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

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

[1]*Department of Software Engineering, University of Granada, Informatics & Telecommunications, 18071 Granada, Spain*

[2]*Processes 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 FCVA 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 *dependability*[1]. 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-

[1]See IEC IEV 191-02-03, IFIP 10.4 Working Group on Dependable Computing and Fault Tolerance.

I. Capel M. and E. Mendoza Morales L..
A Formal Compositional Verification Approach for Safety–Critical Systems Correctness - Model–Checking based Methodological Approach to Automatically Verify Safety Critical Systems Software.
DOI: 10.5220/0004003801050112
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 105-112
ISBN: 978-989-8565-11-2

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.

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

CSP+T Syntax Rules

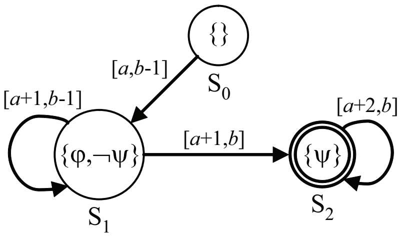


Figure 1: CCTL formula.

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.

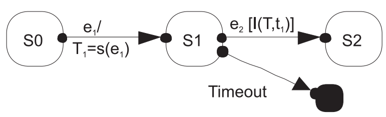
A CSP+T process term $\mathcal{P}$ is defined as a tuple $(\alpha P, P)$, where $\alpha P = Comm\_act(P) \cup Interface(P)$ is called the *communication alphabet* of P. These communications represent the events that process P receives from its *environment* or those that occur internally. CSP+T is a 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.

The event enabling interval $\mathrm{I}(T, t_a) = \{t \in \mathcal{T} \mid rel(t_a, v) \leq t \leq rel(t_a + T, v)\}$ 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 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 (Ruf 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 *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 "LTL-like" temporal op-

Table 1: Example of a map rule from UML–TSM to CSP+T terms.

UML–TSM	Description
	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 $[T1, T1+T]$ (*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) $\frac{S1=e_1 \bowtie t_1 \rightarrow S2}{t_1=s(e_1);\ S2} \begin{pmatrix} S1,S2 \in states; \\ s(e_1) \end{pmatrix}$ 2. e_2 occurrence) $\frac{S2=\mathrm{I}(T,t_1).e_2 \rightarrow S3}{s(e_2);\ S3} \begin{pmatrix} s(e_2) \in [t_1, t_1+T]; \\ S2,S3 \in states \end{pmatrix}$ OR $\mathrm{I}(T,t_1)$ timeout) $\frac{S2=\mathrm{I}(T,t_1) \rightarrow Timeout \rightarrow SKIP}{s(\tau);\ Timeout \rightarrow SKIP} \begin{pmatrix} s(\tau) < t_1+T; S2 \in states; \\ Timeout \in pseudostates \end{pmatrix}$ *Timeout* execution step) $\frac{Timeout \rightarrow SKIP}{s(\tau);\ SKIP} \begin{pmatrix} s(\tau) = t_1+T; \\ Timeout \in pseudostates \end{pmatrix}$	

erators 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 formula ϕ, expressed as a Büchi automaton in Figure 1, is therefore:

$$\phi = \varphi \mathrm{U}_{[a,b]} \psi \ . \qquad (1)$$

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:

$$event/communication/execution\,step)\ \frac{premises}{conclusion}\ (conditions)$$

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, 2007; de Roever et al., 2001) 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 a 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).

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

3.1 Compositional Verification of a Concurrent System

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, 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 (Abadi and Lamport, 1995), to verify the property ϕ_i of component C_i we need to assume some kind of behaviour of the other components (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:*

$$TSM(\mathbb{C}) \vDash (\phi \wedge \psi \wedge \neg\delta) \Leftrightarrow \mathop{\Big\|}_{i:1..n} TSM(C_i) \vDash \bigwedge_{i:1..n} (\phi_i \wedge \psi_i) \wedge \neg\delta, \tag{2}$$

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

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 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.

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 $T(\mathbb{C})$ 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[2] 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 = $\|_{i:1..n}\, d_i$), the invariant ψ_i that each component d_i must satisfy is the conjunction of the replicas properties, but without itself, i.e., $\psi_i = \|_{j:1..n}\, \phi_j | j \neq i$. Thus, at this stage, we only need to address the verification of local properties ϕ_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 interval 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.

[2]Base Transceiver Stations

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, $T(\phi_{RUAC})$, $T(\phi_{RUMM})$, $T(\phi_{LUAC})$, $T(\phi_{LUMM})$ (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:

$$
\begin{array}{rcl}
T(\phi_{LUAC}) \sqsubseteq_T T(AC) & , & T(\phi_{RUAC}) \sqsubseteq_T T(AC) \\
T(\phi_{LUAC}) \sqsubseteq_F T(AC) & , & T(\phi_{RUAC}) \sqsubseteq_F T(AC) \\
T(\phi_{LUMM}) \sqsubseteq_T T(MM) & , & T(\phi_{RUMM}) \sqsubseteq_T T(MM) \\
T(\phi_{LUMM}) \sqsubseteq_F T(MM) & , & T(\phi_{RUMM}) \sqsubseteq_F T(MM)
\end{array}
$$

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 $T(\phi)$ (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 $\mathcal{L}(Act\_Control)$ 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

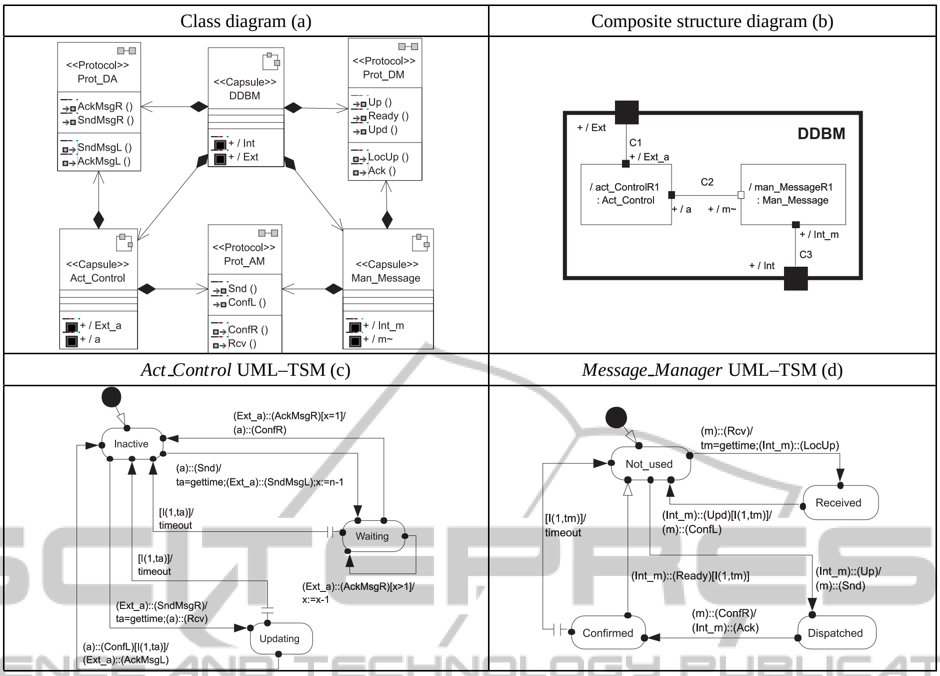


Figure 2: Software Architecture of the *DDBM* Model with MEDISTAM–RT.

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 BTRANSFORMER Tool

The main objective of BTRANSFORMER 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 BTRANSFORMER Properties

Developed using Open Unified Process (OpenUP) methodology (Eclipse.org), BTRANSFORMER 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

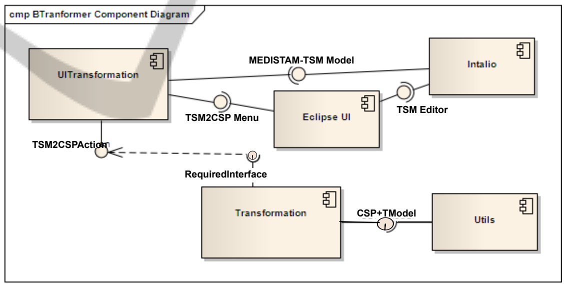


Figure 3: BTRANSFORMER tool components.

for creating and editing UML–TSM to CPS+T transformation rules.

All the plugins needed to implement BTRANSFORMER (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 BTRANSFORMER were in-

Table 3: Results of preliminary tests.

Model	Completeness of elements	Number of processes	Completeness of relation	Behavioural safety
Transport Chain	Yes	12	Yes	Yes
Logistic process in hospitals	Yes	22	Yes	Yes

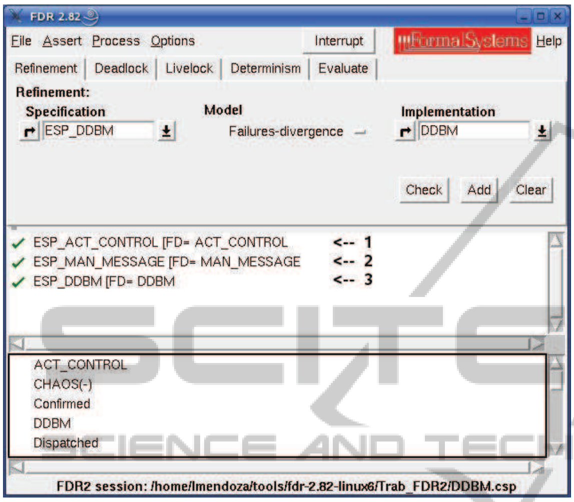


Figure 4: FDR2 screenshot.

put (Figure 4) to FDR2 model–checking tool (Formal Systems Europe Ltd., 2005) to check semantic inconsistencies in syntactical process terms. The cases used to assess the operation of the tool were Transport chain (Koniewski et al., 2006) and Logistics process in hospitals (Baacke et al., 2009). In addition, the following criteria were defined in order to detect inconsistencies between source (MEDISTAM–RT) and target (CSP+T) models.

a. Completeness of elements. All elements of source diagrams appear reflected in their semantically equivalent specifications in CSP+T.

b. Number of processes. There is at least one CSP+T process for each activity defined in the diagram.

c. Completeness of relations. It is possible to establish a relation between two processes as long as they present a previous relation between two or more activities in the model.

d. Behavioral safety. Thus, it has been made sure that the CSP+T model does not "invent" execution sequences which do not exist in the source model.

According to the results of Table 3, we can affirm that in both cases the generated specification was complete, and met the minimum expected of process definitions. Relations were also reflected in the specification and behavioral safety is preserved.

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

Abadi, M. and Lamport, L. (1995). Conjoining specifications. *ACM TOPLAS*, 17(3):507–535.

Baacke, L., Mettler, L., and P.Rohner (2009). Component–based process modelling in health care. *17th Europ. Conference on Information Systems*, 1(a):507–535.

Benghazi, K., Capel, M., Holgado, J., and Mendoza, L. (2007). 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.

Clarke, E., Grumberg, O., and Peled, D. (2000). *Model Checking*. The MIT Press, Cambridge, USA.

de Roever, W., de Boer, F., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., and Zwiers, J. (2001). *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*, volume 54 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK.

Formal Systems Europe Ltd. (2005). *Failures–Divergence Refinement – FDR2 User Manual*. Formal Systems Europe Ltd., Oxford.

Giese, H., Tichy, M., Burmester, S., and Flake, S. (2003). 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. ACM Press.

Hooman, J. (1991). *Specification and Compositional Verification of Real–Time Systems*, volume 558 of *LNCS*. Springer–Verlag, Berlin, Germany.

Jansen, K. (1997). *Coloured Petri Nets*. Springer–Verlag Inc., New York, USA.

Koniewski, R., Dzielinski, A., and Amborski, A. (2006). Use of petri nets and business processes management notation in modelling and simulation of multimodal logistics chains. In *20th European Conference on Modelling and Simulation*, Barcelona, Spain.

Mendoza, L. and Capel, M. (2009). Automatic compositional verification of business processes. *Enterprise Information Systems, LNBIP*, 24:479–490.

Mendoza, L., Capel, M., and Pérez, M. (2012). Conceptual framework for business processes compositional verification. *Information & Software Technology*, 54(2):149–161.

Rabinovich, A. (2007). On compositionality and its limitations. *ACM TOCL*, 8(1):1–26.

Ruf, J. and Kropf, T. (1997). 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.

Schneider, S. (2000). *Concurrent and Real–Time Systems – The CSP Approach*. John Wiley & Sons, Ltd.

Wong, P. and Gibbons, J. (2009). A relative timed semantics for bpmn. *Electronic Notes in Theoretical Computer Science*, 229(2).

Zic, J. (1994). Time–constrained buffer specifications in CSP+T and Timed CSP. *ACM TOPLAS*, 16(6):1661–1674.