10.20378/irb-97634

BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 106

Heterogeneous Specification of Spacecraft Software

Eugene Yip Gerald Lüttgen

August 2024

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG

Lehrstuhl Softwaretechnik & Programmiersprachen
Fakultät Wirtschaftsinformatik & Angewandte Informatik

Heterogeneous Specification of Spacecraft Software

Research support provided by the DFG (German Research Foundation)
under grant nos. LU 1748/3-2 & VO 615/12-2 for the project
"Foundations of Heterogeneous Specifications Using State Machines and Temporal Logic"

Eugene Yip and Gerald Lüttgen
Software Technologies Research Group, University of Bamberg, Bamberg, Germany

August 23, 2024

Abstract: The operational behaviour of a reactive system is commonly specified or modelled as concurrent state machines, where each machine models the possible states or modes of a software component and its interactions with the environment. However, state machines can quickly become verbose when execution constraints between concurrent states need to be modelled. Alternatively, constraints could be modelled declaratively as special edges between states. Such a heterogeneous modelling approach is employed by *Virtual Satellite (VirSat)*, a model-based systems engineering tool from the *German Aerospace Center (DLR)*. The challenge has been to develop a heterogeneous modelling framework that supports an operational and declarative syntax, defines a unified semantics suitable for formal reasoning, and supports the independent and incremental development of software components.

We address this challenge with our *Component State Machine (CSM)* formalism as a means to operationalise VirSat: the operational syntax of VirSat is preserved in CSM, the declarative VirSat constraints are transformed into CSM atomic propositions and parallel conditions, the CSM parallel operator incrementally composes CSMs and resolves the parallel conditions, and design errors are analysed as inconsistent transitions. CSM permits new constraints to be defined without needing to modify the core VirSat semantics. We also propose an intuitive, practical, and mathematically rigorous notion of refinement that aims to encourage a more systematic development of spacecraft software. Lastly, we offer a modular implementation of CSMs as hierarchically scheduled *Executable State Machines (ESMs)* that remains open to further composition and refinement.

Contents

1 Introduction

Virtual Satellite (VirSat) [1, 2] is a *Model-based Systems Engineering* tool for designing all aspects of a spacecraft, including software for managing the operating modes of the spacecraft's equipment. Notably, VirSat offers a heterogeneous modelling language where State Machines (SMs) capture operational behaviour which is overlaid with declarative syntax to capture operational constraints between states. Thus, operational and declarative specifications are tightly coupled together in VirSat. When a spacecraft conducts science experiments, some of its equipment may need to be run concurrently and others exclusively, depending on their resource requirements [3], such as processor time, memory, power, transmission devices, observation time windows, and spatial orientation. In VirSat, such operational and resource requirements are modelled declaratively as special edges (*constraints*) between pairs of concurrent SMs and these need to be respected when the ground station schedules the science experiments. VirSat is not concerned with how the constraints are implemented, but rather their effects on spacecraft operation.

The current formal semantics of VirSat [2] is defined at a global level, where all the SMs of a VirSat design are assumed to be available. Every time the design is modified, even for one SM, the semantics of the entire design has to be recomputed at once. Such a semantics impedes support for the independent and incremental development of spacecraft software. In this article, we propose a semantic framework that is inspired by interface theories, such as *Interface Automata* [4, 5], that address the development of concurrent software components. In such a theory, an initial abstract specification of a system's concurrent components is defined, which is made more concrete as design details are *refined* (worked out) in an incremental manner. Different teams may be tasked with developing and supplying various components, which are then *composed* together into a system implementation. Ideally, the behaviour of an implementation is a subset of its abstract specification. That is, an implementation does not introduce behaviour or errors that were not present in its specification, but can only remove them. Such a refinement of abstract specifications into concrete implementations facilitates the independent and incremental development of components by different teams, and the ability to freely substitute components with refined implementations.

1.1 Contributions

Our proposed semantic framework for VirSat is captured in Figure 1. It begins with the existing *VirSat State Machines (VSMs)* as the frontend language that engineers will continue to use to design their spacecraft (Section 2). The VSMs are transformed into functionally equivalent *Component State Machines (CSMs)* that serve as an intermediate language on which the formal semantics are defined (Section 4). CSM follows the philosophy of Interface Automata in the sense that CSM promotes the independent development, incremental design, and reuse of components, which are all absent from VirSat. These capabilities are enabled by CSM's associative and commutative *parallel operator* for the arbitrary composition of CSMs, *parallel conditions* for defining custom constraints, and a *refinement relation* for safely improving or substituting components without introducing new design errors. Moreover, CSM also extends VSM by supporting nondeterministic transitions for the modelling of incomplete specifications, and fine-grained deadlock analysis via our notion of *inconsistent transitions*.

VSMs are transformed systematically into functionally equivalent CSMs (Section 5): the constrained states of the VSMs are enriched with atomic propositions, and each constraint

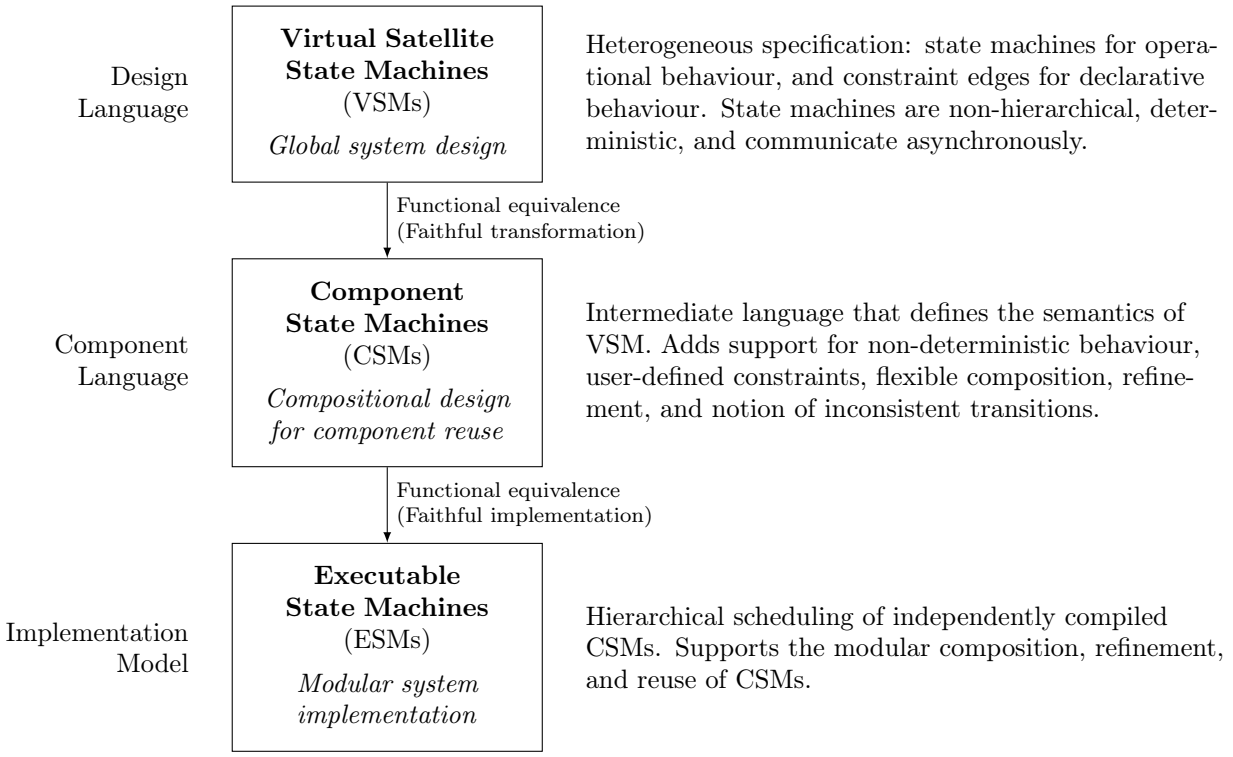


Figure 1: Overview of the Component State Machine framework that extends the VirSat semantics and offers a modular implementation.

is encoded as a parallel condition that is evaluated during parallel composition. Hence, VSMs are transformed into open systems that are capable of further modifications and compositions. This could help tackle the state space explosion problem [2] by facilitating incremental verification of spacecraft subsystems. Currently, VirSat systems are checked in their entirety via a combination of model checking for detecting (local) deadlocks and graph exploration for detecting cyclic dependencies between constraints. Section 6 describes how two new experimental constraints, *Allow* and *Multi-forbid*, can be easily introduced into VirSat by encoding their semantics as CSM atomic propositions and parallel conditions.

In practice, engineers typically "refine" their VirSat systems in an ad hoc manner that is not semantics preserving, i.e., new functionalities and errors may be introduced. Such a system refinement is typically treated as a new system that must be completely reanalysed. We observe that the ability for a spacecraft to reach its operational modes is more important than the commands needed to trigger the mode transitions. Thus, we propose an intuitive and practical refinement approach (Section 3): transitions that are nondeterministic or act as "shortcuts" for sequences of other transitions can be removed. We hope that this will encourage engineers to refine their spacecraft in a more systematic manner.

We demonstrate the executability of CSMs by implementing them as functionally equivalent Executable State Machines (ESMs) that are scheduled in a hierarchical manner (Section 7). Each scheduler is responsible for finding the valid next transitions of its ESMs and the root scheduler is responsible for selecting one such transition to execute. Although the process of implementing a system necessarily closes it off from further refinement and composition, we take a modular approach that retains some flexibility. For example, ESM implementations can be composed together by assigning them to a new root scheduler, and an ESM can be replaced by a refined version.

1.2 Related Work

The operational behaviour of concurrent software is commonly modelled as a parallel composition of *State Machines (SMs)*. Many SM-based tools are in use today, such as UML State Machines [6], MATLAB Stateflow [7], SCADE State Machines [8], Modelica State Machines [9], and KIELER SCCharts [10]. Closely related is *Modechart* [11] for modelling collections of related states as modes with timed transitions, with tool support from MATLAB Simscape Mode Charts [7], Verif Modechart Designer [12], and MT toolset [13]. Although SMs (and Modecharts) are used extensively by the industry, they are not effective at capturing declarative behaviour, such as execution constraints between concurrent states. Such behaviour would have to be modelled as annotations, or explicitly as states and transitions which can obfuscate the intended behaviour of the software [14]. In this case, it is desirable to use a heterogeneous modelling language that concisely captures both operational and declarative specifications.

While many heterogeneous modelling languages for reactive systems do exist, none offer the same tightly coupled operational and declarative behaviour of VirSat and a formal semantics that supports parallel composition and refinement in the sense of Interface Automata [4]. Instead, components specified in different modelling paradigms can be composed together, but each component must be of a specific paradigm [6, 7, 8, 9, 10, 15], e.g., state machine or imperative programming language for operational behaviour, and block diagram or functional programming language for declarative behaviour.

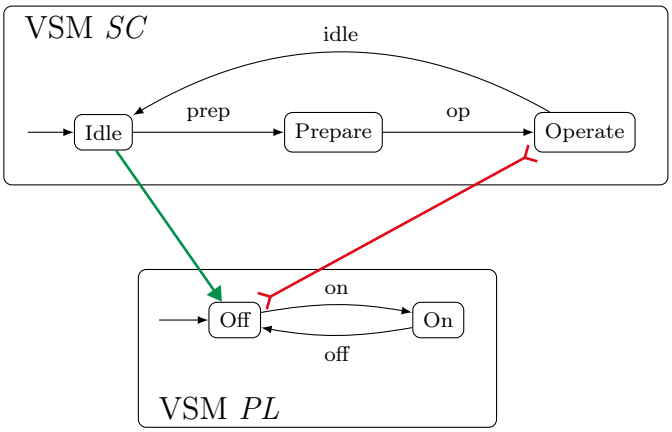
Ptolemy II [16] is a modelling and simulation framework for studying heterogeneous systems with a range of modelling paradigms. Each component is modelled in a specific paradigm and the semantics of their composition is governed by the Ptolemy execution engine. The possible interaction that a paradigm offers is specified using Interface Automata, which allows the compatibility of different paradigms to be checked. While the declarative constraints of VirSat could be defined as a paradigm in Ptolemy, the formal semantics of Ptolemy [17] does not offer a compositionality result to the degree of Interface Automata, and offers no formal rules for component refinement. Closely related is GEMOC Studio [18] for creating heterogeneous languages and models. Again, its formal semantics [19] does not offer the compositionality and refinement results that we demand for VirSat.

2 Virtual Satellite State Machines (VSMs)

To understand the technical concepts behind VirSat [1], we briefly review the syntax and semantics of VirSat State Machine (VSM) as presented by Chrszon et al. [2]. We will build on these concepts in our formalisation of CSM in Section 4, and in our semantics-preserving transformation of VSM to CSM in Section 5. Although we present VSM with slightly different notation, the semantics remains unchanged. A VSM specifies the operating modes and mode transitions of a spacecraft equipment as a state machine. Each mode transition is triggered by a command sent from a ground station.

Definition 1 (VirSat State Machine)**.** A *VirSat State Machine* is a tuple $(P, \Sigma, \rightarrow, p_0)$, where

- P is the set of *states* with $p_0 \in P$ being the *initial state*;
- Σ is the *alphabet* of actions (spacecraft commands) and excludes the distinguished *internal action* τ;
- $\rightarrow \subseteq P \times \Sigma \times P$ is the *transition relation*, and $p \xrightarrow{a} p'$ is written for $(p, a, p') \in \rightarrow$.



(a) Spacecraft and Payload VSMs specifying their operating modes (states), transitions (⟶ between sequential states), and Forbid and Required constraints (✕ and ⟶ between concurrent states, respectively).

$SC = \{\text{Idle, Prepare, Operate}\}$
$\Sigma_{SC} = \{\text{prep, op, idle}\}$

$PL = \{\text{Off, On}\}$
$\Sigma_{PL} = \{\text{on, off}\}$

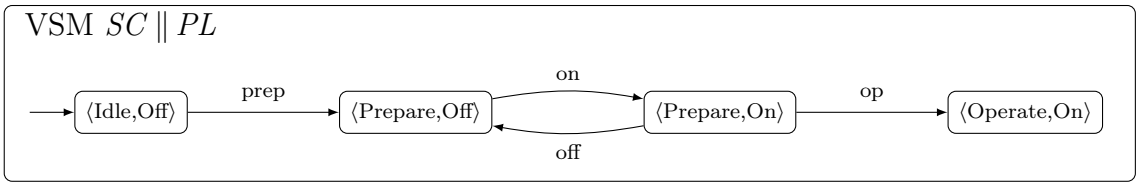
(b) States and actions of the Spacecraft and Payload VSMs.

$\mathcal{F} = \{(\text{SC.Operate, PL.Off})\}$
$\mathcal{R} = \{(\text{SC.Idle, } \{\text{PL.Off}\})\}$

(c) Forbid and Required constraints.

SC.prep, PL.on, SC.op

(d) A possible command sequence.



(e) Parallel composition of the Spacecraft and Payload VSMs.

Figure 2: Example VirSat *Spacecraft (SC)* and *Payload (PL)* components from [2].

The VSM is required to be deterministic, i.e., $p' = p''$ whenever $p \xrightarrow{a} p'$ and $p \xrightarrow{a} p''$ for some $p \in P$ and $a \in \Sigma$;

- (Σ) is the *signature* of the VSM.

We use p and a as representatives of the sets P and Σ, respectively, and simply write P for a CSM $(P, \Sigma, \rightarrow, p_0)$.

A VirSat system consists of VSMs operating together in an interleaving manner with no need to synchronise on common actions, i.e., VSMs have pairwise disjoint actions. The reason is that, particularly for near-Earth spacecraft with negligible communication time, the ground station is responsible for commanding the spacecraft's equipment into the modes needed to fulfil science experiment and mission goals [20]. The ground station generates an operating schedule and sends command sequences to manually manage the spacecraft's equipment. Thus, there is no need for the equipment to actively synchronise their operations with each other. Additionally, spacecraft software typically has a flat structure, so VirSat does not support state hierarchy. A small example of a VirSat system, adopted from [2], is given in Figures 2(a) and 2(b).

A correct operating schedule must at least consider the spacecraft's resource and operating constraints, e.g., the power demand for operating a set of equipment, the concurrent use of an equipment by others, or the dependency of an equipment on another. Such constraints are typically derived from tables [20] that specify the equipment modes required during different spacecraft operations and mission phases. VSM allows so-called *Forbid* and *Required* constraints to be specified between the states of different VSMs.

Definition 2 (VirSat Forbid Constraint)**.** A *Forbid constraint* between two different VSMs, P and Q, is defined as $(p, q) \in P \times Q$. It specifies that states p and q cannot be active together. The set of Forbid constraints in a VirSat system is denoted $\mathcal{F}$.

Definition 3 (VirSat Required Constraint)**.** A *Required constraint* between two different VSMs, P and Q, is defined as $(p, \{q_1, \ldots, q_k\})$ with $p \in P$ and $\{q_1, \ldots, q_k\} \subseteq Q$. It specifies that state p can only be active when one of the states $q_1, \ldots, q_k$ is also active. We call p the *requiring* state and $q_1, \ldots, q_k$ the *required* states. The set of Required constraints in a VirSat system is denoted $\mathcal{R}$.

A VSM can only take a transition to a target state if, after the transition, all constraints of all active states of the system are satisfied. Moreover, while a state is active, all its constraints have to remain satisfied. Thus, constraints restrict the possible interleaving among the VSMs to *valid* state combinations that satisfy all the constraints in $\mathcal{F}$ and $\mathcal{R}$. For the VirSat system of Figure 2(a), Figure 2(c) defines its Forbid and Required constraints, and Figure 2(d) defines a possible command sequence that respects these constraints. In particular, *SC.prep* is necessarily the first command because command *PL.on* would violate the Required constraint; while the *Spacecraft* is in *Idle*, the *Payload* is required to be in *Off*. The second command has to be *PL.on* because command *SC.op* would violate the Forbid constraint; the *Spacecraft* is forbidden from being in *Operate* while the *Payload* is in *Off*.

Definition 4 (VSM Valid State)**.** Let $P_1, \ldots, P_n$ be a set of VSMs and let $G \in P_1 \times \cdots \times P_n$ be a state configuration of the VSMs. G is a *valid state* if all the following conditions hold:

1. $\forall\, (p, q) \in \mathcal{F}.\ \overline{p \in G \wedge q \in G}$;
2. $\forall\, (p, \{q_1, \ldots, q_k\}) \in \mathcal{R}.\ p \in G \Rightarrow \exists q \in \{q_1, \ldots, q_k\}\ .\ q \in G$.

The first condition says that both states of a Forbid constraint cannot be active together. The second condition says that the requiring state of a Required constraint can only be active if one of its required states is also active. While a requiring state is active, it is possible for the required states to transition between each other. Note that all the constraints at a state are evaluated together conjunctively. Indeed, the addition of a new constraint can cause states to become over-constrained, where all their target states become invalid, resulting in local deadlocks that prevent the spacecraft from accepting further commands.

The operational semantics of VirSat is defined in [20] as a transition system over the entire VirSat system of VSMs. The resulting transition system is essentially another VSM, called the parallel composition, with all constraints resolved by limiting transitions to only valid states.

Definition 5 (VirSat Parallel Composition)**.** VSMs $\mathcal{P} =_{\text{df}} \{P_1, \ldots, P_n\}$ are composable if $\forall\, \{P_i, P_j\} \subseteq \mathcal{P}.\ \Sigma_{P_i} \cap \Sigma_{P_j} = \emptyset$. Let $\mathcal{F}$ and $\mathcal{R}$ be the sets of Forbid and Required constraints of $\mathcal{P}$, respectively. The *parallel composition* $P_1 \parallel \cdots \parallel P_n$ of such composable VSMs is defined as $(P_1 \times \cdots \times P_n, \Sigma_{P_1} \cup \cdots \cup \Sigma_{P_n}, \rightarrow, \langle p_{1_0}, \ldots, p_{n_0} \rangle)$, where the transition relation $\rightarrow$ is the least relation satisfying the following rule:

$G \xrightarrow{a} G'$ if $G, G' \in P_1 \times \cdots \times P_n$, G and G' are valid states, $\exists P_i \in \{P_1, \ldots, P_n\}\ .\ p \xrightarrow{a}_{P_i} p'$, $p \in G$, and $p' \in G'$.

A transition in a VirSat parallel composition can only be between two valid state configurations, G and G'. Because the VSMs do not share any actions, the action a in $G \xrightarrow{a} G'$ can only be from one VSM P_i taking a transition $p \xrightarrow{a}_{P_i} p'$. Hence, VSMs execute in an interleaved manner and G' differs from G by exactly one substate when $p \xrightarrow{a}_{P_i} p'$ is not a

self-loop. Figure 2(e) is the parallel composition of the *Spacecraft (SC)* and *Payload (PL)* VSMs from Figure 2(a). Note that, although *SC* and *PL* are cyclic in behaviour, their parallel composition is not; when state $\langle Operate, On\rangle$ is reached, the parallel composition cannot react to further commands and becomes inoperable. By analysing the parallel composition of a VirSat system, it is possible to identify *local deadlocks* where some VSMs are unable to react to further commands because their constraints cannot be satisfied. Such deadlocks point to design issues that have to be rectified.

Definition 6 (VSM Local and Global Deadlock)**.** Let $P = P_1 \parallel \cdots \parallel P_n$ be the parallel composition of VSMs $P_1, \ldots, P_n$, and let $p_i \in P_i$ where $P_i \in \{P_1, \ldots, P_n\}$. Then state configuration $G \in P$ is a *local deadlock* if $p_i \in G$ and $\forall p_i \xrightarrow{a}_{P_i} p_i'$, $\nexists G \xrightarrow{a}_P G'$. If state configuration $G \in P$ is a local deadlock for all $p_i \in G$, we say that G is a *global deadlock*.

Returning to the example of Figure 2(e), state $\langle Operate, On\rangle$ is clearly a global deadlock because neither *SC* no *PL* can take a transition without violating their constraints.

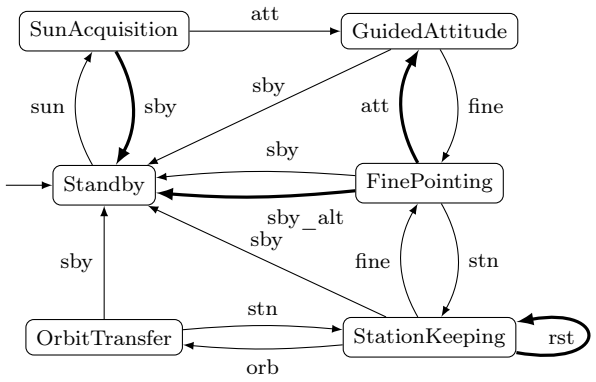
Definition 5 for parallel composition has not been designed to support the incremental composition of VSMs, but instead requires all the VSMs and their constraints to be known upfront. Moreover, every time a VSM is modified, the entire parallel composition has to be computed again. This precludes the ability to simplify a system of VSMs into an intermediate parallel composition of VSMs, whose designs are closed from further modification, alongside VSMs that are still open to modification.

The main hurdle to supporting incremental composition is that Definition 5 is unable to properly evaluate the Required constraints on an intermediate parallel composition. When a requiring state is in the intermediate composition, but the VSM of its required states has yet to be composed, the requiring state would not be able to satisfy Cond. 1 of Definition 4. Thus, the requiring state would be excluded completely and prematurely from the transition relation.

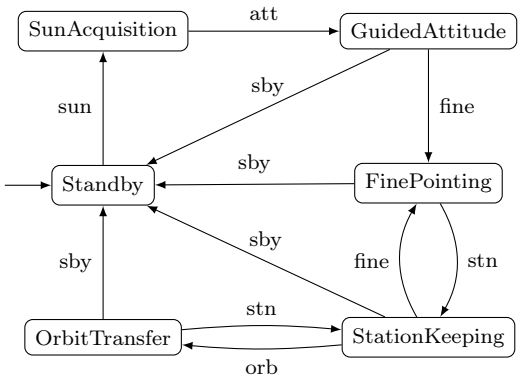
3 Envisioned Refinement of VirSat Systems

In practice, spacecraft engineers refine their systems in an ad hoc manner as they progress through the development phases. For each so-called "refinement", further implementation details are introduced and the resulting refinement often contains behaviour that is incomparable to a prior specification. This can also include the use of different modelling languages and tools. A notion of refinement, which defines a behavioural preorder on VSMs, is required to properly support the independent and incremental development of VSMs. However, this aspect has not been addressed by the VirSat formalism presented by Chrszon et al. [2]. In this section, we discuss an intuitive and practical approach to the refinement of VirSat systems, that can also be formalised (see Definition 13). We have discussed our approach with the lead VirSat developer and they see its potential in facilitating the spacecraft design process.

The engineers responsible for spacecraft operations have to manually assemble commands together for every operational procedure that the spacecraft has been specified to fulfil. While the commands supported by the spacecraft are retrieved from lengthy specification documents, they could instead by gathered from the spacecraft's VSMs. We observe that only a subset of commands may be needed to fulfil all operational procedures, and the ability of a spacecraft to transition through its operational modes is more important than the commands themselves.



(a) VSM of an AOCS specification. Thick arrows denote transitions that have been refined in Figure 3(b).



(b) VSM of refined AOCS.

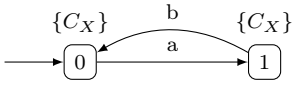
Figure 3: Example *Attitude and Orbit Control System (AOCS)* component, adapted from [2], and a possible refinement.

Intuitively, unnecessary commands can be refined away from the VSMs if the reachability of all its operating modes is preserved. This means that a VSM and its refinement would have exactly the same modes. Moreover, the refinement of two VSMs can be analysed as a graph-theoretic problem, and interactive analysis tools can be developed to communicate refinement problems back to engineers in an intuitive manner.

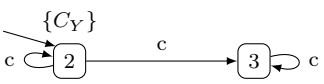
Figure 3(a) is an example specification of an *Attitude and Orbit Control System (AOCS)*, adapted from [2], with a possible refinement in Figure 3(b) where unnecessary transitions have been removed. The mode *FinePointing* in Figure 3(a) has two nondeterministic transitions: $FinePointing \xrightarrow{sby} Standby$ and $FinePointing \xrightarrow{sby\_alt} Standby$. These are nondeterministic in the sense that both have the same destination mode *Standby*, and are alternatives or variants of the same command. In Figure 3(a), the transition $SunAcquisition \xrightarrow{sby} Standby$ could be considered as an abstract transition or "shortcut" of the more concrete transition sequence $SunAcquisition \xrightarrow{att} GuidedAttitude \xrightarrow{sby} Standby$. Likewise, the abstract transition $FinePointing \xrightarrow{att} GuidedAttitude$ could represent the concrete transition sequence $FinePointing \xrightarrow{sby} Standby \xrightarrow{sun} SunAcquisition \xrightarrow{att} GuidedAttitude$. In Figure 3(a), the self-loop on mode *StationKeeping* could represent the ability to restart the mode's operation, but be deemed unnecessary at a later design phase. The rules for refining nondeterministic, shortcut, and self-loop transitions are formalised by Definition 13.

Redundancy is an important aspect to consider when designing a spacecraft. For example, a spacecraft normally uses its reaction wheels to orient itself. When these fail, an alternative method is needed, such as using thrusters. The desired redundancy method would become clearer as the spacecraft's equipment are chosen, leading to the addition of new modes, commands, and transitions to the design. However, in this article, we do not address this perspective of refinement, where a refined system supports more behaviours.

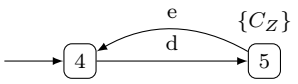
To define a preorder on a set of (refined) VSMs, the VSMs need to carry information about the constraints that their states are involved with. The key challenge is in encoding the constraints, defined in the global sets $\mathcal{F}$ and $\mathcal{R}$, locally at the VSM states and to reconstruct the original constraints when the VSMs are composed together.



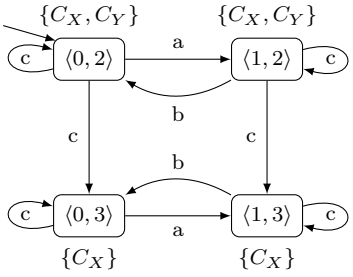
(a) CSM X, where $\Sigma_X = \{a, b\}$, $AP_X = \{C_X\}$, and $AP_X(0) = AP_X(1) = \{C_X\}$.



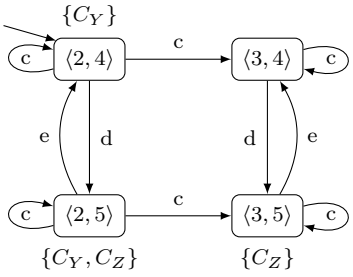
(b) CSM Y, where $\Sigma_Y = \{c\}$, $AP_Y = \{C_Y\}$, $AP_Y(2) = \{C_Y\}$, and $AP_Y(3) = \emptyset$.



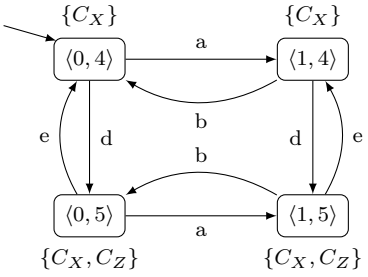
(c) CSM Z, where $\Sigma_Z = \{d, e\}$, $AP_Z = \{C_Z\}$, and $AP_Z(4) = \emptyset$, and $AP_Z(5) = \{C_Z\}$.



(d) CSM $X \otimes Y$, where $\Sigma_{X\otimes Y} = \{a, b, c\}$, $AP_{X\otimes Y} = \{C_X, C_Y\}$.



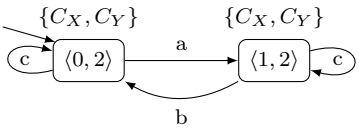
(e) CSM $Y \otimes Z$, where $\Sigma_{Y\otimes Z} = \{c, d, e\}$, $AP_{Y\otimes Z} = \{C_Y, C_Z\}$.



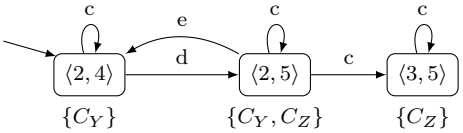
(f) CSM $Z \otimes X$, where $\Sigma_{Z\otimes X} = \{a, b, d, e\}$, $AP_{Z\otimes X} = \{C_X, C_Z\}$.

$\varphi_1 =_{\text{df}} (C_X \Rightarrow C_Y)$
$\varphi_2 =_{\text{df}} (C_Y \vee C_Z)$
$\varphi_3 =_{\text{df}} tt$

(g) Parallel conditions.



(h) CSM $X \parallel_{\varphi_1} Y$.



(i) CSM $Y \parallel_{\varphi_2} Z$.

Figure 4: Example CSMs X, Y, and Z and their parallel products and compositions.

4 Component State Machines (CSMs)

To enable the independent and incremental development of VSMs, *Component State Machine (CSM)* extends VSMs with state labels that represent the constraints that states are involved in. These labels are used to define the semantics of constraints as Boolean formulas, which we call *parallel conditions*. As we will see in Section 6, this approach enables users to define their own constraints via the templating of parallel conditions that encode the intended semantics. The semantics-preserving transformation of VSMs to CSMs is explained in Section 5. We now formally define CSM, which specifies a system component as a labelled transition system with actions on transitions and atomic propositions on states.

Definition 7 (Component State Machine)**.** A *Component State Machine (CSM)* is a tuple $(P, AP, \Sigma, \rightarrow, p_0)$, where P, Σ, $\rightarrow$, p_0 and the related notations are as for VSM (see Definition 1) except that the CSM is not required to be deterministic, AP is the set of *atomic propositions*, and $AP(p) \subseteq AP$ is the function that labels state p with a set of atomic propositions. (P, AP, Σ) is the *signature* of the CSM.

We use C, F, and R as representatives of the set AP. Two CSMs of a parallel composition cannot share an action a; all CSMs execute asynchronously in an interleaved manner. Unlike VSM, CSM transitions can be nondeterministic. Figures 4(a)–4(c) show three CSMs and their definition of actions and atomic propositions. Note that state 2 of CSM Y has nondeterministic transitions on action c.

Definition 8 (CSM Parallel Product)**.** CSMs $(P, AP_P, \Sigma_P, \rightarrow_P, p_0)$, $(Q, AP_Q, \Sigma_Q, \rightarrow_Q, q_0)$ are *composable* if $AP_P \cap AP_Q = \emptyset$ and $\Sigma_P \cap \Sigma_Q = \emptyset$. The *product* $P \otimes Q$ of such composable CSMs is defined as $(P \times Q, AP_P \cup AP_Q, \Sigma_P \cup \Sigma_Q, \rightarrow, \langle p_0, q_0\rangle)$, where the transition relation $\rightarrow$ is the least relation satisfying the following rules:

- $\langle p, q\rangle \xrightarrow{a} \langle p', q\rangle$ if $p \xrightarrow{a}_P p'$;
- $\langle p, q\rangle \xrightarrow{a} \langle p, q'\rangle$ if $q \xrightarrow{a}_Q q'$.

We often drop the index from the transition relation whenever it is clear form the context. Figures 4(d)–4(f) show all pairwise parallel products of CSMs X, Y, and Z.

Execution constraints between CSMs P and Q are specified as a propositional formula φ over the atomic propositions in AP_P and AP_Q. Such constraints enable engineers to specify the conditional execution of concurrent state machines and have the effect of restricting the possible transition interleavings. The formula φ is not defined in terms of the CSM states themselves so as to maintain a black-box view of CSMs, requiring only the knowledge of their signatures. For φ to correctly encode constraints that only involve CSMs P and Q, with no effect on any other CSM, it is necessary for φ to hold on any set of atomic propositions that does not contain elements from AP_P and AP_Q, i.e., let $\overline{AP_P \cup AP_Q}$ be the universe of atomic propositions that excludes elements of AP_P and AP_Q, then $\forall AP' \subseteq \overline{AP_P \cup AP_Q}.\ AP' \vDash \varphi$. Note that the parallel condition $\varphi = tt$ specifies a vacuous constraint.

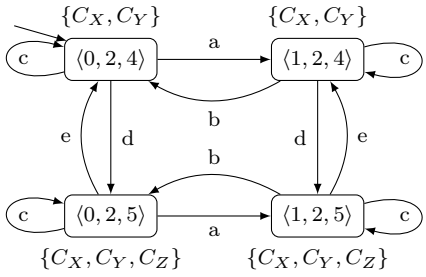
Definition 9 (CSM Parallel Condition)**.** Let $\text{Prop}(AP)$ be the universe of propositional formulas over AP. A *parallel condition* of CSMs P and Q is $\varphi \subseteq_{\text{df}} \text{Prop}(AP_P \cup AP_Q) \cup \{tt\}$ such that $AP' \subseteq \overline{AP_P \cup AP_Q}.\ AP' \vDash \varphi$.

If CSM P is in state p, CSM Q is in state q, $AP(p, q) =_{\text{df}} AP(p) \cup AP(q)$, and $AP(p, q) \nvDash \varphi$, then the state $\langle p, q\rangle$ is *invalid*. Because we assume that a spacecraft's ground station manually triggers the spacecraft to enter specific operating modes, the ground state is responsible for avoiding specific command sequences that would drive the spacecraft into an invalid state. Similar to Interface Automata [4], we take the *optimistic view* that the ground station is a *helpful* environment that avoids driving the CSMs into invalid states. Unlike Interface Automata, CSMs do not have autonomous transitions, so it is sufficient to only prune the invalid states from the parallel product when computing the parallel composition.

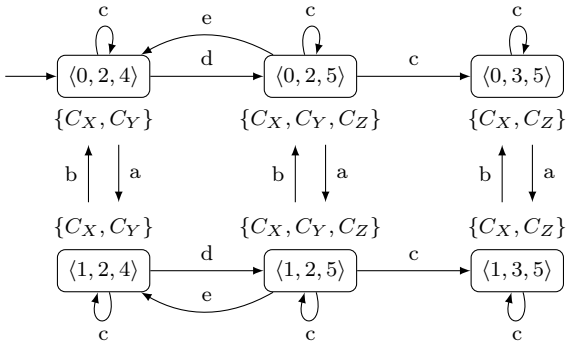
Definition 10 (CSM Parallel Composition)**.** Given a parallel product $P \otimes Q$ of CSMs P and Q with parallel condition φ, a state $\langle p, q\rangle \in P \otimes Q$ is an *invalid state* if $AP(p, q) \nvDash \varphi$. The *parallel composition* $P \parallel_\varphi Q$ is then computed by *pruning* the invalid states and their incoming and outgoing transitions: $P \parallel_\varphi Q =_{\text{df}} P \otimes Q \setminus \{\langle p, q\rangle \in P \otimes Q \mid AP(p, q) \nvDash \varphi\}$. If $\langle p, q\rangle \in P \parallel_\varphi Q$, then states p and q are *compatible* and $p \parallel_\varphi q$ is defined. P and Q are compatible if their initial states are compatible, otherwise $P \parallel_\varphi Q$ is undefined.

Returning to $X \otimes Y$ in Figure 4(d), its parallel composition $X \parallel_{\varphi_1} Y$ with the parallel condition φ_1 defined in Figure 4(g), is Figure 4(h). Likewise, $Y \parallel_{\varphi_2} Z$ with φ_2 defined in Figure 4(g), is Figure 4(i), while $Z \parallel_{\varphi_3} X$ with φ_3 defined in Figure 4(g), is simply $Z \otimes X$. In Section 5, we will show that CSM constraints are general enough to express the VirSat constraints Forbid (Definition 2) and Required (Definition 3).

Unlike VirSat's need to have all VSMs upfront to compute the parallel composition, CSMs can be composed incrementally in an associative and commutative manner. Figure 5(a) illustrates the parallel product of $X \parallel_{\varphi_1} Y$ and Z from Figures 4(c) and 4(h), respectively. The parallel composition $(X \parallel_{\varphi_1} Y) \parallel_{\varphi_2 \wedge \varphi_3} Z$, with parallel conditions φ_2 and φ_3 defined in Figure 4(g), is also Figure 5(a). Figure 5(b) is the parallel product of $Y \parallel_{\varphi_2} Z$ and X from Figures 4(a) and 4(i), respectively. Its parallel composition $(Y \parallel_{\varphi_2} Z) \parallel_{\varphi_1 \wedge \varphi_3} X$, with φ_1 and φ_3 defined in Figure 4(g), is also Figure 5(a).



(a) CSM $(X \parallel_{\varphi_1} Y) \otimes Z$. CSM $(X \parallel_{\varphi_1} Y) \parallel_{\varphi_2 \wedge \varphi_3} Z$, where $\varphi_2 = (C_Y \vee C_Z)$ and $\varphi_3 = tt$, is the same.



(b) CSM $(Y \parallel_{\varphi_2} Z) \otimes X$. $(Y \parallel_{\varphi_2} Z) \parallel_{\varphi_1 \wedge \varphi_3} X$, where $\varphi_1 = (C_X \Rightarrow C_Y)$ and $\varphi_3 = tt$, is the same as Figure 5(a).

Figure 5: Further examples of parallel products and compositions using the CSMs from Figure 4.

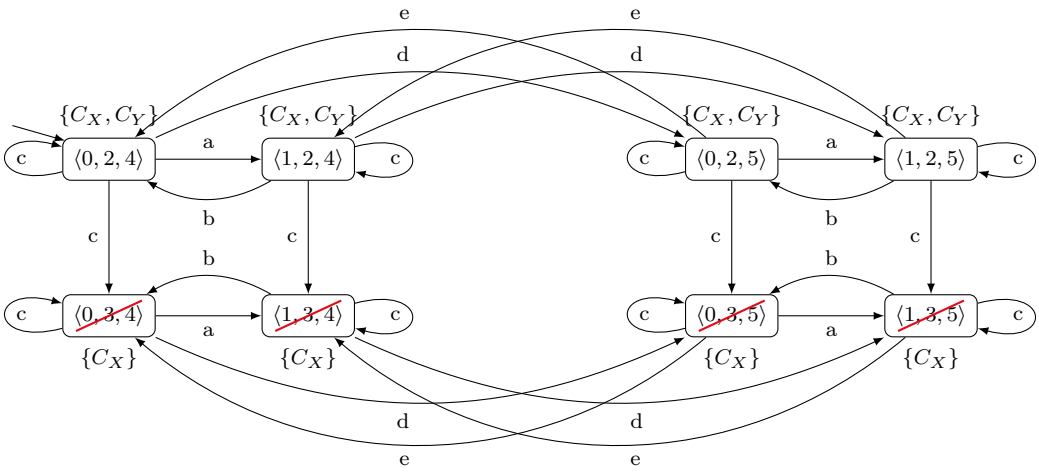


Figure 6: CSM $X \otimes Y \otimes Z$. Crossed-out states do not satisfy the parallel conditions φ_1, φ_2, or φ_3 defined in Figure 4(g).

Proposition 1 (CSM Associativity and Commutativity)**.** *Let P, Q, and R be pairwise composable CSMs. If $P \parallel_{\varphi_1} Q$, $P \parallel_{\varphi_2} R$, and $Q \parallel_{\varphi_3} R$ are defined, then $(P \parallel_{\varphi_1} Q) \parallel_{\varphi_2 \wedge \varphi_3} R$ and $P \parallel_{\varphi_1 \wedge \varphi_2} (Q \parallel_{\varphi_3} R)$ are either both undefined or are both isomorphic. Analogously, $\parallel_{\varphi}$ is commutative.*

Proof. Observe that the CSM parallel product operator $\otimes$ is associative and commutative, because the transition relation (Definition 8) is symmetrical and each product state inherits all the atomic propositions of its constituent states.

Let CSM $PQR =_{\text{df}} P \otimes Q \otimes R \setminus \{\langle p, q, r\rangle \in P \otimes Q \otimes R \mid AP(p, q, r) \not\models \varphi_1 \wedge \varphi_2 \wedge \varphi_3\}$. Let CSM $PQ =_{\text{df}} P \parallel_{\varphi_1} Q$. Since all states that do not satisfy φ_1 have been pruned in PQ, the same is true for $PQ \otimes R$ due to Definition 9. Then, pruning all states that do not satisfy $\varphi_2 \wedge \varphi_3$ from $PQ \otimes R$ results in $PQ \parallel_{\varphi_2 \wedge \varphi_3} R$, which is equivalent to PQR. Similarly, let CSM $QR =_{\text{df}} R \parallel_{\varphi_3} R$, then pruning all states that do not satisfy $\varphi_1 \wedge \varphi_2$ from $P \otimes QR$ results in $P \parallel_{\varphi_1 \wedge \varphi_2} QR = PQR = PQ \parallel_{\varphi_2 \wedge \varphi_3} R$. □

An interesting result of Proposition 1 is that the binary definition of parallel composition

can be generalised to an n-ary parallel composition, i.e., the parallel composition of a set of CSM can be computed equivalently by first taking their parallel product and then pruning all the invalid states. We will rely on this result later to prove that CSM subsumes VSM (see Theorem 4). Figure 6 is the parallel product of the CSMs from Figures 4(a)–4(c) with the invalid states that do not satisfy φ_1, φ_2, or φ_3, defined in Figure 4(g), crossed out. When the invalid states are pruned, we obtain the parallel composition shown in Figure 5(a).

Corollary 1 (CSM N-ary Parallel Composition)**.** *Let* $\mathcal{P} =_{df} \{P_1, \ldots, P_n\}$ *be a set of CSMs, and* $\Phi =_{df} \left\{\varphi_{ij} \subseteq \mathrm{Prop}(AP_{P_i} \cup AP_{P_j}) \cup \{tt\} \mid \{P_i, P_j\} \subseteq \mathcal{P}\right\}$ *be the set of parallel conditions between every pair of CSMs. Then, the* n-ary parallel composition *of CSMs in* $\mathcal{P}$ *under the parallel conditions in* Φ *is*

$$(P_1 \parallel \cdots \parallel P_n)_\Phi =_{df} P_1 \otimes \cdots \otimes P_n \setminus \left\{G \in P_1 \otimes \cdots \otimes P_n \;\middle|\; AP(G) \not\models \textstyle\bigwedge_{\varphi \in \Phi} \varphi\right\}.$$

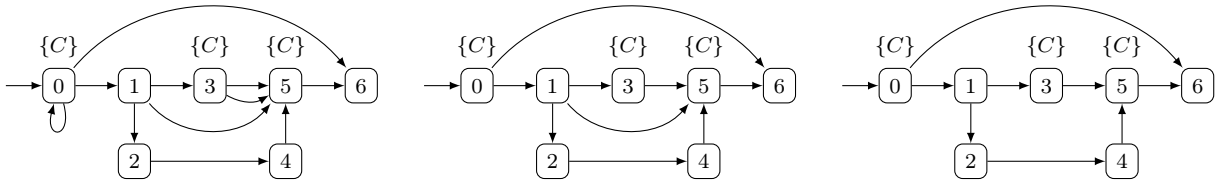
Proof. It follows from Proposition 1 that the parallel conditions in Φ can be evaluated together on the parallel product $P_1 \otimes \cdots \otimes P_n$ to identify all the invalid states. The parallel composition is then obtained by pruning the invalid states from the parallel product. □

A CSM state is *over-constrained* when one or more constraints prevent one of its transitions from being taken under all possible execution scenarios, i.e., taking the transition always leads to an invalid state. Such a transition is called *inconsistent*. In general, the consistency of a transition can only be known after all CSMs have been composed together. Note that the concept of an inconsistent constraint is more fine-grained than that of a VirSat local deadlock (Definition 6), where *all outgoing transitions* $p \xrightarrow{a} p'$ of a CSM state p have to be inconsistent.

Definition 11 (CSM Inconsistent Transition)**.** Let P and Q be two CSMs with parallel condition φ, $R_\otimes =_{\mathrm{df}} P \otimes Q$, and $R_\varphi =_{\mathrm{df}} P \parallel_\varphi Q$. We write $\langle p, \_\rangle$ for a product state where we are only interested in substate p. Then $p \xrightarrow{a}_P p'$ is an *inconsistent transition* if $\langle p, \_\rangle \in R_\varphi$ and $\exists \langle p, \_\rangle \in R_\otimes$ where $\langle p, \_\rangle \xrightarrow{a}_{R_\otimes} \langle p', \_\rangle$, but $\nexists \langle p, \_\rangle \xrightarrow{a}_{R_\varphi} \langle p', \_\rangle$. A transition that is not inconsistent is called a *consistent transition*.

An example of an inconsistent transition can be seen in Figure 4(h) for the parallel composition $X \parallel_{\varphi_1} Y$. State 2 of CSM Y exists in the parallel composition but the transition $2 \xrightarrow{c}_Y 3$ can never be taken, even though it exists in the parallel product $X \otimes Y$ (Figure 4(d)).

When improving the design a CSM, it is helpful to know what modifications are possible such that invalid states and inconsistent transitions can be eliminated without introducing new ones. This leads to the concept of refinement, which aims to establish whether a CSM correctly implements the specification of a more abstract CSM, such that compatibility is preserved when an abstract CSM is substituted by a refined one. Our proposed notion of refinement has been motivated by our discussion in Section 3, and has similarities to the refinement of Interface Automata [4]: while Interface Automata has so-called error states and communication mismatches and components can be refined independently to reduce nondeterminism and communication mismatches, CSM has corresponding invalid states and inconsistent transitions and components can also be refined independently to reduce nondeterminism and inconsistent transitions. To this end, CSM supports the compositional refinement of systems through the notion of bisimulation: intuitively, implementation P refines specification Q if P removes nondeterministic transitions between state pairs, or removes "shortcuts" that can be covered by another finite (and possibly empty) sequence of transitions. Thus, such transitions that are inconsistent can be refined away. It is important



(a) CSM P with $AP_P = \{C\}$. (b) CSM P' with $AP_{P'} = \{C\}$. (c) CSM P'' with $AP_{P''} = \{C\}$.

Figure 7: CSM refinement example. Transition actions have been omitted.

that implementation P preserves the states (operating modes) of specification Q. More precisely, P refines Q if P and Q have the same states and associated atomic propositions, all transitions of Q can be simulated by transition sequences of P, and all transitions of P can be simulated by Q.

Definition 12 (CSM Weak Transition)**.** A *weak transition* $p \Rightarrow p'$ is a finite and possibly empty sequence of transitions with arbitrary actions and intermediate states free of atomic propositions, i.e., $p \Rightarrow p'$ implies that for some $n \geq 0$ there exists $p_0, p_1, \ldots, p_n \in P$ where $p_0 = p$, $p_n = p'$, $p_{i-1} \xrightarrow{a_{i-1}} p_i$ for $1 \leq i \leq n$ and $n \geq 1$, and $AP(p_i) = \emptyset$ for $1 \leq i \leq n-1$ and $n \geq 2$.

Note that actions are not important to the definition of a weak transitions, because we are only interested in a path from p to p'.

Definition 13 (CSM Bisimulation)**.** For CSMs P and Q with the same signature, a relation $\mathcal{R} \subseteq P \times Q$ is a *bisimulation* if the following conditions hold for all $p\mathcal{R}q$:

1. $q \xrightarrow{a} q'$ implies that there exists $p \Rightarrow p'$ where $p = q$, $p' = q'$, $AP(p) = AP(q)$, $AP(p') = AP(q')$, and $p'\mathcal{R}q'$.
2. $p \xrightarrow{a} p'$ implies that there exists $q \xrightarrow{a} q'$ where $p = q$, $p' = q'$, $AP(p) = AP(q)$, $AP(p') = AP(q')$, and $p'\mathcal{R}q'$.

We write $P \sqsubseteq_{CSM} Q$ and say that P CSM-refines Q if there is a bisimulation $\mathcal{R}$ such that $p_0\mathcal{R}q_0$. Also, P is CSM-equivalent to Q if they CSM-refine each other.

Cond. 1 ensures that an abstract but consistent transition is always refined by a sequence of consistent transitions, thus, preserving the reachability of abstract states in the refinement. Cond. 2 prevents a CSM-refinement from adding new transitions, since they cannot be guaranteed to be consistent. In Figure 7, CSM P is a specification that is refined by CSM P' by removing one of the nondeterministic transitions $3 \rightarrow_P 5$ and the self-loop $0 \rightarrow_P 0$. CSM P' is refined by CSM P'' by removing the shortcut transition $1 \rightarrow_{P'} 5$, which has the weak transition $1 \rightarrow_{P'} 2 \rightarrow_{P'} 4 \rightarrow_{P'} 5$. The transition $0 \rightarrow_{P'} 6$ cannot be removed because there is no weak transition $0 \Rightarrow_{P'} 6$. Note that $P'' \sqsubseteq_{CSM} P' \sqsubseteq_{CSM} P$, but $P \not\sqsubseteq_{CSM} P'$, $P' \not\sqsubseteq_{CSM} P''$, and $P \not\sqsubseteq_{CSM} P''$.

Theorem 1 (CSM Preorder)**.** $\sqsubseteq_{CSM}$ *is a preorder.*

Proof. Reflexivity follows immediately from the fact that $P \sqsubseteq_{CSM} P$ is a bisimulation.

For transitivity, let $P \sqsubseteq_{CSM} Q$ and $Q \sqsubseteq_{CSM} R$ due to bisimulation relations $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. We show that $\mathcal{R} =_{\text{df}} \{(p, r) \mid \exists q \in Q.\ p\mathcal{R}_1 q \text{ and } q\mathcal{R}_2 r\}$ is also a bisimulation, implying $P \sqsubseteq_{CSM} R$. Since $p_0\mathcal{R}q_0$ and $q_0\mathcal{R}r_0$, we have $p_0\mathcal{R}r_0$.

Let $p\mathcal{R}r$ due to q with $p\mathcal{R}_1 q$ and $q\mathcal{R}_2 r$. According to Definition 13, for each $q \xrightarrow{a} q'$, (from Cond. 1 when q is the specification of p) there exists $p \Rightarrow p'$ with $p = q$, $p' = q'$, $AP(p) = AP(q)$, $AP(p') = AP(q')$, $p'\mathcal{R}q'$, and simultaneously (from Cond. 2 when q is the implementation of r) there exists $r \xrightarrow{a} r'$ with $q = r$, $q' = r'$, $AP(q) = AP(r)$, $AP(q') = AP(r')$, and $q'\mathcal{R}r'$. We conclude that $p'\mathcal{R}r'$ due to q'. Thus, $p\mathcal{R}r$ is a bisimulation. □

Based on the above theorem, we can show that CSMs support independent and incremental development:

Theorem 2 (CSM Compositionality)**.** *Let P, Q, and R be CSMs, $P \sqsubseteq_{CSM} Q$, Q and R be composable, and φ be a parallel condition. We have: (1) P and R are composable; (2) $P \parallel_\varphi R$ is defined if $Q \parallel_\varphi R$ is, and then $P \parallel_\varphi R \sqsubseteq_{CSM} Q \parallel_\varphi R$.*

Proof. Case (1) follows from the fact that $P \sqsubseteq_{CSM} Q$ requires P and Q to have the same signature. For Case (2), we show that $P \otimes R \sqsubseteq_{CSM} Q \otimes R$ and that the removal of invalid states with respect to φ results in $P \parallel_\varphi R \sqsubseteq_{CSM} Q \parallel_\varphi R$.

- Since $P \sqsubseteq_{CSM} Q$, all transitions taken by states $p \in P$ in $P \otimes R$ and by $q \in Q$ in $Q \otimes R$ are bisimilar. Furthermore, all transitions taken by state $r \in R$ in $P \otimes R$ and $Q \otimes R$ are trivially bisimilar. Since $(q_0, r_0) \in Q \otimes R$ and $p_0 = q_0$ due to $P \sqsubseteq_{CSM} Q$, we have $(p_0, r_0) \in P \otimes R$ and conclude that $P \otimes R \sqsubseteq_{CSM} Q \otimes R$.
- For all $(p, r) \in P \otimes R$ and $(q, r) \in Q \otimes R$ with $p = q$, we have $AP(p, r) = AP(q, r)$. Thus, during parallel composition, the same invalid states are pruned from $P \otimes R$ and $Q \otimes R$ under parallel condition φ, and conclude that $P \parallel_\varphi R \sqsubseteq_{CSM} Q \parallel_\varphi R$. □

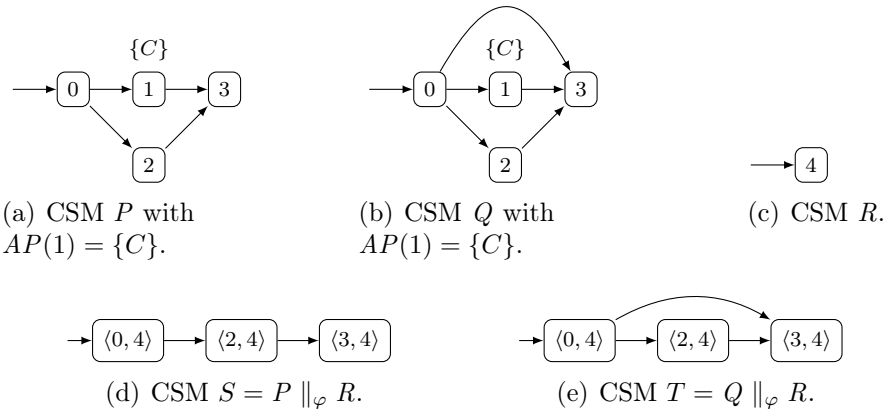
Note that, if $P \parallel_\varphi R \not\sqsubseteq_{CSM} Q \parallel_\varphi R$, then a weak transition $p \Rightarrow_P p'$ with intermediate state $p_i \in P$. $AP(p_i) \neq \emptyset$ was pruned away. However, this violates Definition 12 of a weak transition and $P \not\sqsubseteq_{CSM} Q$ in the first place. We now argue that Definition 12 of weak transition "$\Rightarrow$" is the least restrictive one that supports CSM-refinement. We define a naive weak transition "$\Rightarrow^{\#}$" (Definition 14 below) that is less restrictive than "$\Rightarrow$", and prove (Theorem 3 below) that CSM-refinement based on "$\Rightarrow^{\#}$" is only possible if "$\Rightarrow^{\#}$" is in fact as strong as "$\Rightarrow$".

Definition 14 (CSM Naive Weak Transition)**.** A *naive weak transition* $p \Rightarrow^{\#} p'$ is a weak transition where intermediate states can be labelled with atomic propositions, i.e., $p \Rightarrow^{\#} p'$ implies that for some $n \geq 0$ there exists $p_0, p_1, \ldots, p_n \in P$ where $p_0 = p$, $p_n = p'$, $p_{i-1} \xrightarrow{a_{i-1}} p_i$ for $1 \leq i \leq n$ and $n \geq 1$.

Theorem 3 (CSM Full Abstraction Result)**.** *Definition 12 of weak transition is the least restrictive one for $\sqsubseteq_{CSM}$.*

Proof. Let P, Q, and R be CSMs, $P \sqsubseteq^{\#}_{CSM} Q$ be the naive P CSM-refinement of Q where the naive weak transition "$\Rightarrow^{\#}$" is used in place of the usual weak transition "$\Rightarrow$" in Definition 13, and let $P \sqsubseteq^{\#C}_{CSM} Q =_{\text{df}} \left\{(p, q) \mid \forall p \in P, q \in Q, r \in R, \varphi.\ p \parallel_\varphi r \sqsubseteq^{\#}_{CSM} q \parallel_\varphi r\right\}$ be the greatest pre-congruence for parallel composition under parallel condition φ. We now prove that $P \sqsubseteq_{CSM} Q = P \sqsubseteq^{\#C}_{CSM} Q$:

- Case $P \sqsubseteq_{CSM} Q \subseteq P \sqsubseteq^{\#C}_{CSM} Q$: Follows from the compositionality result of Theorem 2 because "$\Rightarrow$" is stronger than "$\Rightarrow^{\#}$".



(a) CSM P with $AP(1) = \{C\}$.

(b) CSM Q with $AP(1) = \{C\}$.

(c) CSM R.

(d) CSM $S = P \parallel_\varphi R$.

(e) CSM $T = Q \parallel_\varphi R$.

Figure 8: Example of naive refinement that succeeds. Transition actions have been omitted.

- Case $P \sqsubseteq_{CSM} Q \supseteq P \sqsubseteq^{\#C}_{CSM} Q$: Requires us to show that there is at least one context, R and φ, where $P \parallel_\varphi R \sqsubseteq^{\#C}_{CSM} Q \parallel_\varphi R$ such that all transitions "$\Rightarrow^{\#}$" in $P \parallel_\varphi R$ have intermediate states p_i that are necessarily $AP(p_i) = \emptyset$, i.e., all "$\Rightarrow^{\#}$" transitions are indeed "$\Rightarrow$".

  Let P, Q, and R be the CSMs depicted by Figures 8(a)–8(c), respectively. $P \sqsubseteq^{\#C}_{CSM} Q$ because transition $0 \rightarrow_Q 3$ is refined by the naive weak transition $0 \Rightarrow^{\#}_P 3$, i.e., either $0 \rightarrow_P 1 \rightarrow_P 3$ or $0 \rightarrow_P 2 \rightarrow_P 3$. Now compose R in parallel with P and Q under parallel condition $\varphi = \overline{C}$, resulting in $S = P \parallel_\varphi R$ and $T = Q \parallel_\varphi R$, depicted by Figures 8(d) and 8(e), respectively. Note that the product state $\langle 1, 4\rangle$ has been pruned away from S and T because $AP(1) = \{C\}$ and $\{C\} \not\models \varphi$. $S \sqsubseteq^{\#}_{CSM} T$ because transition $\langle 0, 4\rangle \rightarrow_T \langle 3, 4\rangle$ is refined by the naive weak transition $\langle 0, 4\rangle \Rightarrow^{\#}_S \langle 3, 4\rangle$, i.e., $\langle 0, 4\rangle \rightarrow_S \langle 2, 4\rangle \rightarrow_S \langle 3, 4\rangle$ whose intermediate state $\langle 2, 4\rangle$ has $AP(2) = AP(4) = \emptyset$.

  However, if state 2 in P and Q was also labelled with C, i.e., $AP(2) = \{C\}$, then $S = P \parallel_\varphi R$ would simply be the initial state $\langle 0, 4\rangle$ while $T = Q \parallel_\varphi R$ would have the transition $0 \rightarrow_Q 3$, and so $P \parallel_\varphi R \not\sqsubseteq^{\#}_{CSM} Q \parallel_\varphi R$. Thus, a valid refinement of $0 \rightarrow_Q 3$ cannot be a transition sequence that contains a labelled intermediate state. We conclude that all valid transitions sequences of "$\Rightarrow^{\#}$" are exactly those of "$\Rightarrow$". □

5 Transformation of VSMs to CSMs

VSM serves as the heterogeneous specification language that engineers use to design their spacecraft, and its compositional semantics is defined operationally by our CSM formalism. This enables us to equip VirSat with a notion of refinement, support independent and incremental development, and extend VSM with user-defined constraints. Existing VirSat systems, which had to be analysed monolithically as closed systems, can now remain open to further refinement and composition, rather than each system revision being treated as an independent design. This section describes the transformation of VSMs to equivalent CSMs and how the results of semantic analysis at the CSM-level can be transformed back to the VirSat-level for engineers to comprehend. The transformed CSMs act as functionally-equivalent proxies of their original VSMs.

A VSM $(P, \Sigma, \rightarrow, p_0)$ is transformed into the CSM $(P, AP, \Sigma, \rightarrow, p_0)$ by copying the states P, actions Σ, transition relation $\rightarrow$, and initial state p_0, and introducing an empty set of atomic

propositions AP. The set AP will hold the atomic propositions that represent the VirSat constraints that the CSM is involved with. Thus, the core of the transformation is in the encoding of VirSat declarative constraints as CSM operational constraints, expressed as atomic propositions and parallel conditions.

Definition 15 (Encoding of VSM Constraints)**.** Let P and Q be VSMs with the set $\mathcal{F}$ of Forbid constraints and the set $\mathcal{R}$ of Required constraints. Let $P' = \text{CSM}(P)$ and $Q' = \text{CSM}(Q)$ be the transformed CSMs. For each constraint, the involved CSM states are labelled with an atomic proposition that uniquely identifies the constraint and their CSM:

- $F^i = (p \in P, q \in Q) \in \mathcal{F}$: if $(p' \in P') = p$ and $(q' \in Q') = q$, then $AP(p') = AP(p') \cup \{F^i_P\}$ and $AP(q') = AP(q') \cup \{F^i_Q\}$;
- $R^i = (p \in P, \{q_1, \ldots, q_k\} \subseteq Q) \in \mathcal{R}$: if $(p' \in P') = p$ and $1 \leq j \leq k$. $(q'_j \in Q') = q_j$, then $AP(p') = AP(p') \cup \{R^i_P\}$ and $AP(q'_j) = AP(q'_j) \cup \{R^i_Q\}$;
- Symmetrically for $F^i = (q \in Q, p \in P) \in \mathcal{F}$ and $R^i = (q \in Q, \{p_1, \ldots, p_k\} \subseteq P) \in \mathcal{R}$.

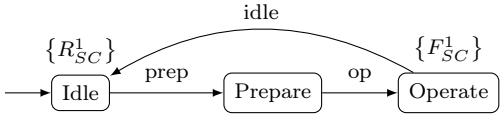
The VirSat constraint conditions (Definition 4) for $\mathcal{F}$ and $\mathcal{R}$ are encoded as a set of CSM parallel conditions $\Phi(\mathcal{F}, \mathcal{R}) =_{\text{df}} \Phi_{\mathcal{F}} \cup \Phi_{\mathcal{R}}$, where

- $\Phi_{\mathcal{F}} =_{\text{df}} \left\{ \overline{F^i_P \wedge F^i_Q} \;\middle|\; F^i = (p \in P, q \in Q) \in \mathcal{F} \right\} \cup$
  $\left\{ \overline{F^i_Q \wedge F^i_P} \;\middle|\; F^i = (q \in Q, p \in P) \in \mathcal{F} \right\}$;
- $\Phi_{\mathcal{R}} =_{\text{df}} \left\{ R^i_P \Rightarrow R^i_Q \;\middle|\; R^i = (p \in P, \{q_1, \ldots, q_k\} \subseteq Q) \in \mathcal{R} \right\} \cup$
  $\left\{ R^i_Q \Rightarrow R^i_P \;\middle|\; R^i = (q \in Q, \{p_1, \ldots, p_k\} \subseteq P) \in \mathcal{R} \right\}$.

The parallel composition of VSMs now proceeds on the transformed CSMs. Returning to the VirSat example of Figure 2, the transformed CSMs of the VSMs SC and PL are shown in Figures 9(a) and 9(b), respectively, as well as their encoded constraints in Figure 9(c). The parallel product of the transformed CSMs is Figure 9(d), while Figure 9(e) is the subsequent parallel composition under the parallel conditions in Figure 9(c). This CSM parallel composition is equivalent to the VSM parallel composition in Figure 2(e).

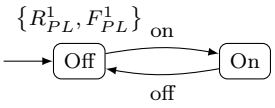
Let us now consider the parallel composition of the three VSMs P, Q, and R in Figure 10(a). We select any pair of VSMs, e.g., P and Q, transform them into equivalent CSMs, and compose them in parallel to obtain CSM S shown in Figure 10(b). Because there are no constraints between P and Q, no states are labelled with atomic propositions and the parallel condition is $\varphi_1 = tt$. CSM S can then be composed with the remaining VSM R by transforming R into a CSM and encoding the constraints between S and R according to Definition 15 with the following adjustment: since S embodies P and Q, any constraint between S and R is really between P and R, and between Q and R. For example, the Required constraint between state 1 of P and state 5 of R in Figure 10(a) means that all states of S containing substate 1 are labelled with the atomic proposition R^1_P, and state 5 of R is labelled with R^1_R. The Required and Forbid constraints are then encoded as $\varphi_2 = (R^1_P \Rightarrow R^1_R)$ and $\varphi_3 = \overline{F^1_Q \wedge F^1_R}$, respectively. In effect, as VSMs are composed together, their transformed CSMs are progressively enriched with atomic propositions. Note that, by indexing the atomic propositions of each CSM by the name of its original VSM, the transformed CSMs are composable (Definition 8) by construction. Figures 10(c) and 10(d) are the parallel product and composition, respectively, of S and R.

The global parallel composition approach of VirSat is still possible with the generalised parallel composition of CSMs (Corollary 1). Given the set $\mathcal{P} =_{\text{df}} \{P_1, \ldots, P_n\}$ of VSMs with



$SC' = \{\text{Idle, Prepare, Operate}\}$
$\Sigma_{SC'} = \{\text{prep, op, idle}\}$
$AP_{SC'} = \{F^1_{SC}, R^1_{SC}\}$

(a) $SC' = \text{CSM}(SC)$.

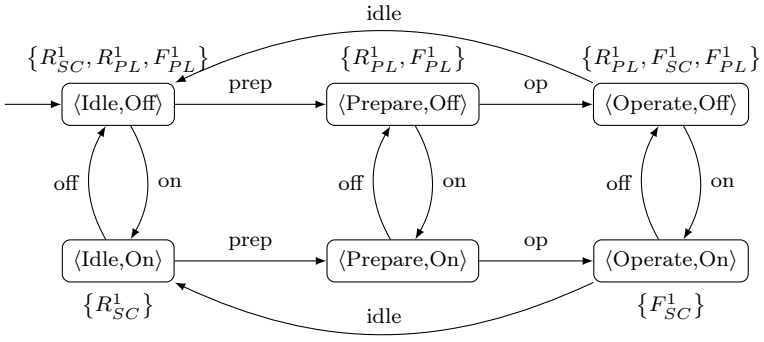


$PL' = \{\text{Off, On}\}$
$\Sigma_{PL'} = \{\text{on, off}\}$
$AP_{PL'} = \{F^1_{PL}, R^1_{PL}\}$

(b) $PL' = \text{CSM}(PL)$.

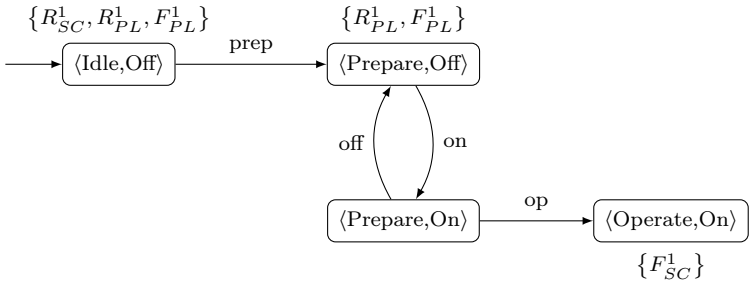
$$\Phi_{\mathcal{F}} = \left\{\overline{F^1_{SC} \wedge F^1_{PL}}\right\} \qquad \Phi_{\mathcal{R}} = \left\{R^1_{SC} \Rightarrow R^1_{PL}\right\}$$

(c) Parallel conditions of the Forbid and Required constraints in Figure 2(c).



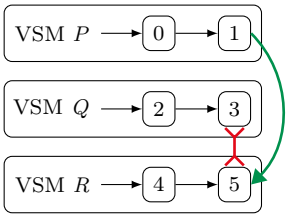
$SC' \otimes PL' = \{\langle\text{Idle,Off}\rangle, \langle\text{Prepare,Off}\rangle, \langle\text{Operate,Off}\rangle,$
$\langle\text{Idle,On}\rangle, \langle\text{Prepare,On}\rangle, \langle\text{Operate,On}\rangle\}$
$\Sigma_{SC' \otimes PL'} = \{\text{prep, op, idle, on, off}\}$
$AP_{SC' \otimes PL'} = \{F^1_{SC}, F^1_{PL}, R^1_{SC}, R^1_{PL}\}$

(d) $SC' \otimes PL'$.

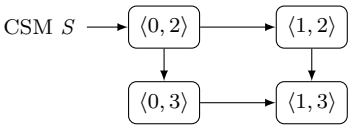


(e) $SC' \parallel_{\varphi} PL'$, where $\varphi = \overline{F^1_{SC} \wedge F^1_{PL}} \wedge (R^1_{SC} \Rightarrow R^1_{PL})$, and is equivalent to the VirSat parallel composition in Figure 2(e).

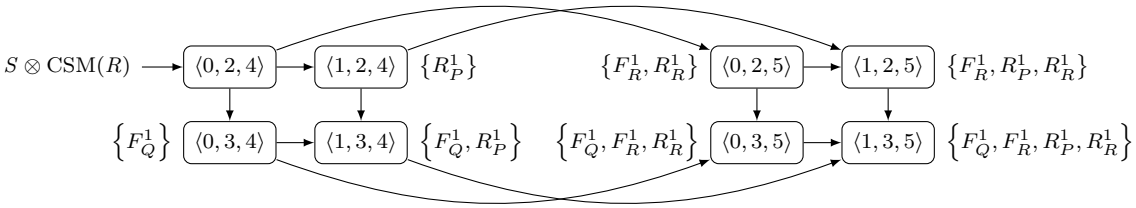
Figure 9: Transformed CSMs of the *Spacecraft (SC)* and *Payload (PL)* VSMs in Figure 2 and their parallel product and composition.



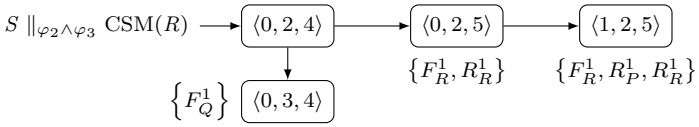
(a) VSMs P, Q, and R with constraints.



(b) CSM $S = \text{CSM}(P) \parallel_{\varphi_1} \text{CSM}(Q)$, with $\varphi_1 = tt$.



(c) $S \otimes \text{CSM}(R)$.



(d) $S \parallel_{\varphi_2 \wedge \varphi_3} \text{CSM}(R)$, with $\varphi_2 = (R_P^1 \Rightarrow R_R^1)$ and $\varphi_3 = \overline{F_Q^1 \wedge F_R^1}$.

Figure 10: Example CSM transformation and parallel composition of three VirSat components. Transition actions have been omitted.

the set $\mathcal{F}$ of Forbid constraints and the set $\mathcal{R}$ of Required constraints, we first transform the VSMs into CSMs $Q_1 = \text{CSM}(P_1), \ldots, Q_n = \text{CSM}(P_n)$. Second, the VirSat constraints between every pair of VSMs are encoded according to Definition 15, resulting in the set $\Phi(\mathcal{F}, \mathcal{R}) = \bigcup_{\{P_i, P_j\} \subseteq \mathcal{P}} \Phi_{P_i P_j}(\mathcal{F}, \mathcal{R})$ of all the encoded parallel conditions, and every transformed CSM being labelled with the constraints they participate in. Finally, the global parallel composition is computed as $(Q_1 \parallel \cdots \parallel Q_n)_{\Phi(\mathcal{F},\mathcal{R})}$.

We can now prove that a parallel composition of VSMs and its proposed CSM-transformation are functionally equivalent to each other. In other words, the semantics of the VirSat parallel composition is simulated by the generalised parallel composition of the transformed CSMs, and vice-versa.

Theorem 4 (VSMs and Their Transformed CSMs are Bisimilar)**.** *Let $\mathcal{P} =_{df} \{P_1, \ldots, P_n\}$ be VSMs with the set $\mathcal{F}$ of Forbid constraints and the set $\mathcal{R}$ of Required constraints. Let $Q_1 =_{df} \text{CSM}(P_1), \ldots, Q_n =_{df} \text{CSM}(P_n)$ be the transformed CSMs with the set of parallel conditions $\Phi(\mathcal{F}, \mathcal{R}) =_{df} \bigcup_{\{P_i, P_j\} \subseteq \mathcal{P}} \Phi_{P_i P_j}(\mathcal{F}, \mathcal{R})$. If $P =_{df} (P_1 \parallel \cdots \parallel P_n)$ is the VSM-defined parallel composition and $Q =_{df} (Q_1 \parallel \cdots \parallel Q_n)_{\Phi(\mathcal{F},\mathcal{R})}$ is the CSM-defined parallel composition, then P and Q are bisimilar.*

Proof. P and Q are bisimilar if, for $\mathcal{R} \subseteq P \times Q$, the following conditions hold for all $p \mathcal{R} q$:

1. $p \xrightarrow{a} p'$ implies that there exists $q \xrightarrow{a} q'$ where $p = q$, $p' = q'$, $p' \mathcal{R} q'$;
2. $q \xrightarrow{a} q'$ implies that there exists $p \xrightarrow{a} p'$ where $p = q$, $p' = q'$, $p' \mathcal{R} q'$.

We show that $\mathcal{R} =_{\text{df}} \left\{ (p, q) \mid \exists p \in P, \exists q \in Q.\ p \xrightarrow{a} p', q \xrightarrow{a} q', p = q, p' = q' \right\}$ is a bisimulation, implying that P and Q are bisimilar. Recall that the states and transition relation

of Q_i are copied from P_i, and the parallel composition P (resp. Q) is constructed by avoiding (resp. removing) invalid product states. Observe that P and Q are trivially bisimilar when $\mathcal{F} = \mathcal{R} = \emptyset$. In the following, we say that the product states $p \in P$ and $q \in Q$ are *equal*, written $p = q$, if and only if their constituent states are also equal.

Cond. 1 implies that VSM states p and p' are valid states, i.e., they satisfy all the constraint conditions of $\mathcal{F}$ and $\mathcal{R}$. CSM states $q = p$ and $q' = p'$ must also be valid states because the transformation ensures that $AP(q), AP(q') \vDash \Phi(\mathcal{F}, \mathcal{R})$. Thus, states q and q' are not pruned in Q, so $q \xrightarrow{a} q'$ exists and $p' \mathcal{R} q'$.

Cond. 2 implies that CSM states q and q' are valid states, i.e., $AP(q), AP(q') \vDash \Phi(\mathcal{F}, \mathcal{R})$. This implies that VSM states $p = q$ and $p' = q'$ satisfy all the constraint conditions of $\mathcal{F}$ and $\mathcal{R}$, and are therefore valid states. Thus, $p \xrightarrow{a} p'$ exists and $p' \mathcal{R} q'$. □

Corollary 2 (CSM Subsumes VSM Semantics)**.** *Let P and Q be the VSM- and CSM-parallel compositions from Theorem 4, respectively. Either both P and Q are undefined, or both are isomorphic.*

Proof. It directly follows from Theorem 4 that $p_0 \in P \Leftrightarrow q_0 \in Q$. When P and Q are defined, their bisimilarity makes them isomorphic. However, CSM's parallel conditions can model more than the constraints supported by VSM. Unlike VSMs, CSMs transitions can be nondeterministic and CSMs can synchronise over shared actions. Thus, CSM is a proper superset of VSM. □

Although a VirSat system can be analysed via CSMs, any analysis result should be conveyed back to engineers in terms of the original VSMs or in VirSat syntax. For example, a CSM parallel composition should be presented as a VSM, an invalid CSM state should be mapped back to its constituent VSM states, a CSM parallel condition should be expressed as a VirSat constraint, and an inconsistent CSM transition should be mapped back to its respective VSM transition.

A VSM can be recovered from a CSM parallel composition by hiding the atomic propositions of each CSM state. The mapping of CSM invalid states and inconsistent transitions back to their original VSMs is simple, because the CSM states and transition relations were directly copied from their VSMs without modification. The CSM constraints can be decoded back to their original VirSat constraints through syntactic analysis:

Proposition 2 (Decoding of CSM Constraints)**.** *Let P and Q be VSMs with the set $\mathcal{F}$ of Forbid constraints and the set $\mathcal{R}$ of Required constraints. If $P' = \mathrm{CSM}(P)$ and $Q' = \mathrm{CSM}(Q)$ are the transformed CSMs, then each parallel condition in $\Phi(\mathcal{F}, \mathcal{R})$ can be decoded back to their original constraint in $\mathcal{F}$ or $\mathcal{R}$.*

Proof. A parallel condition is a conjunction of one of two syntaxes: $\overline{F_P^i \wedge F_Q^i}$ or $R_P^i \Rightarrow R_Q^i$.

- $\overline{F_P^i \wedge F_Q^i}$ is the Forbid constraint $F^i = (p \in P, q \in Q)$ between state $p = (p' \in P')$ with $F_P^i \in AP(p')$ and state $q = (q' \in Q')$ with $F_Q^i \in AP(q')$;
- $R_P^i \Rightarrow R_Q^i$ is the Required constraint $R^i = (p \in P, \{q_1, \ldots, q_k\} \subseteq Q)$ between state $p = (p' \in P')$ with $R_P^i \in AP(p')$ and states $1 \leq j \leq k$. $q_j = (q'_j \in Q')$ with $R_Q^i \in AP(q'_j)$;
- Symmetrically for $\overline{F_Q^i \wedge F_P^i}$ and $R_Q^i \Rightarrow R_P^i$. □

6 User-defined Virtual Satellite Constraints

A key benefit of defining the VirSat semantics in CSM is the support for user-defined constraints. Spacecraft designers can propose new constraints and define their semantics in terms of CSM encoding rules, e.g., similar to how the Forbid and Required constraints were encoded (Definition 15) as CSM atomic propositions and parallel conditions. However, only state-based constraints are supported, because parallel conditions are essential invariants on state labels. In this section, we define two new VirSat constraints, called *Allow* and *Multi-forbid*, and demonstrate how they are supported in our CSM framework.

Definition 16 (VirSat Allow Constraint)**.** An *Allow constraint* between two different VSMs, P and Q, is defined as $(p, q) \in P \times Q$. It specifies that state q can only be active when state p is active. The set of Allow constraints in a VirSat system is denoted $\mathcal{A}$.

Definition 17 (VirSat Multi-forbid Constraint)**.** A *Multi-forbid constraint* between k-different VSMs, $P_1, \ldots, P_k$, is defined as $\{p_1, \ldots, p_k\} \in P_1 \times \cdots \times P_k$. It specifies that none of the states in $\{p_1, \ldots, p_k\}$ can be active together. The set of Multi-forbid constraints in a VirSat system is denoted $\mathcal{M}$.

The Allow constraint is designed to be the dual of the Required constraint and is useful when the inversion of a Required constraint is easier to comprehend. The Multi-forbid constraint is syntactic sugar for expressing a logical collection of Forbid constraints. Figures 11(a) and 12(a) are example VirSat systems demonstrating the Allow and Multi-forbid constraint syntaxes, respectively. Normally, we would need to extend the core semantics of VirSat (see Definition 18) to realise the behaviour of our new constraints. This is a rather intrusive and non-user-friendly approach that can lead to the creation of many variations of VirSat semantics:

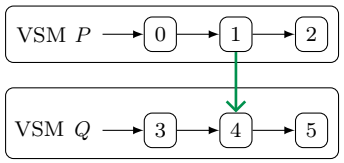
Definition 18 (VSM Valid State (Extension))**.** Let $P_1, \ldots, P_n$ be a set of VSMs and let $G \in P_1 \times \cdots \times P_n$ be a state configuration of the VSMs. Definition 4 for VSM valid state is extended with the following conditions:

1. $\forall\, (p, q) \in \mathcal{A}.\ q \in G \Rightarrow p \in G$;
2. $\forall\, \{p_1, \ldots, p_k\} \in \mathcal{M}.\ |\, \{p_1, \ldots, p_k\} \cap G| \leq 1$.

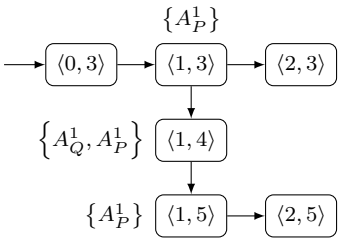
An alternative to modifying the core VirSat semantics is to define the Allow and Multi-forbid semantics via CSM encoding rules: what the constrained states are labelled, and how parallel conditions are generated. This approach isolates the definition of constraint behaviour from the core CSM semantics, which facilitates the creation of a library of constraints and the experimentation of different constraint behaviours.

Definition 19 (Encoding of Allow and Multi-forbid Constraints)**.** Let P and Q be VSMs with the set $\mathcal{A}$ of Allow constraints and the set $\mathcal{M}$ of Multi-forbid constraints. Let $P' = \text{CSM}(P)$ and $Q' = \text{CSM}(Q)$ be the transformed CSMs. For each constraint, the involved CSM states are labelled with an atomic proposition that uniquely identifies the constraint and their CSM:

- $A_i = (p \in P, q \in Q) \in \mathcal{A}$: if $(p' \in P') = p$ and $(q' \in Q') = q$, then $AP(p') = AP(p') \cup \{A_P^i\}$ and $AP(q') = AP(q') \cup \{A_Q^i\}$;
- $M_i \in \mathcal{M}$: if $\{p \in P, q \in Q\} \subseteq M_i$, $(p' \in P') = p$, and $(q' \in Q') = q$, then $AP(p') = AP(p') \cup \{M_P^i\}$ and $AP(q') = AP(q') \cup \{M_Q^i\}$;
- Symmetrically for $A_i = (q \in Q, p \in P) \in \mathcal{A}$.

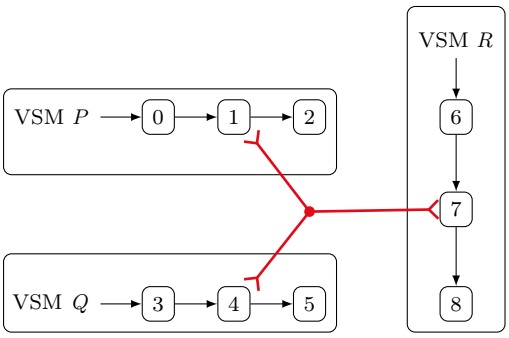


(a) VSMs P and Q with the Allow constraint $(1, 4)$.

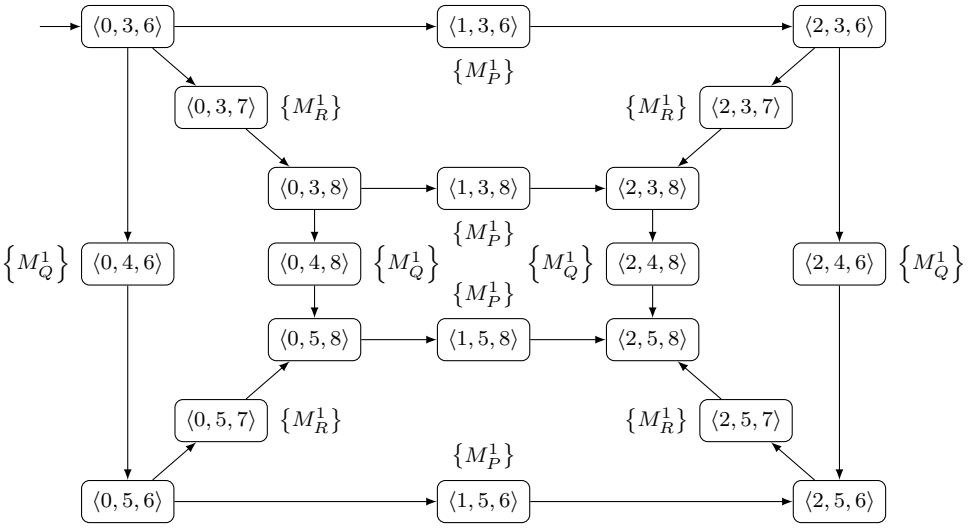


(b) $\mathrm{CSM}(P) \parallel_{\varphi_1} \mathrm{CSM}(Q)$, with $\varphi_1 = (A_Q^1 \Rightarrow A_P^1)$.

Figure 11: Example use of the Allow constraint. Transition actions have been omitted.



(a) VSMs P, Q, and R with the Multi-forbid constraint $\{1, 4, 7\}$.



(b) $(\mathrm{CSM}(P) \parallel_{\varphi_1} \mathrm{CSM}(Q)) \parallel_{\varphi_2 \wedge \varphi_3} \mathrm{CSM}(R)$, with $\varphi_1 = \overline{M_P^1 \wedge M_Q^1}$, $\varphi_2 = \overline{M_P^1 \wedge M_R^1}$, and $\varphi_3 = \overline{M_Q^1 \wedge M_R^1}$.

Figure 12: Example use of the Multi-forbid constraint. Transition actions have been omitted.

The VirSat constraint conditions (Definition 18) for $\mathcal{A}$ and $\mathcal{M}$ are encoded as a set of CSM parallel conditions $\Phi(\mathcal{A}, \mathcal{M}) =_{\text{df}} \Phi_{\mathcal{A}} \cup \Phi_{\mathcal{M}}$, where

- $\Phi_{\mathcal{A}} =_{\text{df}} \left\{ A_Q^i \Rightarrow A_P^i \;\middle|\; A_i = (p \in P, q \in Q) \in \mathcal{A} \right\} \cup$
  $\left\{ A_P^i \Rightarrow A_Q^i \;\middle|\; A_i = (q \in Q, p \in P) \in \mathcal{A} \right\}$;
- $\Phi_{\mathcal{M}} =_{\text{df}} \left\{ \overline{M_P^i \wedge M_Q^i} \;\middle|\; M_i \in \mathcal{M}, \{p \in P, q \in Q\} \subseteq M_i \right\}$.

For a given parallel composition, the complete set of parallel conditions now becomes $\Phi(\mathcal{F}, \mathcal{R}, \mathcal{A}, \mathcal{M}) =_{\text{df}} \Phi_{\mathcal{F}} \cup \Phi_{\mathcal{R}} \cup \Phi_{\mathcal{A}} \cup \Phi_{\mathcal{M}}$. Returning back to the example Allow constraint in Figure 11(a), the VSMs are transformed into CSMs, the constraints are encoded according to Definition 19, and the resulting CSM parallel composition is shown in Figure 11(b). We see that Q is only allowed to enter, stay, and exit state 4 while P is in state 1. The parallel composition of the Multi-forbid constraint example in Figure 12(a) is shown in Figure 12(b). We see that P, Q, and R can only be executed sequentially to satisfy the mutual exclusion of states 1, 4, and 7.

7 Modular Implementation of CSMs

When implementing a Component State Machine (CSM) design for execution on a platform, the design becomes closed-off from further modifications and compositions. Thus, an implementation necessarily sacrifices compositionality for executability, while needing to preserve the consistency of its transitions. Such a trade-off is similar in the modular compilation of reactive systems [21, 22], where causality needs to be preserved when composing separately compiled programs. This section describes a proof-of-concept to the modular implementation of CSMs that allows CSMs to be compiled separately, reused in other implementations, replaced by their refinements, and remain open to future compositions with other CSMs.

The idea is to compile each CSM into a so-called *ESM* and to synthesise *ESM schedulers* that each manage the execution of a pair of ESMs. If X and Y are two ESMs, a "Sch(X, Y)" scheduler would be created. To support an arbitrary number of ESMs, a hierarchical scheduling approach [23, 24] is used, where schedulers are nested to form a tree structure. If W, X, Y, and Z are four ESMs, a possible nesting of schedulers could be $\text{Sch}(\text{Sch}(X, Y), \text{Sch}(W, Z))$ or $\text{Sch}(W, \text{Sch}(\text{Sch}(X, Y), Z))$. To properly enforce the parallel conditions, each scheduler determines the possible transitions that its child ESMs can take, retrieves their corresponding parallel conditions, evaluates the parallel conditions on the transitions, and keeps only the transitions with valid next states. The scheduler passes these transitions up to its parent scheduler, which checks these transitions against the other ESMs it is responsible for. The *root* scheduler, at the top of the scheduler hierarchy, receives all the (valid) transitions and chooses one for execution. This is similar to the ground station choosing one command from a set of possibilities to send to its spacecraft.

We now explain our approach in detail using Algorithms 1–4 and an example ESM implementation (Tables 1–3) based on the CSMs X, Y, and Z in Figures 4(a)–4(c), respectively. Because the implementation of a design with a single CSM is trivial, we focus on designs with multiple CSMs. The CSM X (Figure 4(a)) is compiled into an ESM by generating a collection of tables representing its transition relation (Table 1(a)), labelling of atomic propositions to states (Table 2), and parallel conditions with other CSMs (Table 3). We write Trans(X, s) to return the set of outgoing transitions at source state s in ESM X. For example, using the transition relation of ESM Y in Table 1(b), $\text{Trans}(Y,\ 2) = \{(2, c, 2), (2, c, 3)\}$. We write AP($X$, s) to return the set of atomic propositions at state s in ESM X. For

Table 1: Transition relations (TRANS) of the CSMs X, Y, and Z in Figures 4(a)–4(c). Each table entry represents the target states of a given source state and action

(a) TRANS(X, s)

Source state s	**Actions** a	b
(initial) 0	1	
1		0

(b) TRANS(Y, s)

Source state s	**Action** c
(initial) 2	2, 3
3	3

(c) TRANS(Z, s)

Source state s	**Actions** d	e
(initial) 4	5	
5		4

Table 2: Atomic propositions (AP) of the CSMs in Figures 4(a)–4(c)

(a) AP(X, s)

State s	AP
0	C_X
1	C_X

(b) AP(Y, s)

State s	AP
2	C_Y
3	

(c) AP(Z, s)

State s	AP
4	
5	C_Z

example, using the state labelling of ESM Z in Table 2(c), AP(Z, 5) = $\{C_Z\}$. We write PARCONDS(X, Y) to return the parallel condition between ESMs X and Y. For example, using Table 3(a), PARCONDS(X, Y) = PARCONDS(Y, X) = $(C_X \Rightarrow C_Y)$. We assume that all CSMs are compiled into ESMs with unique names, that each ESM records its current state in a variable *currState*, and that all ESMs are initially in their initial state as indicated in their transition relation table.

Algorithm 1 is used solely by the *root* scheduler, at the top of the scheduler hierarchy, to execute one transition from a collection of compiled ESMs. All other schedulers simply pass scheduling information up and down the hierarchy. Since CSMs only support asynchronous transitions, only one transition can be taken at a time. For example, let *root* = Sch(X, W) and W = Sch(Y, Z). We say that *root* manages two child components, scheduler W and ESM Z, and refer to them using the notation *root.left* and *root.right*, respectively. Line 1 of Algorithm 1 uses Algorithm 2 to obtain all transitions with valid next states from its two children and, recursively, all their children. Line 2 selects the set of transitions belonging to one ESM, from which Line 3 selects one transition for Algorithm 4 to execute (Line 4).

Algorithm 2 retrieves all the outgoing transitions from the current states of the ESMs in its inputs L and R, and returns only the transitions that have valid next states. Lines 1–3 retrieve all outgoing transitions from the current states of L. If L is an ESM, we simply retrieve the output transitions at *L.currState* and store a mapping from L to the transitions in *NextEsmTransL*. However, if L is a scheduler, we call NEXT on its children and descend down the scheduler hierarchy. Lines 4–6 proceed analogously for R. We now need to evaluate

Table 3: Parallel conditions (PARCONDS) from Figure 4(g) of the schedulers

(a) PARCONDS of Sch(X, W)

CSMs	**Parallel Condition**
X, Y	$\varphi_1 =_{\text{df}} (C_X \Rightarrow C_Y)$
Z, X	$\varphi_3 =_{\text{df}} tt$

(b) PARCONDS of Sch(Y, Z)

CSMs	**Parallel Condition**
Y, Z	$\varphi_2 =_{\text{df}} (C_Y \vee C_Z)$

Algorithm 1 STEP(*root*): Given the top-most (*root*) ESM scheduler, execute one step of the ESM system

1: *NextEsmTrans* ← NEXT(*root.left*, *root.right*) ▷ *Mapping of ESMs to their transitions.*
2: $P \mapsto Trans \in NextEsmTrans$ ▷ *Choose an ESM.*
3: $(s, a, s') \in Trans$ ▷ *Choose a transition.*
4: EXEC(*root.left*, *root.right*, P, (s, a, s')) ▷ *Execute chosen transition.*

Algorithm 2 NEXT(L, R): Given ESMs or schedulers L and R, recursively return their set of transitions to valid next states

1: $NextEsmTransL = \emptyset$ ▷ *Valid next transitions of* L.
2: **if** TYPE(L) = ESM **then** $NextEsmTransL \leftarrow \{L \mapsto \text{TRANS}(L, L.currState)\}$
3: **else if** TYPE(L) = Scheduler **then** $NextEsmTransL \leftarrow \text{NEXT}(L.left, L.right)$
4: $NextEsmTransR = \emptyset$ ▷ *Valid next transitions of* R.
5: **if** TYPE(R) = ESM **then** $NextEsmTransR \leftarrow \{R \mapsto \text{TRANS}(R, R.currState)\}$
6: **else if** TYPE(R) = Scheduler **then** $NextEsmTransR \leftarrow \text{NEXT}(R.left, R.right)$
7: ▷ *Get the ESMs that are in* L *and* R. ◁
8: $EsmL \leftarrow \{esm \mid esm \mapsto Trans \in NextEsmTransL\}$
9: $EsmR \leftarrow \{esm \mid esm \mapsto Trans \in NextEsmTransR\}$
10: ▷ *Return only the transitions to valid next states.* ◁
11: **return** FILTER($EsmL$, $NextEsmTransR$) ∪ FILTER($EsmR$, $NextEsmTransL$)

the parallel conditions on every transition in *NextEsmTransL* and *NextEsmTransR*. Because parallel conditions are defined between pairs of ESMs, we first retrieve the ESMs involved in L and R (Lines 8 and 9, respectively), and use Algorithm 3 (explained later) to remove the transitions that do not have valid next states (Line 11).

We illustrate Algorithm 2 on our example (Tables 1–3) by calling NEXT(X, W), where $W = \text{Sch}(Y, Z)$. Assume that all ESMs are currently in their initial state. Line 2 applies because X is an ESM, and *NextEsmTransL* is assigned $\{X \mapsto \{(0, a, 1)\}\}$. Line 6 applies because W is a scheduler, and NEXT(Y, Z) is called. In this second instance of NEXT, Lines 2 and 5 are executed and *NextEsmTransL* and *NextEsmTransR* are assigned $\{Y \mapsto \{(2, c, 2), (2, c, 3)\}\}$ and $\{Z \mapsto \{(4, d, 5)\}\}$, respectively. Next, Lines 8 and 9 assigns $\{Y\}$ and $\{Z\}$ to *EsmL* and *EsmR*, respectively. Finally, Line 11 calls Algorithm 3 (explained below) and returns $\{Y \mapsto \{(2, c, 2)\}, Z \mapsto \{(4, d, 5)\}\}$. Returning to Line 6 in our initial call to NEXT, *NextEsmTransR* is assigned $\{Y \mapsto \{(2, c, 2)\}, Z \mapsto \{(4, d, 5)\}\}$. Next, Lines 8 and 9 assigns $\{X\}$ and $\{Y, Z\}$ to *EsmL* and *EsmR*, respectively, and Line 11 returns $\{X \mapsto \{(0, a, 1)\}, Y \mapsto \{(2, c, 2)\}, Z \mapsto \{(4, d, 5)\}\}$.

Algorithm 3 FILTER(*Esm*, *NextEsmTrans*): Given a set of ESMs *Esm* and a mapping of transitions *NextEsmTrans*, return only the transitions that have valid next states when every ESM in *Esm* remains in its current state

1: $FilteredNextEsmTrans = \emptyset$
2: $AP_{Esm} = \bigcup_{P \in Esm} \text{AP}(P, P.currState)$ ▷ *Atomic props. of all current states in* Esm.
3: **for** $Q \mapsto Trans \in NextEsmTrans$ **do**
4: $\quad \varphi_{Esm} = tt \wedge \bigwedge_{P \in Esm} \text{PARCONDS}(P, Q)$ ▷ *Parallel conds. between* Q *and* Esm.
5: $\quad FilteredNextTrans = \{(s, a, s') \in Trans \mid \text{AP}(s') \cup AP_{Esm} \vDash \varphi_{Esm}\}$
6: $\quad FilteredNextEsmTrans = FilteredNextEsmTrans \cup \{Q \mapsto FilteredNextTrans\}$
7: **return** $FilteredNextEsmTrans$

Algorithm 3 takes a set of ESMs and a mapping of ESMs to transitions, and returns only the transitions that have valid next states, i.e., those that satisfy all the parallel conditions. The assumption is that all the ESMs in the set *Esm* remain in their current state, and only the ESMs in *NextEsmTrans* are able to take one of their mapped transitions. Since the ESMs in *Esm* do not change state, we can compute their collective set of atomic propositions AP_{Esm} (Line 2). Next, for each ESM Q in *NextEsmTrans* (Lines 3–6), we compute the combined parallel condition φ_{Esm} that applies between Q and every ESM in *Esm* (Line 4), we evaluate φ_{Esm} on every target state of Q's mapped transitions and store only those that satisfy φ_{Esm} into *FilteredNextTrans* (Line 5), and we store a mapping from Q to *FilteredNextTrans* in *FilteredNextEsmTrans* (Line 6). After going through all the transitions in *NextEsmTrans*, the resulting *FilteredNextEsmTrans* is returned (Line 7).

We now detail some example computations of the following calls to FILTER (Algorithm 3) using our example above. We again assume that the ESMs are still in their initial states:

- FILTER($\{Y\}$, $\{Z \mapsto \{(4, d, 5)\}\}$): The atomic propositions AP_{Esm} is $\{C_Y\}$. Since *NextEsmTrans* only contains one mapping, from Z to $\{(4, d, 5)\}$, the loop in Algorithm 3 on Line 3 is only executed once. The parallel condition φ_{Esm} is only the one between Y and Z, i.e., $\varphi_{Esm} = (C_Y \vee C_Z)$, which Z's target state, 5, satisfies, i.e., $\{C_Y, C_Z\} \vDash (C_Y \vee C_Z)$. Thus, *FilteredNextEsmTrans* is assigned $\{Z \mapsto \{(4, d, 5)\}\}$, which is returned.
- FILTER($\{Z\}$, $\{Y \mapsto \{(2, c, 2), (2, c, 3)\}\}$): We have $AP_{Esm} = \emptyset$ and $\varphi_{Esm} = (C_Y \vee C_Z)$. Only the transition $(2, c, 2)$ of Y satisfies φ_{Esm}, i.e., $\{C_Y\} \vDash (C_Y \vee C_Z)$, so only $\{Y \mapsto \{(2, c, 2)\}\}$ is returned.
- FILTER($\{X\}$, $\{Y \mapsto \{(2, c, 2)\}, Z \mapsto \{(4, d, 5)\}\}$): We have $AP_{Esm} = \{C_X\}$. In the first iteration of the loop (Line 3) with $Y \mapsto \{(2, c, 2)\}$, $\varphi_{Esm} = (C_X \Rightarrow C_Y)$ and is satisfied by $\{C_X, C_Y\}$. In the second iteration with $Z \mapsto \{(4, d, 5)\}$, $\varphi_{Esm} = tt$ and is trivially satisfied. Thus, $\{Y \mapsto \{(2, c, 2)\}, Z \mapsto \{(4, d, 5)\}\}$ is returned.
- FILTER($\{Y, Z\}$, $\{X \mapsto \{(0, a, 1)\}\}$): We have $AP_{Esm} = \{C_Y\}$ and $\varphi_{Esm} = (C_X \Rightarrow C_Y) \wedge (C_Y \vee C_Z) \wedge tt$, which $\{C_X, C_Y\}$ satisfies. Thus, $\{X \mapsto \{(0, a, 1)\}\}$ is returned.

Algorithm 4 EXEC(L, R, P, (s, a, s')): Given ESMs or schedulers L and R, recursively find ESM P in order to execute its transition (s, a, s')

1: **if** TYPE(L) = ESM and $L = P$ **then** $L.currState \leftarrow s'$ ▷ *Transition is executed.*
2: **else if** TYPE(R) = ESM and $R = P$ **then** $R.currState \leftarrow s'$ ▷ *Transition is executed.*
3: **else if** TYPE(L) = Scheduler **then** EXEC($L.left$, $L.right$, P, (s, a, s'))
4: **else if** TYPE(R) = Scheduler **then** EXEC($R.left$, $R.right$, P, (s, a, s'))

Once the *root* scheduler has chosen a transition, Algorithm 4 can be used to find the correct ESM to execute the transition. The algorithm descends through the scheduler hierarchy (Lines 3 and 4), looking for the chosen ESM P. When it is found (Lines 1 or 2), the current state of P is updated to the target state s'. If the *root* scheduler has direct access to all ESMs in the implementation, then EXEC can be simplified to $P.currState \leftarrow s'$.

When a CSM is refined, only its transition relation table, e.g., Table 1, needs to be updated. When a parallel condition between two CSMs is added or removed, it is added or removed, respectively, from its scheduler's PARCONDS table. To compose a CSM with an existing ESM implementation, the CSM is compiled into a new ESM (TRANS and AP tables), and a new *root* scheduler and PARCONDS table are synthesised to manage the new ESM and

the existing ESM implementation. Similarly, the implementation of a set of CSMs can be reused by "plugging" it into another implementation as a child of the *root* scheduler and updating the *root* scheduler's ParConds table. If new constraints are introduced, the AP and ParConds tables of the constrained CSMs and schedulers will need to be updated with the associated atomic propositions and parallel conditions, respectively.

8 Conclusions

Virtual Satellite (VirSat) is an interesting heterogeneous modelling language that captures both operational and declarative behaviour in the same model. While the authors [2] of VirSat had originally defined its semantics for performing formal verification, it is inadequate for supporting the independent and incremental development of spacecraft software systems. We have addressed these limitations by developing a semantic framework that transforms VSMs into functionally equivalent CSMs for compositional reasoning, refinement, and component reuse, and then into functionally equivalent ESMs for modular compilation and execution. CSM extends VSM with (1) nondeterministic transitions to model incomplete specifications, (2) state-based constraints to model additional and more expressive constraints without needing to redefine the core VirSat semantics, (3) an associative and commutative parallel operator to compose components in a flexible manner, (4) a notion of inconsistent transitions for the fine-grained analysis of deadlocks, and (5) an intuitive and practical notion of refinement for the compositional development of VirSat components.

Our systematic transformation of VSMs into CSMs enables analysis results at the CSM-level to be decoded back in terms of the original VSMs for comprehension by spacecraft engineers. The modular implementation of CSMs as ESMs enables VSMs to be developed independently by different suppliers and for system integrators to perform integration testing on partial compositions.

Our semantic framework could also help simplify the verification challenge [2] of complex spacecraft systems in a modular and incremental fashion. For example, a system could be incrementally verified as its components are composed together, which would be less complex than verifying the entire system at once. Moreover, there is the possibility to limited re-verification when only some components have been modified.

Future work. We plan to implement our semantic framework either in the official VirSat project [1] or in our prototypic toolset [25] developed for Interface Automata. This would enable us to model and analyse larger VirSat systems and to evaluate our refinement approach on realistic engineering use-cases.

So far, VirSat has only supported asynchronous communication, which is adequate for manually controlled spacecraft. However, far-Earth spacecraft, which experience significant communication delays, are typically designed to have greater autonomy to coordinate their equipment modes. Thus, CSM should be extended to support *synchronous communication.* However, the notion of refinement would need to be carefully redefined such that the removal of synchronous transitions does not create inconsistent transitions, and does not change synchronous transitions into asynchronous ones.

Constraints in CSM could be extended to support temporal behaviour, i.e., constraints on the execution history of several states. For example, to model the granting of permission, the Required constraint could be relaxed so that the requiring state can remain active when all of its required states become inactive. Encoding temporal semantics could be achieved

with labelled *observer* CSMs, that synchronise with the execution of the constrained states, and appropriate parallel conditions to restrict the possible observations.

Although VirSat does not support hierarchical state machines because spacecraft designs do not require them, CSM could still be extended with hierarchy to target the development of more general cyber-physical or reactive systems.

There are several useful operators from Interface Automata theory [5] that CSM could be extended with: a *conjunction* operator to compute a CSM that only has functionality common to several CSMs, a *disjunction* operator to compute a CSM that has all the functionality of several CSMs, and a *quotient* operator to compute the CSM that remains when functionality is extracted from an existing CSM.

References

[1] Deutsches Zentrum für Luft- und Raumfahrt, "Virtual Satellite." Available at `https://github.com/virtualsatellite`. Last accessed May 2024.

[2] P. Chrszon, P. Maurer, G. Saleip, S. Müller, P. M. Fischer, A. Gerndt, and M. Felderer, "Applicability of Model Checking for Verifying Spacecraft Operational Designs," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 206–216, IEEE, 2023.

[3] W. T. Scherer and F. Rotman, "Combinatorial Optimization Techniques for Spacecraft Scheduling Automation," *Annals of Operations Research*, pp. 525–556, Dec. 1994.

[4] L. de Alfaro and T. A. Henzinger, "Interface-Based Design," in *Engineering Theories of Software Intensive Systems*, vol. 195 of *NATO Science Series*, pp. 83–104, Springer, 2005.

[5] C. Chilton, B. Jonsson, and M. Kwiatkowska, "An Algebraic Theory of Interface Automata," *Theoretical Computer Science*, vol. 549, pp. 146–174, 2014.

[6] Object Management Group, "Unified Modeling Language, v2.5.1," Dec. 2017. Available at `https://www.omg.org/spec/UML`. Last accessed May 2024.

[7] MathWorks, "Products and Services - MATLAB & Simulink." Available at `https://www.mathworks.com/products`. Last accessed May 2024.

[8] F. X. Dormoy, "SCADE 6 A Model Based Solution For Safety Critical Software Development," in *Embedded Real Time Software and Systems (ERTS)*, Jan. 2008.

[9] H. Elmqvist, F. Gaucher, S. E. Matsson, and F. Dupont, "State Machines in Modelica," in *International MODELICA Conference*, pp. 37–46, Modelica Association and Linköping University Electronic Press, Sept. 2012.

[10] R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. O'Brien, "SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications: HW/SW-Synthesis for a Conservative Extension of Synchronous Statecharts," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 372–383, ACM, June 2014.

[11] F. Jahanian and A. K. Mok, "Modechart: A Specification Language for Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, pp. 933–947, 1994.

[12] J. Fiedor and M. Gach, "Tools for Verification of Time Constrained Systems." Available at `https://www.fit.vutbr.cz/research/groups/verifit/tools/verif`. Last accessed May 2024.

[13] P. C. Clements, C. L. Heitmeyer, B. G. Labaw, and A. T. Rose, "MT: A Toolset for Specifying and Analyzing Real-Time Systems," in *Real-Time Systems Symposium (RTSS)*, pp. 12–22, 1993.

[14] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala, "Developing Mode-Rich Satellite Software by Refinement in Event-B," *Science of Computer Programming*, vol. 78, no. 7, pp. 884–905, 2013.

[15] A. Coen-Porisini, C. Ghezzi, and R. A. Kemmerer, "Specification of Realtime Systems Using ASTRAL," *IEEE Transactions on Software Engineering*, vol. 23, pp. 572–598, Sept. 1997.

[16] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity - The Ptolemy Approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[17] S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee, "A Modular Formal Semantics for Ptolemy," *Mathematical Structures in Computer Science*, vol. 23, no. 4, pp. 834–881, 2013.

[18] E. Bousse, T. Degueule, D. Vojtisek, T. Mayerhofer, J. Deantoni, and B. Combemale, "Execution Framework of the GEMOC Studio (Tool Demo)," in *ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, pp. 84–89, ACM, 2016.

[19] J. Deantoni, P. I. Diallo, J. Champeau, B. Combemale, and C. Teodorov, "Operational Semantics of the Model of Concurrency and Communication Language," Research Report RR-8584, INRIA, Sept. 2014.

[20] J. Eickhoff, *Onboard Computers, Onboard Software and Satellite Operations: An Introduction*. Springer Aerospace Technology, Springer, 2011.

[21] J. Zeng and S. A. Edwards, "Separate Compilation for Synchronous Modules," in *Embedded Software and Systems*, pp. 129–140, Springer, 2005.

[22] R. Lublinerman, C. Szegedy, and S. Tripakis, "Modular Code Generation from Synchronous Block Diagrams: Modularity vs. Code Size," in *Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pp. 78–89, ACM, 2009.

[23] Z. Deng and J. W.-S. Liu, "Scheduling Real-Time Applications in an Open Environment," in *Real-Time Systems Symposium (RTSS)*, pp. 308–319, IEEE, Dec. 1997.

[24] M. Holenderski, R. J. Bril, and J. J. Lukkien, "An Efficient Hierarchical Scheduling Framework for the Automotive Domain," in *Real-Time Systems, Architecture, Scheduling, and Application*, ch. 4, pp. 67–94, IntechOpen, 2012.

[25] N. T. Nguyen and G. Lüttgen, "The IAM Toolset." Available at `https://github.com/uniba-swt/ia-toolset`. Last accessed May 2024.

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989) Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)

Nr. 2 (1990) Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG

Nr. 3 (1990) Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen

Nr. 4 (1990) Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)

Nr. 5 (1990) Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Nr. 6 (1991) Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten

Nr. 7 (1991) Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender

Nr. 8 (1991) Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung

Nr. 9 (1992) Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell

Nr. 10 (1992) Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)

Nr. 11 (1992) Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells

Nr. 12 (1992) Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt

Nr. 13 (1992) Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell

Nr. 14 (1992) Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen

Nr. 15 (1992) Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch

Nr. 16 (1992) Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip

Nr. 17 (1993) Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993) Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994) Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach

Nr. 20 (1994) Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994

Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach - . 2nd edition, November 1994

Nr. 21 (1994) Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen

Nr. 22 (1994) Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems

Nr. 23 (1994) Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle

Nr. 24 (1994) Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen

Nr. 25 (1994) Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme

Nr. 26 (1995) Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes

Nr. 27 (1995) Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995

Nr. 28 (1995) Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach

Nr. 30 (1995) Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit

Nr. 31 (1995) Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse

Nr. 32 (1995) Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger

Nr. 33 (1995) Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?

Nr. 34 (1995) Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -

Nr. 35 (1995) Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme

Nr. 36 (1996) Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996) Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996) Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996

Nr. 39 (1996) Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze

Nr. 40 (1997) Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997

Nr. 41 (1997) Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997

Nr. 42 (1997) Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997

Nr. 43 (1997): Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997

Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998

Nr. 44 (1997) Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung

Nr. 45 (1998) Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998

Nr. 46 (1998) Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997

Nr. 47 (1998) Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998

Nr. 48 (1998) Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering

Nr. 49 (1998) Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999) Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)

Nr. 51 (1999) Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)

Nr. 52 (1999) Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999

Nr. 53 (1999) Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999

Nr. 54 (1999) Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.

Nr. 55 (2000) Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur

Nr. 56 (2000) Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000

Nr. 57 (2000) Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.

Nr. 58 (2000) Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.

Nr. 59 (2001) Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001

Nr. 60 (2001) Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002) Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.

Nr. 62 (2002) Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002

Nr. 63 (2005) Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions

Nr. 64 (2005) Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005

Nr. 65 (2006) Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet

Nr. 66 (2006) Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006

Nr. 67 (2006) Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006

Nr. 68 (2006) Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation

Nr. 69 (2007) Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007

Nr. 70 (2007) Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. February 2007 (out of print)

Nr. 71 (2007) Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007

Nr. 72 (2007) Michael Mendler, Gerald Lüttgen: Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic?

Nr. 73 (2007) Martin Schissler: out of print

Nr. 74 (2007) Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.

Nr. 75 (2008) Karsten Loesing (Hrsg.): Extended Abstracts of the Second *Privacy Enhancing Technologies Convention* (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.

Nr. 76 (2008) Gregor Scheithauer, Guido Wirtz: Applying Business Process Management Systems – A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.

Nr. 77 (2008) Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.

Nr. 78 (2008) Gregor Scheithauer, Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.

Nr. 79 (2008) Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performances. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.

Nr. 80 (2009) Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.

Nr. 81 (2009) Ute Schmid, Emanuel Kitzelmann, Rinus Plasmeijer (Eds.): Proceedings of the ACM SIGPLAN Workshop on *Approaches and Applications of Inductive Programming* (AAIP'09), affiliated with ICFP 2009, Edinburgh, Scotland, September 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 81, Bamberg University, September 2009. ISSN 0937-3349.

Nr. 82 (2009) Ute Schmid, Marco Ragni, Markus Knauff (Eds.): Proceedings of the KI 2009 Workshop *Complex Cognition*, Paderborn, Germany, September 15, 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 82, Bamberg University, October 2009. ISSN 0937-3349.

Nr. 83 (2009) Andreas Schönberger, Christian Wilms and Guido Wirtz: A Requirements Analysis of Business-to-Business Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 83, Bamberg University, December 2009. ISSN 0937-3349.

Nr. 84 (2010) Werner Zirkel, Guido Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 84, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 85 (2010) Jan Tobias Mühlberg und Gerald Lüttgen: Symbolic Object Code Analysis. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 85, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 86 (2010) Werner Zirkel, Guido Wirtz: Proaktives Problem Management durch Eventkorrelation – ein *Best Practice* Ansatz. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 86, Bamberg University, August 2010. ISSN 0937-3349.

Nr. 87 (2010) Johannes Schwalb, Andreas Schönberger: Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 87, Bamberg University, September 2010. ISSN 0937-3349.

Nr. 88 (2011) Jörg Lenhard: A Pattern-based Analysis of WS-BPEL and Windows Workflow. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 88, Bamberg University, March 2011. ISSN 0937-3349.

Nr. 89 (2011) Andreas Henrich, Christoph Schlieder, Ute Schmid [eds.]: Visibility in Information Spaces and in Geographic Environments – Post-Proceedings of the KI'11 Workshop. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 89, Bamberg University, December 2011. ISSN 0937-3349.

Nr. 90 (2012) Simon Harrer, Jörg Lenhard: Betsy - A BPEL Engine Test System. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 90, Bamberg University, July 2012. ISSN 0937-3349.

Nr. 91 (2013) Michael Mendler, Stephan Scheele: On the Computational Interpretation of CKn for Contextual Information Processing - Ancillary Material. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 91, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 92 (2013) Matthias Geiger: BPMN 2.0 Process Model Serialization Constraints. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 92, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 93 (2014) Cedric Röck, Simon Harrer: Literature Survey of Performance Benchmarking Approaches of BPEL Engines. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 93, Bamberg University, May 2014. ISSN 0937-3349.

Nr. 94 (2014) Joaquin Aguado, Michael Mendler, Reinhard von Hanxleden, Insa Fuhrmann: Grounding Synchronous Deterministic Concurrency in Sequential Programming. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 94, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 95 (2014) Michael Mendler, Bruno Bodin, Partha S Roop, Jia Jie Wang: WCRT for Synchronous Programs: Studying the Tick Alignment Problem. Bamberger

Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 95, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 96 (2015) Joaquin Aguado, Michael Mendler, Reinhard von Hanxleden, Insa Fuhrmann: Denotational Fixed-Point Semantics for Constructive Scheduling of Synchronous Concurrency. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 96, Bamberg University, April 2015. ISSN 0937-3349.

Nr. 97 (2015) Thomas Benker: Konzeption einer Komponentenarchitektur für prozessorientierte OLTP- & OLAP-Anwendungssysteme. Bamberger Beiträge zur Wirtschafts-informatik und Angewandten Informatik Nr. 97, Bamberg University, Oktober 2015. ISSN 0937-3349.

Nr. 98 (2016) Sascha Fendrich, Gerald Lüttgen: A Generalised Theory of Interface Automata, Component Compatibility and Error. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 98, Bamberg University, March 2016. ISSN 0937-3349.

Nr. 99 (2014) Christian Preißinger, Simon Harrer: Static Analysis Rules of the BPEL Specification: Tagging, Formalization and Tests. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 99, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 100 (2016) Cedrik Röck, Stefan Kolb: Nucleus - Unified Deployment and Management for Platform as a Service. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 100, Bamberg University, March 2016. ISSN 0937-3349.

Nr. 101 (2016) Michael Mendler, Partha S. Roop, Bruno Bodin: A Novel WCET Semantics of Synchronous Programs. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 101, Bamberg University, June 2016. ISSN 0937-3349.

Nr. 102 (2017) Joaquín Aguado, Michael Mendler, Marc Pouzet, Partha Roop, Reinhard von Hanxleden: Clock-Synchronised Shared Objects for Deterministic Concurrency. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 102, Bamberg University, July 2017. ISSN 0937-3349.

Nr. 103 (2018) Eugene Yip, Erjola Lalo, Gerald Lüttgen, Michael Deubzer, Andreas Sailer: Optimized Buffering of Time-Triggered Automotive Software. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 103, Bamberg University, September 2018. ISSN 0937-3349.

Nr. 104 (2018) Daniel Hallmann: Die COCOMO-Modelle im Licht der agilen Softwareentwicklung. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 104, Bamberg University, October 2018. ISSN 0937-3349.

Nr. 105 (2021) Johannes Manner: SeMoDe – Simulation and Benchmarking Pipeline for Function as a Service. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 105, Bamberg University, October 2021. ISSN 0937-3349.

Nr. 106 (2024) Eugene Yip, Gerald Lüttgen: Heterogeneous Specification of Spacecraft Software. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 106, Bamberg University, August 2024. ISSN 0937-3349.