Abstract—We present an OCL framework and tool for the description and verification of consistency rules of extra-functional properties (EFPs) in component and connector (C&C) models. The framework is based on our previously defined structure of EFP consistency rules using selection, aggregation, and comparison operators, and provides C&C specific OCL functions and configurations that allow engineers to succinctly express EFP consistency rules for C&C models. Further, the extension of OCL is twofold. First, constraints may contain C&C specific expressions and second, expressions natively support measurement units as required in specifications of EFPs. We have extended the OCL verification process to support the novel extensions and to automatically generate meaningful positive and negative witnesses for consistency and inconsistency. We implemented the approach within the MontiCore framework for the C&C modeling language MontiArc. Initial evaluation shows that it is expressive and scales to large, industrial sized C&C models.

I. INTRODUCTION

Component and connector (C&C) models are used in many application domains of software engineering, from cyber-physical and embedded systems to web services and enterprise applications. The structure of C&C models consists of components at different containment levels, their typed interaction points, and connectors between them [13]. In addition to functional properties, extra-functional properties (EFPs) play an important role in the development of C&C models [6], [19]–[21]. Important examples of EFPs include worst-case-execution-time (WCET), memory and power consumption, security properties, and traceability [1], [16], [18].

Recently, we have presented a non-invasive technique for adding EFPs to C&C models by means of tagging languages [12]. Importantly, this technique requires no changes to the meta-model of the C&C language. In addition, we have suggested a generic structure for EFP consistency rules based on selection, aggregation, and comparison operators. Conceptually, a consistency rule states for a C&C element and an EFP whether the EFP value is consistent with the EFP values of other C&C elements. Technically, consistency rules are constraints that are satisfied iff the EFP value is consistent. Consistency is determined by comparison to aggregated EFP values of related C&C elements as defined by the selection, aggregation, and comparison operators of consistency rules.

In this paper we are interested in the concrete formalization of EFP consistency rules and in their automated analysis. We propose to leverage the popular and expressive Object Constraint Language [25] (OCL) and its analysis capabilities for the definition and analysis of EFP consistency of C&C models.

Our first contribution is a C&C-specific extension of OCL. The extension allows the definition of C&C consistency constraints based on clean mathematical C&C definitions; all implementation specific details, such as more complex types from the meta-model or abstract-syntax representations are abstracted away by providing C&C specific OCL configurations and powerful type-inference mechanisms when checking consistency.

Our second contribution extends OCL with support for measurement units and an automatic mechanism to produce witnesses for EFP consistency checking results. First, many EFPs, describing physical properties of C&C models, e.g., WCET, power consumption, or memory usage, contain physical units, e.g., seconds, Watts, or sizes. Thus, to enable domain experts define EFP consistency rules, a constraint language for EFPs naturally should support automatic unit comparison and conversion. Second, our previously defined mathematical framework [12] structures the definition of consistency rules. This general structure allows to automatically generate positive and negative witnesses for consistency checking results. These witnesses intuitively demonstrate the reasons for consistency or inconsistency of a C&C model.

Our third contribution is an initial evaluation of checking EFP consistency of an industrial sized C&C model. Specifically, we used our tool to verify four selected consistency rules on an industrial sized Autonomous Driver Assistance System (ADAS) model. The ADAS model consists of about 1,400 components and 4,500 ports, and 2,800 tagged EFP values. Our examples and evaluation show that (i) our OCL based approach is expressive enough for defining complex consistency constraints, and (ii) that the performance of our implementation for checking these consistency rules scales to large models.

In the next section we present a running example. Sect. III gives background on C&C models and consistency rules of EFPs. Sect. IV and Sect. V present our OCL framework and example constraints, Sect. VI presents the implementation, and Sect. VII presents a case study in which the framework is used. Sect. VIII present related work, and Sect. IX concludes.

II. RUNNING EXAMPLE

Consider the sensor block of a weather balloon system as shown in Fig. 1 (for textual syntax, see Fig. 2). The sensor...
We provide background on C&C models, extra-functional properties, and their consistency rules.

A. Component and Connector Models

Component and connector models describe components, their points of interaction, and their hierarchical composition. We repeat a definition of C&C models as, e.g., given in [11], in Def. 1 which represents the essence of component models [13] as formalized by ADLs ACME [4], AADL [3], and MontiArc [7], or in tools AutoFOCUS [8] and Simulink [24].

Definition 1 (Component and Connector model [11]): A C&C model is a structure \( ccmn = \langle Cmps, Ports, Cons, Types, subs, ports, type \rangle \) where

- \( Cmps \) is a set of named components, \( cmp \in Cmps \) has a set of ports \( ports(cmp) \subseteq Ports \) and a (possibly empty) set of immediate subcomponents \( subs(cmp) \subseteq Cmps \),
- \( Ports = InPorts \cup OutPorts \) is a disjoint union of input and output ports where each port \( p \in Ports \) has a name, a type \( type(p) \in Types \), and belongs to exactly one component \( p \in ports(cmp) \),
- \( Cons \) is a set of directed connectors \( con \in Cons \), each of which connects two ports \( con.src, con.tgt \in Ports \) of the same type, which belong to two sibling components or to a parent component and one of its immediate subcomponents, and
- \( Types \) is a finite set of type names.

C&C models from Def. 1 are well-formed iff no component is its own (transitive) subcomponent and has at most one direct parent and subcomponents are connected legally (see [11] and [17] for complete definitions).

While some formalisms directly express C&C models, e.g., [8], [24], others provide C&C type definitions and their instantiation to define C&C models, e.g., [3], [7].

Definition 2 (Component Type Definition [17]): A component type definition is a structure \( ct = \langle cType, CPorts, CSubs, CCons \rangle \in CTDefs \) where

- \( cType \) uniquely identifies the component type,
- \( CPorts \) is a set of input and output port definitions where each port \( p \in CPorts \) has a name and a type,
- \( CSubs \subseteq Name \times CTDefs \) is a set of named subcomponent declarations, and
- \( CCons \) is a set of directed connector definitions \( con \in CCons \), each of which connects two port definitions \( con.src, con.tgt \) of the same type, which belong to two
instances must be at most the certificates common to all ports.

A component type \( t \in CTDefs \) is instantiated to a C&C model by creating a component for \( c \) with subcomponents, that are instances of \( t.CSubs \) with connectors according to \( t.CCons \). For a more detailed definition including well-formedness rules and instantiation see [17]. For an instantiated \( cmp \in Cmps \) the function \( CType(cmp) \) returns the corresponding component type.

B. Consistency of EFPs in C&C Models

It is important to note that the consistency of an EFP value may depend on multiple other C&C model elements, their relations, and their EFP values. Some advanced examples of consistency relate to component instantiation and composition in C&C models. In addition, consistency may be very specific to the EFP type, e.g., checking consistency of WCET values is different from checking memory consumption.

To address the challenge of ensuring consistency of EFP values, in a previous paper [12] we defined a general structure of EFP consistency rules shown in Def. 3. First, each rule defines what EFP value of which kind of C&C model element it checks. Second, the rule specifies how to select relevant C&C model elements for the check. Third, the rule defines how to aggregate tagged values over the selected elements. Finally, the aggregated value is compared to the value of the checked element, to determine its consistency.

**Definition 3 (EFP Value Consistency Rule [12]):** A consistency rule is a structure consisting of:

- **checks** name of tag and element checked by rule;
- **selection** selects relevant C&C elements to check consistency;
- **aggregation** aggregates values of selected elements; and
- **comparison** compares values to decide consistency.

The next two subsections illustrate some examples on consistency definition rules according to Def. 3. More examples are available in [12], Sect. III-B1 presents rules for the consistency of EFP values in the context of component type instantiation. Sect. III-B2 presents rules in the context of composition.

1) **Instantiation Consistency:** Instantiation consistency checks whether the EFPs of component instances are consistent to the EFPs of their component type definitions.

**Rule 1 (InstPower [12]):** The power consumption of an instance is at most the power consumption of its type:

- checks: tag power of \( cmp \in Cmps \)
- selection: \( S := CType(cmp) \in CTDefs \)
- aggregation: \( v := \sum_{sct.power \in S} \)
- comparison: \( v \leq ct.power \)

**Rule 2 (InstCertificates):** The certificates of component instances must be at most the certificates common to all ports of the component:

- checks: tag cert of \( cmp \in Cmps \)
- selection: \( P := cmp.ports \)
- aggregation: \( v := \bigcap_{p \in P} p.cert \)
- comparison: \( v \geq cmp.cert \)

2) **Composition Consistency:** Composition consistency checks whether the EFPs of C&C model elements are consistent across their composition. The following example rules address consistency at the type level. Similar rules can be defined at the instance level.

**Rule 3 (CompPower [12]):** The combined power consumption of all subcomponents is at most the power consumption of the composed component:

- checks: tag power of \( ct \in CTDefs \)
- selection: \( S := cts.CSubs \)
- aggregation: \( v := \min_{\text{number}(asil) <= 5} \)
- comparison: \( v \geq \text{number}(cmp.asil) \)

**Rule 4 (ASIL):** The ASIL (Automotive Safety Integrity Level) of all subcomponents must be higher or equal than the ASIL of the composed component:

- checks: tag asil of \( cmp \in Cmps \)
- selection: \( S := cmp.subs \)
- aggregation: \( v := \max \) \( \sum_{\text{sc.wcet} \in S} \)
- comparison: \( v \leq cmp.wcet \)

**Rule 5 (WCET Single Core):** The WCET of a component instance is at most the WCET of its subcomponent instances.

- checks: tag wcet of \( cmp \in Cmps \)
- selection: \( S := cmp.subs \)
- aggregation: \( v := \sum_{\text{sc.wcet} \in S} \)
- comparison: \( v \leq \text{cmp.wcet} \)

**Rule 6 (WCET Multi Core):** The WCET of a component instance is at most the maximum of the WCET of parallel executable direct subcomponent paths.

Note that our framework does not limit the selection, aggregation, and comparison operators in any way. Our previous paper [12] contains more examples with different and more complicated operators.

IV. OCL FRAMEWORK FOR C&C MODELS

This section presents our first contribution: a C&C-specific extension of OCL.

We start with an example. The top part in Fig. 3 presents an OCL invariant expressing that the source and target ports of a connector have the same data type (l. 3). Typically, OCL constraints are written against a concrete metadata; this, however, makes the OCL constraints dependent on a specific language implementation. As an example, our C&C language implementation MontiArc provides references to all connectors in a C&C model by:

```
List<ConnectorSymbol> connectors = globalScope.resolve(ConnectorSymbol.KIND) [14].
```

1Q = 0, ASIL A = 1, ASIL B = 2, ASIL C = 3, ASIL D = 4

2directSubCmpPaths(cmp) computes the set of all paths from a subcomponent of \( cmp \) to an output port of \( cmp \).
language-implementation-specific OCL constraint would reference the type ConnectorSymbol in Fig. 3 l. 3 rather than the general C&C type Con from Def. 1.

To address this problem, we decided to extend our OCL implementation to support importing of configuration files. A configuration file serves as a bridge from the mathematical definitions to the implementation-specific classes. Configuration files contain rewrite rules that rewrite names of C&C elements from Def. 1 and Def. 2 to implementation-specific OCL expressions. This concept is generic and extensible to multiple target implementations. The C&C model OCL constraints can be reused with different C&C meta-models as long all rewrite rules can be expressed using corresponding meta-model-specific OCL expressions.

The bottom part in Fig. 3 shows an excerpt of our MontiArc specific configuration file, which provides C&C specific functions in OCL constraints. It imports all required, implementation-specific Java classes (Fig. 3 bottom, l. 5) and contains rewrite rules (ll.7-10), similar to C preprocessor’s #define macros, to map C&C domain names to implementation-specific names. In addition to rewrite rules, the configuration file imports and provides graph functions (Fig. 3 bottom, l. 6) with C&C specific method implementations. For example, the function path computes all C&C elements (components, ports, and connectors) on a path connecting an input to an output port. As another example, these graph functions can easily express the OCL invariant that each output port depends on at least one input port.

The complete configuration file for rewriting C&C concepts into MontiArc concepts consists of 11 rewrite rules and imports 12 helper functions for use in C&C model specific constraints. There are two kinds of helper functions. The first kind consists of graph-based functions that return a graph or chains of C&C elements connected (directly or indirectly) by two given C&C elements; they come in different versions to return only specific C&C elements (e.g., only components, or only components and ports), to avoid creating large graphs with unnecessary elements and accelerate the verification process. The second kind of helper functions are mathematical helper functions for EFPs such as min, max, sum, prod, union, or intersection.

Fig. 3: Top: OCL Expression for same port type of connected ports. Bottom: Excerpt of &C MontiArc specific configuration file.

V. DEFINING EFP CONSTRAINTS IN OCL

This section presents our second contribution: extending OCL with support for measurement units in order to allow modeling of EF constraint constraints in OCL and to automatically generate witnesses for OCL constraint consistency or inconsistency. The OCL constraints follow the selection, aggregation, and comparison structure of EF value consistency rules (Def. 3) we introduced in [12].

We structure this section by examples presenting different features of our OCL framework. The examples we show are an OCL constraint for InstPower (Rule 1) with a generated positive witness, an OCL constraint for CompPower (Rule 3) with a generated negative witness, and finally an OCL constraint for WCET Multi Core (Rule 6), which is the most complex example based on connection properties of C&C models.

A. Example: Power Consumption for Component Instantiation

Fig. 4 shows the complete OCL code for instantiation consistency of EF power (Rule 1). Lines 1 and 2 import the configuration files for C&C specific extensions (see Sect. IV above) as well as for EF extensions containing special helper functions (such as min, max, or sum) with units as parameters. Including the configuration file EFExt.conf ensures that all OCL constraints follow the structure of Def. 3 with a check (Fig. 4 l. 4), selection (selectedValue in Fig. 4 l. 6), aggregation (aggregatedValue in Fig. 4 l. 7), and a comparison (Fig. 4 ll. 9-10) element. Component instances without power tags are assigned the value 0W (Fig. 4 l. 7) and component type definitions without power tags are assigned the value +ooW (Fig. 4 l. 10). These interpretations are up to the engineers defining the consistency rule.

All components in our example C&C model in Fig. 1 satisfy the OCL constraint from Fig. 4. In case of satisfaction our tool can generate a positive witness. Fig. 5 shows a (textual) positive witness that component gps1 satisfies the OCL constraint from Fig. 4 its power consumption is below
The power consumption of its type GPS. Note that our OCL engine automatically uses the correct interpretation of the different units mW in tag of GPS and W in tag of gps1.

The witness is meant to help the engineer in understanding the reasons for consistency. It contains the checked C&C element, i.e., component instance gps1 (Fig. 5 l. 6), the selected elements to check against, i.e., component type GPS (l. 5), and all EFP tags used for the computation of the aggregated value (l. 3) and in the comparison step.

Witness generation is completely automated for the satisfaction or violation of consistency rules. Our tool relies on the structure of the OCL constraints following Def. 3 to construct meaningful witnesses. Concretely, the witnesses contain the checked element (context of OCL invariant), the elements selected for comparison (OCL variable selectedValue(s), or selectedPaths), all EFP tags appearing in the aggregation (computation of OCL variable aggregatedValue(s)), and all EFP tags appearing in the final comparison. To further demonstrate satisfaction or violation, the aggregated value is shown within the witness.

B. Example: Power Consumption for Component Definitions

Fig. 6 shows the complete OCL code for composition consistency of EFP value power (Rule 3).

Our running example does not satisfy the consistency rule in Fig. 6 and a negative witness is generated for component type definition WBalloonSens. The graphical representation of the generated negative witness in Fig. 7 shows the reason for inconsistency: the aggregated value of 5.41 W (400mW + 2500mW + 2500mW + 10mW) is greater than the power tag value of component type definition WBalloonSens.

C. Example: Worst-Case Execution Time

Fig. 8 shows the complete OCL code for composition consistency of EFP value wcet (Rule 6 multi core).

The OCL variable selectedPaths in l. 6 stores a list of paths, which are lists of subcomponents. For each path in the list, l. 7-9 calculate the WCET value by summation of the values on the path, and l. 11-12 compare each value in the list against the WCET value of the parent component. Storing the list instead of the maximal value in the aggregatedValues variable makes generated witnesses easier to understand: witnesses will contain all paths instead of only one value.

The positive witness that component WBalloonSens satisfies the OCL constraint of Fig. 8 contains the following selectedPaths with their aggregatedValues:

- wBalloonSens → controller → wBalloonSens with 50ms ≤ 2s
- temp→controller→wBalloonSens with 550ms ≤ 2s
- gps1→controller→wBalloonSens with 1000ms ≤ 2s
- gps2→controller→wBalloonSens with 800ms ≤ 2s

VI. IMPLEMENTATION

Our consistency verification tool takes a textual MontiArc C&C model, EFP tag files tagging the C&C model, and an OCL constraint as input. The verification process is fully automated and consists of the following steps: (1) process the textual model (parsing text to an AST and creating symbol table) using the MontiCore framework [9]; (2) process the textual OCL constraint; (3) execute all rewrite rules of all imported configuration files; (4) generate Java code from the OCL constraint based on the AST and symbol table; and (5) execute the generated Java code to check the C&C model for consistency and to generate witnesses.

One challenge for implementing the generation of Java code from OCL was that while OCL is a type-less language, Java requires types for every declaration. Our generator infers the variable types from OCL by looking up the return types of constructs (e.g., l. 7-9 in Fig. 8).

Another challenge was adding a unit concept to OCL. Many EFPs contain values with measurement units. Thus we developed an additional language SIUnit, which our
OCL implementation extends. SIUnit is capable of parsing imperial, SI, and unofficial SI units with metric prefixes and derived constructs, e.g., 28.2km/h and 9.81m*s⁻². It supports additional symbols for plus and minus infinity. For unit conversions we use the JScience framework.

VII. Case Study on ADAS

In our case study we want to evaluate the following two research questions:

**RQ1** Does our framework scale to large industry provided C&C models?

**RQ2** Which steps of automated EFP value consistency checking are most time consuming?

### A. Case Study Data:

For our case study we used a C&C model of an Autonomous Driver Assistance System (ADAS) provided by Daimler AG. The available model did not contain EFP values. Therefore, we derived values for EFP wcet and power tags from reference materials and by evaluation as follows. To approximate realistic EFP values for wcet, we have translated the Simulink models to C-Code, compile this C-Code to executable code, i.e., checking the 119 or 1,396 components, and witness generation, ranges from 0.1s for single-value aggregations to 183s for nested lists aggregations. And running times for verification and witness generation are reasonable running times.

<table>
<thead>
<tr>
<th>EFP</th>
<th>Model</th>
<th>Parsing C&amp;C Model</th>
<th>Parsing OCL</th>
<th>OCLJava Generator</th>
<th>Java compilation</th>
<th>Verification/ Witness Generation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstPower</td>
<td>small complete</td>
<td>1.6s 3.3s 0.2s</td>
<td>0.5s 0.1s</td>
<td>5.7s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>complete</td>
<td>3.2s 3.3s 0.2s</td>
<td>0.5s 0.1s</td>
<td>7.3s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CompPower</td>
<td>small complete</td>
<td>1.6s 10.0s 0.2s</td>
<td>0.5s 3.5s</td>
<td>15.7s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>complete</td>
<td>3.2s 10.0s 0.2s</td>
<td>0.5s 3.5s</td>
<td>20.7s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCET</td>
<td>small complete</td>
<td>1.6s 16.3s 0.2s</td>
<td>0.5s 19.0s</td>
<td>38.0s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>complete</td>
<td>3.2s 16.3s 0.2s</td>
<td>0.5s 19.0s</td>
<td>39.0s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCET</td>
<td>small complete</td>
<td>1.6s 34.0s 0.2s</td>
<td>0.5s 11.0s</td>
<td>47.0s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>complete</td>
<td>3.2s 34.0s 0.2s</td>
<td>0.5s 11.0s</td>
<td>47.0s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This leads us to the following answers to our research questions. (**RQ1**) We can automatically verify the consistency of EFP values of an industrial sized model in reasonable running times. (**RQ2**) It is interesting to see that OCL processing times and running times for verification and witness generation are consuming most of the time. However, the OCL processing step for code generation has to be executed only once (and every time a consistency rule changes). Finally, one may separate verification from witness generation in order to speed up consistency checking when witnesses are not required.

For inspection and further research we make all mentioned C&C models and OCL constraints used in our case study available at: http://www.se-rwth.de/materials/OCLVerifyTool4CnCEFP.zip

VIII. Related Work

Acme [5] allows one to specify first-order predicates on the structure of architectures, dealing, e.g., with connectedness of components. Our extension of OCL for C&C models provides similar features. However, to the best of our knowledge the constraint language of Acme does not support EFPs.

Grunske [6] presents an evaluation framework for EFPs, consisting of four elements. We focus on defining and checking consistency rules rather than on evaluation of EFPs. Nevertheless, Grunske [6] also observed the need for a general language to formulate evaluation of different EFP types.

Sentilles et al. [21] present a meta-model for integrating non-functional properties into C&C models. They focus on handling multiple EFPs for the same entity and did not implement rules for EFP consistency.

Cicchetti et al. [2] introduce a framework for evolution of EFP values and present, as an example, how the change of the WCET of a component requires updating the WCET of its parent component. To define validity conditions, they use the Epsilon Validation Language [27], which is similar to OCL. It is possible to extend our framework to support similar evolution scenarios.

Leveque and Sentilles [10] present refinement of EFPs through instantiation and subtyping of components. Engineers can use OCL constraints to filter EFP attributes of components. The OCL constraints are written against the meta-model of the C&C implementation. We use C&C-specific OCL constraints to define consistency rules beyond component EFP refinement.
Finally, Sapienza et al. \cite{20} present EFP consistency rules for component composition. The tools used in \cite{20} and a related case study \cite{22} go beyond information stored with model elements, e.g., they measure EFP values by simulation. It is unclear whether their approach can be easily extended to additional or alternative EFP constraints.

IX. CONCLUSION

We presented an OCL framework and tool for the description and verification of consistency rules of EFPs in C&C models. The work includes a C&C-specific extension of OCL with native support for measurement units, and an automatic mechanism for verifying consistency and producing witnesses for EFP consistency checking results. We implemented the approach within the MontiCore framework for the C&C modeling language MontiArc. Our initial evaluation on an industrial sized model shows that it is expressive and scales to large C&C models. Our framework works for comparable extra-functional properties, these include quantitative properties (e.g., worst-case execution time, power, or memory consumption) and qualitative properties (e.g., traceability, portability flags, ASIL security, or encryption levels).

Acknowledgements This research was supported by a Grant from the GIF, the German-Israeli Foundation for Scientific Research and Development.

REFERENCES

\cite{1} J. P. Cavano and J. A. McCall. A framework for the measurement of software quality. SIGSOFT Softw. Eng. Notes, 3(5), 1978.
\cite{3} P. H. Feiler and D. P. Gluch. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley, 2012.
\cite{19} M. Saadatmand, A. Cicchetti, and M. Sjödin. UML-based modeling of non-functional requirements in telecommunication systems. In ICSEA, 2011.