A Compositional Scheme and Framework for Safety Critical Systems Verification

Manuel I. Capel[1] and Luis E. Mendoza-Morales[2]

[1]Department of Software Engineering, Informatics & Telecomunications Bldg.,
University of Granada, 18071 Granada, Spain
[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 [1],[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 [3],[4], [5],[6],[7],[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

I. Capel M. and E. Mendoza-Morales L..
A Compositional Scheme and Framework for Safety Critical Systems Verification.
DOI: 10.5220/0004097300150026
In *Proceedings of the 10th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems and 1st International Workshop on Web Intelligence* (MSVVEIS-2012), pages 15-26
ISBN: 978-989-8565-14-3

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.

Formalism	Work	Strategy	Properties	Tool	Encapsulation
Automata	[4]	Model the environment using additional interface processes.	Liveness, Mutual exclusion. Limited to loosely coupled systems.	No	Yes
	[6]	Propose a preorder for use with a subset of CTL*.	Safety, simple liveness, and a notion of fairness.	Ad-hoc	No
	[3]	Incrementally refinement using partial models.	Safety and liveness.	Ad-hoc	No
	[11]	MC to deal with control issues and deductive method to handle data-intensive elements.	Safety.	SPIN	Yes
	[5]	Incremental assume-guarantee reasoning.	Safety.	LTSA	Yes
	[12]	Compose complex software systems from domain–specific patterns.	Safety.	RAVEN	Yes
	[13]	Modular analysis based on the Assumption/Commitment method.	Safety.	HyTech	No
	[7]	Propose a modular verification approach.	Safety.	SMV	No
	[14]	Automatically generating component assumptions based on the behaviour of the environment.	Do not specify.	SPIN	Yes
	[8]	Using the equivalence relation among processes.	Deadlock, Starvation, and Parallelism.	Ad-hoc	Yes
Process	[15]	Presents a simple formulation of AGR using CSP.	Safety.	FDR	Yes
Algebra	[16]	Proposes a compositional technique for traces refinement checking.	Safety.	FDR	Yes
	[17]	Constructing decompositions to efficient AGR.	Safety.	FDR	Yes

2 Formal Background

The essence of safety–critical processes behaviour and the sequence and communica-

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 $\mathcal{P}$ is defined as a tuple (αP, P), where $\alpha P = Comm\_act(P) \cup Interface(P)$ 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 $a, \star \in \Sigma$ (communication alphabet); $A, N \subseteq \Sigma$; $v \in \mathcal{M}$ (marker variables); $\mathrm{I} \in \mathcal{I}$ (time intervals); $P, Q, X, \widetilde{P} \in \mathcal{P}$ (process names); $t_0, t_a, t_1 \in \mathcal{T}$; and $T \in \mathbb{N}$ (time instants), and the function $s(t_a.a)$ which return the occurrence time of symbol a.

Table 2. CSP+T Syntax Rules.

$SKIP$	$:\equiv$ $success$ (successful termination)	$P\|[A]\|Q$	$:\equiv$ (parallel composition) P in parallel with Q
$STOP$	$:\equiv$ $deadlock$	in alphabet A	(alphabetized composition)
$t_a.a \rightarrow P$	$:\equiv$ $t_a.a$ then P (prefix)	$P \mid\mid\mid Q$	$:\equiv$ P interleave Q (interleaving)
$t0.\star \rightarrow \widetilde{P}$	$:\equiv$ $(\star \wedge s(\star) = t0)$ then $\widetilde{P}$ (process instantiation)	$\mathrm{I}(T_a, t_a).a \rightarrow P\|[A]\|$ $\mathrm{I}(T_b, t_b).b \rightarrow Q$	$:\equiv$ $P\|Q$ if $(a = b) \wedge$ $(\mathrm{I}(T_a, t_a) \cap \mathrm{I}(T_b, t_b) \neq \emptyset)$ $:\equiv$ $P \mid\mid\mid Q$ if $(a \neq b) \wedge$ $(\mathrm{I}(T_a, t_a) \cap \mathrm{I}(T_b, t_b) \neq \emptyset)$ $:\equiv$ $STOP$ if $\mathrm{I}(T_a, t_a)$ $\cap \mathrm{I}(T_b, t_b) = \emptyset$
$t_a.a \bowtie v \rightarrow P$	$:\equiv$ $(t_a.a \wedge s(a) = t_a)$ then P (marker variable)		
$P \fatsemi Q$	$:\equiv$ P (successfully) followed by Q (sequential composition)		
$P \sqcap Q$	$:\equiv$ P or Q (non–deterministic)	$\mu X@P$	$:\equiv$ the process X such that $X = P(X)$ (recursion)
$P \Box Q$	$:\equiv$ P choice Q (deterministic or external choice)	$\Box_{i=1}^{m} : N \bullet P(i)$	$:\equiv$ $i : N \rightarrow P(i)$ (external choice indexed)
$P \backslash A$	$:\equiv$ P without A (hidding)	$\sqcap_{i=1}^{m} : N \bullet P(i)$	$:\equiv$ $P((\tau -) action)$ (internal choice indexed)
$P \triangle Q$	$:\equiv$ P interrupted by Q	$\|\|\|_{i=1}^{m} : N \bullet P(i)$	$:\equiv$ $i : N \rightarrow \|\|\|_{i=1}^{m} P(i)$ (indexed interleaving)
$\mathrm{I}(T, t_1).a \rightarrow P$	$:\equiv$ $(t_a.a \wedge t_a \in [rel(t_1, v), rel(t_1 + T, v)])$ then P (event–enabling interval)	$\|\|_{i=1}^{m}[A] : N \bullet P(i)$	$:\equiv$ $i : N \rightarrow \|\|_{i=1}^{m} P(i)$ (partial interleaving)
$\mathrm{I}(T, t_1) \rightarrow \widetilde{P}$	$:\equiv$ $t > rel(t_1 + T, v)$ then $\widetilde{P}$ (delay)	$\|\|_{i=1}^{m} : N \bullet A(i) \circ P(i)$	$:\equiv$ $i : N \rightarrow \|\|_{i=1}^{m} A(i) \circ P(i)$ (parallel combination)
$P \| Q$	$:\equiv$ P in parallel with Q		

The event enabling interval $\mathrm{I}(T, t_a) = \{t \in \mathcal{T} | rel(t_a, v) \leq t \leq rel(t_a + T, v)\}$ indicates the time span where any event is accepted. $rel(x, v) = x + v - t_0$, t_0 corresponds to the preceding *instantiation event* ($\star$), 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 $\mathrm{I}(T, t_a) = [t_a, t_a + T]$ if the instantiation event, after which the event a can occur, corresponds to the origin ($t_0 = 0$) of the rt-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

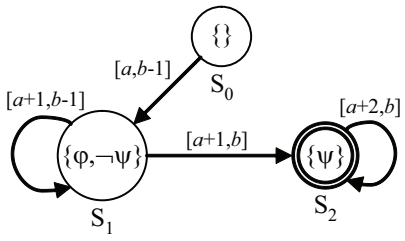


Fig. 1. CCTL formula.

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 ϕ be the CCTL formula 1 which states that ψ must become true within the interval $[a, b]$ and, that the formula φ must be valid at all previous time steps. The CCTL specification of the formula ϕ in Figure 1 is therefore:

$$\phi = \varphi \mathsf{U}_{[a,b]} \psi \ . \tag{1}$$

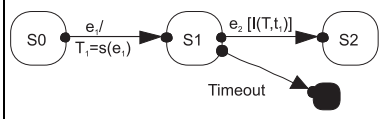
2.3 Transformation Rules

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:

$$event/communication/execution\ step) \quad \frac{premises}{conclusion}(conditions) \tag{2}$$

Table 3. Example of a map rule from UML–TSM to CSP+T terms.

UML–TSM	Description
	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 $[T1, T1+T]$ (*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 $[T1, T1+T]$ (i.e., the event–enabling interval completely runs), then reaches a pseudostate *Timeout*. $T_1, T \in \mathbb{N}^*$ (i.e., natural numbers without zero).
CSP+T Structural Operational Semantics	
1. e_1 occurrence) $\frac{S1=e_1 \bowtie t_1 \to S2}{t_1=s(e_1);\ S2}$ $\begin{pmatrix} S1, S2 \in states; \\ s(e_1) \end{pmatrix}$ 2. e_2 occurrence) $\frac{S2=\mathrm{I}(T,t_1)\cdot e_2 \to S3}{s(e_2);\ S3}$ $\begin{pmatrix} s(e_2) \in [t_1, t_1+T]; \\ S2, S3 \in states \end{pmatrix}$ OR $\mathrm{I}(T, t_1)$ timeout) $\frac{S2=\mathrm{I}(T,t_1) \to Timeout \to SKIP}{s(\tau);\ Timeout \to SKIP}$ $\begin{pmatrix} s(\tau) < t_1 + T; S2 \in states; \\ Timeout \in pseudostates \end{pmatrix}$ $Timeout$ execution step) $\frac{Timeout \to SKIP}{s(\tau);\ SKIP}$ $\begin{pmatrix} s(\tau) = t_1 + T; \\ Timeout \in pseudostates \end{pmatrix}$	

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 $\mathbb{C}$ is assumed to be structured into several verified software components working in parallel, i.e., $\mathbb{C} = \|_{i:1..n} C_i$, where each C_i satisfies the *property* ϕ_i, i.e., $C_i \vDash \phi_i$, which represents the specification of the expected behaviour of the component. Regarding the proposed decomposition strategy,

Table 4. *Verification–compositionality* (VC) of different properties.

Name	TL–denotation	Fulfils VC?
Safety	AG	Yes
Liveness	$\mathsf{AG}(req \rightarrow \mathsf{AF}sat))$	No
Reachability	$\mathsf{EF}\phi$	No
Deadlock freeness	$\mathsf{AGEX}true$	Yes
Fairness	$\mathsf{AGAF}\phi$	Yes

we assume that $\mathbb{C}$ can be decomposed until a set of components, whose behaviour can be specified using a TSM, is found. In addition to the local properties ϕ_i, each C_i must also satisfy the invariant expression ψ_i that represents the behaviour of other system components with respect to C_i. Since, according to [23], to verify the property ϕ_i of component C_i we need to assume the other components' behaviour (i.e., ψ_i).

Theorem 1: System Compositional Verification. Let the system $\mathbb{C}$ be structured into several components working in parallel, $\mathbb{C} = \|_{i:1..n}\, C_i$. For a set of $TSM(C_i)$ describing the behaviour of components C_i, properties ϕ_i, invariants ψ_i, and deadlock δ, with $\bigcap_{i:1..n} \Sigma_i = \emptyset$, $\bigcap_{i:1..n} \Omega_i = \emptyset$, and $\bigcap_{i:1..n} \mathcal{L}(TBA(C_i)) = \emptyset$, the following condition holds:

$$TSM(\mathbb{C}) \vDash (\phi \wedge \psi \wedge \neg\delta) \Leftrightarrow \Big|\Big|_{i:1..n} TSM(C_i) \vDash \bigwedge_{i:1..n} (\phi_i \wedge \psi_i) \wedge \neg\delta, \quad (3)$$

where $TBA(\mathbb{C}) = \|_{i:1..n}\, TBA(C_i)$.

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 $\mathbb{C}$ can be obtained by conjunction of local properties ϕ_i (i.e., $\bigwedge_{i:1..n} \phi_i \Rightarrow \phi$) and invariants ψ_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.

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 $T(\mathbb{C})$ 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.

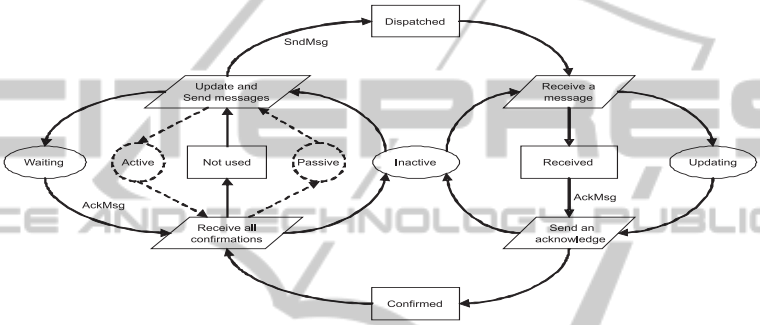


Fig. 2. Graphical model of a DDBM communication protocol (in [24]).

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.

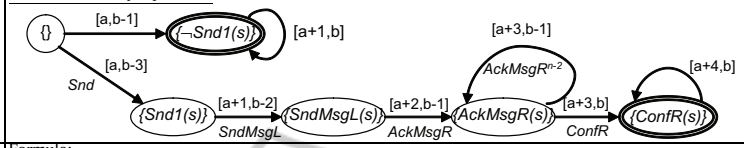
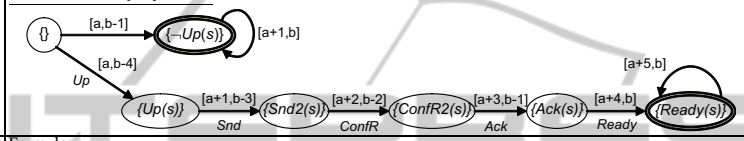
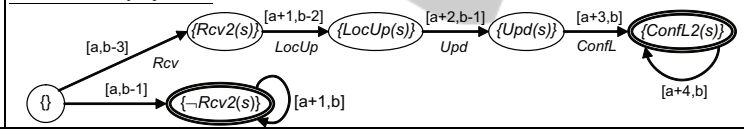
We present a simple case study, but conceptually relevant. It is real–life scenario where five BTSs[3] (A to E) exchange messages between them, i.e., *SndMsg(s)*; acknowledgement message, *AckMsg(s)*; and receive confirmation, *RcvConf(s)*.

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 = $\|_{i:1..n}\ d_i$). When a manager d_i makes an update to its own copy, it must send a message (denoted as *SndMsg(s)*) to all the other managers to ensure consistency between the n copies of the data base, d_i: $SndMsg(s) = \{(s,r)|s,r \in \mathrm{DDBM} \wedge s \neq r\}$.

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.

Property	Specification
(a) Remote update request by $Act\_Control$ (RUAC)	Formula: $\phi_{RUAC} := \mathsf{AG}_{[a,b]}(Snd1(s) \rightarrow \mathsf{A}[SndMsgL(s)\ \mathsf{U}_{[a+1,b-1]}\ (\bigwedge_{x:1..n-1} AckMsgR(s_x) \wedge \mathsf{A}[\bigwedge_{x:1..n-1} AckMsgR(s_x)\ \mathsf{U}_{[a+1,b]}\ ConfR(s)])])$ TBA semantically equivalent: 
(b) Remote update request by $Message\_Manager$ (RUMM)	Formula: $\phi_{RUMM} := \mathsf{AG}_{[a,b]}(Up(s) \rightarrow \mathsf{A}[Snd2(s)\ \mathsf{U}_{[a+1,b-2]}\ (ConfR(s) \wedge \mathsf{A}[ConfR(s)\ \mathsf{U}_{[a+2,b-1]}\ (Ack2(s) \wedge \mathsf{A}[Ack2(s)\ \mathsf{U}_{[a+3,b]}\ Ready(s)])])])$ TBA semantically equivalent: 
(c) Local update by $Act\_Control$(LUAC)	Formula: $\phi_{LUAC} := \mathsf{AG}_{[a,b]}(SndMsgR(s) \rightarrow \mathsf{A}[Rcv1(s)\ \mathsf{U}_{[a+1,b-2]}\ (ConfL(s) \wedge \mathsf{A}[ConfL(s)\ \mathsf{U}_{[a+2,b-1]}\ (AckMsgL(s) \wedge \mathsf{A}[AckMsgL(s)\ \mathsf{U}_{[a+3,b]}\ RcvConfL(s)])])])$ TBA semantically equivalent: 
(d) Local response by $Message\_Manager$ (LUMM)	Formula: $\phi_{LUMM} := \mathsf{AG}_{[a,b]}(Rcv2(s) \rightarrow \mathsf{A}[LocUp(s)\ \mathsf{U}_{[a+1,b-1]}\ (Upd(s) \wedge \mathsf{A}[Upd(s)\ \mathsf{U}_{[a+2,b]}\ ConfL(s)])])$ TBA semantically equivalent: 

by rhomboids named, 'Update and Send Messages', 'Receive a Message', 'Send an Acknowledgement' and 'Receive All Confirmations'.

4.1 Properties Specification

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 ϕ, 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, $\phi_{Act\_Control} = \phi_{RUAC} \wedge \phi_{LUAC}$ and $\phi_{Message\_Manager} = \phi_{RUMM} \wedge \phi_{LUMM}$, respectively. Since the DDBM protocol model is conformed by n replicas of the same component (i.e., DDBM = $\|_{i:1..n}\ d_i$), the in-

variant ψ_i that each component d_i must satisfy is the conjunction of the properties of the n replicas, but without itself, i.e., $\psi_i = \|_{j:1..n}\ \phi_j | j \neq i$, we need not include the invariants ψ_i as part of the verification process. Our method, at this stage, needs only to address verification of local properties ϕ_i. From the practical viewpoint, if we included invariants ψ_i in the verification process, we would be double–checking the satisfaction of property ϕ_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 fulfilment 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 behaviour 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:

$$T(\phi_{LUAC}) \sqsubseteq_T T(AC)\ ,\ T(\phi_{RUAC}) \sqsubseteq_T T(AC) \tag{4}$$

$$T(\phi_{LUAC}) \sqsubseteq_F T(AC)\ ,\ T(\phi_{RUAC}) \sqsubseteq_F T(AC) \tag{5}$$

$$T(\phi_{LUMM}) \sqsubseteq_T T(MM)\ ,\ T(\phi_{RUMM}) \sqsubseteq_T T(MM) \tag{6}$$

$$T(\phi_{LUMM}) \sqsubseteq_F T(MM)\ ,\ T(\phi_{RUMM}) \sqsubseteq_F T(MM) \tag{7}$$

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 behaviour 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:

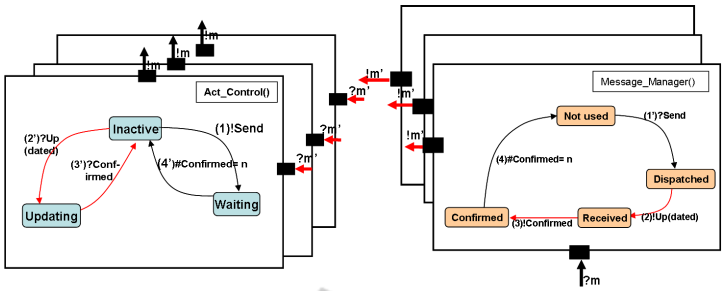


Fig. 3. DBM Composite Structure Communications.

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. 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 $\mathcal{L}(Act\_Control)$ and $\mathcal{L}(Message\_Manager)$ 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 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 $Act\_Control$ and $Message\_Manager$ system components are therefore "composable",

$$Act\_Control \| Message\_Manager \models \phi_{Act\_Control} \wedge \phi_{Message\_Manager} \tag{8}$$

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

$$d_i = Act\_Control \| Message\_Manager \text{ and } \phi_{d_i} = \phi_{Act\_Control} \wedge \phi_{Message\_Manager}, \tag{9}$$

And hence, the entire system's model represented by each replicated DB manager d_i satisfies the property ϕ_{d_i} that represents every manager's behaviour,

$$d_i \models \phi_{d_i}. \tag{10}$$

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

1. A. M. Ben Amram, S. Genaim, and A. N. Masud. On the termination of integer loops. In In: Viktor Kumcak and Andy Rybalchenko editors, *Verification, Model–Checking and Abstract Interpretation*, Lecture Notes in Computer Science (to appear), Springer–Verlag, D, 2012.
2. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model–checking. Advances in Computers, 58:117–148, 2003.
3. T. Bultan, J. Fischer, , and R. Gerber. Compositional verification by model checking for counter–examples. In ISSTA '96: Proc. of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 224–238, New York, USA, 1996. ACM Press.
4. E. Clarke, D. Long, and K. McMillan. Compositional model checking. In Proc. of the Fourth Annual Symposium on Logic in Computer Science, pages 353–362, Piscataway, USA, June 1989. IEEE Press.
5. J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. LNCS, 2619(0):331–346, 2003.
6. O. Grumberg and D. E. Long. Model checking and modular verification. ACM TOPLAS, 16(3):843–871, 1994.
7. B. Lukoschus. Compositional Verification of Industrial Control Systems: Methods and Case Studies. PhD thesis, Universitaet zu Kiel, Technischen Fakultaet der Christian-Albrechts, July 2005.
8. W. Wong and M. Young. Compositionality reachability analysis using process algebra. In Proc. of the Symposium on Testing, Analysis, and Verification: TAV4, pages 49–59, New York, USA, 1991. ACM Press.
9. Luis E. Mendoza Morales and Manuel I. Capel. Automatic compositional verification of business processes. Enterprise Information Systems, LNBIP, 24:479–490, 2009.
10. Luis Eduardo Mendoza, Manuel I. Capel, and María A. Pérez. Conceptual framework for business processes compositional verification. Information & Software Technology, 54(2):149–161, 2012.
11. Y. Kesten, A. Klein, A. Pnueli, and G. Raanan. A perfecto verification: Combining model checking with deductive analysis to verify real–life software. LNCS, 1708:173–194, 1999.
12. H. Giese, M. Tichy, S. Burmester, and S. Flake. Towards the compositional verification of real–time UML designs. In ESEC/FSE–11: Proc. 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 38–47, New York, USA, 2003. ACM Press.
13. G. Frehse, O. Stursberg, S. Engell, R. Huuck, and B. Lukoschus. Modular analysis of discrete controllers for distributed hybrid systems. In The XV IFAC World Congress, pages 21–26, Barcelona, Spain, 2002. IFAC.
14. C. de la Riva and J. Tuya. Automatic generation of assumptions for modular verification of software specifications. Journal of Systems and Software, 79(9):1324–1340, 2006.

15. N. Moffat and M. Goldsmith. Assumption—commitment support for CSP model checking. Journal of Automated Reasoning, 41(3-4):365–398, 2008.
16. H. Wehrheim and D. Wonisch. Compositional CSP traces refinement checking. Electronic Notes in Theoretical Computer Science, 250(2):135–151, 2009.
17. B. Metzler, H. Wehrheim, and D. Wonisch. Decomposition for compositional verification. In Proceedings of the 10th International Conference on Formal Methods and Software Engineering, ICFEM '08, pages 105–125, Heidelberg, Germany, 2008. Springer-Verlag.
18. C. A. R. Hoare. Communicating Sequential Processes. International Series in Computer Science. Prentice–Hall International Ltd., Hertfordshire UK, 1985.
19. J. Zic. Time–constrained buffer specifications in CSP+T and Timed CSP. ACM TOPLAS, 16(6):1661–1674, 1994.
20. J. Ruf and T. Kropf. Symbolic model checking for a discrete clocked temporal logic with intervals. In Proc. of the IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods, pages 146–163, 1997.
21. Kawtar Benghazi Akhlaki, Manuel I. Capel-Tuñón, Juan Antonio Holgado Terriza, and Luis E. Mendoza Morales. A methodological approach to the formal specification of real-time systems by transformation of uml-rt design models. Science of Computer Programming, 65(1):41–56, 2007.
22. A. Rabinovich. On compositionality and its limitations. ACM TOCL, 8(1):1–26, 2007.
23. M. Abadi and L. Lamport. Conjoining specifications. ACM TOPLAS, 17(3):507–535, 1995.
24. K. Jansen. Coloured Petri Nets. Springer-Verlag Inc., New York, USA, 1997.
25. FormalSystemsEuropeLtd. Failures–Divergence Refinement – FDR2 User Manual. Formal Systems Europe Ltd., Oxford, 2005.
26. S. A. Schneider. Concurrent and Real–Time Systems – The CSP Approach. John Wiley & Sons, Ltd., 2000.