Checking SysML Models for Co-Simulation

Nuno Amálio[1], Richard Payne[2], Ana Cavalcanti[3], and Jim Woodcock[3]

[1] Birmingham City University (UK) nuno.amalio@gmail.com
[2] Newcastle University (UK) richard.payne@newcastle.ac.uk
[3] University of York (UK) {ana.cavalcanti, jim.woodcock}@york.ac.uk

Abstract. Cyber-physical systems (CPSs) are often treated modularly to tackle both complexity and heterogeneity; and their validation may be done modularly by *co-simulation*: the coupling of the individual subsystem simulations. This modular approach underlies the FMI standard. This paper presents an approach to verify both healthiness and well-formedness of an architectural design, expressed using a profile of SysML, as a prelude to FMI co-simulation. This checks the *conformity of component connectors* and the absence of *algebraic loops*, necessary for co-simulation convergence. Verification of these properties involves theorem proving and model-checking using: Fragmenta, a formal theory for representing typed visual models, with its mechanisation in the Isabelle/HOL proof assistant, and the CSP process algebra and its FDR3 model-checker. The paper's contributions lie in: a SysML profile for architectural modelling supporting multi-modelling and co-simulation; our approach to check the adequacy of a SysML model for co-simulation using theorem proving and model-checking; our verification and transformation workbench for typed visual models based on Fragmenta and Isabelle; an approach to detect algebraic loops using CSP and FDR3; and a comparison of approaches to the detection of algebraic loops.

Keywords: Co-simulation, FMI, CSP, SysML, algebraic loops

1 Introduction

Cyber-physical systems (CPSs) are designed to actively engage with the physical world in which they reside. They tend to be heterogenous: their subsystems tackle a wide variety of domains (such as, mechanical, hydraulic, analogue and a plethora of software domains) that mix phenomena of both continuous and discrete nature, typical of physical and software systems, respectively.

CPSs are often handled modularly to tackle both heterogeneity and complexity. To effectively separate concerns, the global model of the system is decomposed into subsystems, each typically focussed on a particular phenomenon or domain and tackled by the most appropriate modelling technique. Simulation, the standard validation technique of CPSs, is often carried out modularly also, using co-simulation [18] – the coupling of subsystem simulations. This constitutes the backdrop of the industrial Functional Mockup Interface (FMI) standard [5,4] for co-simulation of components built using distinct modelling tools.

This paper presents an approach to formally verify the well-formedness and healthiness of SysML CPS architectural designs as a prelude to co-simulation. The designs are described using INTO-SysML [3], a profile for multi-modelling and FMI co-simulation. The well-formedness checks verify that designs comply with all the required constraints of the INTO-SysML meta-model; this includes *connector conformity*, which checks the adequacy of the connections between SysML blocks (denoting components) with respect to the types of the ports being wired. The healthiness checks concern detection of *algebraic loops*, a feedback loop resulting in instantaneous cyclic dependencies; this is relevant because a desirable property of co-simulation, which often reduces to coupling of simulators, is *convergence* – whether numerical simulations approximates the solution –, which is dependent on the structure of the subsystems and cannot be guaranteed if this structure contains algebraic loops [18,6]. The work presented here demonstrates the capabilities of our verification workbench for modelling languages and engineering theories, which rests on Fragmenta [2], a theory to formally represent designs of visual modelling languages, and its accompanying mechanisation in the Isabelle proof assistant [22], and the CSP process algebra [13] with its accompanying FDR3 refinement-checker [12].

Contributions. The paper's contributions are as follows:

- A novel SysML profile for architectural modelling of CPSs that tackles heterogeneity by providing support for multi-modelling and co-simulation in compliance with the FMI standard.
- An approach to statically check the adequacy of a SysML architectural model for co-simulation, supporting connector conformity and algebraic loops detection, by using a theorem prover and a model-checker.
- A prototyping environment for Fragmenta [3], a mathematical theory to represent typed visual models, based on the proof assistant Isabelle/HOL that enables model verification and transformation.
- A CSP-based solution to the detection of algebraic loops, which is based on a novel approach to represent graphs in CSP.
- An evaluation of approaches to the detection of algebraic loops.

Outline. The remainder of the paper gives some background on Fragmenta and CSP (Section 2). It presents our approach to represent architectural designs in INTO-SysML, highlighting verification of well-formedness (Section 3), and our approach for representing directed graphs in CSP and detecting algebraic loops through a FDR3 refinement check (Section 4). It evaluates our CSP-based approach (Section 5). Finally, the paper discusses its results (Section 6), compares them against related work (Section 7) and draws the conclusions (Section 8).

2 Background

We give some background on two main ingredients of the work presented here: Fragmenta and CSP.

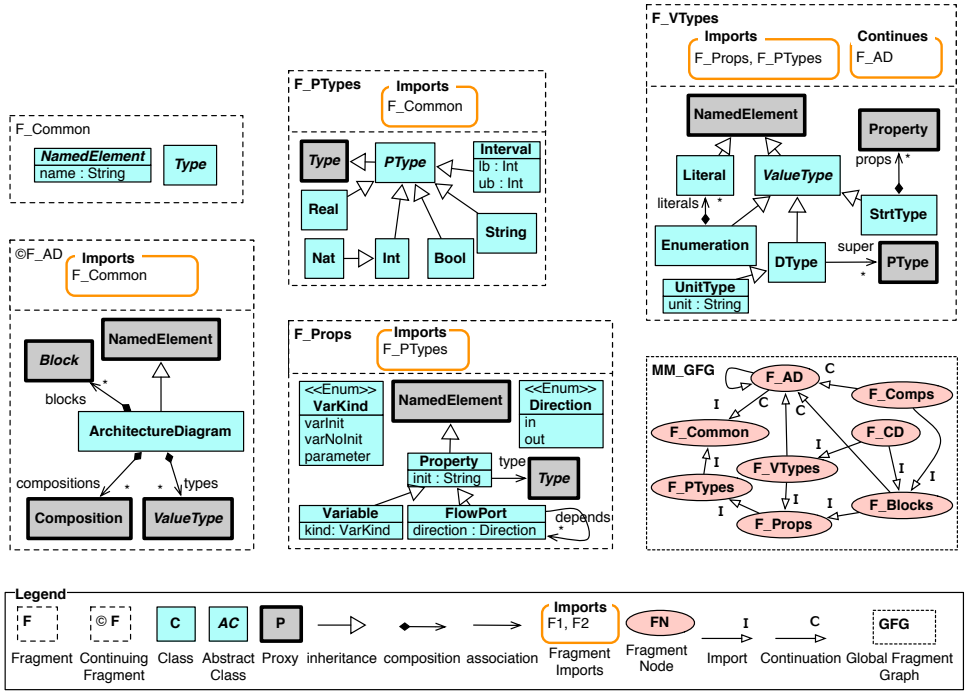


Fig. 1: Some fragments of metamodel of INTO-SysML.

2.1 Fragmenta and its Isabelle Mechanisation

FRAGMENTA [2] is a graph-based theory to represent modularised (or fragmented) typed class models. It is based on the algebraic theory of graphs and their morphisms [8]. FRAGMENTA represents designs of visual modelling languages whose structure is defined by class metamodels – domain-specific languages (DSLs) – and their resulting instance models. Its overall models are a collection of sub-models called *fragments*. Type and instance models are related through morphisms. A major novelty lies in FRAGMENTA's *proxies* – representatives of other nodes. A fragment is as a graph that supports proxies.

Figure 1 portrays five fragments and one global fragment graph (GFG) from INTO-SysML's metamodel. It highlights how fragments build up on other fragments either in a bottom-up (through imports) or top-down (through continues) fashion and the use of proxies for inter-fragment referencing. Importing is bottom-up because the bigger fragments are built from smaller ones. Continuation is top-down because it starts by specifying a summary model (or a skeleton) with points of continuation, represented as proxies, to be continued by other fragments. Fragment `F_PTypes` is an increment to `F_Common`; node `Type` from `F_Common` is referenced through the proxy with same name; likewise in `F_Props` with proxy `NamedElement`. Fragment `F_AD`, which summarises meta-

model of INTO-SysML architecture diagrams (ADs), is a continuing fragment; `F_VTypes` continues `F_AD`. The GFG (`MM_GFG`) describes the continues and imports relations between fragments.

FRAGMENTA proposes two composition operators: (a) union composition ($\cup_F$) merges fragments without resolving the proxies, and (b) colimit composition (based on category theory) joins fragments by resolving the proxies.

The theory introduces the following sets (see [2] for details):

- *Fr*, of well-formed fragments, requires that: (a) the underlying graph is well-formed, (b) the inheritance hierarchy is acyclic, (c) the source of composition relations has multiplicity 1 or $0\,..\,1$ and (d) proxies do not inherit[4]. All fragments in Fig. 1 are members of *Fr*.
- *GFGr*, of acyclic GFGs – `MM_GFG` (Fig. 1) $\in$ *GFGr*.
- *Mdl*, of all well-formed models, requires that the model's fragments are disjoint. A model M is a tuple (GFG, fd), made up of a $GFG \in GFGr$ and a total function $fd : Ns_{GFG} \to Fr$ mapping *GFG* nodes to fragments. INTO-SysML's metamodel, partially described in Fig. 1, is a member of *Mdl*.
- $F_1 \to_F F_2$, of all well-formed fragment morphisms, which impose the required graph commuting constraints in the setting of fragments.
- *FrTy*, of well-formed typed fragments $FT = (F, TF, ty)$; F and TF are instance and type fragments, respectively: $F, TF \in Fr$, and $ty \in F \to_F TF$.
- *FrTyConf*, of conformant fragments, a subset of *FrTy*, imposes the following constraints on instances: abstract nodes may not have direct instances, containments are not shared, instance relations satisfy metamodel multiplicities, and instances of containments form a forest.
- *MdlTy*, of all well-formed typed models $MT = (M, TM, ty)$, where M and TM are instance and type models – $M, TM \in Mdl$ –, and the type morphism is conformant – $(UFs\ M, UFs\ TM, ty) \in FrTyConf$, where *UFs* makes a single fragment out of the union of model fragments.

FRAGMENTA's Isabelle mechanisation[5] provides a verification and transformation environment for metamodel designs. One can check that:

- The individual fragments of both model and metamodel are locally consistent and well-formed. For fragment `F_Common` of Fig. 1, for instance, we need to prove $\vdash F\_Common \in Fr$[6]; likewise for the remaining fragments.
- GFGs are well-formed also. For GFG of Fig. 1: $\vdash MM\_GFG \in GFGr$.
- Overall models and metamodels are also consistent and well-formed. For the metamodel *INTO_SysML* of Fig. 1: $\vdash INTO\_SysML \in Mdl$.
- Instance models conform to the constraints imposed by the type model.

Section 3 gives further details on INTO-SysML inside FRAGMENTA/Isabelle.

[4] A local check that ensures the compositionality of FRAGMENTA's union operator.
[5] Available at `https://github.com/namalio/Fragmenta`
[6] Such membership predicates are represented in Isabelle as functions to booleans and they capture the well-formedness constraints associated with a FRAGMENTA set.

2.2 CSP and FDR3

The CSP process algebra [13] describes communicating processes and interaction-driven computations. CSP's major structuring concept, the process, represents a self-contained component made up of interfaces to enable interaction with a multitude of environments.

Processes communicate by transmitting information along channels. A CSP channel carries messages and has, therefore, a set of associated events, corresponding to all messages that may be transmitted. Process expressions are built using a number of operators, which include:

- Event prefixing, expressed as $e \rightarrow P$, describes a process that expects event e and then behaves as process P.
- External choice, $P_1 \Box P_2$, gives the environment the choice of events offered by P_1 and P_2. Replicated external choice $\Box\ i : \mathbb{N} \bullet P(i)$ composes the resulting processes using external choice.
- Internal choice, $P_1 \sqcap P_2$, non-deterministically chooses to act like P_1 or P_2.
- Parallel composition, $P_1 \underset{A}{\parallel} P_2$, executes the two processes in parallel synchronising on the set of events A.

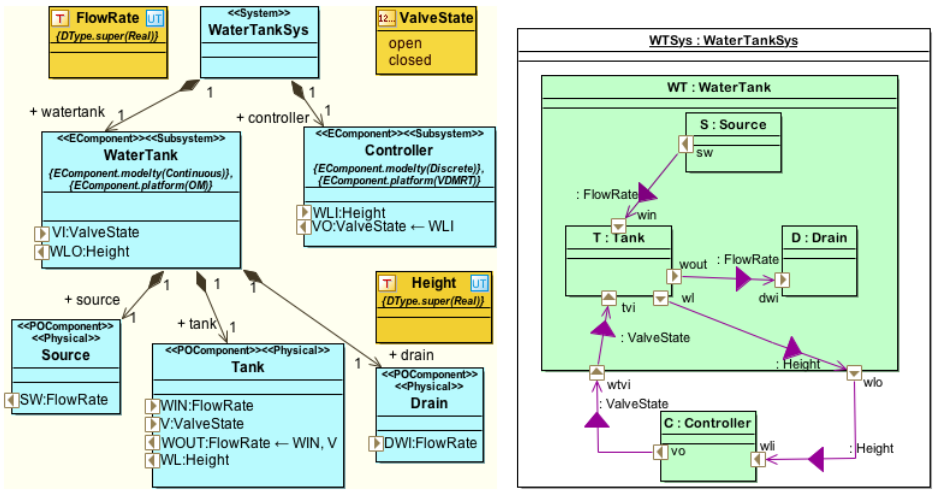
FDR3 [12] is CSP's refinement checker. It checks refinement according to CSP's denotational models (including traces, failures and failures-divergences), and other properties, including deadlock- and livelock-freedom, and determinism.

3 Architectural Modelling in INTO-SysML

The Systems Modelling Language (SysML) [25] is a general-purpose notation for systems engineering that builds up on the Unified Modelling Language (UML). The INTO-SysML profile [3] customises SysML for architectural modelling in a setting of multi-modelling and FMI co-simulation. It embraces the many themes of the INTO-CPS project[7], namely, tool interoperability, heterogeneity, holistic modelling and co-simulation, and constitutes the gateway into modelling in the INTO-CPS approach.

The profile introduces specialisations of SysML blocks (known as stereotypes) to represent different types of CPS components, constituting the building blocks that enable a hierarchical description of the CPS architectures that we need. A component is a logical or conceptual unit of the system, corresponding to a software or a physical entity. The profile's component constructs comprise: `System`, `EComponent` (encapsulating component) and `POComponent` (part-of component). A system is decomposed into subsystems (represented as `EComponents`), which are further decomposed into `POComponents`. `EComponents` and `POComponents` may be further classified as `Subsystem` (a collection of inner components), `Cyber` (an atomic unit that inhabits the digital or logical world)

[7] The INTO-CPS project aims to create an integrated "tool chain" for comprehensive model-based design of CPSs. For further information, see `http://into-cps.au.dk/`



(a) INTO-SysML AD (b) INTO-SysML CD

Fig. 2: The INTO-SysML model of the water tanks system

or `Physical` (an atom unit pertaining to the physical world). Furthermore, their characterising phenomena may be classified as `discrete` or `continuous`.

Currently, INTO-SysML comprises two diagram types: *architecture diagrams* (ADs) and *connections diagrams* (CDs), specialising SysML block definition and internal block definition diagrams, respectively. They are as follows (see Fig 2):

- ADs (see Fig. 2a) describe a decomposition in terms of the types of system components and their relations. They emphasise multi-modelling: certain components encapsulate a model built using some modelling tool (such as VDM/RT [20], 20-sim [17] or Open Modelica [11]).
- CDs (see Fig. 2b) are AD instances. They convey the configuration of the system's components, highlighting flow and connectedness.

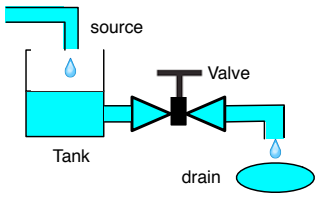


Fig. 3: Water Tanks system.

The water tanks system, sketched in Fig. 3, is this paper's running example. A source of water fills a tank whose water outflow is controlled by a valve; when the valve is open the water flows into the drain. The valve, managed by a software controller, is opened or closed depending on the tank's water level. We also consider a variant of this system with the drain connected to the tank.

Fig. 2 portrays the architectural model of water tanks, built using INTO-SysML's Modelio implementation[8]. The AD of Fig. 2a is as follows:

[8] Available from `http://forge.modelio.org/projects/intocps-modelio34`.

- The overall system (WaterTankSys) comprises two major subsystems, WaterTank and Controller, which are EComponents – they encapsulate separate models. WaterTank deals with continuous phenomena modelled in Open Modelica. Controller is discrete and modelled in VDM/RT.
- WaterTank has three physical sub-components: Source, Tank and Drain – they are POComponents (part-of a subsystem).
- Enumeration ValveState captures the valve's state. Unit types FlowRate and Height, built from reals, deal with flow rates and water levels.
- Each component provides flow ports to enable communication and the flow of material; the outputs indicate the inputs ports on which they depend.

CD of Fig. 2b describes the system instance (WTSys) composed of one WaterTank (WT) with its sub-components. The Controller instance (C) receives the water height from WT and, in return, directs WT to open or close the valve.

3.1 Well-formedness Checking using Fragmenta/Isabelle

Many things are checked in order to deem a INTO-SysML model, such as one of Fig. 2, consistent, well-formed and type conformant. Such checks are performed on the FRAGMENTA typed representation illustrated in Fig. 4. Fig. 4b gives the FRAGMENTA representation of CD in Fig. 2b and Fig 4a is the metamodel of INTO-SysML CDs; the correspondence from CD to metamodel, entailed by the type morphism, is represented as labels with numbers. In Fig. 4b, the proxies reference elements from the AD of Fig. 2a, nodes labelled 4 correspond to the connectors of the CD, and those labelled 5 correspond to ports.

From the FRAGMENTA base sets of Section 2.1, we build a set of well-formed INTO-SysML models *INTO_Mdls*, catering for all profile-specific invariants. The AD invariants are: (i) there is one system block, (ii) EComponents are not nested, and (iii) POComponents are contained by EComponents. The CD invariants are: (iv) instance ports are correctly typed with respect to AD flow ports, (v) connection's flow types correspond to types consistent with the ports being connected (conformity of connectors), and (vi) the CD satisfies multiplicities imposed by AD.

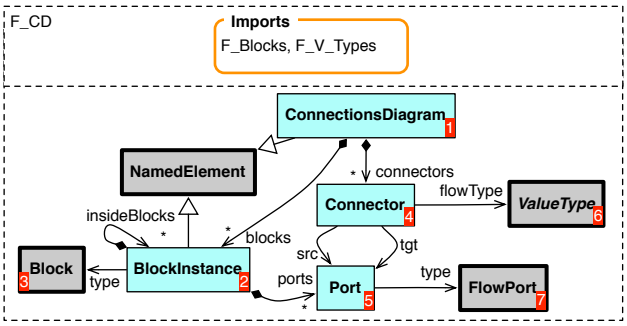
The model of Fig. 2, referred as M_WTs, is subject to the following checks:

- Fragments of AD and CD are well-formed: $\vdash F\_AD \in Fr$, $\vdash F\_CD \in Fr$.
- The model's GFG is well-formed: $\vdash GFG\_WTs \in GFGr$.
- Overall model is well formed: $\vdash M\_WTs \in Mdl$.
- M_WTs must be a valid INTO-SysML model. Given a type morphism ty (illustrated in Fig 4b), we prove: $\vdash (M\_WTs, ty) \in INTO\_Mdls$, which entails $\vdash (M\_WTs, INTO\_SysML, ty) \in MdlTy$.

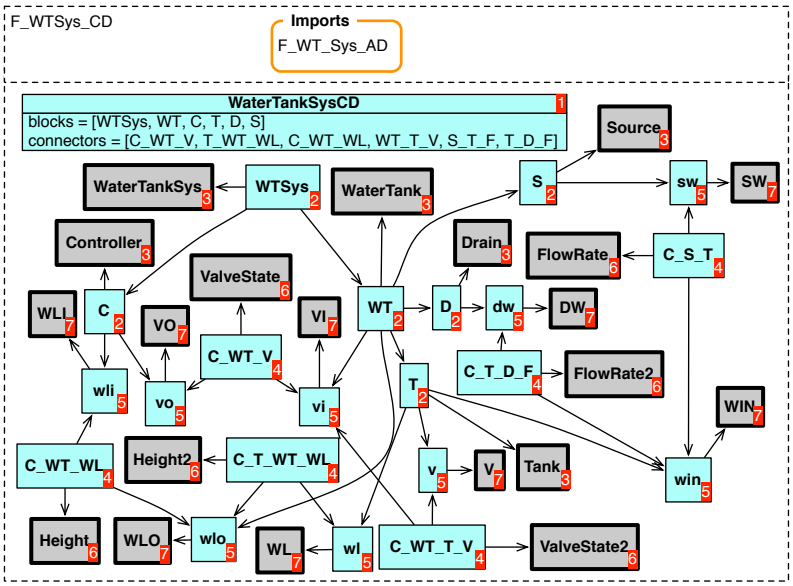
These are the checks required for any INTO-SysML model.

3.2 Fragmenta/Isabelle as a Transformation Engine

To enable usage of model-checkers, FRAGMENTA/Isabelle is used as a transformation engine in the algebraic loops check, which finds cycles in a topology of dependencies in instantaneous component communication (Fig. 5).



(a) Metamodel of INTO-SysML CDs

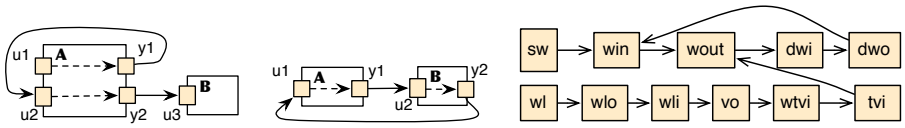


(b) INTO-SysML CD of water tanks system (Fig. 2b) in FRAGMENTA

Fig. 4: Metamodel, models and typing morphism (numbered labels) in FRAGMENTA illustrated with INTO-SysML CDs

Fig. 5a portrays a self-cycle component that is algebraic loop free. Output `y1` of `A` is connected to `A`'s input `u2`, but this does not entail an algebraic loop. The topology in Fig. 5b, on the other hand, contains an algebraic loop.

Finding algebraic loops equates to detecting cycles in a directed graph describing port dependancy relations. An edge between two ports indicates that the target node is instantaneously dependent on the source. This constitutes a *port dependancy graph* (PDG), illustrated in Fig. 5c, which portrays a PDG



(a) A topology without algebraic loops (b) A topology with an algebraic loop (c) PDG of a variant of Fig. 2 with algebraic loop

Fig. 5: The algebraic loops check is about finding cycles in a system topology emphasising component communication

with an algebraic loop corresponding to the variant of the INTO-SysML model of Fig. 2 that connects the `Drain` to the `Tank` (`dwo` to `win`).

Fig. 6: From INTO-SysML models to CSPm

The Isabelle mechanisation introduces a function that produces a PDG from a INTO-SysML model. The resulting PDG, obtained from the ports and connections of CD and the internal dependancies between output and input port types of AD, is derived from both metamodel and model. Another function takes the PDG and produces the CSPm specification to be checked in FDR3. This is summarised in the diagram of functions of Fig. 6.

4 Algebraic Loop Verification using CSP

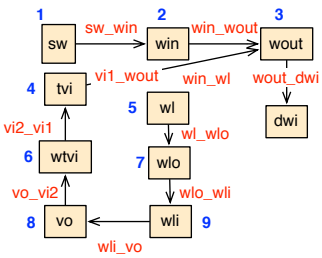


Fig. 7: A PDG with labelled nodes and edges

We represent graphs in CSP and detect cycles on them via a traces-refinement check executed in the FDR3 refinement checker[9]. This is illustrated with the PDG of Fig. 7 (derived from model of Fig. 2), containing labelled edges and numbers assigned to nodes with outgoing edges. We represent edges as CSP channels and nodes as CSP processes. The edges result in the following channel declaration:

$$\textbf{channel}\ sw\_win, win\_wout, wout\_dwi, tvi\_wout, wtvi\_tvi, vo\_wtvi, wli\_vo, \ldots$$

The overall graph is a CSP process constructed from sub-processes representing each node. The node processes are an external choice of CSP prefixed expressions for each edge that starts at the node. They offer the events on the corresponding channel and then behave as the process at the end of the edge. An edge to a sink node (no outgoing edges) results in a transition to $SKIP$. The main process is

[9] `https://www.cs.ox.ac.uk/projects/fdr/`

the external choice of all sub-processes. The process for PDG of Fig. 7 is:

$$
\begin{array}{l}
PortDependancyGraph = \\
\quad \text{let } P(1) = sw\_win \rightarrow P(2) \\
\qquad P(2) = win\_wout \rightarrow P(3) \\
\qquad P(3) = wout\_dwi \rightarrow SKIP \\
\qquad \vdots \\
\quad \text{within } \Box\ i : 1..9 \bullet P(i)
\end{array}
$$

Cycles are detected through traces refinement. The abstract CSP process to be refined defines all finite paths whose size is at most the number of edges in the graph (those that can be built by combining the graph's edges):

$$
\begin{array}{l}
edges = \{sw\_win, win\_wout, wout\_dwi, tvi\_wout, wtvi\_tvi, vo\_wtvi, \ldots\} \\
Limited = \\
\quad \text{let } Limited0(E, n) = \\
\qquad \text{if } n > 0 \text{ then } \Box\ e : E \bullet e \rightarrow Limited0(E, n-1) \sqcap SKIP \text{ else } STOP \\
\quad \text{within } Limited0(edges, 9)
\end{array}
$$

The traces refinement check to be executed in FDR3 is then:

$$\textbf{assert } Limited \sqsubseteq_{\mathrm{T}} PortDependancyGraph$$

All counter-examples are cycles. The function *toCSP* of Fragmenta/Isabelle (Section 3.2) yields CSP specifications as outlined above. For the PDG of Fig. 7, FDR3 gives no counter-examples; for Fig. 5c FDR3 yields one counter-example.

5 Evaluation

FDR3 is a tool based on model-checking, a verification technique whose drawback is scalability. We compare our CSP approach to detect algebraic loops (Section 4) against one approach based on Alloy [14] and one graph algorithm [15], to gauge scalability.

5.1 Experimental Setup

Scalability is evaluated against growing PDGs based on the water tanks running example (Fig. 3). We keep adding tanks to a base water tanks systems to produce systems of cascading water tanks having two versions: one with algebraic loops (drain is connected to first tank) and one without (as per Fig. 3).

The generation of files to execute in either FDR3, Alloy 4[10] or the implementation of Johnson's algorithm in JGraphT[11], involves Isabelle functions that yield PDGs given the number of tanks. We then define functions from PDGs

[10] `http://alloy.mit.edu/alloy/download.html`.
[11] A Java library of graph algorithms – `https://github.com/jgrapht/jgrapht`.

to the abstract syntax of CSP (as per Section 4), Alloy (see below) and Graph ML[12] as per diagram of Fig. 8[13].

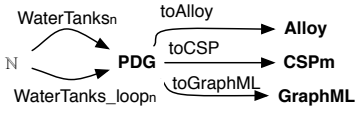


Fig. 8: The experiment's generation functions

The graph checks and data collection were performed by a Java program that reads the files and calls either Alloy 4 (using the minisat SAT solver), FDR3 or JGraphT, executed on a MacBook Pro with a 2.5 GHz Intel core i7 processor and 16GB RAM memory. The resulting data was subject to a statistical analysis carried out in the R statistical package [24].

5.2 The Alloy Model

Alloy [14] is a declarative modeling language based on first-order logic with transitive closure. It is used for data modelling and provides an automatic bounded analysis of a model. Our Alloy model of PDGs is based on the signature *Port*:

```
abstract sig Port {tgt :  set Port}{tgt ≠ this}
```

Above, we declare a set of `Port` instances – `abstract` says that `Port` has no instances of its own and that all its instances belong to its extensions (subsets) – with the relation `tgt` between `Port`s declared to be non-reflexive: the `tgt` of some `Port` cannot be itself (`this`).

The actual nodes of the PDG of Fig. 7 extend `Port`:

```
one sig sw, win, wout, dwi, wl, wlo, wli, vo, wlvi, tvi
   extends Port  {}
```

Above, the nodes are singletons (constraint `one`) that subset `Port` (`extends`).

The following Alloy fact defines the edges of the graph:

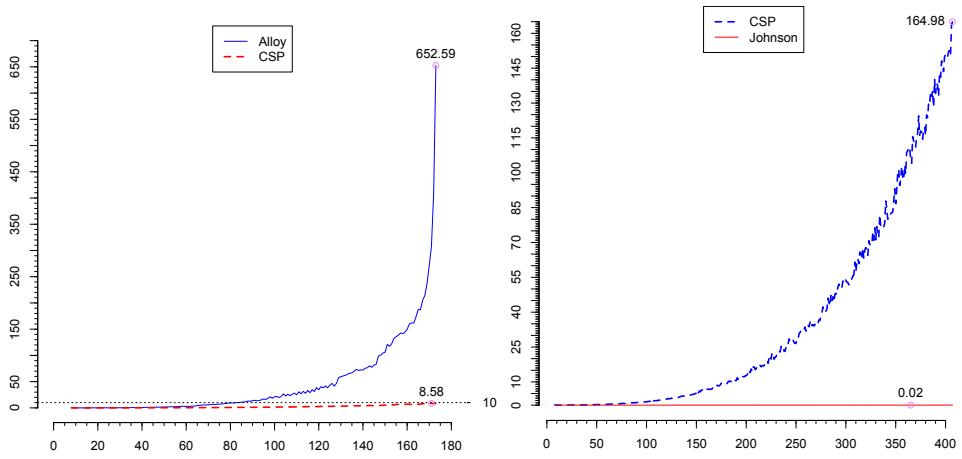
```
fact {sw.tgt = win
   win.tgt = wout
   wout.tgt = dwi
   no dwi.tgt ... }
assert AcyclicTgt {no ^tgt & iden}
check AcyclicTgt for 10
```

Above, each edge is declared through relation `tgt`: `sw.tgt =win` says that there is an edge from `sw` to `win` – operator `.` is the relational image –, `win.tgt =wout` says that there is an edge from `win` to `wout`, and `no dwi.tgt` says that `dwi` has no outgoing edges (set is empty).

Finally, we assert the acyclicity of the relation `tgt` representing the PDG and declare the command to check the assertion:

[12] A standard for graphs exchange that enables a direct representation of PDGs – `http://graphml.graphdrawing.org/`.

[13] The Isabelle file that performs the generation, the actual generated files, and the Java code that runs the three approaches, can be found at `http://bit.ly/1WKTIC7`.



(a) Alloy vs CSP

(b) CSP vs Johnson algorithm

Fig. 9: The performances of the Alloy and CSP solutions (seconds on the ordinate and number of nodes of a graph on the abscissa)

```
assert AcyclicTgt {no ^tgt & iden}
check AcyclicTgt for 10
```

Above, the assertion says that there can be no elements (operator `no`) in the set resulting from the intersection (operator `&`) of the relation's transitive closure (`^tgt`) with the identity relation (`iden`). The `check` command includes a scope declaration: the analysis should consider at most 10 PDG nodes.

5.3 Comparisons

The plots of Fig. 9 depict the data obtained from running the experiments. They display the number of nodes of the analysed graph in the abscissa and the duration of the check (in seconds) in the ordinate.

Fig. 9a shows that there is an overwhelming difference in favour of CSP against Alloy. CSP's maximum duration is $8.58s$, Alloy's is $652.59s$. The two approaches start to diverge with small to medium size graphs (number of nodes > 17). The p-value, obtained from the paired data plotted in Fig. 9a using the Wilcoxon statistical test[14], of $< 2.2^{-16}$ (< 0.001) indicates a very large difference. We derived estimates of functions that fit the data of both Alloy and CSP to yield estimates of time complexity: Alloy has complexity $O(Exp)$, whereas CSP has complexity $O(n^3)$ – n is number of nodes of graph.

Fig. 9b, on the other hand, shows that Johnson's algorithm performs substantially better than CSP. The former's maximum duration is $0.02s$, CSP's is

[14] It is a non-parametric test that compares the two sampled distributions without assuming that they follow the normal distribution.

$164.98s$. The p-value of $< 2.2^{-16}$ (< 0.001) signals a very large difference. The estimated function that fits the data endorses the algorithm's linearity claim.

6 Discussion

The following discusses the results presented in the paper.

A prelude to co-simulation. The work presented here statically checks an architectural design of a CPS in preparation for co-simulation. This is done at the high-level architectural design to provide early warnings of any issues so that the appropriate remedial action can be taken. It is a preliminary check – done before delving into the details of global co-simulation and local modelling and analysis of each component – to ensure that the models to be co-simulated are, among other things, free of connector inconformities and algebraic loops. These checks are performed using the Isabelle proof assistant and the FDR3 model-checker; both constitute an intimate part of our verification toolset.

Into-SysML profile. The paper presents a profile of SysML (defined in [3]), designed as a DSL, for architectural modelling of CPSs supporting multi-modelling and FMI co-simulation. The profile embodies an implicit systems decomposition paradigm driven by multi-modelling: the overall system architecture is a decomposition of subsystems (`E-components`), encapsulating their own models, which are further decomposed into `POComponents` to give an account of the inner structure of each subsystem. The profile enables a holistic algebraic loop analysis that considers the inner details of each subsystem. Guidance on the definition of SysML models for multi-modelling is provided in [10], aiding CPS engineers in modelling a CPS architecture both holistically and in a decomposed form suitable for co-simulation.

The profile's design caters for FMI co-simulation. The `E-component` subsystems of the architecture result in FMI's Functional Mock-up Units (FMUs) to be co-simulated; FMUs are generated by the corresponding modelling framework.

Fragmenta/Isabelle as prototyping environment. The profile's DSL design was brought to life by Fragmenta and its accompanying Isabelle mechanisation. Fragmenta/Isabelle, built as part of the work presented here, constitutes a prototyping environment built on top of Fragmenta's mathematical theory that provides reasoning and transformation capabilities for metamodels and their instances. As this paper demonstrates, it can be used in real-world settings; ideally, however, Fragmenta designs should be specialised and optimised as part of fully fledged visual modelling environments.

Algebraic loops. The algebraic loops healthiness check is performed on a graph describing the instantaneous dependencies between ports extracted from INTO-CPS architectural models; external port connections are derived from the CD and internal ones from the AD. Internal and external port dependencies of the INTO-SysML model must be consistent with the underlying model equations.

It is interesting to contrast the two model-based approaches to check algebraic loops. Alloy represents a graph directly (Section 5.2) as a relation between

nodes; the property to check is stated as an ordinary relational calculus formula. The CSP approach (Section 4), on the other hand, is edge-oriented to suit CSP's communication model based on channels; a graph is the communications established between nodes (CSP processes) chosen from the environment (external choice); the property is expressed in an ingenious, but less evident way: through an abstract process and a traces refinement check.

FDR3 and Alloy 4 are both based on model-checking; however, the CSP solution outperforms Alloy overwhelmingly. Alloy's exponential time complexity is attributed to the complexity of SAT whose worst-case time complexity is exponential [21,19] – the Alloy solution resorts to the transitive closure, a computationally demanding operation (specialised algorithms do it in $O(n^3)$). An important factor in CSP's lower $O(n^3)$ time complexity lies in the use of traces refinement, founded on the simplest denotational model of CSP and with the least expensive time complexity – polynomial according to [16].

Our CSP solution is beaten by Johnson's algorithm, but it is used in our verification approach, which employs FDR3 for more sophisticated checks of FMI co-simulations [1]. It is difficult for general-purpose model-checking to outperform specialised algorithms taking advantage of problem specificities.

The experimental setup varies size but not structure, which remains essentially the same throughout the different water tanks systems. However, as the results show, this is enough to expose differences; furthermore, as discussed above, the obtained results are consistent with theoretical results.

7 Related Work

Feldman et al [9] generate FMI model descriptions from Rhapsody SysML models and FMUs from statecharts to enable integration with continuous models. Unlike our work, this does not define a profile embodying a paradigm designed for multi-modelling and FMI-co-simulation; furthermore, formal static checks covering connector conformity and absence of algebraic loops are not covered. Pohlmann et al [23] propose a UML-based DSL for real-time systems; FMI FMUs are generated from model components described as real-time statecharts; our work specialises the SysML block diagrams, a standard notation for architectural modelling, and supports multi-modelling.

This paper applies the Fragmenta theory presented in [2] to a real-world problem. This required an extension to the Isabelle/HOL theory of [2], developed to prove that paper's main theorem. This extension builds an infrastructure to support automated verification and transformation for visual modelling languages. Fragmenta/Isabelle constitutes a prototyping environment supporting all the novel aspects of Fragmenta, namely: a formal theory of proxies and its verified theory of decomposition and the support for fragmentation strategies. This is the first time that the novel theory of modularity with its Isabelle mechanisation is applied to a real-world application. To our knowledge, this is also the first prototyping environment based on a proof assistant that provides formal reasoning and transformation capabilities for visual models.

The approach to connector conformity used here is based on typing. It supports sub-typing according to the inheritance relations specified in the metamodel; for instance, in INTO-SysML, natural numbers may be used when integers are expected because the metamodel says that the former is a subtype of the latter. This is checked as part of FRAGMENTA's typing morphisms. This is different from the *connector compatibility* of [7], which performs validations based on interface contracts, a relation between allowed inputs and outputs [26].

Broman et al [6] require that FMI component networks are algebraic-loop free as a pre-condition to the deterministic composition results of their FMI master algorithms, proposing port-dependency graphs as a means to perform such checks. Unlike the work presented here, [6] does not study different approaches to detect algebraic loops; it suggests algorithms that topologically sort a graph, which yield an error if the graph has a cycle. Our algebraic loop analysis provides actual cycles as feedback to designers.

8 Conclusions

This paper has presented our approach to check a SysML model in preparation for co-simulation. This involves checking the consistency and well-formedness of the INTO-SysML model, which involves checking the conformance of the model with respect to its metamodel based on FRAGMENTA's representation. The actual checks are carried out using FRAGMENTA's Isabelle mechanisation, ensuring, among other things, connector conformity. The paper then showed how the INTO-SysML models could be transformed into other modelling languages to perform a check for the absence of algebraic loops using FRAGMENTA's Isabelle mechanisation as a transformation engine. The paper presented a novel CSP approach to detect algebraic loops by checking a traces refinement in FDR3. The paper's evaluation highlighted how our CSP approach based on refinement-checking performs well when compared with an Alloy SAT-based model-checking approach, but that it does not perform better than a special-purpose graph algorithm. The work presented in this paper is done in tandem with the effort on the formal semantics of FMI in CSP [1].

Acknowledgements. This work was supported by the EU project INTO-CPS (Horizon 2020, # 644047, `http://into-cps.au.dk/`). Thanks are due to Etienne Brosse, who implemented the INTO-SysML profile in the Modelio tool, and Bernhard Thiele, who provided useful feeedback on the work presented here.

References

1. Amalio, N., Cavalcanti, A., König, C., Woodcock, J.: Foundations for FMI Co-Modelling. Tech. rep., INTO-CPS Deliverable, D2.1d (December 2015)
2. Amálio, N., de Lara, J., Guerra, E.: FRAGMENTA: A theory of fragmentation for MDE. In: MODELS 2015. IEEE (2015)
3. Amalio, N., Payne, R., Cavalcanti, A., Brosse, E.: Foundations of the SysML profile for CPS modelling. Tech. rep., INTO-CPS Deliverable, D2.1a (December 2015)

4. Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A.: The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In: Modelica Conference. Munich, Germany (2012)
5. Blochwitz, T.: Functional mock-up interface for model exchange and co-simulation. https://www.fmi-standard.org/downloads (July 2014), torsten Blochwitz Editor
6. Broman, D., Brooks, C., Greenberg, L., Lee, E., Masin, M., Tripakis, S., Wetter, M.: Determinate composition of FMUs for co-simulation. In: EMSOFT (2013)
7. Dragomir, I., Preoteasa, V., Tripakis, S.: Compositional semantics and analysis of hierarchical block diagrams. In: SPIN 2016. pp. 38–56 (2016)
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
9. Feldman, Y., Greenberg, L., Palachi, E.: Simulating Rhapsody SysML Blocks in Hybrid Models with FMI. Modelica Conference pp. 43–52 (2014)
10. Fitzgerald, J., Gamble, C., Payne, R., Pierce, K.: Method Guidelines 1. Tech. rep., INTO-CPS Deliverable, D3.1a (December 2015)
11. Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press (January 2004)
12. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: FDR3 — A Modern Refinement Checker for CSP. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 8413, pp. 187–201 (2014)
13. Hoare, T.: Communication Sequential Processes. Prentice-Hall International, Englewood Cliffs, New Jersey 07632 (1985)
14. Jackson, D.: Software abstractions: logic, language, and analysis. MIT Press (2012)
15. Johnson, D.B.: Finding all the elementary circuits in a directed graph. SIAM Journal of Computing 4(1), 77–84 (1975)
16. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. Information and Computation 86(1), 43–68 (1990)
17. Kleijn, C.: Modelling and Simulation of Fluid Power Systems with 20-sim. Intl. Journal of Fluid Power 7(3) (November 2006)
18. Kübler, R., Schiehlen, W.: Two Methods of Simulator Coupling. Mathematical and Computer Modelling of Dynamical Systems 6(2), 93–113 (2000)
19. Kullmann, O.: New methods for 3-sat decision and worst-case analysis. Theoretical Computer Science 223(1–2), 1–72 (1999)
20. Larsen, P.G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., Verhoef, M.: The Overture Initiative – Integrating Tools for VDM. SIGSOFT Softw. Eng. Notes 35(1), 1–6 (January 2010)
21. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2n steps. Discrete Applied Mathematics 10(3), 287–295 (1985)
22. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer (2014)
23. Pohlmann, U., Schäfer, W., Reddehase, H., Röckemann, J., Wagner, R.: Generating Functional Mockup Units from Software Specifications. In: Modelica Conference (2012)
24. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2015), `https://www.R-project.org/`
25. OMG Systems Modeling Language (OMG SysML^{TM}). Tech. Rep. Version 1.3, SysML Modelling team (June 2012), http://www.omg.org/spec/SysML/1.3/
26. Tripakis, S., Lickly, B., Henzinger, T.A., Lee, E.A.: A theory of synchronous relational interfaces. ACM TOPLAS 33(4) (2011)