A Formal Compositional Verification Approach for Safety–Critical Systems Correctness
Model–Checking based Methodological Approach to Automatically Verify Safety Critical Systems Software

Manuel I. Capel¹ and Luis E. Mendoza Morales²

¹Department of Software Engineering, University of Granada, Informatics & Telecommunications, 18071 Granada, Spain
²Processes and Systems Department, Simón Bolívar University, P.O. box 89000, Baruta, Caracas 1080-A, Venezuela

Keywords: Safety–Critical Systems, Compositional Verification, Model–Checking, Software Specification, Software Verification, Methodological Approach.

Abstract: The complexity of modern Safety–Critical Systems (SCS) together with the absence of appropriate software verification tools is one reason for the large number of errors in the design and implementation of these systems. Moreover, exhaustive testing is hard and highly complex because of the combinatorial explosion in the great number of states that an SCS can reach when it executes. A methodological approach named FCV A that uses Model–Checking (MC) techniques to automatically verify SCS software is presented here. This approach facilitates decomposition of complex SCS software into independently verified individual components, and establishes a compositional method to verify these systems using state–of–the–art MC tools. Our objective in this paper is to facilitate the description of an SCS as a collection of verified components, allowing complete complex SCS software verification. An application on a real–life project in the field of mobile phone communication is discussed to demonstrate the applicability of FCVA.

1 INTRODUCTION

The ever increasing complexity of current software systems has reached application areas where the trustworthiness of a computing system must guarantee the reliance on the service it delivers. Safety–Critical Systems (SCS), including energy production, automotive, medical systems, avionics and modern telecommunications are typical industrial systems where availability, performance, safety, integrity, maintainability, real–time response are crucial. All the above features are included in the computer systems concept of dependability¹. New verification methods and software tools for “design prediction” of dependability attributes of SCS are being intensely investigated now. Thus, this paper proposes a compositional scheme that can be applied to the verification of properties that express the certainty of a future event or system action (safety), or to verify that the system is not undergoing a deadlock situation or to affirm that every needed state of the system must eventually entered in an infinite computation (fairness).

Deductive techniques combined with advanced Model–Checking (MC) techniques are seen as the silver bullet to face the enormous complexity of SCS verification (Hooman, 1991; de Roever et al., 2001). However, it is not a simple task to export local verification results using a formal deductive language, such as Predicate Logic, including conjunctive propositional logic operators and, at the same time, to preserve the semantic correctness of the automatically performed proofs of verified system’s components.

Formal Compositional Verification Approach (FCVA) is proposed here to verify a SCS from individual components (Mendoza and Capel, 2009), based on a conceptual framework that transforms a graphical oriented model of the system and its properties into a specific process calculus. FCVA offers a methodological infrastructure for compositional verification made up of: (1) a formal specification/modelling notation supported by CSP–based (Schneider, 2000) compositional reasoning that enables the preservation of the component properties, and (2) “conceptual hooks” that facilitate the integration of CSP–based model–checkers into the verifica-
tion process of the entire system. FCVA variants can be applied to modelling timed and untimed systems. The untimed infinite traces is used for the analysis of liveness properties, which may contain failures.

In the following section, the formal background to our approach is described. Afterwards, the conceptual framework behind the FCVA is presented. Thereafter, a real-life project regarding mobile phone communications that has to meet critical time requirements. Finally, our conclusions and future work are discussed.

2 FORMAL BACKGROUND

The essence of safety-critical processes behaviour and the sequence and communication synchronization that it should represent are described by CSP and CSP+T models in our proposed method.

\[ \text{SKIP} \equiv \text{success (successful termination)} \]
\[ \text{STOP} \equiv \text{deadlock} \]
\[ t_a \rightarrow P \equiv t_a \text{ then } P \text{ (prefix)} \]
\[ t_0, t_1 
\]

\[ P \cap Q \equiv P \text{ or } Q \] (sequential composition)
\[ P 
\]

\[ \text{P choice } Q \] (deterministic or external choice)
\[ P[A] \equiv P \text{ without } A \] (hiding)
\[ P \parallel Q \equiv P \text{ in parallel with } Q \] (parallel composition)
\[ P[A] \parallel Q \equiv P \text{ in parallel with } Q \text{ in alphabet } A \] (alphabetized composition)
\[ P[[Q] \equiv P \text{ interleave } Q \text{ (interleaving)} \]
\[ I(T, t_0), a \rightarrow P[A] \equiv I(T, t_0)[a] \rightarrow P[A] \]
\[ I(T, t_0), b \rightarrow Q \equiv I(T, t_0)[b] \rightarrow Q \]
\[ \mu X @ P \equiv \text{the process } X \text{ such that } X = P(X) \] (recursion)
\[ \boxplus_{e}^{n} : N \bullet P(i) \equiv i : N \rightarrow P(i) \] (external choice indexed)
\[ \boxplus_{e}^{n} : N \bullet P(i) \equiv i : N \rightarrow P(i) \] (internal choice indexed)
\[ \boxplus_{e}^{n} : N \bullet P(i) \equiv i : N \rightarrow P(i) \] (indexed interleaving)
\[ \boxplus_{e}^{n} : N \bullet A(i) \circ P(i) \equiv i : N \rightarrow \text{success}(i) \circ P(i) \] (partial combination of processes)

CSP+T Syntax Rules

2.1 Specification of the System Model

CSP+T (Zic, 1994) is a real-time specification language which extends Communicating Sequential Processes (CSP) allowing the description of complex event timings, within a single sequential process.

A CSP+T process term \( \tau \) is defined as a tuple \((\alpha, P, P)\), where \( \alpha = \text{Comm}_a(P) \cup \text{Interface}(P) \) is called the communication alphabet of \( P \). These communications represent the events that process \( P \) receives from its environment or those that occur internally. CSP+T is a superset of CSP, the latter being changed by the fact that traces of events become \( pairs \) denoted as \( t_a \), where \( t \) is the time at which event \( a \) is observed. where \( a, x \in \Sigma \) (communication alphabet); \( A, N \subseteq \Sigma \); \( v \in \mathcal{M} \) (variable names); \( I \in I \) (time intervals); \( P, Q, X, \tau \in \mathcal{P} \) (process names); \( t_0, t_1, t_1 \in T \); \( t \in T \) (time instants), and the function \( s(t_a, a) \) which return the occurrence time of symbol \( a \).

The event enabling interval \( I(T, t_0) = \{ t \in T \mid \text{rel}(t_0, v) \leq t \leq \text{rel}(t_0 + T, v) \} \) indicates the time span where any event is accepted. \( \text{rel}(x, v) = x + v - t_0 \), \( t_0 \) corresponds to the preceding instantiation event \( (*) \), occurred at some absolute time \( t_0 \), and \( x \) is the value held in the marker variable \( v \) at that time. The time interval expression can be simplified to \( I(T, t_0) = [t_0, t_0 + T] \) if the instantiation event, after which the event \( a \) can occur, corresponds to the origin \( (t_0 = 0) \) of the real-time clock.

2.2 Abstract Specification of the Properties

Property specification languages are used to obtain a formal specification of the expected SCS behaviour according to the user requirements. CCTL (Raf and Kropf, 1997) is a temporal interval logic that extends Computation Tree Logic (CTL) (Clarke et al., 2000) with quantitative bounded temporal operators, i.e., temporal operators interpreted over time intervals. CCTL includes CTL with the operators \( \text{until} \) (U) and the operator \( \text{next} \) (X) and other derived operators in LTL, such as \( \text{release} \) (R), \( \text{weak until} \) (W), \( \text{cancel} \) (C) and \( \text{since} \) (S). All “LTL-like” temporal op-
Table 1: Example of a map rule from UML–TSM to CSP+T terms.

<table>
<thead>
<tr>
<th>UML–TSM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td>The state $S_1$ precedes the state $S_2$ and these states are reached when events $e_1$ and $e_2$ occur, respectively. But to reach the state $S_2$, the event $e_2$ (restricted event) must occur within the time interval $[T_1, T_1 + T]$ (event–enabling interval), where $T_1$ is the maker variable of the event $e_2$ (marker event). If the restricted event $e_2$ does not occur within the time interval $[T_1, T_1 + T]$ (i.e., the event–enabling interval completely runs), then reaches a pseudostate $\text{Timeout}_1$, $T \in \mathbb{N}^*$ (i.e., natural numbers without zero).</td>
</tr>
</tbody>
</table>

CSP+T Structural Operational Semantics:

1. $e_1$ occurrence  
   $\frac{s_{e_1}(x; t_1) \rightarrow S_2}{\exists t_1 = s(e_1); S_2 \in \text{states}} \quad (S_1, S_2 \in \text{states}; \quad s(e_1))$

2. $e_2$ occurrence OR
   $\frac{S_2 = \text{Timeout}(t_1), e_2 \rightarrow S_3}{S_2, S_3 \in \text{states}} \quad (t_1 = t_1 + T; \quad \text{Timeout} \in \text{pseudostates})$

Timeout execution step

3. $\frac{S_2 = \text{Timeout}(t_1) \rightarrow \text{Timeout} \rightarrow \text{SKIP}}{s(\tau) = t_1 + T; \quad \text{Timeout} \in \text{pseudostates}}$

Generators are preceded by a run quantifier (A universal existential) which determines whether the temporal operator must be interpreted over one run (existential quantification) or over every run (universal quantification). These temporal operators start in the current configuration. For instance, let $\phi$ be the CCTL formula (1) which states that $\psi$ must become true within the interval $[a, b]$ and, that the formula $\phi$ must be valid at all previous time steps. The CCTL formula $\phi$, expressed as a Büchi automaton in Figure 1, is therefore:

$$\phi = \phi_0[u_{[a,b]}]\psi .$$

2.3 Transformation Rules

The formalisation of UML–RT given by MEDISTAM–RT (Benghazi et al., 2007) is of interest here because it allows us to obtain and verify a SCS model from UML diagrams. MEDISTAM–RT (Spanish acronym of Method for System Design based on Analytic Transformation of Real–Time Models) can be described as a series of system views represented by UML for Real Time (UML–RT), with class diagrams, composite structure diagrams, and UML timed state machines (UML–TSM). The expressiveness of UML–TSM is augmented by including new constructs adopted from CSP+T syntax, such that TSMs make now possible to model timing issues and time dependencies among tasks.

Table 1 shows a graphical example of the transformation rules application for obtaining CSP+T process terms from UML–TSMs. We will only present one of the proposed rules, mainly to demonstrate the applicability of FCVA and to show that our approach can be integrated to MC tools like FDR2. A complete description of the system of transformation rules can be found in (Benghazi et al., 2007). The transformation is performed by mapping (1) every UML–TSM state to a CSP+T process term, (2) every transition to a prefixed CSP+T process, (3) every discrete time guard to a CSP+T event–enabling interval, and (4) two or more outgoing transitions to two or more prefixed CSP+T process separated by an external choice operator.

We define the transformation rules according to the Structural Operational Semantics (SOS), which is usually used to formally describe the semantics of programming languages. SOS is compositional, because it allows the semantics of complex process terms to be defined from simpler ones.

The application of the transformation rules’ pattern:

$$\frac{\text{event/communication/execution step}}{\text{premises}} \quad \frac{\text{conclusion}}{\text{conditions}}$$

can be understood as a transformation between two syntactical terms that occur as a consequence of a communication between concurrent processes or an execution step or event occurrence in a sequential process. Thus, each rule defines the premises of the UML–RT element to be transformed and the conditions that must be satisfied before transforming the referred element into the syntactical CSP+T process term indicated in the conclusion of the rule.

3 COMPOSITIONAL VERIFICATION OF SCS

Compositional verification of properties for a given temporal logic has recently been studied intensively by several authors (Giese et al., 2003; Rabinovich,
In order to achieve practical application of MC techniques to the verification of software systems, Temporal Logic (TL) formulas that express the possibility of entering in the state in the future (reachability), or properties expressing liveness, are not preserved by compositionality (Table 2).

### Table 2: Verification–compositionality (VC) of different properties, see (Rabinovich, 2007).

<table>
<thead>
<tr>
<th>Name</th>
<th>TL–denotation</th>
<th>Fulfils VC?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>AG</td>
<td>Yes</td>
</tr>
<tr>
<td>Liveness</td>
<td>AG(req → AFsat)</td>
<td>No</td>
</tr>
<tr>
<td>Reachability</td>
<td>EF</td>
<td>No</td>
</tr>
<tr>
<td>Deadlock freeness</td>
<td>AGEX</td>
<td>Yes</td>
</tr>
<tr>
<td>Fairness</td>
<td>AGAF</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### 3.1 Compositional Verification of a Concurrent System

In a formal way, the system model \( C \) is assumed to be structured into several verified software components working in parallel, i.e., \( C = \parallel_{i=1}^{n} C_i \), where each \( C_i \) satisfies the property \( \phi_i \), i.e., \( C_i \models \phi_i \), which represents the specification of the expected behaviour of the component. Regarding the proposed decomposition strategy, we assume that \( C \) can be decomposed into a set of components, whose behaviour can be specified using a TSM, is found. In addition to the local properties \( \phi_i \), each \( C_i \) must also satisfy the invariant expression \( \psi_i \) that represents the behaviour of other system components with respect to \( C_i \). Since, according to (Abadi and Lamport, 1995), to verify the property \( \phi_i \) of component \( C_i \) we need to assume some kind of behaviour of the other components (i.e., \( \psi_i \)).

**Theorem 1. System Compositional Verification.** Let the system \( C \) be structured into several components working in parallel, \( C = \parallel_{i=1}^{n} C_i \). For a set of TSM(\( C_i \)) describing the behaviour of components \( C_i \), properties \( \phi_i \), invariants \( \psi_i \), and deadlock \( \delta \), with \( \bigcap_{i=1}^{n} \Sigma_i = \emptyset \), \( \bigcap_{i=1}^{n} \Omega_i = \emptyset \), and \( \bigcap_{i=1}^{n} \mathcal{L}(\text{TBA}(C_i)) = \emptyset \), the following condition holds:

\[
\text{TSM}(C) \models (\phi \land \psi \land \neg \delta) \iff \bigwedge_{i=1}^{n} \text{TSM}(C_i) \models (\phi_i \land \psi_i) \land \neg \delta,
\]

where \( \text{TBA}(C) = \parallel_{i=1}^{n} \text{TBA}(C_i) \).

#### 3.1.1 Interpretation of SCV Theorem

If the properties used to specify the system components are circumscribed to the class of composable properties for verification (see Table 2), then property \( \phi \) and the invariant \( \psi \) that are satisfied by the system \( C \) can be obtained by conjunction of local properties \( \phi_i \) (i.e., \( \bigwedge_{i=1}^{n} \phi_i \Rightarrow \phi \)) and invariants \( \psi_i \) (i.e., \( \bigwedge_{i=1}^{n} \psi_i \Rightarrow \psi \)), respectively. The special symbol \( \neg \delta \) is used to denote deadlock absence, i.e., a state without any outgoing transition cannot be reached on any system execution.

A more complete description and practical aspects of our conceptual scheme are detailed in (Mendoza and Capel, 2009).

#### 3.2 Formal Compositional Verification Approach

The rationale of FCVA is that the behavioural correctness of SCS software components can be individually verified, in isolation, based on Theorem 1 and the well–defined communications behaviour specified by MEDISTAM–RT capsule component (Benghazi et al., 2007). FCVA uses the CSP+T specification language, which has a simple but powerful form of composition given by concurrent composition and hiding operators, to describe formally capsules and TSM diagrams. And thus, the automata interpretation, intrinsic to the use of CSP–like notation, allow us to implement complex system behaviour in a easy and direct way (Schneider, 2000). CSP+T–based language allows us to calculate initial events of any syntactical process term as well as when the events occur and what the process is doing after the event occurrence.

Methodologically, our approach establishes that both the formal description of the system’s behaviour and the specification of its properties must be directed by the system’s user requirements. And thus, FCVA consists of the following integrated processes according to MC technique and the automata theory.

**System Interpretation.** Firstly, the complete description of the system’s behaviour, modelled by the CSP+T process term \( T(C) \) is interpreted into a set of CSP+T process terms \( T(C_i) \) by using MEDISTAM–RT. In (Benghazi et al., 2007), the modelling process is detailed.

**Properties Specification.** Then, requirements and temporal constraints that the system must fulfill are specified in CCTL, which is based on the interval structure and time–annotated automata (Ruf and Kropf, 1997). Afterwards, these properties are expressed by CSP+T process terms \( T(\phi_i) \), \( T(\psi_i) \), \( T(\neg \delta) \), following the algorithm described in (Mendoza and Capel, 2009) and then applying the procedure also presented here.
Verification. Finally, we proceed to verify the system behaviour component-by-component.

Thus, we use formal specification/modelling notations supported by CSP–based compositional reasoning that enables the preservation of the component properties throughout the compositionality.

4 APPLICATION

The application of FCVA presented here relates to monitoring the state of mobile devices within the cells that constitute a mobile phone communication network. We present here a real–life scenario where a series of BTSs exchange messages between them, i.e., send message, SndMsg(s); acknowledgement message, AckMsg(s); and receive confirmation, RcvConf(s). The DDBM model represents the functioning of a small distributed database system, which is needed to keep consistent the communication information locally stored in the base stations.

To understand this model of protocol, we need to think of it as a set of finite state automata with symmetries. Each automaton represents $n$ symmetric replicated automata that describe the states of the $n$ managers $d_i$ and the state of the messages transmitted by each $d_i$ during DDBM protocol functioning. The transitions that each automaton must undergo are named, ‘Update and Send Messages’, ‘Receive a Message’, ‘Send an Acknowledgement’ and ‘Receive All Confirmations’ (Jansen, 1997).

4.1 Properties & Software Specification

The complete set of CCTL formulas that formally define the properties fulfilled by the DDBM model’s behaviour are detailed in (Mendoza and Capel, 2009) and derived from user’s requirements. Since the DDBM protocol model is conformed by $n$ replicas of the same component (i.e., DDBM = $\parallel_1..n d_i$), the invariant $\psi_i$, that each component $d_i$ must satisfy is the conjunction of the replicas properties, but without itself, i.e., $\psi_i = \parallel_1..n \phi_j | j \neq i$. Thus, at this stage, we only need to address the verification of local properties $\phi_i$.

We can use an RT-software design method like MEDISTAM–RT (Benghazi et al., 2007), which introduces temporal annotations to UML–TSM to formally describe the protocol (Figure 2). Time labels on the state machines are necessary to assure the fulfilment of maximum time constraints that the real-time DDBM protocol requires. By using these inter-val and time instants specifications, we can guarantee that none of the $d_i$ managers will enter in a blocking state and new occurrences will be disregarded.

4.2 System Components Verification

Once we have obtained the automata,
- $T(d_i), T(AC), T(MM)$, which represent system components, DDBM Manager, Act_Control, and Message_Manager (Figure 2), respectively.
- As well as the ones corresponding to the properties, $T(\phi_{RUAC}), T(\phi_{RUUM}), T(\phi_{LUAC}), T(\phi_{LUUM})$ (Mendoza and Capel, 2009).

We can proceed to the verification of the DDBM system, component by component. Then, under the semantic domain of CSP–based process calculus, we can automatically check with the help of FDR2 (Formal Systems Europe Ltd., 2005) tool that the following relations of refinement are satisfied:

$\forall \psi \in \phi \subseteq T(\phi)$

We say that there is a refinement relation between two formal automata $T(\phi) \subseteq T(\psi)$ if every trace of execution of $T(\psi)$ is included in the set of traces and failures that defines the behaviour of the automaton $T(\phi)$ (Schneider, 2000).

According to the conditions of System Compositional Verification Theorem 1 (see section 3.1), and based on the detailed design of Act_Control and Message_Manager components shown in Figure 2, we must determine now whether the individual verification of these components is “composable”. We must verify that the following 2 conditions of Theorem 1 are always fulfilled:

1. The input signals ($\Sigma_{Act\_Control}$ and $\Sigma_{Message\_Manager}$) and the output signals ($\Omega_{Act\_Control}$ and $\Omega_{Message\_Manager}$) of both components are disjoint. The encapsulation of the automata that only communicate through dedicated input/output ports ?m and !m, respectively, makes this condition always true.

2. The labelling sets of both components $L(Act\_Control)$ and $L(Message\_Manager)$ are disjointed. This can also be easily verified since transition and state labels of each automaton are only visible inside the capsule.

The main interest of Theorem 1 is to address the difficult problem of proving that the satisfaction of

---

2Base Transceiver Stations
a complex property of the system can be determined by the individual verification of simpler properties of its components and the rules used to combine them. In our case, the proposed adaptation of (Abadi and Lamport, 1995) Theorem has as its most important consequence the fact that compositional verification of an SCS becomes reduced to proof the reliability of a communication protocol between deterministic CSP+T processes with interfaces and communication alphabets previously defined.

4.3 BT RANSFORMER Tool

The main objective of BT RANSFORMER tool is to generate a CSP+T specification from UML–TSM diagrams of any reactive system. In previous work (Mendoza et al., 2012), we have improved the semantic proposed in (Wong and Gibbons, 2009) by incorporating the CSP+T operators, which allow the definition of a timed semantics of TSM and Composite Structure Diagrams of UML/MEDISTAM–RT.

4.3.1 BT RANSFORMER Properties

Developed using Open Unified Process (OpenUP) methodology (Eclipse.org), BT RANSFORMER has the capability to read input/output models written in standard XML and it can be used with different operating systems: Windows, Linux and MacOS. It allows the analyst to have access to the TSM2CSP menu options for creating and editing UML–TSM to CPS+T transformation rules.

All the plugins needed to implement BT RANSFORMER (Figure 3) are based on the Eclipse platform, especially those that allow the implementation of the interfaces. The integration of transformation languages was achieved with the editor Intalio, also integrable with the Eclipse platform.

The plugin Intalio controls the entire modelling process from the initial source model (including temporal annotations on TSMs) and helps to integrate the transformation notations. In its part, the plugin Utils handles the reading of MEDISTAM–RT modelling entities in the source model and helps to write CSP+T processes in the object model.

4.3.2 Preliminary Tests

CSP+T models yielded by BT RANSFORMER were in-
Table 3: Results of preliminary tests.

<table>
<thead>
<tr>
<th>Model</th>
<th>Completeness of elements</th>
<th>Number of processes</th>
<th>Completeness of relation</th>
<th>Behavioural safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport Chain</td>
<td>Yes</td>
<td>12</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Logistic process in hospitals</td>
<td>Yes</td>
<td>22</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

5 CONCLUSIONS

In this paper we have presented FCVA for compositional software verification from independently verified individual components. MC was used to prove the correctness of individual components and a CSP-based process calculus inspired formal language was integrated in order to foster the composition of SCS, aided by concurrent composition operators.

We have shown the value and practicality of our approach by means of the application to a real-life project in the field of mobile communications, which has to meet time critical requirements. The CSP+T specification of the system components at the design phase can be verified against the CCTL specification of the individual system component properties.

ACKNOWLEDGEMENTS

We would like to thank A. González from Simón Bolívar University in Caracas (Venezuela) for implementing the BTRANSFORMER in the Eclipse platform.

REFERENCES


