A Compositional Scheme and Framework for Safety Critical Systems Verification

Manuel I. Capel\textsuperscript{1} and Luis E. Mendoza-Morales\textsuperscript{2}

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

Abstract. Safety–Critical Systems (SCS) must satisfy dependability requirements such as availability, reliability, and real-time constraints, in order to justify the reliance of the critical service they deliver. A verification framework named Formal Compositional Verification Approach (FCVA) is presented here. FCVA establishes a compositional method to verify safety, fairness and deadlock absence of SCS. Software components of a given critical system are model–checked to verify the aforementioned properties. Our objective in this paper is to facilitate the design of an SCS from a collection of verified simpler components, and hence 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

Safety–Critical Systems (SCS), including energy production, automotive, medical systems, avionics, modern telecommunications . . . , they are industrial systems where availability, performance, safety and the other dependability attributes should justify the reliance on the critical service they deliver to their customers. The baseline for obtaining a verifiable design of SCS is to previously develop a specification of the target system using at least one formal language to perform the subsequent verification of the system. Based on component abstraction and system modularity, advanced Model–Checking (MC) techniques \cite{1},\cite{2}, have become an active area of research, and are frequently used to uncover well–hidden bugs in sizeable industrial SCS (see Table 1). However, SCS automatic verification can be impeded by the state explosion problem that a model checker tool has to tackle when the system model is huge and complex \cite{3},\cite{4}, \cite{5},\cite{6},\cite{7},\cite{8}.

The objective here is to facilitate the description of an SCS as a collection of simpler verified components, then allowing the verification of safety, fairness and deadlock absence of a complex SCS software.

However, it becomes very difficult to export local verification results using a formal language with conjunctive propositional logic operators, and preserve, at the same time, the semantic correctness of these demonstrations when they are compound. In
our view, the lack of compositionality that automatic verification techniques exhibit is mainly due to semantic and syntactical problems, caused by the incorrect integration of different specification formalisms, i.e., property specification languages (CTL, ACTL, etc.) and modelling notations based on states (formal automata, Promela, etc.), which, depending on the formal combination language, may induce semantic errors in the system’s verification. So far, the notational integration and semantical integration has not been solved.

A unique underlying common semantic domain, within which the different specification formalisms used in SCS verification are interpreted, may help to obtain the desired compositionality of verified system components. Therefore, a new Formal Compositional Verification Approach (FCVA) is proposed to verify a SCS from individual components, based on a conceptual framework that transforms the model and properties of the SCS into a CSP–based formal language. An automatable instance of our framework FCVA is introduced [9],[10]. FCVA gives a methodological infrastructure for verification made up of: (1) a formal specification/modelling notation supported by CSP–based compositional reasoning that enables the preservation of the component properties throughout the compositionality to be demonstrated, and (2) conceptual hooks that facilitate the integration of CSP–based MC tools into the verification process.

The paper is organized as follows. In the following section the formal background to our approach is described. Afterwards, the conceptual framework behind the FCVA is presented, followed by a complete description of how the FCVA is designed. Thereafter, we demonstrate the value and practicality of our approach through the application on a real–life project in the field of mobile phone communications, which has to meet critical time requirements. Finally, in the last section, our conclusions and future work are discussed.

Table 1. Main characteristics of related work.

<table>
<thead>
<tr>
<th>Formalism Work</th>
<th>Strategy</th>
<th>Properties</th>
<th>Tool</th>
<th>Encapsulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6] Propose a preorder for use with a subset of CTL*.</td>
<td>Safety, simple liveness, and a notion of fairness.</td>
<td>Ad-hoc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>[11] MC to deal with control issues and deductive method to handle data-intensive elements.</td>
<td>Safety.</td>
<td>SPIN</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>[5] Incremental assume-guarantee reasoning.</td>
<td>Safety.</td>
<td>LTSA</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>[12] Compose complex software systems from domain-specific patterns.</td>
<td>Safety.</td>
<td>RAVEN</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>[7] Propose a modular verification approach.</td>
<td>Safety.</td>
<td>SMV</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>[14] Automatically generating component assumptions based on the behaviour of the environment.</td>
<td>Do not specify.</td>
<td>SPIN</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Algebra [16]</td>
<td>Proposes a compositional technique for traces refinement checking.</td>
<td>Safety.</td>
<td>FDR</td>
<td>Yes</td>
</tr>
<tr>
<td>[17] Constructing decompositions to efficient AGR.</td>
<td>Safety.</td>
<td>FDR</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

2 Formal Background

The essence of safety–critical processes behaviour and the sequence and communic-
tion synchronization that it should represent are described by CSP [18] and CSP+T [19] models in our proposed method.

2.1 Specification of the System Model

CSP+T 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 \( P \) is defined as a tuple \((\alpha P, P)\), where \( \alpha P = \text{Comm–act}(P) \cup \text{Inter-face}(P) \) 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 \( a, \ast \in \Sigma \) (communication alphabet); \( A, N \subseteq \Sigma \); \( v \in M \) (marker variables); \( I \in \mathcal{T} \) (time intervals); \( P, Q, X, \bar{P} \in \mathcal{P} \) (process names); \( t_0, t_a, t_1 \in T \); and \( T \in \mathbb{N} \) (time instants), and the function \( s(t,a) \) which return the occurrence time of symbol \( a \).

Table 2. CSP+T Syntax Rules.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{STOP} )</td>
<td>deadlock</td>
</tr>
<tr>
<td>( t \cdot a \rightarrow P )</td>
<td>( t ) is the time at which event ( a ) is observed</td>
</tr>
<tr>
<td>( 0.\ast \rightarrow \bar{P} )</td>
<td>( 0 ) is the time at which event ( \ast ) is observed</td>
</tr>
<tr>
<td>( I(t_a, t) \cdot \alpha \rightarrow P )</td>
<td>process instantiation</td>
</tr>
<tr>
<td>( P \cap Q )</td>
<td>( P ) and ( Q ) are both present</td>
</tr>
<tr>
<td>( P \cdot Q )</td>
<td>( P ) and ( Q ) are parallel</td>
</tr>
<tr>
<td>( P \Delta Q )</td>
<td>( P ) is interrupted by ( Q )</td>
</tr>
<tr>
<td>( I(t, t_a) \cdot a \rightarrow P )</td>
<td>event enabling interval</td>
</tr>
</tbody>
</table>

The event enabling interval \( I(t, t_a) = \{ t \in \mathcal{T} \mid \text{rel}(t_a, v) \leq t \leq \text{rel}(t_a + T, v) \} \) 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 (\( \ast \)), 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 \( I(t, t_a) = [t_a, t + T] \) if the instantiation event, after which the event \( a \) can occur, corresponds to the origin \( (t_0 = 0) \) of the r-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 [20] is a temporal
interval logic that extends *Computation Tree Logic* (CTL) with quantitative bounded temporal operators, i.e., temporal operators interpreted over time intervals. CCTL is used to deal with sequences of states, where a state gives a temporal interpretation of a set of *atomic propositions* (AP) at a certain time interval and time instants are isomorphic to the set of non-negative integers.

CCTL includes CTL with the operators *until* ($U$) and the operator *next* ($X$) and other derived operators in LTL, such as *release* ($R$), *weak until* ($W$), *cancel* ($C$) and *since* ($S$). All of them have proved to be useful to facilitate the definition of the properties included in reactive systems classes —such as the SCS one— requirements specification. All “LTL-like” temporal operators are preceded by a run quantifier (A universal, E 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 specification of the formula $\phi$ in Figure 1 is therefore:

$$\phi = \phi U_{[a,b]} \psi.$$  \hspace{1cm} (1)

2.3 Transformation Rules

The formalisation of UML–RT given by MEDISTAM–RT [21] is of interest here because it allows us to obtain and verify a SCS model from UML diagrams. MEDISTAM–RT (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 state–machines (UML–SM) is augmented by including new modelling constructs adopted from CSP+T syntax, such that TSMs make now possible to model timing issues and time dependencies among tasks. Table 3 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 [21].

The application of the transformation rules’ pattern:

$$\text{event/communication/execution step} \quad \text{premises (conditions)} \quad \text{conclusion (conditions)}$$ \hspace{1cm} (2)
Table 3. 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 S1 precedes the state S2 and these states are reached when events e_1 and e_2 occur, respectively. But to reach the state S2, 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_1 (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 Timeout. T_1, T ∈ N^* (i.e., natural numbers without zero).</td>
</tr>
</tbody>
</table>

CSP+T Structural Operational Semantics

1. \( e_1 \) occurrence \( S_1 = e_1 \overset{\tau_1}{\rightarrow} S_2 \) \( (s(a_1), S_1, S_2) \in \text{states}; \)
2. \( e_2 \) occurrence \( S_2 = \text{I}(T, \tau_1) \overset{\tau_2}{\rightarrow} S_3 \) \( (s(e_2) \in [T_1, T_1 + T]; S_2, S_3) \in \text{states}; \)

OR

\( \text{I}(T, \tau_1) \) timeout \( S_2 = \text{I}(T, \tau_1) \overset{T \text{imeout}}{\rightarrow} \text{TIMEOUT} \) \( (s(\tau) < T_1 + T, S_2) \in \text{states}; \)

\( \text{Timeout} \in \text{pseudostates} \)

Timeout execution step \( \text{TIMEOUT} \overset{\tau}{\rightarrow} \text{SKIP} \) \( (s(\tau) = T_1 + T, \text{TIMEOUT}) \in \text{pseudostates}; \)

\( \text{TIMEOUT} \in \text{pseudostates} \)

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 [12], [22], in order to solve a fundamental problem of practical application of MC techniques to the verification of software systems.

A compositional scheme can be applied to the verification of temporal formulae 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 be eventually entered in an infinite computation (fairness) (see Table 4). In contrast, Temporal Logic (TL) formulas that express the possibility of entering in a state in the future (reachability) are not preserved by compositionality, nor properties expressing that something is unavoidable in the future provided that some other thing occurs (liveness).

3.1 Compositional Verification of a Concurrent System

FCVA is aimed at performing compositional verification of behavioral properties of SCS. In a formal way, the system model C is assumed to be structured into several verified software components working in parallel, i.e., \( C = \prod_{i=1}^{n} C_i \), 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,
Table 4. Verification–compositionality (VC) of different properties.

<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>EFφ</td>
<td>No</td>
</tr>
<tr>
<td>Deadlock freeness</td>
<td>AGEXtrue</td>
<td>Yes</td>
</tr>
<tr>
<td>Fairness</td>
<td>AGAFφ</td>
<td>Yes</td>
</tr>
</tbody>
</table>

we assume that \( C \) can be decomposed until 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 [23], to verify the property \( \phi_i \) of component \( C_i \) we need to assume the other components’ behaviour (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 \( \bigcap_{i=1}^n L(TBA(C_i)) = \emptyset \), the following condition holds:

\[
TSM(C) = (φ ∧ ψ ∧ ¬δ) ⇒ \bigwedge_{i=1}^n TSM(C_i) = \bigwedge_{i=1}^n (φ_i ∧ ψ_i) ∧ ¬δ \tag{3}
\]

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

**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 4), then property \( φ \) and the invariant \( ψ \) that are satisfied by the system \( C \) can be obtained by conjunction of local properties \( \phi_i \) (i.e., \( \bigwedge_{i=1}^n \phi_i \Rightarrow φ \) and invariants \( \psi_i \) (i.e., \( \bigwedge_{i=1}^n \psi_i \Rightarrow ψ \)), respectively. The special symbol \( ¬\delta \) is used to denote deadlock absence, i.e., a state without any outgoing transition cannot be reached on any system execution.

### 3.2 Formal Compositional Verification Approach

Based on previous concepts and ideas, we propose a possible instantiation of the conceptual scheme called FCVA. 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 UML/MEDISTAM–RT *capsule* component [21]. 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 [21].
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 \[20\]. Afterwards, these properties are expressed by CSP+T process terms \(T(\phi_i), T(\psi_i), T(\neg\delta)\), following the algorithm described in \[9\]. In this way, we translate the properties to the same semantic domain of the system model in order to perform the verification process.

Verification. Finally, we proceed to verify the system behaviour component by component.

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

![Graphical model of a DDBM communication protocol (in [24]).](image)

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 a simple case study, but conceptually relevant. It is real-life scenario where five BTSs\(^3\) (A to E) exchange messages between them, i.e., \(\text{SndMsg}(s)\); acknowledgement message, \(\text{AckMsg}(s)\); and receive confirmation, \(\text{RcvConf}(s)\).

The DDBM model shown in Figure 2 represents the functioning of a small distributed database system, which is needed to keep consistent the communication information locally stored in the base stations. Each site contains a copy of the entire database and this copy is handled by a replicated local data base manager (DDBM = \(\parallel_{i=1}^n d_i\)). When a manager \(d_i\) makes an update to its own copy, it must send a message (denoted as \(\text{SndMsg}(s)\)) to all the other managers to ensure consistency between the \(n\) copies of the data base, \(d_i; \text{SndMsg}(s) = \{(s,r)|s, r \in \text{DDBM} \land s \neq r\}\).

To understand the model of this DDBM communication protocol, we need to think of it as a set of finite state automata with symmetries. The automaton on Figure 2 represents \(n\) symmetric replicated automata that describe the states (ellipses) of the \(n\) managers \(d_i\) and the state of the messages (rectangles) transmitted by each \(d_i\) during DDBM protocol functioning. The transitions that each automaton must undergo are represented

\(^3\)Base Transceiver Stations
Table 5. Properties for components that implements the DDBM communication protocol.

<table>
<thead>
<tr>
<th>Property</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Remote update request by Act_Control (RUAC)</td>
<td>$\Phi_{RUAC} := \neg A[Upd(s)] \land A[Upd(s)] \land A[ConfL(s)] \land A[ConfR(s)]$</td>
</tr>
<tr>
<td>(b) Remote update request by Message_Manager (RUMM)</td>
<td>$\Phi_{RUMM} := \neg A[Upd(s)] \land A[Upd(s)] \land A[ConfL(s)] \land A[ConfR(s)]$</td>
</tr>
<tr>
<td>(c) Local update by Act_Control (LUAC)</td>
<td>$\Phi_{LUAC} := \neg A[LocU(s)] \land A[LocU(s)] \land A[ConfL(s)] \land A[ConfR(s)]$</td>
</tr>
<tr>
<td>(d) Local response by Message_Manager (LUMM)</td>
<td>$\Phi_{LUMM} := \neg A[LocR(s)] \land A[LocR(s)] \land A[ConfL(s)] \land A[ConfR(s)]$</td>
</tr>
</tbody>
</table>

The complete set of CCTL formulas that formally define the properties fulfilled by the DDBM model’s behaviour are detailed in [9] and derived from user’s requirements. Table 5 shows the interpretation of the property $\phi$, expressing the guarantee of processing one message at a time, according to the DDBM ‘Active’ and ‘Passive’ states, respectively. When a $d_i$ manager enters the ‘Active’ state (i.e., request a remote update), the Act_Control component must engage in a sequence of events that corresponds with the fulfillment of properties (a)-(d) set in Table 5. Therefore, the properties that express the expected behaviour of Act_Control and Message_Manager components can be expressed as conjunctions of simpler properties, $\Phi_{Act\_Control} = \Phi_{RUAC} \land \Phi_{LUAC}$ and $\Phi_{Message\_Manager} = \Phi_{RUMM} \land \Phi_{LUMM}$, respectively. Since the DDBM protocol model is conformed by $n$ replicas of the same component (i.e., DDBM = $\parallel_{i=1..n} d_i$), the in-
variant $\psi_i$ that each component $d_i$ must satisfy is the conjunction of the properties of the $n$ replicas, but without itself, i.e., $\psi_i = \bigwedge_{j \neq i} \phi_j$, we need not include the invariants $\psi_i$ as part of the verification process. Our method, at this stage, needs only to address verification of local properties $\phi_i$. From the practical viewpoint, if we included invariants $\psi_i$ in the verification process, we would be double-checking the satisfaction of property $\phi_i$ in each automaton, which is neither efficient nor necessary.

4.2 Software Specification

We can use an RT-software design method like MEDISTAM [21], which introduces temporal annotations to UML–TSM to formally describe the protocol. Time labels on the state machines are necessary to assure the fulfillment of maximum time constraints that the real-time DDBM protocol requires. By using these interval and time instants specifications, we can guarantee that none of the $d_i$ managers will enter in a blocking state and hence new updating occurrences will be disregarded.

4.3 System Components Verification and Discussion

Once we have obtained the automata, $T(d_i), T(AC), T(MM)$, which represent system components, DDBM Manager, Act-Control, and Message Manager, respectively.

- As well as the ones corresponding to the properties, $T(\phi_{RUAC}), T(\phi_{RUMM}), T(\phi_{LUAC}), T(\phi_{LUMM})$ (Table 5).

We can proceed to the verification of the DDBM system, component by component.

According to our approach, we must verify that the behavior of the above components fulfills the properties specified in section 4.1. Then, under the semantic domain of CSP–based process calculus, we can automatically check with the help of FDR2 [25] tool that the following relations of refinement are satisfied:

$$T(\phi_{LUAC}) \sqsubseteq_T T(AC), T(\phi_{RUAC}) \subseteq_T T(AC)$$
$$T(\phi_{LUAC}) \sqsubseteq_F T(AC), T(\phi_{RUAC}) \subseteq_F T(AC)$$
$$T(\phi_{LUMM}) \sqsubseteq_T T(MM), T(\phi_{RUMM}) \subseteq_T T(MM)$$
$$T(\phi_{LUMM}) \sqsubseteq_F T(MM), T(\phi_{RUMM}) \subseteq_F T(MM)$$

We say that there is a refinement relation between two formal automata $T(\Phi) \sqsubseteq_T T(Component)$ if every trace of execution of $T(Component)$ is included in the set of traces and failures that defines the behavior of the automaton $T(\Phi)$ [26], i.e., the automaton $T(Component)$ “formally implements” the specification described by automaton $T(\Phi)$.

**Compositional Verification.** According to the conditions of System Compositional Verification Theorem 1 (3.1), and based on the detailed design of Act_Control and Message_Manager components shown in Figure 3, 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 ($\Sigma_{\text{Act\_Control}}$ and $\Sigma_{\text{Message\_Manager}}$) and the output signals ($\Omega_{\text{Act\_Control}}$ and $\Omega_{\text{Message\_Manager}}$) of both components are disjoint. In Figure 3 it can be seen how the encapsulation of the automata that only communicate through dedicated input/output ports $?m$ and $!m$ makes this condition always true.

2. The labelling sets of both components $\mathcal{L}(\text{Act\_Control})$ and $\mathcal{L}(\text{Message\_Manager})$ 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 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 [23] 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.

Finally, from (1) and (2), we can conclude that $\text{Act\_Control}$ and $\text{Message\_Manager}$ system components are therefore “composable”,

$$\text{Act\_Control} \parallel \text{Message\_Manager} \models \phi_{\text{Act\_Control}} \land \phi_{\text{Message\_Manager}} \quad (8)$$

and because of that we can affirm the compositional property of the entire system,

$$d_i = \text{Act\_Control} \parallel \text{Message\_Manager} \quad \text{and} \quad \phi_{d_i} = \phi_{\text{Act\_Control}} \land \phi_{\text{Message\_Manager}} \quad (9)$$

And hence, the entire system’s model represented by each replicated DB manager $d_i$ satisfies the property $\phi_{d_i}$, that represents every manager’s behaviour,

$$d_i \models \phi_{d_i} \quad (10)$$

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.

References