Structural Contracts – Motivating Contracts to Ensure Extra-Functional Semantics

Gregor Nitsche, Ralph Görgen, Kim Grüttner, Wolfgang Nebel

Structural Contracts – Motivating Contracts to Ensure Extra-Functional Semantics

Gregor Nitsche[1], Ralph Görgen[1], Kim Grüttner[1], and Wolfgang Nebel[12]

[1] OFFIS – Institute for Information Technology, Oldenburg 26121, DE
[2] Carl von Ossietzky University Oldenburg, Oldenburg 26121, DE
{nitsche,goergen,gruettner,nebel}@offis.de

Abstract. In our work we aim at a composable and consistent specification and verification of *contracts* for *extra-functional* properties, such as power consumption or temperature. To this end, a necessary precondition for the semantical correctness of such properties is to ensure the structurally correct modeling of their interdependences.
While this can be solved by a tailoring of the *Component Based Design (CmpBD)* frameworks, the resulting design constraints are specific to tools and viewpoints, not being sufficiently configurable for the designers. To solve this problem within the contract framework, *Contract Based Design (CBD)* with explicit port variables provides also no configurable but sound methodology for structurally relating the properties between different *components* and *views*. For that, we propose the idea of *Structural Contracts*. Using implicit *structural ports*, *structural guarantees* can be given according to the Component Based Design structure. Expressing structural design constraints by the means of *structural assumptions*, the CmpBD constraints can become part of the Contract Based Design framework and, thus, can be checked for compatibility and refinement.
As a result, structural contracts enable the contract based specification and verification of structural rules for the correct modeling of functional and extra-functional interdependences. Providing both, property specifications and Component Based Design constraints by contracts, the approach is flexible and sound.

Keywords: contracts, contract based design, components, component based design, compositionality, composability, compatibility, structure, extra-functional, view, aspect, type, semantics

1 Introduction

Following the increasing opportunities to integrate more functionality and improved performance in today's integrated microelectronic systems has lead to continuously growing design complexity and an increasing number and heterogeneity of design requirements. As a result, the specification, modeling and

This work is supported by the ANCONA Project, funded by the German Federal Ministry of Research and Education (BMBF) under Grant Agreement 16ES021.

verification of such heterogeneous systems became a challenging task, requiring a reliable collaboration of specialists from different design and verification domains.

Following the paradigms of *encapsulation*, *divide and conquer* and *separation of concerns* the concepts of *components* and *viewpoints* have been introduced to master design complexity by *Component Based Design (CmpBD)* [9]. Commonly a component is considered to be a design element, which internally encapsulates its behavior, solely restricting its interaction with the environment to its well-defined *port interface*. Hence, a main precondition of Component Based Design is the components' behaviors to be *compositional*. That means, that for each point of time the interaction between connected components is clearly determined by solely one of the components, controlling the information exchange across this connection without being affected by undelayed influence from the environment.

Additionally, considering the further refinement and implementation of *subcomponents* to proceed independently, the *compatibility* of connected ports has to be ensured. To this end, *type systems* [6] are applied, to declare the *type* of the components' ports and to verify that connected ports have compatible types. Beyond the most common notion of untimed *static types*, such as *boolean*, *integer* etc., *Contract Based Design (CBD)* [3,11] enables a more dynamic notion of compatibility. By the means of the *contracts*' *assumptions* A all acceptable inputs of the components M are formally described by *timed traces*, declaring interconnections incompatible if the corresponding *environment* E is allowed to provide a timed sequence of outputs which violates the components' assumptions. Differently, when the assumptions are satisfied, the components provide outputs according to the *guarantee* G, which correspond to the satisfied assumption.

Nevertheless, static type and *compatibility checking* of contracts is not flexible enough to be applicable for the consistent specification and verification of the interactions between the properties from multiple *extra-functional* viewpoints, largely comprising physical properties w. r. t. power or temperature etc. Differently, a more flexible declaration of designer-defined types would be necessary, to allow for complex, derived and configurable types, which appropriately combine the value and time semantics of the ports with a viewpoint-specific physical interpretation, such as 'average power consumption per operation in μW'.

Considering viewpoint-specific models of a heterogeneous (multi-viewpoint) component to be viewpoint-specific components themselves, we assume that a sufficiently flexible but sound declaration and verification of designer-defined types can be achieved by extending the compatibility criteria of contracts to also structural properties, which constrain the basic design elements of the decomposition, such as the identifiers of components and ports. Since Contract Based Design allows contracts to constrain only the explicitly declared ports of components we propose a first concept of *Structural Contracts*. Based on an *introspection* of the component structure we implicitly instantiate a *decomposition component* plus *structural ports* and *structural nets*, to enable the contract based *reflection* of the component structure via these explicit ports. As a result, the usage of *structural assumptions* allows us to systematically constrain the instantiation and connection of subcomponents based on their component and port names.

To motivate our idea of structural contracts, Section 2 explains an artificial example, for which the extra-functional failure of the design becomes hidden – i. e. erroneously not visible – because of a semantically incorrect connection of extra-functional ports. Next, in Section 3 we summarize the related work, before we introduce the formal basics of components and contracts in Section 4. In Section 5 we present our approach of structural contracts based on the introspection, component extension and contract based reflection of the component structure. Then, in Section 6 we outline a first proof of concept, using our initial example to successfully invalidate the previously hidden false negative verification using structural contracts. In Section 7 we conclude and give an outlook to future work.

2 Motivating Example

To motivate structural contracts we consider the simplified, artificial example of a composed component M given in Fig. 1, which is specified to hold an average power consumption of at maximum 20μW. The component has one functional *input port* x_0, indicating two different operating modes, and one extra-functional *output port* y_p, denoting the average power consumption per clock cycle.

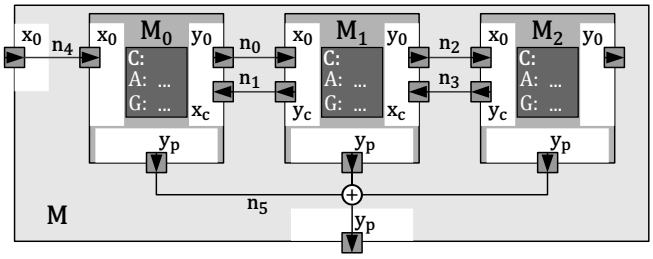

Fig. 1. Motivating example, composing a component M from subcomponents M_i, connecting their inputs $x_{(\cdot)}$ resp. outputs $y_{(\cdot)}$ from multiple extra-functional viewpoints (functional, capacitive load and power) by interconnecting nets $n_{(\cdot)}$ to evaluate the composed system's power consumption.

Refining M by a composition of three different subcomponents, the subcomponents are M_0, M_1 and M_2, with all of them having one functional input x_0 and one functional output y_0, decoding their operating mode in one bit. Moreover, for the purpose of calculating the power consumption, the subcomponents provide inputs x_c and outputs y_c, which describe the components' input or load capacitance $\overline{C}^e_{sw}$, responsible for consuming a certain amount of power during the interaction of both components. Of course, in addition to that, each component consumes also an internal amount of power, based on the internally switched capacitance $\overline{C}^i_{sw}$. For simplification, the example contains no further ports for the supply voltage V_{DD} or clock frequency f_{clk}, considering both of them to be constant at $V_{DD} = 1.0$V and $f_{clk} = 2.0$MHz for components M_1 and M_2 and $M_0.V_{DD} = 1.0$V and $M_0.f_{clk} = 6.0$MHz for component M_0. Above that, following $\overline{P} = \frac{1}{2} V^2_{DD} f_{clk} \overline{C}_{sw}$ the average power consumption $y_p = \overline{P}$ depends on the average switched capacitance $\overline{C}_{sw} = \overline{C}^i_{sw} + \overline{C}^e_{sw}$, internally resp. externally

switched according to the functional activity of the component. Appropriately to these specifications, Table 1 outlines the functional and extra-functional behavior of the components w. r. t. their input resp. output ports. For example, the switched capacitance of M_1 is 3pF for $M.x_0 = 1$. This is because of $\overline{C}^{i}_{sw} = 2$ and $M_1.x_c = M_2.y_c = \overline{C}^{e}_{sw} = 1$, following from exchanging the functional and extra-functional information according to the interconnection of Fig. 1, so that $M_0.x_0 = 1$, $M_1.x_0 = 1$ and $M_2.x_0 = 0$. As a result of this evaluation, the given composition of M finally does not hold the specification of an average power consumption of at maximum 20μW, consuming 22μW for the case of $M.x_0 = 1$.

Table 1. Overview of the example's functional and extra-functional component characteristics, which belong to the functional, capacitive load and power viewpoints. To consider the dependency on the functional inputs the expression $(x_0 \,?\, a : b)$ shall logically denote $(x_0 \rightarrow a) \wedge (\neg x_0 \rightarrow b)$.

viewpoint	$M.x_0$	M_0	M_1	M_2	$M.y_p$
fct	0/1	$y_0 = x_0$	$y_0 = \neg x_0$	$y_0 = x_0$	
$\overline{C}^{i}_{sw}\backslash$[pF]	0/1	$x_0 \,?\, 4 : 1$	$x_0 \,?\, 2 : 1$	$x_0 \,?\, 3 : 1$	
$y_c = \overline{C}^{e}_{sw}\backslash$[pF]	0/1	–	$x_0 \,?\, 2 : 1$	$x_0 \,?\, 1 : 1$	
$\overline{C}^{i}_{sw} + \overline{C}^{e}_{sw}\backslash$[pF]	0/1	$x_0 \,?\, 4 + 2 : 1 + 1$	$x_0 \,?\, 2 + 1 : 1 + 1$	$x_0 \,?\, 3 : 1$	
$V_{DD}\backslash$[V]	0/1	1.0	1.0	1.0	
$f_{clk}\backslash$[MHz]	0/1	6.0	2.0	2.0	
$\overline{P}\backslash[\mu$W]	0/1	$x_0 \,?\, 12 + 6 : 3 + 3$	$x_0 \,?\, 2 + 1 : 1 + 1$	$x_0 \,?\, 3 : 1$	$x_0 \,?\, 22 : 11$

Contrary to this, the same calculation of the power consumption asserts 'valid' when during *refinement* the following – semantically inconsistent – connection error occurs as given in Fig. 2. Refining M with n_6 and n_7 instead of n_1 and n_3, the externally switched capacitances $M_0.y_0$ and $M_1.y_0$ are interchanged, leading to the erroneous power calculation $M.y_0 = (x_0 \,?\, 20 : 11)$ in Table 2.

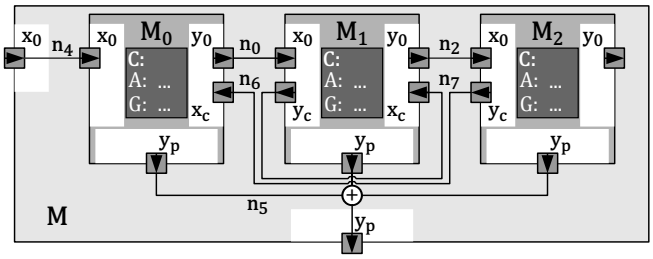

Fig. 2. Erroneous composition resp. refinement of M from the same subcomponents M_i, introducing semantically false connections n_6 and n_7 instead of n_1 and n_3, which violate the physical semantics of allocating the capacitive loads in correspondence with the real worlds functional interconnections n_0 and n_2, falsely verifying $M.\overline{P} < 20\mu$W.

Erroneously refining M with n_6 and n_7 instead of n_1 and n_3, the semantical meaning of the capacitive loads, corresponding to the real worlds functional interconnections n_0 and n_2 gets violated. Without the possibility of strictly relating these ports by some concepts of derived complex types, such – and similar – semantic errors can easily remain unrevealed. As a solution to this, we propose

Table 2. Extra-functional characteristics of the erroneously refined example, leading to a false negative verification of the power consumption, satisfying $M.\overline{P} < 20\mu\text{W}$.

viewpoint	$M.x_0$	M_0	M_1	M_2	$M.y_p$
$\overline{C}^i_{sw} + \overline{C}^e_{sw} \backslash [\text{pF}]$	0/1	$x_0 \,?\, 4 + 1 : 1 + 1$	$x_0 \,?\, 2 + 2 : 1 + 1$	$x_0 \,?\, 3 : 1$	
$V_{DD} \backslash [\text{V}]$	0/1	1.0	1.0	1.0	
$f_{clk} \backslash [\text{MHz}]$	0/1	6.0	2.0	2.0	
$\overline{P} \backslash [\mu\text{W}]$	0/1	$x_0 \,?\, 12 + 3 : 3 + 3$	$x_0 \,?\, 2 + 2 : 1 + 1$	$x_0 \,?\, 3 : 1$	$x_0 \,?\, 20 : 11$

structural contracts, to allow the designers to add *composition constraints*, and an introspection and reflection of the component structure, making the *structural decomposition* part of the contract-based specification and verification approach.

3 Related Work

Considering the related work – to the authors' best knowledge – no other work aims at ensuring the *semantical consistency* of different components and viewpoints – i. e. a verifiable but flexible type system with complex, designer-defined types – by a contract based formulation of constraints for the logical decomposition structure.

The probably most common approach to support semantical consistency and compatibility would be to provide only a limited set of fixed types of components and ports resp. *design rules*, which are defined and checked by the component based design framework. While this *tailoring* may support complex type systems, as e. g. the polymorphic and *structured types* [12] in *Ptolemy II*, it lacks flexibility w. r. t. defining viewpoint-specific compatibility and refinement rules, meaning constraints on how these types can *bottom-up* be constructed resp. *top-down* refined, checking value-, causality- and time-aware construction rules.

In *interface theories* [1, 2] the distinction between bottom-up components for *compositional abstraction* on the one hand, and top-down interfaces for *compositional design* on the other hand have lead to the general concepts of compatibility and refinement checking for component specifications using assumptions and guarantees. Applying timed languages to describe assumptions and guarantees, the contracts allow to specify and to verify the compatibility of the components' interaction protocols according to value and time resp. causality criteria. Nevertheless – while building the general foundation for contracts based design – without some introspection and reflection of the interface variables and their interconnection relations via additional ports, top-down constraints w. r. t. the logical decomposition structure are not possible that way.

Differently, to investigate and define behavioral types, in [4, 5] the concepts of *glue operators* and *glue constraints* are defined for the *BIP (behavior, interaction, priority) framework*. Providing connectors with priorities and their own memoryless behavior the interaction between components connected by a connector can appropriately be synchronized w. r. t. to some timed or untimed causality relation. Hence, again compatibility is meant only in the sense of interaction protocols, not concerning extra-functional semantics of e. g. different viewpoints.

4 Formal Basics

As given in Fig. 1, Component Based Design allows to structurally compose the behavior of higher level components M from instantiating lower level subcomponents $M_i \in M_M^* = \{M_0, \ldots, M_j\}$, $j \in \mathbb{N}$. These subcomponents' behaviors can interact via the directed ports of the components' interface declaration $p_i \in \bigcup_m \chi_m$, $\chi_m = \chi_m^{in} \cup \chi_m^{out}$, $m \in \{M\} \cup M_M^*$. Its input ports are given by $x_i \in \chi_m^{in} = \{x_0, \ldots, x_j\}$, $j \in \mathbb{N}$ and its output ports are given by $y_i \in \chi_m^{out} = \{y_0, \ldots, y_j\}$, $j \in \mathbb{N}$. Their interconnection is denoted by directed nets $n_i = (p_{src}, p_{snk}) \in N_M = N_M^A \cup N_M^D = \{n_0, \ldots, n_j\}$, $j \in \mathbb{N}$. Among these, the *assembly nets* $n_i \in N_M^A = \{n_i | (p_{src} \in \chi_{M_M^*}^{out}) \wedge (p_{snk} \in \chi_{M_M^*}^{in})\}$ denote the connections between the different subcomponents M_M^* of M. In contrast, the *delegation nets* $n_i \in N_M^D = \{n_i | ((p_{src} \in \chi_M^{in}) \wedge (p_{snk} \in \chi_{M_M^*}^{in})) \vee ((p_{src} \in \chi_{M_M^*}^{out}) \wedge (p_{snk} \in \chi_M^{out}))\}$ denote the connections between subcomponents M_M^* and the composed component M. Assuming both, the behavior of the components as well as their communication, to be compositional, a top-down refinement resp. bottom-up *virtual integration* of the composed behavior becomes possible, reducing the design complexity by a – possibly hierarchical – structural decomposition.

Hence, we consider a component M as $M = (tp(M), \chi_M, S_M, D_M, B_M)$, with χ_M, S_M, D_M, B_M being tuples or sets and $tp(M)$ denoting a function to resolve the component's type name. For the top-level of a decomposition the type $tp(M) = `M'$ is also considered to represent the component's instance name, normally given by $id(M_i) = `M_i'$ for the lower levels of a decomposition. Similar to the notation in Section 2, the component's port interface is defined by the set $\chi_M = \chi_M^{in} \cup \chi_M^{out}$ of input ports $x_i \in \chi_M^{in}$ and output ports $y_i \in \chi_M^{out}$. Besides a function $id(\cdot)$ to resolve a port's name, the declaration of each port $(\cdot)$ defines also functions $\nu(\cdot)$ to resolve its *value domain* – e. g. boolean or integer – and $dir(\cdot) \in \{in, out\}$ to resolve its *direction* as input resp. output.

Using an extended linear temporal logic, contracts $C_i := (A_i, G_i)$ are used, to formally specify the assumptions A_i of a component M w. r. t. to the timed behavior of its environment E, combined with its guarantees G_i, provided for the case that the corresponding assumptions A_i are satisfied. To this end, the assumptions describe expressions, which observe (read) only the *input* variables, while the guarantees are expressions, which control (write) only the *output* variables. Semantically, both expressions are interpreted as sets $[\![\mathrm{A}]\!]$ resp. $[\![\mathrm{G}]\!]$ of timed traces s_{x_i} resp. s_{y_i} with $[\![A]\!] = \{(s_{x_0}, \ldots, s_{x_j}) | (\bigcup_{i=0}^{j} x_i = \chi_M^{in}) \wedge ([\![A]\!] \models A)\}$, $[\![G]\!] = \{(s_{y_0}, \ldots, s_{y_j}) | (\bigcup_{i=0}^{j} y_i = \chi_M^{out}) \wedge ([\![G]\!] \models G)\}$, satisfying the corresponding assertions A resp. G, and with $s_{x_i} = \{e_0(x_i), e_1(x_i) \ldots\}$ resp. $s_{y_i} = \{e_0(y_i), e_1(y_i) \ldots\}$ describing the timed traces of x_i resp. y_i as possibly infinite sequences of *events* $e_\iota(x_i) = (v(x_i), t_\iota)$ resp. $e_\iota(y_i) = (v(y_i), t_\iota)$ with variable value $v(x_i) \in \nu(x_i)$ resp. $v(y_i) \in \nu(y_i)$, time $t_\iota : (t_\iota \in \mathbb{R}_0^+) \vee (t_\iota \in \mathbb{N})$ and $\iota \in \mathbb{N}_0^+$. Using the contracts' *saturated* interpretation $(A_i \rightarrow G_i')$, with $G_i' := (A_i' \rightarrow G_i)$ and $A_i' \subseteq A_i$, compositional assume/guarantee reasoning becomes possible to prove the compatibility and refinement within a component based decomposition.

Thus, we provide the component's contract based *specification* $S_M = \bigcup_i C_i$, which we onwards denote as behavioral specification. Accordingly, B_M describes the corresponding behavioral implementation, e. g. given as an executable program, automata or formula. Furthermore, the tuple $D_M = (M_M^*, N_M)$ describes the component's structural decomposition, either for the purpose of a structural top-down refinement of the initial specification as well as for the structural bottom-up *implementation* by instantiation and *integration* of available components. Finally, the norm $|\cdot|$ shall for all sets denote their number of elements and, if necessary for unambiguousness, we prefix identifiers and symbols by component identifiers M or M_i, using a dot as delimiter, as e. g. $M_0.C_0$, $M_1.C_0$, etc. For simplicity we identify components, contracts and ports by names equal to their symbols, so that $id(M) = `M'$, $id(M_i) = `M_i'$, $id(x_i) = `x_i'$, $id(y_i) = `y_i'$ etc.

5 Structural Contracts

In general, being based on interface theories, Contract Based Design is limited to such specifications S_M, declaring only the externally observable 'behavioral properties' of the component – meaning 'behavioral' in that sense, that its properties refer only to the components' explicitly declared ports. Differently, the component's inherently contained 'structural properties' can neither be specified nor verified that way – meaning 'structural' not necessarily w. r. t. the physical but w. r. t. the logical structure, such as available ports, the instantiated subcomponents or the interconnection of a structural decomposition D_M etc. As a solution, we suggest an introspection and reflection of these structural properties to introduce a structural point of view, enabling for structural contracts. According to the interface declaration $(\chi_M^{in}, \chi_M^{out})$ and the formal decomposition D_M we extract the available structural information and systematically add an implicit interface χ_M^{struc} of structural ports, which provide explicit access to the structural information. As a result, the component's decomposition structure becomes specifiable and verifiable via contract based constraints for its original interface declaration, structural decomposition resp. the instantiation and integration of the component within a hierarchical composition.

To explain our approach in detail, we follow its sequential steps according to:

1. extract structural information
2. build structural data types
3. insert introspection ports
4. add introspection components
5. add introspection subnets
6. add *structural guarantees*

First, the components' structural information $(tp(M), \chi_M, S_M, D_M, B_M)$ are derived from the component model, to build the data structure of M according to Sec. 4. In the second step – to avoid complete string analysis for the first approach – we generate *structural data types* according to the following enumerations:

dt_cId : set of all components identifiers $tp(M)$ and $id(M_i)$ $\forall M_i \in M_M^*$
plus one additional 'open' symbol to denote ports without a connection

dt_pId : set of all port identifiers $id(p_i) \in \chi_m$ of M and $M_i \in M_M^*$
plus one additional 'open' symbol to denote ports without a connection

Based on these structural data types, we then introduce the structural introspection ports according to Fig. 3. That is, for each port $p_i \in \chi_{M_i}$ of each subcomponent $M_i \in M_M^*$ of the decomposition D_M two additional input ports of type *dt_cId* resp. *dt_pId* are added. For each input port $p_i = x_i \in \chi_{M_i}^{in}$ the ports are named $M_i.id(x_i)\_cSrc$ and $M_i.id(x_i)\_pSrc$ resp. $M_i.id(y_i)\_cSnk$ and $M_i.id(y_i)\_pSnk$ for each output port $p_i = y_i \in \chi_{M_i}^{out}$. Using these additional ports the components are enabled to receive information about the connections $n_i \in \{N_M, \text{'}open\text{'}\}$ between their original ports, meaning the identifiers of the 'source component' and 'source port' resp. the 'sink component' and 'sink port' connected via the net n_i, resp. to receive 'open' if ports remained unconnected.

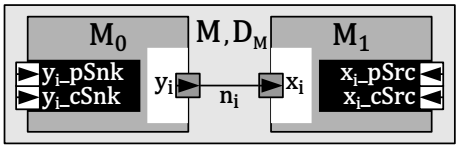

Fig. 3. Overview of the implicitly inserted structural introspection ports, extending the port interface of each subcomponent within a decomposition D_M of a component M.

In the fourth step, an additional introspection component M_{DM} is added to the decomposition structure – i. e. $M_M^* := M_M^* \cup M_{DM}$ – to reflect the component's structural information via the introspection ports. To this end, M_{DM} provides the structural output ports $M_{DM}.id(M_i.x_i)\_cSrc$, $M_{DM}.id(M_i.x_i)\_pSrc$, $M_{DM}.id(M_i.y_i)\_cSnk$ and $M_{DM}.id(M_i.y_i)\_pSnk$, building the corresponding counter part to the structural ports we introduced in step three.

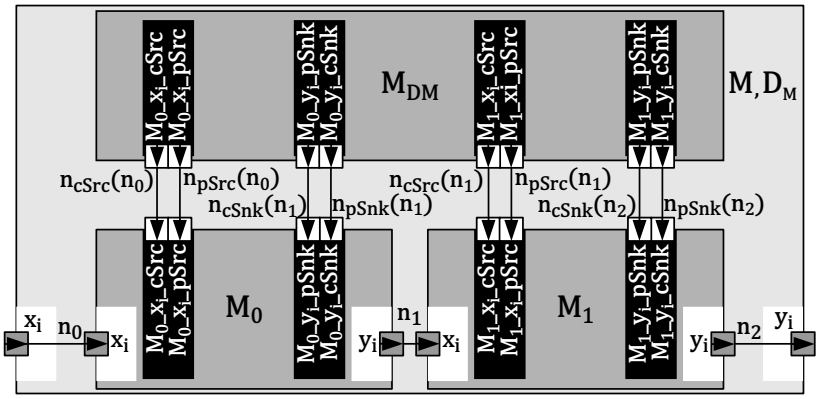

Fig. 4. Extension of a component's decomposition D_M by an introspection component M_{DM} and structural nets $n_{cSrc}(n_i)$, $n_{pSrc}(n_i)$, $n_{cSnk}(n_i)$, $n_{pSnk}(n_i)$, connecting the introspection component with the subcomponents.

In the fifth step, we complete the communication structure of the structural view according to Fig. 4, connecting the structural introspection ports of M_{DM} with the corresponding subcomponents $M_i \in M_M^*$. To this end, for all nets $n_i \in N_M$ additional subnets $N_{DM} = \bigcup_{n_i} n_{cSrc}(n_i) \cup \bigcup_{n_i} n_{pSrc}(n_i) \cup \bigcup_{n_i} n_{cSnk}(n_i) \cup \bigcup_{n_i} n_{pSnk}(n_i)$ are inserted to N_M, according to the following rules:

$$n_{cSrc}(n_i) = (M_{DM}.id(M_i.x_i)\_cSrc, M_i.id(M_i.x_i)\_cSrc),$$
$$n_{pSrc}(n_i) = (M_{DM}.id(M_i.x_i)\_pSrc, M_i.id(M_i.x_i)\_pSrc),$$
$$n_{cSnk}(n_i) = (M_{DM}.id(M_i.y_i)\_cSnk, M_i.id(M_i.y_i)\_cSnk),$$
$$n_{pSnk}(n_i) = (M_{DM}.id(M_i.y_i)\_pSnk, M_i.id(M_i.y_i)\_pSnk).$$

Finally, the introspection component M_{DM} is annotated with structural guarantees $C : ((A : true), (G : \langle struc\_port\rangle = \langle struc\_value\rangle))$, reflecting the information of the original component's interconnections with $struc\_value \in \nu(struc\_port)$ and:

$$struc\_port \in \bigcup_{x_i \in \chi^{in}_{M_i}} M_{DM}.id(M_i.x_i)\_cSrc \cup \bigcup_{x_i \in \chi^{in}_{M_i}} M_{DM}.id(M_i.x_i)\_pSrc$$
$$\cup \bigcup_{y_i \in \chi^{out}_{M_i}} M_{DM}.id(M_i.y_i)\_cSnk \cup \bigcup_{y_i \in \chi^{out}_{M_i}} M_{DM}.id(M_i.y_i)\_pSnk$$

Differently, for the subcomponents $M_i \in M^*_M$ the corresponding structural introspection ports allow to constrain the components' interconnections by structural assumptions $C : ((A : \langle struc\_port\rangle = \langle struc\_value^*\rangle), (G : true))$, with $struc\_value^* := \{\langle struc\_value\rangle | \langle struc\_port\rangle\}$, $struc\_value \in \nu(struc\_port)$ and:

$$struc\_port \in \bigcup_{x_i \in \chi^{in}_{M_i}} M_i.id(M_i.x_i)\_cSrc \cup \bigcup_{x_i \in \chi^{in}_{M_i}} M_i.id(M_i.x_i)\_pSrc$$
$$\cup \bigcup_{y_i \in \chi^{out}_{M_i}} M_i.id(M_i.y_i)\_cSnk \cup \bigcup_{y_i \in \chi^{out}_{M_i}} M_i.id(M_i.y_i)\_pSnk$$

That way, the structural information of a decomposition are treated as properties, which are provided from the compositional environment which embedds the instantiated components, consequently allowing for a contract based assume-guarantee reasoning in this structural view. Following from this, structural assumptions can be specified top-down, becoming part of the functional and extra-functional contracts and thus an additional validity constraints during the compatibility and refinement checking within multiple viewpoints.

6 Proof of Concept

For a first proof of concept, we evaluated our approach of structural contracts for the motivating example, outlined in Sec. 2. To this end, we implemented the component interfaces of M and its subcomponents M_0, M_1 and M_2 and provided them with contracts according to the functional and extra-functional properties given in Tab. 1. Based on this implementation we showed that structural contracts are able to reveal the false negative verification of the erroneous logical structure depicted in Fig. 2. Furthermore, we showed that for a correct structural decomposition our structural extension does not influence compatibility and refinement checking of the other functional and extra-functional properties. For the implementation and evaluation we used *OTHELLO (Object Temporal with Hybrid Expressions Linear-Time LOgic)* [8] for the specification of contracts and *OCRA (OTHELLO Contracts Refinement Analysis)* [7] to describe the components as *OSS (OCRA System Specification)* [7] and to check their compatibility and their refinement. To reproduce our study, our example is online available at [10].

7 Conclusion

Based on an artificial example we motivate the need for structural contracts and show how structural contracts increase the reliability of composed extra-functional multi-domain models. By the evaluation of the example we show that structural contracts can reveal failures in the logical composition structure, which otherwise remain hidden, enabling false negatives during extra-functional verification.

In the future work, we want to investigate the abstraction and refinement of our structural contracts and evaluate how structural contracts can be propagated throughout the design and abstraction hierarchies. Furthermore, we plan to examine if and which additional port annotations will become necessary enable this hierarchies and to allow for the seamless and composable integration of designer-defined extra-functional semantics based on structural contracts.

References

1. de Alfaro, L., Henzinger, T.: Interface-based design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, T. (eds.) Engineering Theories of Software Intensive Systems, NATO Science Series, vol. 195. Springer, Netherlands (2005)
2. Alfaro, L.d., Henzinger, T.A.: Interface theories for component-based design. In: Proceedings of the First International Workshop on Embedded Software (EMSOFT). Springer, London, UK (2001)
3. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T., Larsen, K.: Contracts for systems design. Technical Report RR-8147, Research Centre Rennes – Bretagne Atlantique, Rennes Cedex (2012)
4. Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008 - Concurrency Theory, Lecture Notes in Computer Science, vol. 5201. Springer (2008)
5. Bliudze, S., Sifakis, J.: Synthesizing glue operators from glue constraints for the construction of component-based systems. In: Proceedings of the 10th International Conference on Software Composition (SC). Springer, Zurich, CHE (2011)
6. Cardelli, L.: Type systems. ACM Comput. Surv. 28(1) (1996)
7. Cimatti, A., Dorigatti, M., Tonetta, S.: Ocra: A tool for checking the refinement of temporal contracts. In: 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2013)
8. Cimatti, A., Roveri, M., Susi, A., Tonetta, S.: Validation of requirements for hybrid systems: A formal approach. ACM Transactions on Software Engineering and Methodology (TOSEM) 21(4) (2012)
9. Lee, E.A., Sangiovanni-Vincentelli, A.L.: Component-based design for the future. In: Design, Automation & Test in Europe (DATE) (2011)
10. Nitsche, G.: Structural contracts - conceptual example in OCRA, `https://vhome.offis.de/gnitsche/paper/iees2015/example/`
11. Sangiovanni-Vincentelli, A., Damm, W., Passerone, R.: Taming dr. frankenstein: Contract-based design for cyber-physical systems. European Journal of Control 18(3) (2012)
12. Zhao, Y., Xiong, Y., Lee, E.A., Liu, X., Zhong, L.C.: The design and application of structured types in ptolemy ii. Int. J. Intell. Syst. 25(2) (2010)