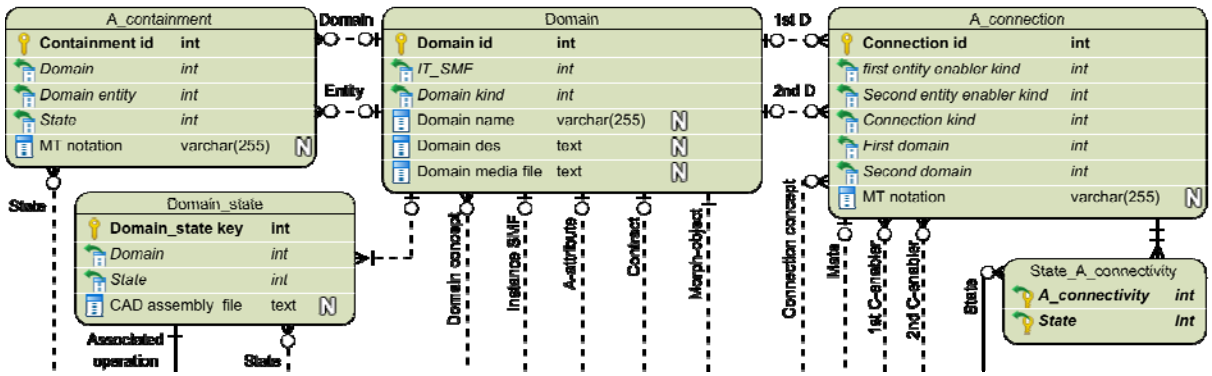

Fig. 5 Construct for arranging relational tables for capturing architectural domain containment and connectivity relationships of SMF instances

3.2.1 ISC for handling assigned metric and attributive values

Due to interaction with other associated instances, variable values of SMF instances may temporarily change. For example, in the case of an electromotor built into a model, the output values of speed and torque will change according to the input value of the electric current. These dependencies are captured as numerical relations between inputs and outputs, and described by 'methods' stored as mathematical equations. The temporal changes of variable values can be specified by either methods or states of operation. These two ways of capturing the changes support both 'continuous-in-time' and 'discrete event-based' simulations.

In the process of instantiation, the values of operation-related parameters (e.g., time stamps, duration, stream parameters, and operation attribute parameters) and architecture-related parameters (e.g., morphological parameters, mate parameters, architecture attribute parameters, and contract parameters) should be specified. The values of operation-related parameters can be (1) defined in interaction with other SMF instances, (2) calculated based on the mathematical equations stored in the 'method' table, (3) specified by the simulation engine, and/or (4) inserted manually. From computational and data management perspectives, architecture parameters and their values should be defined based on the associated states (of the components or whole of the system at hand). In the organization of the model database, we have to take into consideration that architecture parameters may be different in start states and end states. This also applies to the descriptive architectural parameters of domains.

In the database of the model warehouse, the 'value assigner' table is in charge of assigning values to all architecture and operation parameters (Fig. 6). This table can be considered an input gate for values received from the instantiator module, the simulation engine, and the user interface. It has a many-to-many relation to: (1) the morphology parameter table, (2) the mate parameter table, (3) the architecture attribute parameter table, (4) the contract parameter table, and (5) the operation parameter table. Five associated tables are allocated to capture the many-to-many relations. The association tables related to architectural parameters accommodate another column to assign the states. The foreign key of 'state' is imported from the 'state' table.

3.2.2 ISC for handling values assigned to duration and point of time parameters

The designed construct is shown in Fig. 7. As one approach, temporal and event values are partly handled by the 'sequence_constraint' table. This table captures transposition and chronological relations among events. The states are linked to events, and events are defined based on temporal specifications. Some temporal constraints can be defined by the link between the 'event' and 'time stamp' tables. Another way of setting the temporal values is to specify the actual time of timestamps. The actual time specifications are handled by the 'simulation engine'. The 'time_assigner' table is created and linked to a 'time_stamp' in order to support assignment of values to temporal parameters. The 'time_assigner' table is considered a gatekeeper for specifying the time stamp values by the 'simulation engine'. To specify the accurate duration of an operation, the 'duration assigner' table has been created and linked to the 'parameter assigner' table. The values of this table are calculated based on the actual time of the start, end, and

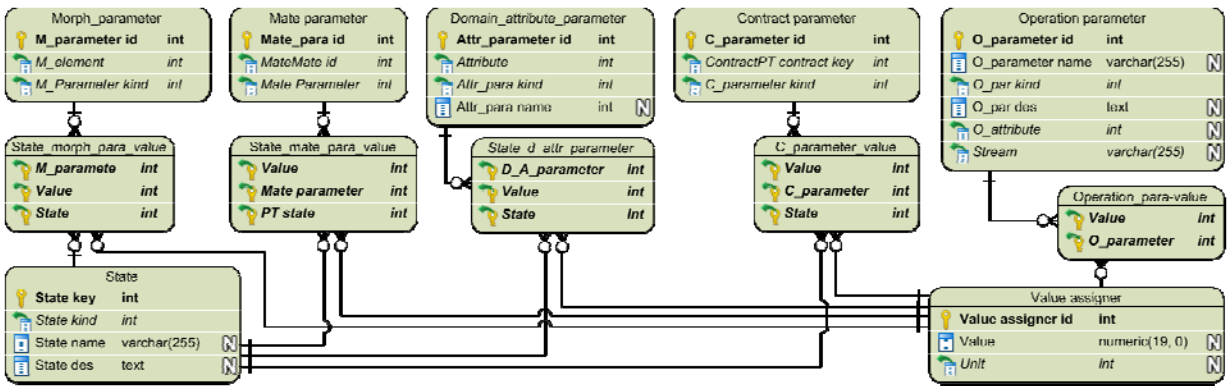

Fig. 6 Construct for arranging relational tables for assigning values to parameter variables of the SMF instances

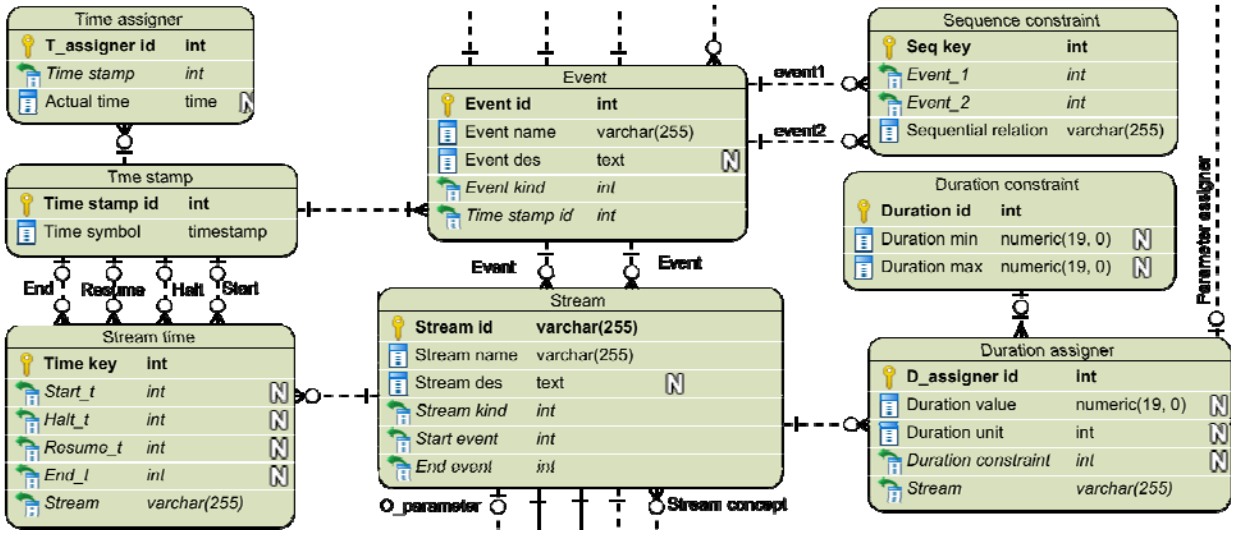

Fig. 7 Construct for arranging relational tables for assigning the duration constraint and time to operations of SMF instances

halt and resume events. The 'duration constraint' table supports the process of checking the composability of temporal constraints.

4 ISCs for meta-level knowledgebase of the model warehouse

The model warehouse module also includes the computational mechanisms for meta-level management of SMF instances based system models in its database. The warehouse keeps track of and records (1) from which GT the instantiated PTs have been derived, (2) which PTs have been composed and instantiated as parts of the model, (3) what interfaces of the instantiated PTs are coupled, and (4) the history of assigning values to their parameters. As a first step in discussing the ISCs used for structuring the meta-level knowledgebase of the model warehouse, we present an overview of its external schema in Fig. 8. It shows the logic of the connections among the relational tables. Maintaining these chunks of information facilitates addition and/or subtraction of SMFs, and any in-process partial modification of the model.

4.1 Recording composition and parameterization history

Creation of a system model involves coupling, instantiation, and composition of chosen PTs. To couple PTs operationally, their interfaces (i.e., external streams and their associated events and states) should be connected. This process is called unification. Depending on the subject, various unification mechanisms have been defined (e.g., stream unification, event unification, and state unification). For instance, to describe complex operation processes, the end event of the first operation should be unified with the start event of the next operation. In association with this, information about mating of the PTs in the architectural aggregation process should also be recorded.

4.1.1 ISC for recording the origin of the PTs and handling the registry of SMFs

Fig. 9 shows the construct designed for recording the PTs included in the system model. Typically, PTs are derived from GTs. As an alternative to this, PTs that are stored in the PT warehouse can also be used for composing parts of system models. Coupling of PTs creates 'compound' SMF instances. In the case of compound SMF instances, there is no need to record the identifier of the parent GTs, since this information can be retrieved from the database of the concerned PTs. Therefore, in this case, the 'GT key' in the 'browsed PT' table is 'nullable'. Another consideration is that, by alternative structural parameterization, multiple SMF phenotypes can be derived from a particular GT and multiple SMF instances can be derived from one PT. This explains why there are one-to-many relations among the relational tables shown in Fig. 8.

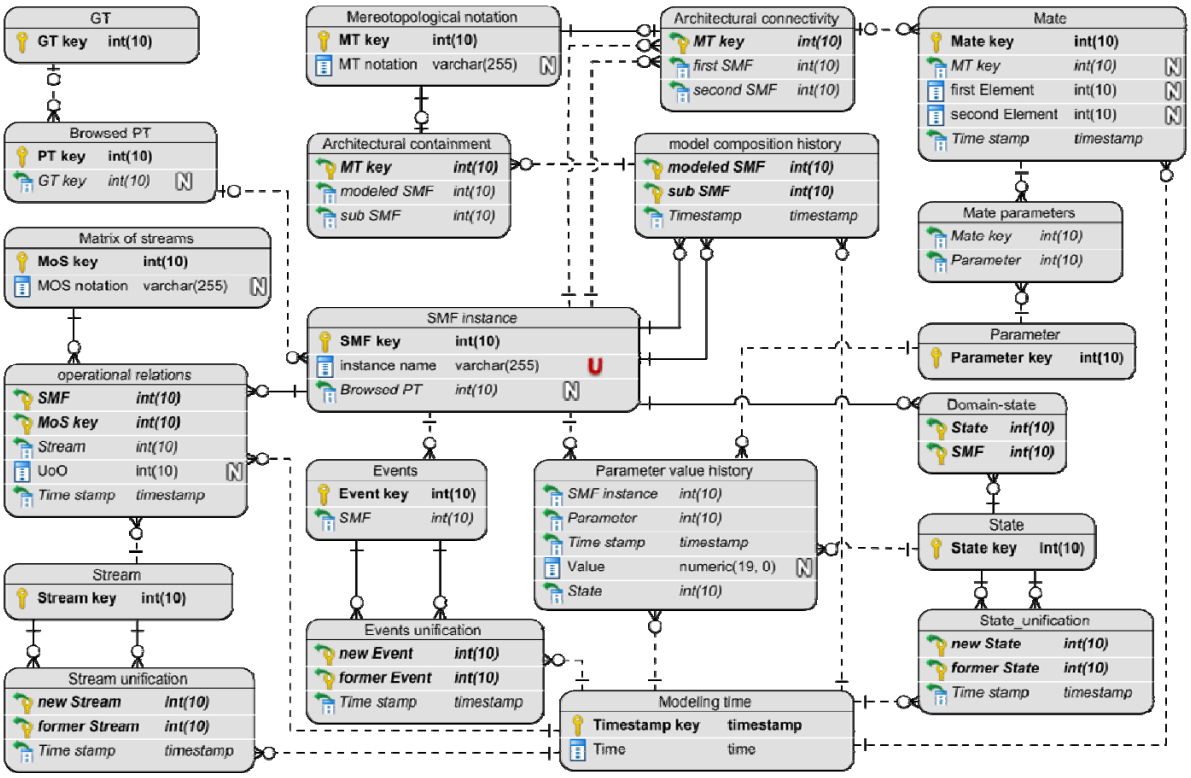

Fig. 8 Overview of the meta-level external schema of the model warehouse

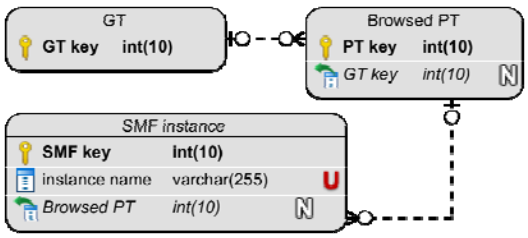

Fig. 9 Construct for recording the origin of the PTs included in the system model

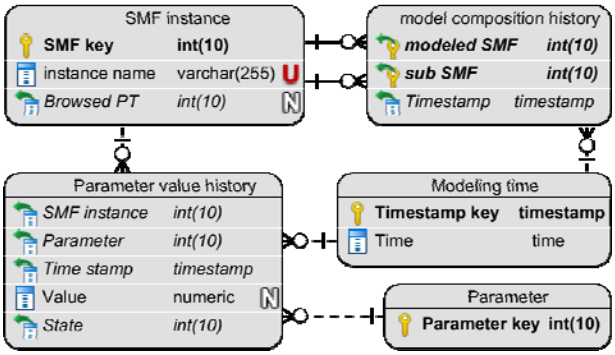

Fig. 10 Construct for recording the history of PT composition and parameterization

4.1.2 ISC for recording the composition and parameterization history

Fig. 10 shows the construct designed for recording the history of PT composition and parameterization. Actually, this is the main construct of supporting meta-level model database management. It consists of two history recording tables. The ‘model composition history’ table stores the information about the composition of SMFs during the modeling process, and the ‘parameter value history’ table captures the changes of the values of the various parameters in the course of model composition.

There are two types of SMF instances stored in the ‘SMF instance’ table. The instances that are derived from browsed PTs belong to the first type. This is why the ‘browsed PT’ keys are specified for this type. The compound SMFs that are created by composing several PTs in the model database belong to the second type. The ‘model composition history’ table captures the relations among these two implementations of SMFs. The ‘modeled SMF’ key refers to the newly generated SMFs and the ‘sub SMF’ key refers to the

instantiated PTs. This table also captures the timestamp of composition and consequently the sequence of composing SMF instances into a model. As a compound SMF can be treated computationally as a novel SMF, they have been identified in the construct as 'new SMFs'. The value of 'browsed PT' is null for the new SMFs.

The composition of SMFs changes their parameter values and attributes. Some of the parameters should be redefined, while values of some others should be updated according to the results of model composition. For example, the center of gravity of a component may change due to building in additional components into the model, or the fuel consumption changes the weight in airplanes when its behavior is simulated. In the first example, the parameter changes should be captured according to the modeling timestamp, while in the second example, the changes of parameters should be captured according to the simulated operational states. The 'parameter value history' table captures all pieces of information required for tracking these changes in the model. The parameters shared among the instantiated PTs and the new SMFs are captured by a many-to-many relation among the 'SMF instance' and 'parameter' tables. This relation is realized by the 'parameter value history' association table, which accommodates timestamps to track history of changes.

4.2 Recording state, event, and stream unification history

A PT can specify two types of internal and external states. External states are 'observable' from outside and are regarded as part of interfaces of PTs. When multiple PTs are coupled in a model composition process, some of the external states may be turned into internal states. Just as an example, if FoO_1 of PT_1 is also associated with PT_2 as FoO_2, then the end-state (S_1^e) of FoO_1 is considered the start-state (S_2^s) of FoO_2. In the model composition process these two states are unified into an internal state (S_3^m) of the new compound SMF. The construct designed to capture the records of state unification and the related timestamps is shown in Fig. 11. This construct facilitates ordering of the stored states, including the internal and external states of instantiated PTs, as well as the states newly created by state unifications. The records of state unifications and their respective timestamps are accommodated in the 'state_unification' table.

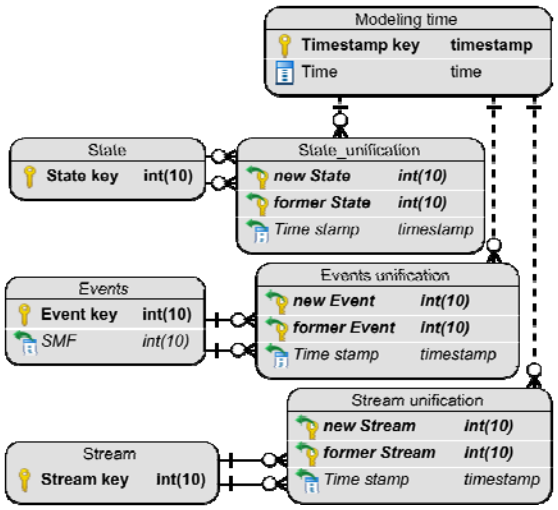

Fig. 11 Construct for recording state, event, and stream unification history

When a start-event and an end-event are unified, a new middle-event is created and related to both coupled SMFs. Accordingly, the former start and end events are no longer related to the coupled SMFs. However, the information about the relations of the former events is needed for modification of the model. Consequently, the history of event unification should be captured. If the output stream of one PT is the same as the input stream of the other, they can be operationally coupled. For composition, the two external (input and output) streams should be unified. The stream unification replaces the external streams with an internal stream in the database. Capturing the history of stream unification facilitates separation of SMFs, which is needed for model modification.

The construct named 'stream unification history' resembles the other constructs related to interfaces of SMFs. The main difference is that the streams are indirectly connected to the SMFs. In fact, the streams are connected to SMFs through an association table named 'operational relations', which is a part of the constructs presented in the next paragraph. It is worth mentioning that the values and parameters related to the constraints of streams may need to be updated due to the coupling. These changes are captured by the 'composition and parameterization history' construct, which has been introduced and explained above.

4.3 Capturing operational and architectural relations on meta-level

The concept of 'matrix of streams (MoS)' is used as a means for capturing operation relations (Pourtalebi and Horváth, 2016a). An MoS specifies operational relations within an FoO by capturing its UoOs and the streams connecting them. Considering FoOs operations of the modeled CPS, UoOs are operations of instances of SMFs composed into the model. The streams (which are connecting these instances) are the newly created streams (as a result of stream unification).

4.3.1 ISC for recording operation relations on meta-level

This construct is shown in Fig. 12. MoSs are stored in the 'matrix of streams' table. The streams and UoOs are linked by the association table named 'operational relations'. This table accommodates imported SMF and MoS keys as compound primary keys. The stream key is imported from the 'stream' table. Moreover, the 'time stamp' key that is imported from the 'modeling time' table specifies the sequence of model composition. In the case of model modification, the time stamp also helps identify the records in multiple tables that are related to one model composition action.

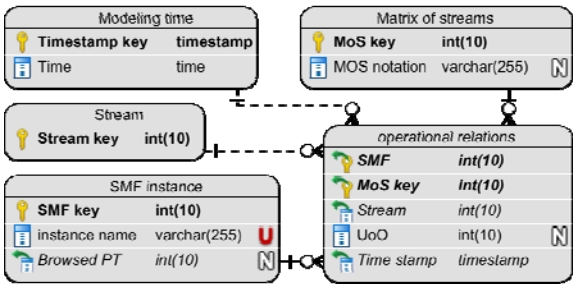

Fig. 12 Construct for capturing operational relations in the composed system model

4.3.2 ISC for recording architecture relations on meta-level

This construct is shown in Fig. 13. In Pourtalebi and Horváth (2016a) it was explained that the architectural relations are captured by mereotopological notations. These notations are stored in the 'mereotopological notation' table. The elements of the notations are the SMFs that are stored in the 'SMF instance table'. The containment notations are linked to SMFs through the 'architectural containment' table. The connectivity notations are linked to SMFs through the 'architectural connectivity' table.

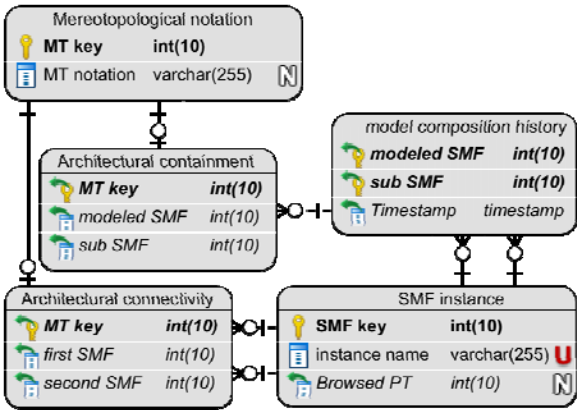

Fig. 13 Construct for capturing architectural relations in the composed system model

4.3.3 ISC for capturing mating elements and history of mates

This construct is shown in Fig. 14. Recording the mate history is needed for tracking the physically driven aggregation of SMFs, and modifying the model whenever instances of SMFs are added to or subtracted from the model. Another benefit of having the 'mate history' construct is that it provides opportunity for calculating the mating constraints. For example, in a practical pre-embodiment design case, the tables of this construct may keep a record of the occupation of the hard disk space by the installed software applications, or the 2D space remaining on a surface after connecting some components.

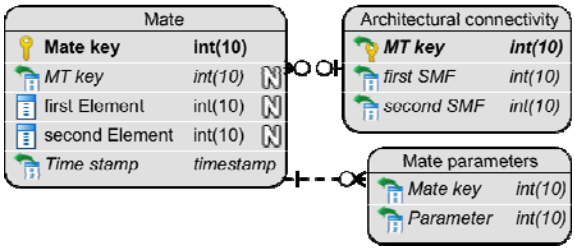

Fig. 14 Construct for capturing mating elements and history of mates

Mates are associated with the architectural connectivity relations. Specifically, each of the architectural connectivity relations may be described by more

than one mate. This is the reason why a one-to-many relation is established among the 'architectural connectivity' and 'mate' tables. The 'mate' table accommodates keys of: (1) two morphological elements imported from the model database, (2) the timestamp when the mate was created, and (3) their mereotopological relations. The mate parameters are captured by an association table that supports many-to-many relations among the 'mate' and 'parameter' tables.

Although the precise value of parameters should be defined for instances, the constraints imported from PTs should be kept and updated. These constraints are needed for regulating the value assignment (e.g., during simulation) and analysis of constraints satisfaction (Munir *et al*., 2015). The composer module of SMF-TB is responsible for checking and validating all constraints before and during composing instances into a system model, and for updating them after composing. However, to point out weak points or bottlenecks, the total performance analysis of the CPS might need redundant constraint analysis. Additionally, the simulation engine is in charge of calculation of the values that should be specified. Sometimes, the purpose of simulation is to understand the situations where some constraints are violated. The resultant operational and architectural data are assigned to the model by the instantiator module.

5 Computational issues of SMF-based modeling

Instantiation of PTs and assigning values to parameter variables are done by the workbench part of SMF-TB. Multiple modules support the execution of instantiation, for instance: (1) the PT browser module which is in charge of finding the PTs that can be used for modeling of the CPS at hand, presenting the information sets to the user, and sending the information sets to the composer, (2) the composer module which receives the data packages of the chosen PTs, updates them with new information, and sends the updated data packages to the instantiator, (3) the instantiator module which determines and assigns values and constraints to parameter variables, and (4) the model warehouse module which stores the consolidated and tested model data.

5.1 Collecting chunks of information by entry forms

The entry forms make the modeling process interactive and conversational. There are three entry forms designed for collecting the required chunks of information from the system designers, each of which is realized in a separate tab of the user interface. These entry forms successively collect chunks of information concerning: (1) list of PTs that are composed, (2) operational composition of the selected PTs, and (3) architectural composition of the selected PTs.

In the first entry form, CPS designers need to select the PTs that will be used in the system model. Name and description of any new instance of SMFs should be included in the first partition of the form. In the second partition, the designer will be asked to provide a name and description for the architectural domain of the concerned SMF. There is an option for uploading a media file. In the third partition, name and descriptions of operations of the SMF should be determined. Since there might be several operations for an SMF, for each operation, associating UoOs of PTs should be selected. For example, for composing a new SMF of a smartphone, PTs of battery, screen, camera, processor, etc. should be instantiated. Multiple operations could be defined for the new SMF, e.g., navigation, gaming, video calling, photo shooting, and music playing. Accordingly, in the third partition of the first entry form, operations of the new SMF (FoOs) could be defined by selecting multiple operations (UoOs) imported from the chosen PTs. The following declarations represent the fields of the first entry form:

Entry form 1: composition basics

1.0. First partition:
  1.0.1. Composed SMF name <text box>
  1.0.2. Composed SMF descriptions <text area>
  1.0.3. Select phenotypes to compose <select box, add button>
1.1. Second partition (architecture of composed SMF):
  1.1.1. Composed domain name <text box>
  1.1.2. Composed domain descriptions <text area>
  1.1.3. Composed domain media file <brows file button>
1.2. Third partition (operation of composed SMF):
  1.2.1. Composed operation name <text box>
  1.2.2. Composed operation descriptions <text area>
  1.2.3. Select operations of phenotypes to compose <select box, add button>
Add new operation (1.2) <button>

The second entry form is created for configuring operations of the model. The first partition is allocated to select an operation (among the operations defined in the first tab). The state transition should be defined in the second partition. The states could be selected from the states formerly defined in PTs. The I/O transformation should be defined in the third partition. The streams represented as options in this partition are invoked from external streams of the chosen PTs. In this way, the interfaces of the PTs that remain as interfaces of the new SMF could be specified. The next partition specifies the external streams of the PTs that should be turned into internal streams in the new SMF. To couple the operation interfaces of the chosen PTs, their external streams should be linked together. In fact, the FoOs of PTs (which are considered here UoOs) should be linked together as procedural elements of FoOs of the new SMF.

Sequential constraints of the events have already been defined in the PTs. However, after composing them, we need to define sequential constraints among the events in various PTs. For instance, if the end state of one PT is the start state of the other, their respective events should be defined as coincident. In the fifth partition, the sequential relations of the events will be specified. The same applies to defining conditions. There may be a need to define some additional conditions that regulate the operational relationship among PTs. This kind of condition can be defined in the sixth partition. Since all external events are formerly created in PTs, there is no need to generate a new event. The events are invoked from the database to be selected by designers. The composer module and instantiator module are in charge of defining the operational procedure and methods of the newly composed SMF, respectively. The following declarations represent the fields of the second instance form:

Entry form 2: operational composition

2.0. First partition:
  2.0.1. Select operation <select box>
2.1. Second box (state transition):
  2.1.1. Start state <select box>
  2.1.2. End state <select box>
  2.1.3. Transition descriptions <text area>
Add new state transition (2.1) <button>
2.2. Third partition (input/output transformation):
  2.2.1. Input streams <select box, add button>
  2.2.2. Output streams <select box, add button>
  2.2.3. Transformation descriptions <text area>
2.3. Fourth partition (stream unification):
  2.3.1. Stream <select box, add button> is the same as <text>
  2.3.2. Stream <select box, add button>
2.4. Fifth partition (event sequence constraints):
  2.4.1. Events <select box, add button>
  2.4.2. Sequential relation <select box>
  2.4.3. Events <select box, add button
Add new sequence constraint (2.4) <button>
2.5. Sixth partition (conditions):
  2.5.1. Condition description <smart text area>
Add new condition (2.5) <button>
Compose new operation (2) <button>

Following the specification of the state transitions in the second tab, the first partition in the third tab (architectural composition) is to select the state in which the architectural relations should be specified. The other partitions are allocated to specify architecture containment (partition 2), architecture connectivity (partition 3), and morphological mates (partition 4). Since the included domain entities and the connections among them are specified in association with states, the changes in the architectural domain of the new SMF should be defined in this tab. The domain entities of the new SMF are the domains of the chosen PTs that are being instantiated here. The contract defined in PTs could be used for defining connections in this step. The architectural composition of the new SMF is the result of these configuration and instantiation actions completed by CPS designers. After specifying the architectural connections, morphological mates should be specified. This partition is more or less similar to the partition allocated to morphological mates in the PT derivation step. The following declarations represent the fields of the third instance form:

Entry form 3: architectural composition

3.0. First partition:
  3.0.1. Select state key <select box>,
  3.0.2. CAD assembly file <brows file button>
3.1. Second partition (architecture containment):
  3.1.1. Contained domain <select box, add button>
3.2. Third partition (architecture connection):
  3.2.1. Connection kind <select box>
  3.2.2. Select first domain <select box>
  3.2.3. First connection enabler kind <select box>
  3.2.4. Select second domain <select box>
  3.2.5. Second connection enabler kind <select box>
3.3. Fourth partition (morphological mate):
  3.3.1. First morph-element <select box>
  3.3.2. Second morph-element <select box>
  3.3.3. Mate kind <select box>

3.3.4. Mate_parameter <select box>
3.3.5. Mate_parameter_value <text box>
3.3.6. Mate_para_value_unit <select box>
Add mate parameter (3.3.4–3.3.6) <button>
Add new mate (3.3) <button>
Add new connection (3.2) <button>
Add new domain state (3) <button>

5.2 Information processing by the composer and instantiator modules

The modeling process of a CPS is considered an ordered set of systematic procedures in which several actors are involved. Accordingly, this procedure can be investigated in many aspects. The user workflow, interaction among modules of the modeling tool, and information processing are examples of these aspects which are in the focus of our research. Obviously, there are multiple aspects that are excluded from our research, to be able to reduce it to a manageable amount of work.

To capture the procedure of model composition and instantiation, the concept of 'procedural computational schema (PCS)' has been created. PCSs represent algorithmic activity flows that could be implemented through programming languages. In the following, the PCSs of generating SMF-based models are demonstrated. Here, we present mainly the activities of the composer and instantiator modules. The activity cycles represented in Fig. 2 will be revisited and referred to in the elaboration below of the process of modeling. The process starts with copying content of the relational tables of the chosen PTs from the PT database to the model database. This information transfer is performed according to a mapping table.

5.2.1 First activity cycle of instantiation

In the first step of instantiation, the information sets of the chosen PTs should be copied to the model database. A new SMF should also be created and stored in the same relational tables. The connections between the new SMF and the copied PTs are stored in the 'operation containment' table. The following declarations represent the procedure construct proposed to be employed by the composer and the instantiator in the first activity cycle:

PCS 1: Check constraints (actor: composer)

- Start with a newly specified operation (FoO)
- Extract the operations (UoOs) of the chosen phenotypes composed for delivering the new operation (FoO)
- Compare the 'operation kind' (specified for the new operation) with a list of the 'operation kind' defined for the possible super operations of the chosen phenotypes
- Notify the designer about the mismatches
- Receive approves or changes
- If changes are applied, repeat the comparison
- Create operation containment relations
- Record the relations in the meta-level knowledgebase of the model warehouse

PCS 2: Create a new SMF (actor: instantiator)

- Receive the name and descriptions of the new SMF (and its domain) from the entry form and store it in the model warehouse
- Receive the data sets of the chosen phenotypes and store them in the respective table of the model database
- Receive the operation containment relations among the new SMF and the chosen phenotypes, made by the composer and store them in the 'operation containment' table
- Create a report and store it in the meta-level knowledgebase of the model warehouse

5.2.2 Second activity cycle of instantiation

In the second activity cycle, the composer identifies 'operation interfaces' and provides them as options for the designer. Following the designer's decision, the parameters should be merged and constraints should be redefined by the composer. Subsequently, the instantiator modifies the respective keys of parameters in order to link them to the methods and to the newly defined constraints and values. The generic algorithms used by the composer and the instantiator are represented below:

PCS 3: Identify operation interfaces (actor: composer)

- Group operations of the chosen phenotypes regarding their participation in the operations defined for the new SMF
- Perform the rest of this process for each group separately
- Identify external streams of the selected operations of the phenotypes
- Identify the streams associated with the 'I/O transformation' table
  - Identify the streams linked to the 'UoO connectivity' table that are missing one side of either cause operation or effect operation
  - Group them as external 'input streams' and 'output streams'
- Identify the interface events of the selected operation of the phenotypes
  - Trace the events associated with external streams represented in the 'stream' table as the 'start event' and 'end event'
  - Identify the events represented in the 'state transition'

table as the 'triggering event' and 'concluding event'
  - Group them as external 'start events' and 'end events'
- Identify the interface states of the selected operation of the phenotypes
  - Extract the list of states associated with the chosen events in the table of 'state transition'
  - Group them as 'start states' and 'end states'
- Create a report and store it in the meta-level knowledge-base of the model warehouse

All of the selected and sorted streams, events, and states should be sent to the UI to get displayed to the designer. The various pieces of the entered information are sent back to the composer to be processed and composed.

PCS 4: Connect operation parameters and redefine constraints (actor: composer)

- Identify the parameters that should be merged
  - Identify the streams of the phenotypes that are unified (turned to the internal streams)
  - Trace the 'operation parameters' that host the foreign keys of the unified streams
- Consider the identified parameters mergeable parameters
- Compare the 'parameter kinds' of the mergeable parameters
- If there is any difference, notify the designer
- Trace the 'constraints' associated with the parameters
- If there is any mismatch in constraints, notify the designer
- Redefine new constraints by combining the constraints of the parameters
- Send the result of parameters composition and new constraints to the instantiator
- Create a report and store it in the meta-level knowledge-base of the model warehouse

PCS 5: Modify operation parameter keys and generate new constraints and values (actor: instantiator)

- Link the chosen states of the phenotypes to the new 'state transitions' associated with the operations of the new SMF
  - Create new 'state transitions' for the operations of the new SMF (specified by the user)
  - Link the states chosen by the user to the created state transitions
- Link the streams chosen as input and output of the new I/O transformations to the operations of the new SMF
  - Create an I/O transformation for each of the operations defined for the new SMF
  - Link the external streams of the phenotypes (selected by the user to be considered input and output of the new operations) to the created I/O transformations
- Update names and relations of the unified streams
- Modify relations between timestamps and events
  - Trace the events associated with the unified streams
  - Unify the events through changing their names and relations
  - Assign the respective time stamps to the newly unified events
- Calculate new durations and link them to the streams
- Refine the parameters merged by the composer
- Refine the constraints associated with the merged parameters
- Update the sequential constraints of the events
- Update conditions and generate new ones according to the user input
- Create a report and store it in the meta-level knowledge-base of the model warehouse

5.2.3 Third activity cycle of instantiation

To provide contents of the UI in the third activity cycle, the composer should identify the architecture interfaces of the composed phenotypes. Containment and connectivity relationships are the most important issues regarding domain states. On the other hand, contracts should be used as architecture interfaces that provide options for domain connections. The morphological elements that are associated with the contracts will be seen as ports that could provide connection with other domains. The pieces of information filled in by the user will be sent to the composer for connecting the parameters and refining the respective constraints. Subsequently, the instantiator will create a new database entity and modify the available primary keys. The generic algorithms used by the composer and the instantiator are represented below:

PCS 6: Identify architecture interfaces (composer)

- Identify the list of states regarding the operations defined for the new SMF
- Identify the list of domains respective to the phenotype operations participating in the new operations
- Group and sort the domains according to the associated operations
- Trace the contracts regarding each of the listed domains
- Identify the complementary contracts in different domains for creating smart suggestions
- Trace the domain elements associated with the contacts
- Create a report and store it in the meta-level knowledge-base of the model warehouse

According to the chunks of information collected by the third entry form, three groups of information will be defined by the user at this step related to (1) architecture containment, (2) architecture connectivity, and (3) morphological mate. The 'connection kind' and 'connection enabler kind' suggested to the user are derived according to the contracts of the selected

domains. The user inputs should be processed by the composer and the instantiator according to the algorithms below:

PCS 7: Connect architecture parameters and redefine constraints (actor: composer)

- Identify the contracts used for creating the connections
- Identify the morphological elements associated with the connections
- Compare the morphological elements associated with the contracts to the ones associated in the mates
- Notify the user if there is any difference
- Trace the new mate parameters and the former contract parameters that are the same
- Check the constraint of the former contract parameters with the values of the new mate parameters
- Notify the user if there is any mismatch
- Create a list of modification and send it to the instantiator
- Derive the list of contracts that are not used for connections according to the domain state of the new SMF
- Decide if the contracts converted to connections are still available as contracts for the new SMF
- Send the list of available contracts to the instantiator
- Create a report and store it in the meta-level knowledgebase of the model warehouse

PCS 8: Modify architecture parameter keys and generate new constraints and values (actor: instantiator)

- Assign new architecture containment relations regarding the states of the new SMF
- Assign new architecture connectivity relations regarding the states of the new SMF
- Assign new morphological mates to the specified connections
- Replace the former contract parameters with the new mate parameters in the 'parameter assigner' table
- Assign newly specified values to the mate parameters
- Summarize and send the report to the model warehouse

The process of composing the model of an SMF is conducted here. However, the functions of the composer and instantiator are not completed. The instantiator should assign values to the variable parameters during simulation and the composer should check all constraints at the same time.

6 Modifying SMF-based system models

'Modification' is a generic term for all kinds of changes that need to be applied manually in the process of or after composing a model. It could be divided into two groups of modifications: (1) parameter and context modification, and (2) partial modifications of the SMFs instances based model.

6.1 Parameter value and context modification

Parameter modification implies changing values of the variables of the parameters. These parameters belong to a particular instance of a PT, composed in the model. For example, if a multifunctional PT is instantiated in a model to use only one of its functions, changing its operating mode could be considered parameter modification. Or, if a PT with some knobs for adjusting the operation or architecture parameters is instantiated in a model, setting up the knobs could also be considered parameter modifications. Accordingly, parameter modification could be handled both by the simulation engine and manually. Simulation of embedded customization could be enabled through parameter modification. The options are embedded in PTs as interface events which are linked to various streams and states.

Context modification implies changing the operating contexts of the modeled CPS (Seiger *et al.,* 2014). The simulation engine defines contexts of operation through specifying variable parameters that are defined for this purpose (e.g., parameters of the external streams that define input and output of the CPS). The context can be modified based on the communication of the simulation engine with the CPS designer. The instantiator module is the main actor in the course of the above described parameter and context modifications. It captures all applied changes and stores them in the meta-level knowledgebase of the model warehouse. The knowledgebase is accessible by the simulation engine if it is required.

6.2 Modification of an SMF instances based model

SMF modification implies subtracting, adding, or replacing SMF instances composed in a model. To re-design a CPS, SMF instances might be replaced or added in the CPS model several times. Adding a new SMF to the model is considered similar to model composition, consisting of two PTs (the model and the new PT). Replacing an SMF can be seen as first subtracting an SMF and then adding and instantiating a new PT. Consequently, it can be computationally realized by combining a subtraction mechanism with the procedure of model composition as explained before.

In the following the procedural construct of subtraction will be explained.

This procedural computational schema has three actors, namely the composer, instantiator, and model warehouse (Fig. 15). The main challenge in SMF subtraction is separating unified interfaces (i.e., events, streams, states, and mates) of the composed PTs that are currently turned into internal events and streams, etc. After identifying the unified interfaces, they should be separated and the former interfaces should be rebuilt. Former interfaces imply the interfaces of PTs before composition.

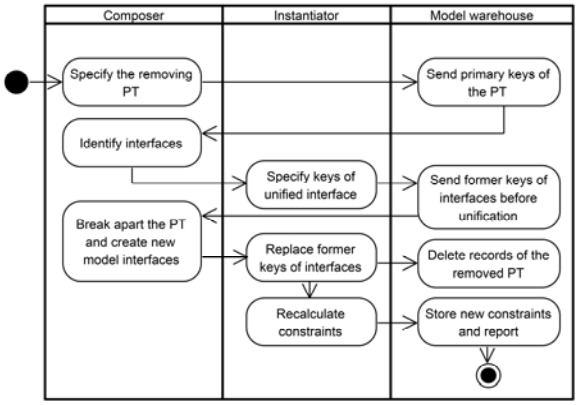

Fig. 15 Procedural computational schema of PT subtraction

The procedure starts with receiving a command from the designer indicating that the SMF should be subtracted from the model. The composer specifies the PT composed in the model and asks for its removal. A list of the related primary keys, which are stored in the meta-level knowledgebase of the warehouse, are invoked from the 'events_unification', 'stream_unification', 'mate_parameters', and 'state_unification' tables, and are sent back to the composer. In tight interaction with the composer, the instantiator specifies the keys of interfaces. These changes are applied in the 'stream_unification', 'events_unification', and 'state_unification' tables. Although in the model the unified streams and events are internal streams and events, they will be turned into external streams and events after subtraction.

The 'model_composition_history' table and 'parameter_value_history' table contain all required chunks of information about the composed SMFs and their parameters. The former keys of interfaces are extracted from the meta-level knowledgebase and sent to the composer to break apart the SMFs and create the new model interfaces. The keys are replaced and all records of the subtracted SMF are removed from the model database.

The last activity of the instantiator module is to recalculate constraints of the parameters in the model after modification. The constraints are updated in the 'constraint' table of the model database. All modifications stored in the relational tables of the meta-level database are assigned with the timestamp key of the 'modeling_time' table. This is the modeling timestamp and should not be confused with the timestamp of the modeled CPS (which is stored in the database).

7 Discussion and reflection

7.1 Consideration of impacts on functionality, usability, and utility aspects of CPS modeling

The proposed ISCs (together with the proposed entry forms and procedural computational schemata) support and confirm feasibility, usability, and utility of SMFs-based modeling of CPSs. From the feasibility perspective, they provide a solid and flexible basis for programming and implementation of the software toolbox. As logical units, they support implementation as follows:

1. The introduced ISCs formalize and regulate large and heterogeneous chunks of SMFs-related information as well as the relationships between them.

2. The entry forms formalize the process of inputting information. Programming of the user interface is viable based on the introduced entry forms.

3. The PCSs determine the information processing and reasoning needed for refining chunks of information, creating connections among them, and modifying the stored information. They can be used as bases for generating procedural codes of SMF-TB.

From the perspective of usability, the proposed constructs and computation enablers operationalize the theoretical specifications that are inevitable for a system-level configuration and conceptualization of CPSs. Usability should be judged by the eyes of CPS designers and having their requirements in mind, since they are the decisive stakeholders. These issues are considered below:

1. CPS designers do not have to bother with the definition of SMFs. SMFs are available for them in a

relatively high level of preprocessing. The task of specifying the required chunks of information about components and accommodating them into SMFs, is delegated to knowledge engineers. So, designers can focus on the creative part of system-level configuration and conceptualization of CPSs. At the same time the design thinking and composition process is supported by the systematic computational workflow.

2. The proposed SMF-TB supports various user activities such as SMF definition, content conversion, CPS model composition, CPS modification, and model simulation, in an integral manner. This has been made possible by the rigorously defined theoretical/methodological frameworks, and the uniform implementation of data processing and process execution resources.

3. The information structures and defined contents of SMFs are reusable. Large-scale reuse of information structures and contents contributes to efficiency and transparency in both design and knowledge engineering activities.

4. SMFs can be retrieved, adapted, and manipulated in the whole CPS model composition and modification process. This is particularly important in the pre-embodiment design phase. The SMFs-based implementation even allows suggesting components according to the list of requirements of designers. The taxonomically aided warehouse management helps designers find the most appropriate components for model composition.

From the perspective of utility, the whole methodology and proposed constructs support CPS designers in tackling system composability challenges. The major indicators of utility are as follows:

1. Our methodology assumes that CPSs are composed of SMFs. It imposes a strictly physical view, which is obeyed by ISCs and PCSs. This physical view has been transferred to the organization of the warehouse databases.

2. The proposed constructs play the role of cognitive enablers of programming, and oriented thinking introduces a meso-level in between micro-level entity management and macro-level model management.

3. The created CPS models are extensively and rapidly modifiable. The proposed constructs capture the architectural and operational data of models through the meso-level chunks of information and relations. On the other hand, they can manage structural re-parameterization and attribute change in a holistic way (Hadorn *et al*., 2015).

4. The proposed constructs lend themselves to effective constraints management, with the capability originating in their abovementioned meso-level manifestation. They support not only modeling but also simulation of operations as the resources needed for simulation (e.g., methods, relations, parameters, and temporal relations) are all accommodated in ISCs.

5. The proposed construct-driven schemas and database management allows the interoperation of SMF-TB with other modeling and simulation software tools. The chunks of information and the relations can be converted to the entities and relations of micro-level database schemas. However, in this case, many of the advantages of SMF instances based modeling are lost.

7.2 Ignoramus et ignorabimus?

This Latin maxim (meaning ‘not known and not knowable’) was used in the nineteenth century to express the position on the limits of knowing. In the context of this paper, it fairly expresses our position about validation of the ISCs proposed for instantiation and composition of SMFs in pre-embodiment modeling of cyber-physical systems. Although we have argued with defendable reasons that the proposed computational constructs oriented thinking supports feasibility, utility, and usability of SMF-TB, we must recognize that only a fully-fledged implementation can provide practical evidence for this assertion.

Using ISCs for database development and management for instantiation and composition of SMFs is a novel and affordance-rich approach. ISCs help tackle the heterogeneity problem and support multi-granularity. In the current stage of our research, we do not have the means that would be needed for a comprehensive and systematic validation of the proposition. The testbed implementation provided some initial insight into the above mentioned three aspects, but it cannot replace long-term testing in benchmark applications. This remains for our future work.

Our reported work concentrated on the methodology of computation, rather than on the methodology of using the proposed SMFs-based modeling in various application contexts. For this reason it is understandable that the presented stage of development may give rise to some skepticism. However, the major

question is whether the impact of the concept of ISCs on the targeted SMFs-based toolbox is just 'ignoramus' or 'ignorabimus'. Without being able to build it completely, nor applying it to a wide set of practical cases and testing it exhaustively, we tried to consider everything based on which we could forecast its merits from feasibility, usability, and utility perspectives. Based on our recent findings, it seems that our arguments will be defensible. However, the catch-22 situation and the dilemma related to that cannot be resolved.

References

Bhave, A., Krogh, B., Garlan, D., *et al*., 2010. Multi-domain modeling of cyber-physical systems using architectural views. Proc. Analytic Virtual Integration of Cyber-Physical Systems Workshop.

Broman, D., Lee, E.A., Tripakis, S., *et al*., 2012. Viewpoints, formalisms, languages, and tools for cyber-physical systems. Proc. ACM 6th Int. Workshop on Multi-paradigm Modeling, p.49-54. https://doi.org/10.1145/2508443.2508452

Derler, P., Lee, E.A., Vincentelli, A.S., 2012. Modeling cyber–physical systems. *Proc. IEEE*, **100**(1):13-28. https://doi.org/10.1109/JPROC.2011.2160929

Edwards, J.R., Bagozzi, R.P., 2000. On the nature and direction of relationships between constructs and measures. *Psychol. Methods*, **5**(2):155-174. https://doi.org/10.1037/1082-989X.5.2.155

Erbas, C., Pimentel, A.D., Thompson, M., *et al*., 2007. A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP J. Embed. Syst.*, **2007**(1):082123. https://doi.org/10.1155/2007/82123

Frevert, R., Haase, J., Jancke, R., *et al*., 2005. System level modeling. *In*: Modeling and Simulation for RF System Design. Springer, Boston, MA, p.25-38. https://doi.org/10.1007/0-387-27585-1_4

Gavrilescu, M., Magureanu, G., Pescaru, D., *et al*., 2010. Accurate modeling of physical time in asynchronous embedded sensing networks. Proc. IEEE 8th Int. Symp. on Intelligent Systems and Informatics, p.477-482. https://doi.org/10.1109/SISY.2010.5647308

Hadorn, B., Courant, M., Hirsbrunner, B., 2015. Holistic system modelling for cyber physical systems. Proc. 6th Int. Multi-conf. on Complexity, Informatics and Cybernetics.

Horváth, I., Pourtalebi, S., 2015. Fundamentals of a Mereo-Operandi theory to support transdisciplinary modeling and co-design of cyber-physical systems. Proc. ASME Int. Design Engineering Technical Conf., p.1-12. https://doi.org/10.1115/DETC2015-46702

Lee, E.A., 2015. The past, present and future of cyber-physical systems: a focus on models. *Sensors*, **15**:4837-4869. https://doi.org/10.3390/s150304837

Lee, G., Sacks, R., Eastman, C., 2007. Product data modeling using GTPPM: a case study. *Autom. Constr.*, **16**(3):392-407. https://doi.org/10.1016/j.autcon.2006.05.004

Macal, M.C., North, J.M., 2006. Tutorial on agent-based modeling and simulation. Part 2: how to model with agents. Proc. 38th Winter Simulation Conf., p.73-83.

Munir, S., Ahmed, M., Stankovic, J., 2015. EyePhy: detecting dependencies in cyber-physical system Apps due to human-in-the-loop. Proc. 12th EAI Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services, p.170-179. https://doi.org/10.4108/eai.22-7-2015.2260045

Petnga, L., Austin, M., 2016. An ontological framework for knowledge modeling and decision support in cyber-physical systems. *Adv. Eng. Inform.*, **30**(1):77-94. https://doi.org/10.1016/j.aei.2015.12.003

Pourtalebi, S., Horváth, I., 2016a. Towards a methodology of system manifestation features-based pre-embodiment design. *J. Eng. Des.*, **27**(16):232-268. https://doi.org/10.1080/09544828.2016.1141183

Pourtalebi, S., Horváth, I., 2016b. Procedures for creating system manifestation features: an information processing perspective. Proc. Int. Symp. on Tools and Methods of Competitive Engineering, p.1-16.

Pourtalebi, S., Horváth, I., 2016c. Information schema constructs for defining warehouse databases of genotypes and phenotypes of system manifestation features. *Front. Inform. Technol. Electron. Eng.*, **17**(9):861-884. https://doi.org/10.1631/FITEE.1600997

Richter, G., 1981. Utilization of data access and manipulation in conceptual schema definitions. *Inform. Syst.*, **6**(1):53-71. https://doi.org/10.1016/0306-4379(81)90018-1

Seiger, R., Keller, C., Niebling, F., *et al*., 2014. Modelling complex and flexible processes for smart cyber-physical environments. *J. Comput. Sci.*, **10**:137-148. https://doi.org/10.1016/j.jocs.2014.07.001

Simko, G., Levendovszky, T., Maroti, M., *et al*., 2014. Towards a theory for cyber-physical systems modeling. Proc. 4th ACM SIGBED Int. Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems, p.56-61. https://doi.org/10.1145/2593458.2593463

Zhou, K.L., Liu, B.B., Ye, C., *et al*., 2013. Design support tools of cyber-physical systems. *In*: Leung, V., Chen, M. (Eds.), Cloud Computing. Springer, Cham, p.258-267. https://doi.org/10.1007/978-3-319-05506-0_25