2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems

Why Is My Component and Connector Views Specification Unsatisfiable?

Shahar Maoz, Nitzan Pomerantz, Jan Oliver Ringert, Rafi Shalom
School of Computer Science, Tel Aviv University, Israel

***Abstract*—Component and connector (C&C) views specifications, with corresponding verification and synthesis techniques, have been recently suggested as a means for formal yet intuitive structural specification of component and connector models. One challenge for effective use of C&C views synthesis relates to the case where the specification is unsatisfiable.**

In this work we present an approach to deal with unsatisfiable C&C views specifications. First, we define a notion of a C&C views specification core, a locally minimal unsatisfiable subset of the views specification. Second, based on the core, we generate explicit, concrete, structured natural-language report, which explains the cause of unsatisfiability. Finally, we extend our work to support specifications with architecture styles, library components, and Boolean formulas beyond simple conjunctions.

Our views core computation relies on a new translation to SAT, via Alloy, which is refined enough to allow the extraction of detailed explanations. We implemented our work and evaluated it using 12 synthetic and real-world C&C views specifications. The evaluation examines the cost of the core computation and its effectiveness in reducing the size of the specification.

I. Introduction

Component and Connector (C&C) models, described using languages such as SysML [17] and AADL [1], [8], are used extensively in software and systems engineering. Recently, we have presented C&C views [13], as a means to formally and intuitively specify constraints on the structure of C&C models. The views allow engineers to specify constraints on hierarchy and connectivity, using partial examples, while crosscutting the implementation-oriented system/sub-system hierarchy of the target model. The verification problem of checking a C&C model against a view was investigated in [14]. The synthesis problem of automatically generating a C&C model satisfying a given views specification, if one exists, was studied in [13].

One challenge for C&C views synthesis concerns the case where the views specification is unsatisfiable. Since in this case no C&C model can be synthesized, the engineer who is informed about the unsatisfiability of her views specification is left in the dark as to the problems in it and the causes for its unsatisfiability.

We aim to address this challenge using two different powerful tools: a well-defined locally minimal unsatisfiable subset of the specification and a comprehensive natural-language report. Both have to be provided in a fully automated way; the engineer need not be aware of the details of the performed computations.

Specifically, in this work we make the following contributions. First, **we define a notion of a C&C views core, and show how to compute it**. A views core consists of a minimal conjunctive specification over a subset of views from the original specification, where each of the views, by itself, is stripped down to contain only a minimal subset of its components, connectors, and ports.

Second, **based on the core on the one hand and the semantics of views on the other hand, we generate explicit, concrete, structured natural-language explanations about the root cause of unsatisfiability**. The generated explanation lists the unsatisfiability problem and the names of the concrete components and/or ports involved in it. As such, it provides most clear and precise explanation for the reason of the unsatisfiability at hand.

Finally, **we extend our work to support specifications with architecture styles, library components, and Boolean formulas beyond conjunctions. These extensions are important for the more general applicability of our work**.

To implement and automate our ideas, we use Alloy [12] as an intermediate language on the way to SAT. This design choice is based on Alloy's readability, which helps us in accelerating our work and in gaining confidence in its correctness. Naively, one would hope to combine the translation of [13] with Alloy's support for UNSAT core [19] and obtain a C&C views core. However, an Alloy core is a minimal set of facts and predicates, while a C&C views core is a minimal set of partial views; the naive approach fails. Although the translation of [13] is correct, it is not *regular* in the sense of [19] and thus the algorithm of [19] sometimes fails to produce C&C views cores based on this translation. We have thus revised and refined the translation from [13] so that the UNSAT core result can be effectively mapped back to the C&C views level. We consider this refined translation to be an important technical contribution of our work (see Sect. IV-C).

Finally, we note that we fully automated all analysis using Alloy's APIs and embedded SAT solver; the engineer need not see the generated Alloy module and SAT formula.

We have performed an evaluation of our work using 12 synthetic and real-world views specifications, taken from [1], [2], [10]. The evaluation examines the C&C view core cost in computation time and its effectiveness in reducing the size of the specification.

II. Preliminaries

A. C&C Views Specifications

We provide background on the syntax and semantics of C&C views, as presented in [13].

978-1-5386-3492-9/17 $31.00 © 2017 IEEE
DOI 10.1109/MODELS.2017.26

Definition 1 (Component and Connector model [13]): A C&C model is a structure $m = \langle Cmps, Ports, Cons, Types, subs, ports, type \rangle$ where

- $Cmps$ is a set of named components, each of which has a set of ports $ports(cmp) \subseteq Ports$ and a (possibly empty) set of immediate subcomponents $subs(cmp) \subset Cmps$,
- $Ports$ is a disjoint union of input and output ports $Ports = PortsIn \cup PortsOut$ 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, each of which connects two 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.

Some additional well-formedness rules apply [15], e.g., that the subcomponents relation is a strict partial order, that every port has at most one incoming connector, and that port names are unique within their component.

Definition 2 (Component and Connector view [13]): A C&C view is a structure $v = \langle Cmps, Ports, AbsCons, Types, subs, ports, type \rangle$ where

- $Cmps$ is a set of named components, each of which has a (possibly empty) set of ports $ports(cmp) \subseteq Ports$ and a (possibly empty) set of subcomponents $subs(cmp) \subset Cmps$,
- $Ports$ is a disjoint union of sets of input and output ports $Ports = PortsIn \cup PortsOut$ where each port $p \in Ports$ has a (possibly unknown) name, a (possibly unknown) type $type(p) \in Types \cup \perp$, and belongs to exactly one component $p \in ports(cmp)$,
- $AbsCons$ is a set of abstract connectors, each of which connects components (optionally) via ports of the same type or an unknown type, and
- $Types$ is a finite set of type names.

Note that in a C&C view, abstract connectors are not required to connect only two sibling components or a parent component and one of its immediate subcomponents. Again, the subcomponents relation is a strict partial order.

A C&C model satisfies a C&C view iff the types, components, and ports mentioned in the view are contained in the model, the model respects the subcomponent relation induced by the view, two ports connected by an abstract connector in the view are connected by a chain of connectors in the model (respecting direction, names, and types), and all component's ports in the view belong to the same component in the model with corresponding name, type, and direction. More formally:

Definition 3 ($m \models v$ [13]): A C&C model m satisfies an C&C view v iff:

- $v.Types \subseteq m.Types$, $v.Cmps \subseteq m.Cmps$, $v.Ports \subseteq m.Ports$,
- $\forall cmp_1, cmp_2 \in v.Cmps$: $cmp_1 \in v.subs(cmp_2)$ iff $cmp_1 \in m.subs^+(cmp_2)$ (we use $^+$ to denote the transitive closure),
- $\forall ac \in v.AbsCons$ $\exists$ chain of connectors in m, $c_1, \ldots, c_n$ with $ac.srcCmp = c_1.srcCmp$ and $ac.tgtCmp = c_n.tgtCmp$ with matching port names and types, if specified, and
- $\forall cmp \in v.Cmps$:
  (1) $v.ports(cmp) \subseteq m.ports(cmp)$, and
  (2) $\forall p \in v.ports(cmp)$: $p \in v.PortsIn$ iff $p \in m.PortsIn \wedge v.type(p) \in \{\perp, m.type(p)\}$
  (similarly for unknown and given port names).

Definition 4 (C&C views specification [13]): A C&C views specification (S, V) is a Boolean expression S over a set of C&C views V. A C&C model m satisfies a specification (S, V) iff replacing each view v in S with the value of $m \models v$ satisfies S: $m \models S \Leftrightarrow S[v/(m \models v)]_{v \in V}$.

B. Alloy

Alloy [11], [12] is a textual modeling language based on relational first-order logic. An Alloy module consists of signature declarations, fields, facts and predicates. Each signature denotes a set of atoms, which are the basic entities in Alloy. Relations between two or more signatures are represented using fields and are interpreted as sets of tuples of atoms. Facts are statements that define constraints on the elements of the model. Predicates are parametrized constraints. A predicate can be included in other predicates or facts.

Alloy modules can be analyzed using Alloy Analyzer, a fully automated constraint solver. This is done by a translation of the module into a Boolean expression, which is analyzed by SAT solvers embedded within the Analyzer. The analysis is based on an exhaustive search for instances of the module, bounded by a user-specified scope, which limits the number of atoms for each signature in an instance of the system that the solver analyzes. For a complete and detailed account of Alloy see [12].

C. UNSAT Core

Some work study the problem of finding unsatisfiable cores of unsatisfiable constraints written as propositional satisfiability formulas [6], [21]. Given an unsatisfiable CNF formula, a minimal unsatisfiable sub-formula, a minimal core, is a subset of its clauses that is both unsatisfiable and minimal, i.e., any subset of it is satisfiable. There may be many independent reasons for a formula's unsatisfiability and hence more than one core, but extracting all of them is computationally expensive.

Torlak et al. [19] proposed an efficient algorithm for the extraction of a single core of declarative specifications based on the resolution refutation proofs generated by SAT solvers and theorem provers. The Recycling Core Extractor algorithm (RCE), returns an unsatisfiable core of specifications written in the Alloy language that is guaranteed to be sound (constraints not included in the UNSAT core are definitely irrelevant to the unsatisfiability proof) and irreducible (removal of any constraint from the set would make the remaining formula satisfiable). Cores have been shown to be useful in the identification of over-constrained models, weak theorems, and insufficient scopes while checking models. In this work, we specialize [19]

to the domain of C&C views, in order to localize and explain unsatisfiability of C&C views specifications.

III. Example

We use an example to demonstrate the challenge of localizing and explaining unsatisfiability of views specifications and the means we provide to address it. The example is based on a specification of an industrial robot arm, which we borrow from [13]. This section is semi-formal and the examples shown are relatively simple. We give formal definitions and details about more complicated cases and extensions later in the paper.

A. Example I

Fig. 1 shows the C&C views specification S_1, consisting of six views. We start with an overview of their semantics.

The C&C view `RJFunction` describes the system architecture from the point of view of the team responsible for its function: the `RotationalJoint` contains a `Cylinder` and a `Sensor` that is connected to an `Actuator`. As a C&C view, `RJFunction` is partial, so it may not contain all the system's components. Moreover, while the components shown inside the joint must actually be inside the joint, they may be nested within some of its subcomponents (not shown in this model). On the other hand, the C&C view specifies that the three subcomponents, `Cylinder`, `Sensor`, and `Actuator` are not nested within one another. Finally, `Sensor` and `Actuator` must be connected, but their connection is not necessarily direct and the port names and types are not given in the view.

The C&C view `BodySensorOut` describes the C&C model from the point of view of the team responsible for a component named `Body` and focus on its internal structure. It specifies three subcomponents of `Body` (not necessarily direct sub components, not necessarily all of them) and the connections between them (again, not necessarily all connections, not necessarily direct ones). Another component, `Sensor`, is placed outside `Body`.

The C&C view `SensorConnections` describes the point of view of the engineer responsible for `Sensor`. It shows that `Sensor` is connected to `Cylinder` and to a component named `JointLimiter`. Again, the C&C view is partial, thus in the complete model the connections shown may be indirect and `Sensor` may be connected to additional components.

The C&C view `RJStructure` provides a high-level description of the `RotationalJoint` structure, some of the components it contains and the connections between them. It describes the knowledge of the senior engineer responsible for the joint. It also shows the name `angle` and type `float` of an incoming port of the `Cylinder` for a connection (not necessarily direct) coming from `Body`.

The C&C view `ASDependence` shows an unwanted placement of components `Actuator` and `Sensor` inside `Body`. Thus, it is used in the specification (see below) in a negated form, to not allow an architecture where `Actuator` and `Sensor` are independent components inside component `Body`.

Finally, the C&C view `OldDesign` specifies that `Actuator` is connected to `Cylinder` and that both components are contained inside `Body` (although not necessarily directly). It also shows the name `angle` and type `int` of the `Cylinder`'s incoming port for a connection (not necessarily direct) coming from `Actuator`.

The Boolean expression for the C&C views specification S_1 is $\texttt{RJFunction} \wedge \texttt{BodySensorOut} \wedge \texttt{SensorConnections} \wedge \neg\texttt{ASDependence} \wedge \texttt{RJStructure} \wedge \texttt{OldDesign}$.

Is there a C&C model that satisfies this specification? The tool presented in [13] tells the engineer that there is no such model. But why? Here the engineer was left in the dark. Our new work provides a concrete and detailed answer to this question.

Fig. 2 shows the output of our tool. First, an automatically generated views core, a minimal subset of the specification S_1 that is already unsatisfiable, where each view by itself is also stripped down to contain only a minimal subset of its elements, relevant to the unsatisfiability. In our case, the core is a specification S_1' made of a conjunction of only two of the six original views of S_1, views `RJStructure` and `OldDesign`. Note that the two views are stripped down from irrelevant elements: no connectors are shown and component `ServoValve` is not shown in view `RJStructre` because indeed these elements are redundant in explaining the unsatisfiability. Second, an automatically generated unsatisfiability report, a concrete, structured textual explanation for the cause of unsatisfiability; in this case about the specific hierarchy conflict that was found in S_1.

B. Example II

Given the views core and report shown in Fig. 2, the engineer may decide to create a view similar to `OldDesign`, but with component `Cylinder` outside `Body`. Fig. 3 presents this view, named `OldDesignExternalCylinder`. She now considers the specification S_2, which is a modified version of S_1 where `OldDesign` is replaced by `OldDesignExternalCylinder`. Is S_2 satisfiable?

The answer is negative and our tool provides a views core and a concrete explanation, shown in Fig. 4. Again, only two views are included in the core, and they are stripped down to include only the necessary elements related to the reason for unsatisfiability. The unsatisfiability report provides details about the conflicting port types problem.

IV. Dealing with UNSAT C&C Views Specifications

We are now ready to present the contribution of our work, namely our approach to dealing with UNSAT C&C views specifications.

A. Solution Architecture and Overview

Fig. 5 shows a flow chart of our solution architecture. The input to the synthesis process is a C&C views specification: the C&C views and the Boolean formula. If the Boolean formula is not satisfiable (by itself, e.g., is of the form $v_1 \wedge \neg v_1$),

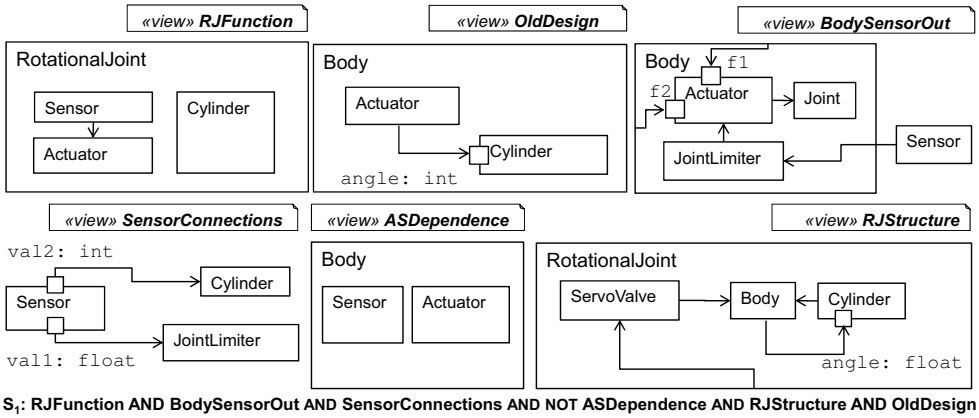

Fig. 1. A C&C views specification S_1. Note `ASDependence` is negated, so it must not be satisfied by the model.

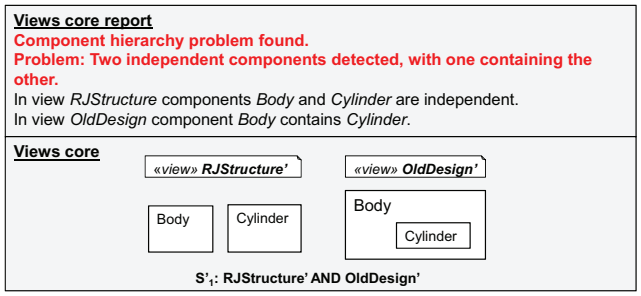

Fig. 2. Generated C&C views core and unsatisfiability report for specification S_1. The resulting core specification is called S'_1.

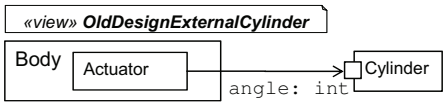

Fig. 3. The view `OldDesignExternalCylinder`, a modified variant of view `OldDesign`, where component `Cylinder` is outside `Body`.

there is obviously no solution regardless of the content of the views. We use a SAT solver to check this and report this simple case separately. Otherwise, we apply the synthesis process described in [13], i.e., translate the specification into an Alloy module, and use a SAT solver in order to synthesize a C&C model. Most importantly, rather than using the original translation from [13], we use a new translation to generate an Alloy module that supports the computation of a views core (see details in Sect. IV-C). Finally, if the Alloy analyzer cannot find a satisfying assignment, we do not only report unsatisfiability, but also extract an Alloy core. Our views core generator uses the Alloy core to produce the views core and the views core report. These are the main contribution of our present paper, described in the next three subsections.

Remark 1: In this section we assume the specification formula consists of a conjunction of views (including possibly negated ones). We address advanced cases of more complicated formulas with disjunctions and implications in Sect. V-C.

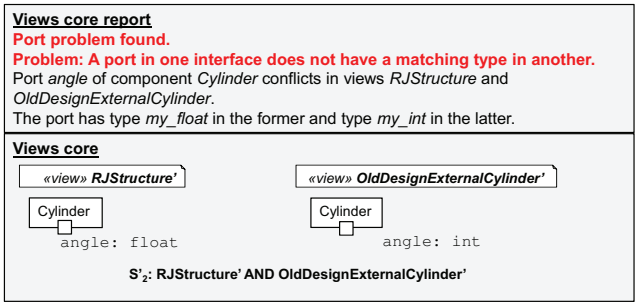

Fig. 4. Generated C&C views core and unsatisfiability report for specification S_2. The resulting core specification is called S'_2.

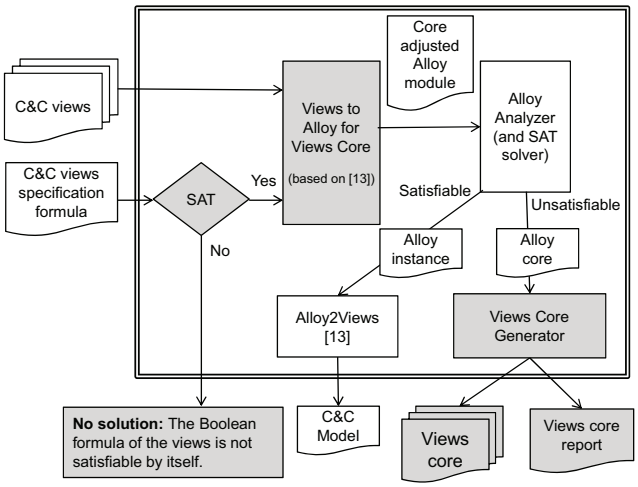

Fig. 5. Solution architecture for the main contributions of this paper for generating C&C views cores and core reports. The new contribution is emphasized in gray background. We discuss the new Alloy translation changed from [13] in Sect. IV-C.

B. Views Core

Intuitively, a views core of an unsatisfiable C&C views specification is a minimal unsatisfiable views specification consisting of a fragment of the original set of views and the original specification formula. Minimality of the core is defined in two dimensions, the views elements dimension and

the specification formula dimension. To formalize, we start by defining the following partial order on C&C views.

Definition 5 (C&C view partial order): Two C&C views $v = \langle Cmps, Ports, AbsCons, Types, subs, ports, type\rangle$ and $v' = \langle Cmps', Ports', AbsCons', Types, subs', ports', type'\rangle$, are ordered $v' \sqsubseteq v$ iff $Cmps' \subseteq Cmps$, $Ports' \subseteq Ports$, $AbsCons' \subseteq AbsCons$, $\forall c \in Cmps' : subs'(c) = subs(c) \cap Cmps'$, $\forall c \in Cmps' : ports'(c) = ports(c) \cap Ports'$, and $\forall p \in Ports' : type'(p) = type(p)$.

Note that Def. 5 relates the structure of C&C views but also implies a relation on their semantics (Def. 3), for any C&C model m and any two views $v' \sqsubseteq v$, if $m \models v$ then $m \models v'$ We use this partial order in our definition of views core of a conjunction of views and their negations below: a views core of an unsatisfiable views specification in conjunctive form is an unsatisfiable views specification that (1) consists of views that are smaller (or equal) to the original views according to the partial order $\sqsubseteq$ and (2) is minimal in the sense that removing any of these views or replacing them with smaller views (according to $\sqsubseteq$), will make it satisfiable. Formally:

Definition 6 (Conjunctive C&C views specification): For any two sets of views V_p, V_n such that $V_p \cap V_n = \emptyset$, we define their conjunctive specification $Cnj(V_p, V_n) = (S, V_p \cup V_n)$ where $S = (\bigwedge_{v \in V_p} v) \wedge (\bigwedge_{v \in V_n} \neg v)$.

Definition 7 (C&C views core): A views core of an unsatisfiable specification $Cnj(V_p, V_n)$ is a minimal unsatisfiable specification $Cnj(V'_p, V'_n)$ such that

- $V'_p \subseteq \{v' | \exists v \in V_p : v' \sqsubseteq v\}$;
- $V'_n \subseteq V_n$;
- For any $v_1 \in V'_p$, and any v_2 such that $v_2 \sqsubseteq v_1$ and $v_2 \neq v_1$, $Cnj(V'_p \setminus \{v_1\} \cup \{v_2\}, V'_n)$ is satisfiable;
- For any $v \in V'_p$, $Cnj(V'_p \setminus \{v\}, V'_n)$ is satisfiable; and
- For any $v \in V'_n$, $Cnj(V'_p, V'_n \setminus \{v\})$ is satisfiable.

For example, the specification S'_1 shown in Fig. 2 is a views core of the unsatisfiable specification S_1 of Fig. 1. S'_1 consists of only two views, each of which is smaller (according to $\sqsubseteq$) than a view in S_1: `OldDesign'` $\sqsubseteq$ `OldDesign` and `RJStructure'` $\sqsubseteq$ `RJStructure`. As a specific example, in the view `OldDesign'` of S'_1, component `Actuator` does not appear and component `Cylinder` has no ports specified, because these are not necessary for unsatisfiability.

Note that in Def. 5, ports and connectors are considered complete units. Thus, while unnamed and untyped ports are allowed in views specifications, rendering a port unnamed or untyped while keeping unsatisfiability is not part of our views core definition. Based on Def. 5 and Def. 7, a port either appears exactly the same in the core as in the original specification, or it is not a part of the core.

In Def. 7 negated views (elements of V_n) remain the same in the views core, if they are a part of it, because smaller negated views impose stronger, not weaker, constraints. Formally, assume that v' is a negated core view, and v is its corresponding view in the original specification. Then from $v' \sqsubseteq v$ follows that for every C&C model m, $m \models \neg v'$ implies $m \models \neg v$ Thus replacing v' with v in the core would result in a core which is both less restrictive, and still unsatisfiable. Therefore $v' = v$ is always the best choice under these conditions.

It is important to note that the views core definition Def. 7 is monotonic with regard to unsatisfiability.

Theorem 1 (Core Monotonic): Given an unsatisfiable specification $Cnj(V_p, V_n)$, any extension of V_p, V_n with a view v or replacing $v \in V_p$ by a view v' with $v \sqsubseteq v'$ keeps the specification unsatisfiable.

Proof-sketch: Proof for any extension in Def. 7 and Def. 5 that it strengthens the (already unsatisfiable) specification.

Thanks to this monotonicity, we can guarantee that the reason for unsatisfiability in the core (which we report to the engineer, see Sect. IV-C below), is also a reason for unsatisfiability in the original specification. Thus, although our core is an abstraction of the specification, it never reports spurious reasons for unsatisfiability.

Finally, note that the views core definition, like other UNSAT core definitions in the literature (see, e.g., [19], [21]), is about local minimality, not a global one. Thus, an unsatisfiable views specification may have many different view cores (for example, specification S_1 from Fig. 1 has multiple cores). Like other works, our implementation presents the first core it finds.

C. Computing the Views Core

Given a C&C views specification as input, the work in [13] produces an Alloy module that has an instance iff the specification is satisfiable (in a bounded scope). However, the original translation from [13], although correct for synthesizing C&C models, is inadequate for producing C&C views cores with the algorithm of [19] in case no satisfying instance exists; it fails to identify view elements and to produce minimal C&C views according to Def. 5.

Notably, the algorithm of [19] requires translations to be *regular*, i.e., translation results are *equisatisfiable* and all conjuncts are *context-independent*. The translation from [13] produces equisatisfiable results but it is not regular. Thus, we present here a new translation that revises and refines the one from [13]. The new translation is regular and UNSAT core result can be effectively mapped back at the required abstraction level. Below we demonstrate the limitations of the original translation and the key features of the new translation.

Fig. 6 (left) shows an excerpt of an Alloy module for the example specification S_1, as generated by the original translation, and its Alloy core (highlighted). The Alloy core consists of the existence of component `Cylinder` and the bodies of the predicates `contains` and `independentSet`. This core of one signature and two predicates expresses only very generic view constraints and is missing important information on the specific instances that lead to unsatisfiability (e.g., S_1 has 8 instances of `contains`).

Our new translation presents three major differences from the original translation. It allows the identification of view elements and the computation of C&C views cores, as we demonstrate in an excerpt from the result of our new translation and corresponding highlighted Alloy core in Fig. 6 (right).

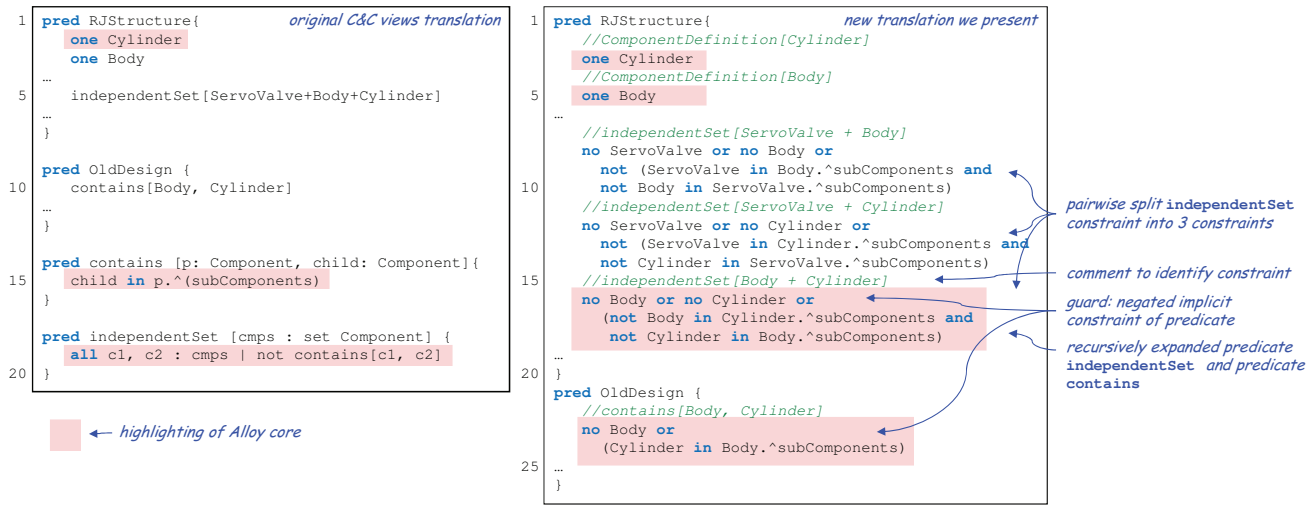

Fig. 6. Two excerpts of Alloy translations for views `RJStructure` and `OldDesign` from Fig. 1: The original translation from [13] (left) and the new translation of this paper (right). For each we highlight the unsatisfiable Alloy core as computed by Alloy Analyzer 4.2 (top-level, locally minimal core).

First, the new translation recursively instantiates and expands Alloy predicates into constraints. As an example, ll. 9-10 (right) contains the expanded predicate `independentSet` and its recursively expanded child predicate `contains` instantiated for component signatures `ServoValve` and `Body`. This modification ensures the designation of specific constraints on view elements instead of generic predicates.

Second, the new translation splits constraints over sets of signatures for components or ports into multiple constraints over pairs of signatures. For example, the independence of components `ServoValve`, `Body`, and `Cylinder` from l. 5 (left) is expanded to three pairwise constraints in ll. 7-18 (right). In addition, the new translation suppresses implicit constraints by adding guards. As an example, consider the relational join of `p` with `subComponents` in l. 15 (left). The join implicitly makes `p` non-empty, i.e., the component represented by `p` has to exist. The guards we add are disjunctions with the negation of the implicit constraint, e.g., l. 23 (right) negates the non-emptiness of `Body`. These modifications ensure that translated constraints are context-independent, i.e., independent of other constraints in the C&C view.

Finally, the new translation adds annotations that identify specific view constraints, e.g., l. 15 (right) `independentSet`. Together, these modifications allow us to compute a C&C view core from the Alloy core. It is easy to see that the highlighting in Fig. 6 (right) corresponds to the views core shown in Fig. 2.

Theorem 2 (Translation regular): The new translation is regular (in the sense of [19, Def. 2] within a bounded scope) and the core calculation of [19] implemented in Alloy will result in a views core as defined by Def. 5 and Def. 7.

Proof-sketch: Proof for any C&C view extension following Def. 5 that the translation adds an equisatisfiable, context-independent conjunct, i.e., that it is regular. Restate [19, Thm. 2] for conjunctive specifications of Def. 6, our regular translation, and cores of Def. 7.

D. Core Data Classification and Problem Extraction

Core computation results in core data, a minimal set of facts and annotated constraints that together cannot be satisfied. The annotated constraints contain predicate names and related information that point to specific views elements.

From the core, we extract reasons for unsatisfiability using an extensible two-levels framework. First, we identify a problem type: a high-level classification of the problems that cause unsatisfiability. Second, we identify specific problems. We instantiate the framework for specific problems, including all of the ones in this paper.

For the high-level clasification, we distinguish between three problem types shown in Table I (left): hierarchy problems, where the core consists only of hierarchy constraints (e.g., `contains[Body, Cylinder]` from Fig. 6, right, l. 22); port problems, where the core consists only of port constraints; and connection problems, where the core contains connection constraints (and might also contain hierarchy and port constraints). This classifies core data into high-level types of unsatisfiability causes. As an example, the core data we report in Fig. 2 consists only of facts and constraints related to hierarchy, while the core data in Fig. 4 consists only of facts and constraints related to ports.

The low-level extraction is based on problem definitions consisting of a specific problem name, as shown in Table I (right), and a matching of facts and annotated constraints from core data. For example, for problem H3 from Table I we go through all pairs of components in core constraints annotated `independentSet`, and look for `contains` annotations indicating a (possibly transitive) containment between them. This matches the core in Fig. 2 to problem H3. For problem P1 the matching looks for multiple port annotations, e.g., `portOfComponent`, of the same port with conflicting directions or conflicting types (for typed ports).

Remark 2: With language extensions of C&C views and more complex specifications (see Sect. V), additional specific

TABLE I
CLASSIFICATION OF BASIC UNSATISFIABILITY PROBLEMS

Problem Type	Specific Problem
Hierarchy	H1 Cyclic containment H2 Independent components contain a third component H3 An independent component contains the other HF Other hierarchy problem
Port	P1 Inconsistent port definition in two interfaces PF Other port problem
Connection	C1 Connection type mismatch CF Other connection problem

problems can manifest in core data. Thus, we implemented specific problems identification as an extensible framework and added generic problem kinds, HF, PF, and CF (see Table I), for core data with no specific match.

Remark 3: If there are negated views in the core data, we report the list of negated views, and the parts of the positive views they conflict with.

E. Generating Natural-Language Explanations

In most cases, it is possible to match the core data to a specific problem, which allows the automatic generation of a structured natural-language explanation that we call *views core report*. The report provides a reason for unsatisfiability, and arranges the information in a way that clarifies the problem.

The first line of a report indicates the type of the problem. The second line indicates the specific problem, possibly one version of such a problem according to the findings. The third part of the report contains structured core information. Reports may also contain a suggestion.

For example, the views core report in Fig. 4 indicates problem P1. The first line indicates a port problem. The second line indicates a port type mismatch. The first sentence of the third part explains where the port with the conflicting type appears, and its second sentence shows the names of its conflicting types.

The reports also extract and arrange information. For example, component containment may result from a containment chain. Our explanations automate the extraction of all containment chains. Suppose component C_1 contains C_2 through component C_3. The description formats the containment to first show the chain C_1=>C_3=>C_2, and then list the views and the containments according to their order within the chain. See the second report in Fig. 7 for an example. We do not show a containment chain in containments that involve only two components. For example, the chain `Body=>Cylinder` does not appear in Fig. 2. This format is used throughout the many possible problems that involve containments, thus standardizing its presentation.

The generic problems HF, PF, and CF handle cases not matched to any specific problem. They provide an arrangement of core data and/or indications of problems not directly evident from the core data. For example, if the classification of the core data indicates a port problem, but no specific port problem was detected, we suggest that the specified port scope was too small. In this case the core report advises the engineer to increase the port scope.

TABLE II
CLASSIFICATION OF ARCHITECTURAL STYLE RELATED PROBLEMS

Style	Specific Problem
CS L	H6 Top component is contained
CS L	H7 Two top components contain the same component
CS L	H8 Top components do not contain all components
L	H9 A layer cannot contain a required component
H	C2 Circular connection

V. EXTENSIONS AND ADVANCED TOPICS

Synthesis was extended in [13] to include architectural styles, restrictions over components, and more complex Boolean specifications. We use the extensible mechanism of identifying specific problems for the former two extensions. The additional specific problems appear in Tables II and III. Their enumeration in the tables show their type according to Sect. IV-D, and they are applied only when indicative core facts and constraints appear in the core.

A. Architectural Styles

Architectural styles systematize successful architectural design practices in terms of constraints on architectural elements and their composition into systems [16], [18]. C&C views synthesis, as defined in [13], supports three architectural styles: a hierarchical style, whose essence is to forbid the C&C models from having directed cycles of connected components, a client-server style, whose essence is to identify one of the components as a single server and to forbid any direct communication between clients, and a layered style, which forces a partition of the components into a sequence of layers and allows direct connectors only within layers and between consecutive layers. As styles impose additional constraints on the synthesis problem, they may be related to a specification's unsatisfiability. Therefore, **we extended our work to support the three styles mentioned above, i.e., to provide style-related views core reports, when applicable**.

Table II shows the style-related specific problems we address. H, CS, and L refer to hierarchical, client-server, and layered styles. The client-server and layered styles require that certain specified components are top-level components. They must be non-contained, independent, and contain all other components (see problems H6, H7, and H8). In layered style it is possible to require that a specific component be contained in a specific layer, which may create problem H9.

As an example, the top part in Fig. 7 shows a report of problem C2, in hierarchical style, which forbids the C&C models from having directed cycles of connected components. The report presents three specific ports that are connected in a circular way. It shows the circular path and lists the views in which each of the connections appears.

As another example, the bottom part in Fig. 7 shows a report of problem H7, in layered style with layers `TL1`, `TL2`, and `TL3`: component `Motor` is contained in layer `TL1` and (by a chain of containment) in layer `TL3`. All containments are listed with the views in which they appear.

Views core report
Connection problem found.
Problem: A circular connection of ports in hierarchical style.
The connections : *Door.openSig -> Motor.roll -> Sensor.pad -> Door.openSig*.
In view *H2* connection *Door.openSig* -> Motor.roll.
In view *H1* connection *Motor.roll -> Sensor.pad*.
In view *H1* connection *Sensor.pad -> Door.openSig*.

Views core report
Component hierarchy problem found.
Problem: Layers contain the same component in layered style.
TL1 and *TL3* should be independent top components in the specified style.
Both contain component *Motor*.
In view *Lay1* component *TL1* contains *Motor*.
Component *TL3* contains *Motor* : *TL3 => Portal => Encapsulator => Motor*.
In view *Lay3* component *TL3* contains *Portal*.
In view *Lay2* component *Portal* contains *Encapsulator*.
In view *Lay2* component *Encapsulator* contains *Motor*.

Fig. 7. Examples of generated style-related views core reports. The notation $C_1.p_1 \rightarrow C_2.p_2$ means a connection from port p_1 of component C_1 to port p_2 of component C_2.

B. Library Components, Atomic, and Interface Complete

Most C&C models reuse library components, pre-defined or existing components adopted from other systems. Indeed, C&C views synthesis, as defined in [13], supports the integration of library components by adding files that contain their details, and specifying them as library components. Those are not C&C views, they do not appear in the Boolean formula, and cannot be negated. A successful synthesis inserts them as they are, according to the constraints set by the views.

Library components are both *atomic* (that is, views may not add internal components into them), and *interface-complete* (they have a fixed list of ports that cannot be changed or extended, and none of the ports are unnamed or untyped). The designer may decide to declare a component within a view to be atomic and/or interface-complete using stereotypes.

We have extended the views core and report to support unsatisfiable specifications with library, interface-complete, and atomic components. In the first column of Table III, problems marked with A are related to atomic components, and ones marked IC are related to complete interfaces - ones originating both from stereotypes and from library components. We give a brief overview of the extension.

Atomic components may cause component hierarchy problems. Problems H4 and H5 are about such component found to be containing other components.

We also handle interface-complete related port problems. A component may have two complete-interface specifications with non-identical ports (problem P2), or an interface that contradicts a complete interface (problem P3). For example, if a component has an interface-complete declaration with three ports $p1, p2$, and $p3$, and another interface declaration with two ports, $p1$ that matches the one of the interface-complete specification, and another unnamed port, depending on the types and directions of $p2$ and $p3$ the unnamed port may or may not be matched. In case it cannot be matched, the problem is presented in the views core report.

Complete interfaces are considered complete units just like ports, in order to keep unsatisfiability monotonic, since adding ports to an interface-complete component may turn an unsatisfiable specification into a satisfiable one, and removing such ports may cause a satisfiable specification to become unsatisfiable. Thus when an interface-complete component appears in a core view it is unchanged. Similarly, library components either appear in the core without change, or they are not part of it.

TABLE III
CLASSIFICATION OF COMPONENT RESTRICTION RELATED PROBLEMS

Feature	Specific Problem
A	H4 Library component contains a component
A	H5 Atomic component contains a component
IC	P2 Two complete interfaces are unequal
IC	P3 Complete interface specification is contradicted
IC	C3 Connection prohibited by lack of ports in an interface-complete component

C. Beyond Conjunctions

C&C views specifications, as originally described in [13] and supported in our tool, allow not only conjunctions but also disjunctions (of several literals), implications, and binary exclusive-or operators. For example, if v_1, v_2 and v_3 are views, $\neg v_1 \otimes v_2$, $v_1 \rightarrow \neg v_2$, and $v_1 \vee v_2 \vee \neg v_3$, are allowed, and their conjunction is a valid C&C views specification Boolean formula. This allows engineers to write concise specifications that are as expressive as any propositional Boolean formula. **Our work on C&C views core supports such more complex specification formulas**, as we describe below.

Consider, for example, a case where $v_1 \vee v_2 \vee \neg v_3$ is part of the core as returned by Alloy. In the presence of such a disjunction, Alloy highlights the entire disjunction. Although correct, we consider this to be not best for our purpose, because we cannot minimize the views in the core.

Our current solution provides core information for one of the alternatives by replacing the satisfiable core Boolean formula with a satisfiable refined Boolean formula (for example, if the core Boolean formula is $\neg v_1 \wedge (v_1 \vee \neg v_2) \wedge v_3$ we choose $\neg v_1 \wedge \neg v_2 \wedge v_3$ rather than $\neg v_1 \wedge v_1 \wedge v_3$), which is a Boolean formula that has the form of the formulas in Def. 7, and one that has only positive views, if there is such an alternative.

Specifically, we implement this as follows. When disjunctions, implications, or exclusive-or expressions are found in the core Boolean formula, we use the Boolean equivalences $A \rightarrow B \equiv \neg A \vee B$ and $A \otimes B \equiv (\neg A \wedge B) \vee (A \wedge \neg B)$ in order to translate implications and exclusive-or expressions respectively into disjunctions. We then go over all disjunctions one by one, and choose one of the operands of each disjunction which is consistent with the rest of the Boolean expression, with priority to alternatives without negated views. We rerun Alloy core on the new specification with the original port and component scopes, to ensure that the rerun does not introduce a new scope problem.

An example for the above process is as follows. Suppose the engineer adds a view `BodySensorIn`, which is exactly the same as `BodySensorOut`, only with component `Sensor` inside component `Body`, and uses the specification $S_3 = \texttt{RJFunction} \wedge (\texttt{BodySensorIn} \vee \texttt{OldDesign}) \wedge$

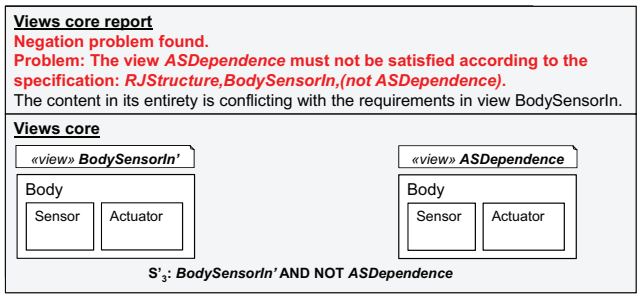

Fig. 8. One of the two alternative cores for specification S3

SensorConnections ∧ ¬ASDependence ∧ RJStructure. The core Boolean formula returned by Alloy is (BodySensorIn ∨ OldDesign) ∧ ¬ASDependence ∧ RJStructure, so we replace the disjunction BodySensorIn ∨ OldDesign by either BodySensorIn or OldDesign.

Both BodySensorIn or OldDesign are consistent with the rest of the core Boolean formula, and both are positive views. If we choose OldDesign, after we rerun synthesis with the Boolean formula OldDesign∧¬ASDependence∧ RJStructure the resulting core is the same as in Fig. 2. If we choose BodySensorIn, after a rerun with the formula BodySensorIn ∧ ¬ASDependence ∧ RJStructure we obtain the core report and views core shown in Fig. 8.

The above implementation provides views core information by restricting the core Boolean formula to a satisfiable Boolean formula of the form of Def. 7, that still leaves the views specification unsatisfiable. However, note that this solution is no longer monotonic because removing disjuncts makes the Boolean formula stricter. We still decide to use this heuristic in this case in order to provide the engineer with a smaller views specification that contains a concrete reason for unsatisfiability.

VI. Implementation and Evaluation

We have implemented the C&C views core computation of Sect. IV, and the extensions discussed in Sect. V, on top of the prototype implementation provided in [5].

To evaluate our approach to dealing with unsatisfiable C&C views we consider the following research questions:

- R1 What is the performance cost of C&C views core computation?
- R2 Does core computation effectively reduce the number of views?

A. Corpus of Specifications

Table IV lists the 12 specifications we used in our evaluation, which model a robot arm, a pump-station, and an avionics system, adapted from [1], [2], [10]. The first 4 specifications in the list are S1 and S2 from Sect. III, specification S3 from Sect. V-C, and a version of specification S3 with an exclusive-or instead of a disjunction. The last 8 specifications are the unsatisfiable specifications from the evaluation in [13]. The first 4 specifications of these 8 specifications have a complex Boolean specification, yet have a simple core specification, thus they require only a single run of the solver, including in setup Core. Specifications S3, S3_xor, and specPhySim1 have a complex Boolean specification that require two runs of the solver in setup Core as described in V-C. The last 2 specifications are unsatisfiable as a result of a low port scope.

TABLE IV
Running times and views counts

Specification	Time (ms)			Size (#v/#c)	
	Orig	Core	Ratio	Orig	Core
S1	466	1829	3.92	6/22	2/4
S2	503	3071	6.10	6/22	2/2
S3	744	4461	5.99	6/22	2/6
S3_xor	816	6229	7.63	6/22	2/6
paperS1_ClientServer	1337	4600	3.44	6/24	1/2
paperS2	936	2234	2.38	7/27	2/4
paperS2_ExtCyl	978	4763	4.87	7/27	2/2
specAll_NoLayers	312	534	1.71	9/25	1/2
specAllAndEmergency	6634	33931	5.11	7/28	2/2
specPhySim1	516	134135	259.95	4/12	2/5
specPhySim2_LowScope	110	51556	468.69	3/10	2/6
specAll_LowScope	133	4932	37.08	9/25	5/11

B. Validation

We validated the correctness of our implementation on existing and synthetic specifications. We have created C&C views specifications that cover all the problems in Table I, including advanced features described in Sect. V in Tables II and III, and manually inspected the results of running synthesis and core computation on them. Additionally, we have run synthesis over the core specifications our tool produces for all the specifications in Table IV. As expected, all of the core specifications were found to be unsatisfiable and minimal. Computing the core of each views core, with the original scopes, generated the same views core.

Our implementation, the synthetic specifications used for validation, all the specifications in our experiments below, and the test that produces the experiment results, are available from http://smlab.cs.tau.ac.il/cncviews/unsat/. We invite the interested reader to inspect them.

C. R1: Performance Cost

To examine the performance of our work, relative to the original satisfiability check presented in [13], we have compared the running times of two setups, with and without core computation. Setup Orig uses the translation to Alloy and the MiniSAT solver that were presented and used in [13]. Setup Core uses the translation to Alloy presented in Sect. IV-C and the MiniSATCore solver.

We obtained running times on an ordinary desktop computer with an i7 3.4 GHz CPU running Windows 7 64Bit, Java 1.7 64Bit, Alloy 4.2, and MiniSAT 64Bit. The times we report are wall-clock-times measured by the Java API. We executed each experiment 12 times and report average running times.

The table shows that in our experiments, the running time ratio varies from less than twice to two orders of magnitude slower (the median ratio is 5.56). Note that the three exceptionally high ratios were obtained for two specifications with a low port scope, and one with an additional SAT run for a core

alternative. Note also that our tool notifies the engineer that the specification is not satisfiable early, and it is the engineer's choice whether to wait for the more informative response that includes the views core and report.

To answer question R1: **Views core computation does not come for free**, indeed it is **in 50% of the cases more than 5.5 times slower than satisfiability checking alone.** Still, this seems a price one would be willing to pay in order to enjoy the benefits of concise and concrete explanation for the cause of unsatisfiability.

D. R2: Effectiveness of Core Reduction

Table IV (right) compares the sizes of the original specification and the core. We measure size by the number of views (before the slash) and the total number of components within the views (after the slash, when a component appears in multiple views we count all its occurrences). In almost all cases, the core consisted of only one or two views [1]. The single exception belongs to a specification with insufficient port scope. In this case the views core may be composed of many unrelated elements of the views because the problem is not local to any small subset of them. Nevertheless, the views core report is helpful in this case too, as it explicitly reports that no other kind of problem (hierarchy etc.) is relevant.

To answer question R2: Core computation **effectively reduces the number of views** to consider by a factor of at least 3 in 75% of the specifications, **and the total number of components** by a factor of over 6 in 50% of the specifications.

E. Threats to Validity

Internal. Our implementation may have bugs. To mitigate, we have validated it as described in Sect. VI-B.

External. Although comprehensive, the set of specifications we have used in our experiments is small; although some of the specifications are adapted from real-world models (the robot arm, pump-station, and avionics systems), others were designed with our implementation in mind. This may limit the generalizability of our results. We did not do a study with engineers to evaluate the contribution of the core and report to the identification and comprehension of the unsatisfiability problem. Still, the smaller size of the core (as evident in our evaluation) and the additional textual report, hint that they will be easier to understand by engineers.

VII. Related Work

In [13] we introduced C&C views synthesis, but did not deal with the challenge of locating and explaining unsatisfiability. In [14] we considered the verification problem of checking whether a C&C model satisfies a given C&C view. We generated short natural-language texts that explained the reasons for satisfaction or non-satisfaction. This motivated us to generate natural-language texts to explain unsatisfiability in the context of C&C views synthesis.

Bagheri and Sullivan [3], [4] present a different approach to using Alloy for specifying structural properties of component and connector models. To the best of our understanding, they do not handle a case of unsatisfiability.

Torlak et al. [19] find minimal cores of Alloy modules specifications efficiently by iterating over refutations of translations of fragments of the module to CNF formulas. They showed that translations that have specific properties are enough to ensure correctness. D'Ippolito et al. [7] suggest a heuristic for fast approximation of cores of Alloy modules.

While many works discuss the potential of the use of UNSAT core in addressing software engineering related problems, we have only found a few that actually try to realize this potential. Wille [20] suggest and demonstrate the use of UNSAT core for debugging of UML/OCL models. Gopinath et al. [9] used UNSAT core using Alloy to improve fault localization. Our work uses the work of Torlak et al. [19]. We are not aware of other works that have directly used [19].

VIII. Conclusion

We presented an approach to deal with unsatisfiable C&C views specifications. We define a notion of a C&C views specification core, and show how to compute it via a new translation to Alloy leveraging UNSAT core techniques; moreover, we generate a C&C views core report, which presents explicit, concrete, structured natural-language explanations about the cause of unsatisfiability. Finally, we extend our work to support specifications with architecture styles, library components, and Boolean formulas beyond simple conjunctions.

We implemented our work and evaluated it using synthetic and real-world C&C views specifications. In the evaluation we examined the cost of the core computation and its effectiveness in reducing the size of the specification. We have made the code and data required in order to reproduce our experiments available to the readers of the paper.

Beyond its direct contribution to the field of component and connector modeling, our work provides an interesting, working demonstration of the potential of existing UNSAT core technologies to assist in addressing challenges in software engineering on the one hand, and the work required to realize this potential on the other hand.

We suggest several directions for future research. First, obviously, an extended evaluation with additional specifications as well as a user study that will examine the difficulty of identifying and manually explaining C&C views unsatisfiability as well as the effectiveness of our generated core and report in helping engineers identify and fix unsatisfiability problems.

Second, experimenting with alternative means to generate UNSAT cores, with and without Alloy, e.g., [7].

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

[1] A core of only a single unsatisfiable view may occur in advanced style specifications that add restrictions on the component hierarchy (see Sect. V-A).

REFERENCES

[1] AADL website. http://www.aadl.info/. Accessed 4/2017.

[2] AutoFocus3 website. http://autofocus.informatik.tu-muenchen.de/. Accessed 4/2017.

[3] H. Bagheri and K. J. Sullivan. Monarch: Model-based development of software architectures. In *MoDELS (2)*, volume 6395 of *LNCS*, pages 376–390. Springer, 2010.

[4] H. Bagheri and K. J. Sullivan. Model-driven synthesis of formally precise, stylized software architectures. *Formal Asp. Comput.*, 28(3):441–467, 2016.

[5] Supporting materials on C&C views verification. http://www.se-rwth.de/materials/cncviews/.

[6] N. Dershowitz, Z. Hanna, and A. Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In A. Biere and C. P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 36–41. Springer, 2006.

[7] N. D'Ippolito, M. F. Frias, J. P. Galeotti, E. Lanzarotti, and S. Mera. Alloy+HotCore: A Fast Approximation to Unsat Core. In M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, editors, *Abstract State Machines, Alloy, B and Z, Second International Conference, ABZ 2010, Orford, QC, Canada, February 22-25, 2010. Proceedings*, volume 5977 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 2010.

[8] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis & design language (AADL): An introduction. Technical report, Software Engineering Institute, Carnegie Mellon University, 2006.

[9] D. Gopinath, R. N. Zaeem, and S. Khurshid. Improving the effectiveness of spectra-based fault localization using specifications. In M. Goedicke, T. Menzies, and M. Saeki, editors, *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 40–49. ACM, 2012.

[10] F. Hölzl and M. Feilkas. Autofocus 3 - a scientific tool prototype for model-based development of component-based, reactive, distributed systems. In *Model-Based Engineering of Embedded Real-Time Systems*, volume 6100 of *LNCS*, pages 317–322. Springer, 2007.

[11] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[12] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[13] S. Maoz, J. O. Ringert, and B. Rumpe. Synthesis of component and connector models from crosscutting structural views. In B. Meyer, L. Baresi, and M. Mezini, editors, *ESEC/SIGSOFT FSE*, pages 444–454. ACM, 2013.

[14] S. Maoz, J. O. Ringert, and B. Rumpe. Verifying component and connector models against crosscutting structural views. In P. Jalote, L. C. Briand, and A. van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 95–105. ACM, 2014.

[15] J. O. Ringert. *Analysis and Synthesis of Interactive Component and Connector Systems*. Aachener Informatik-Berichte, Software Engineering, Band 19. Shaker Verlag, 2014.

[16] M. Shaw and D. Garlan. *Software architecture - perspectives on an emerging discipline*. Prentice Hall, 1996.

[17] OMG SysML website. http://www.omgsysml.org/. Accessed 4/2017.

[18] R. N. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[19] E. Torlak, F. S. Chang, and D. Jackson. Finding minimal unsatisfiable cores of declarative specifications. In J. Cuéllar, T. S. E. Maibaum, and K. Sere, editors, *FM 2008: Formal Methods, 15th International Symposium on Formal Methods, Turku, Finland, May 26-30, 2008, Proceedings*, volume 5014 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008.

[20] R. Wille, M. Soeken, and R. Drechsler. Debugging of inconsistent UML/OCL models. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1078–1083, 2012.

[21] J. Zhang, S. Li, and S. Shen. Extracting minimum unsatisfiable cores with a greedy genetic algorithm. In A. Sattar and B. H. Kang, editors, *AI 2006: Advances in Artificial Intelligence, 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, December 4-8, 2006, Proceedings*, volume 4304 of *Lecture Notes in Computer Science*, pages 847–856. Springer, 2006.