redin Revista Facultad de Ingeniería, Universidad de Antioquia, No. 83, pp. 92-101, 2017

Architectural capability analysis using a model-checking technique

Análisis de Capacidades en arquitecturas utilizando técnicas de evaluación de modelos

Darío José Delgado-Quintero[1*], Jormary Noguera-Muños[2], Gerson Alonso Flores-Rojas[2], Clara Isabel López-Gualdron[3], Ricardo Llamosa-Villalba[3]

[1] Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones, Facultad de Ingenierías Físicomecánicas, Universidad Industrial de Santander. Carrera 27, Calle 9. C. P. 680002. Bucaramanga, Colombia.
[2] Escuela de Ingeniería de Sistemas, Facultad de Ingenierías Físicomecánicas, Universidad Industrial de Santander. Carrera 27, Calle 9. C. P. 680002. Bucaramanga, Colombia.
[3] Escuela de Diseño Industrial, Facultad de Ingenierías Físicomecánicas, Universidad Industrial de Santander. Carrera 27, Calle 9. C. P. 680002. Bucaramanga, Colombia.

ARTICLE INFO
Received October 11, 2016
Accepted April 04, 2017

KEYWORDS
Capability analysis, enterprise architectures, DoDAF, model-checking, requirements, TOGAF

Análisis de capacidades, arquitecturas empresariales, DoDAF, evaluación de modelos, requerimientos, TOGAF

ABSTRACT: This paper describes a mathematical approach based on a model-checking technique to analyze capabilities in enterprise architectures developed by using DoDAF and TOGAF architecture frameworks. Such approach base is the requirements' validation related to the enterprise capabilities by employing operational or business artifacts associated with the dynamic behavior processes. We show how this approach can be used to quantitatively verify if the operational models in an enterprise architecture can achieve the enterprise capabilities by using a case study connected to a capability integration problem.

RESUMEN: Este trabajo describe un enfoque matemático basado en una técnica de validación de modelos para analizar capacidades en arquitecturas empresariales construidas utilizando los marcos arquitecturales DoDAF y TOGAF. La base de este enfoque es la validación de requerimientos relacionados con las capacidades empresariales empleando artefactos arquitecturales operacionales o de negocio asociados con el comportamiento dinámico de los procesos. Se muestra cómo este enfoque puede ser utilizado para verificar, de forma cuantitativa, si los modelos operacionales en una arquitectura empresarial pueden satisfacer las capacidades empresariales. Para ello, se utiliza un estudio de caso relacionado con un problema de integración de capacidades.

1. Introduction

In systems designing, early fault detection is one of the biggest challenges due to software crises. In Software, fault detection is less than 10% in the conceptual design phase and around 40% of the failures are introduced in this phase. Thus compared with the cost to fix a fault during the design phase, a fault in an operational or testing phase is more expensive to fix [1, 2]. This is especially true during most of the systems development procedures; fault detection during the design phase usually consists of models' logical verification (UML, SysML, BPMN, etc.) and the requirement compliance in the design models only [2-5]. However, logical verification of models only ensures the fault detection associated with the stakeholders needs, but that approach cannot ensure that the system could achieve its behavioral goals. There exist other approaches to reduce the fault insertion by goal-oriented requirement engineering (GORE) [5]. Those approaches try to ensure that elicitation, analysis, elaboration and refinement, specification and modeling of requirements are related to the specified goals for a particular solution. As a result, this article is focused on fault-detection confirmation approaches by using logical verification of models that seek to ensure the achievement of their behavioral goals.

ISO/IEC/IEEE 24765 [6] (System and software engineering - Vocabulary) defines a requirement as "a condition or capability needed by a user to solve a problem or achieve an objective". In an architectural environment, business requirements support the business capabilities designing, architectural models– especially in the business or operational views– represent those capabilities, and the capabilities designs seek to satisfy the requirements and

* Corresponding author: Darío José Delgado Quintero
e-mail: dario.delgado@correo.uis.edu.co
ISSN 0120-6230
e-ISSN 2422-2844

DOI: 10.17533/udea.redin.n83a12

achieve the organizational goals [7, 8]. An organizational capability represents skills that organizations have that can create value [9]. This paper shows an approach based on a model-checking methodology [1] to verify models in terms of capabilities. This approach seeks to detect faults related to the organizational goals.

In order to implement the capability model-checking approach, a Custom Implants Design System (CIDS) [10] is employed with an architecture implementation that uses DoDAF and TOGAF architectural frameworks [11, 12]. To describe the approach, this document is organized as follows: the first part describes the model-checking technique, the second part shows the approach for capability model-checking. Finally, the analysis and conclusions are presented. All the parts contain an application example by including the CIDS system.

2. Model-checking as a fault detection technique

The model-checking (MC) technique proposed by Clarke & Emerson [13] and Queille & Sifakis [14] is an automatic technique for finite state systems verification regarding some temporary logical specifications. The standard approach focuses on the semi-formal models: SysML (System modeling language), UML (Unified Modeling Language), BPMN (Business Process Model and Notation), etc. Especially those that describe the system behavior with behavioral models to develop formal ones (executable) aim at verifying dynamic aspects of the systems and detecting failures through a model-checking tool. See Figure 1.

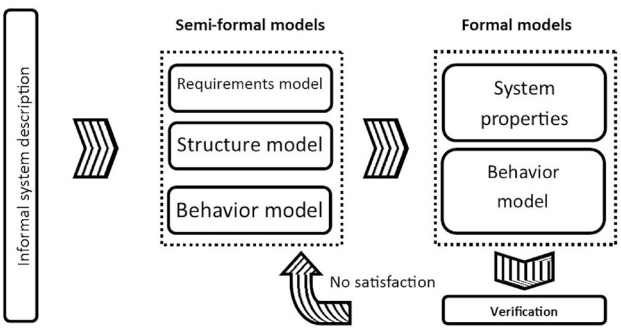

Figure 1 Requirements validation general scheme using model-checking

Failure detection using MC follows the next steps:

1. Convert the system behavioral models into formal models (LTS-Labeled Transition Systems).
2. Specify the system requirements in terms of logic temporal specifications (LTL).
3. Use an MC tool to execute the LTL and determine compliance.

2.1 Behavioral models and Kripke structures

Systems architectures are developed to work as a bridge between requirements and design in complex systems [15] by using an ADM (Architecture Development Method) cycle [16] that describes the system organization in term of its components and interactions. SAs divide the analyses into structural component and functional component analyses, which are both related, but conduct different activities. See Figure 2.

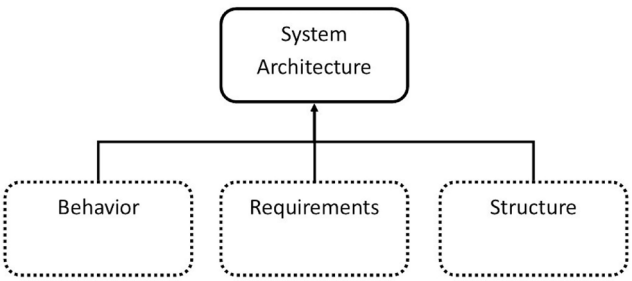

Figure 2 A generic scheme for systems architecture

Architectural models are developed by using modeling languages such as UML, SysML, BPMN, among others, that display different classifications according to their viewpoints. However, the architectural viewpoints, in general, are classified in terms of behavior, requirements, and structure. For example, SysML [17] architectural model classification is particularly considered in this paper (see Figure 3). Regarding other architectural frameworks as TOGAF or DoDAF, the possible models have a different classification. TOGAF, for instance, categorizes them as business, information and systems, and technology views. In contrast, DoDAF categorizes the architecture as capability, operational, services, and systems views.

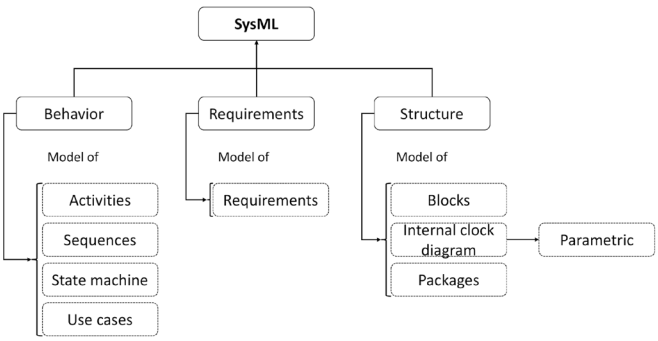

Figure 3 Possible architectural models using SysML

For the purpose of this paper, two AF (Architectural Frameworks), TOGAF and DoDAF are mixed in order to

standardize a set of capability models that DoDAF includes combined with the implementation method that TOGAF provides, namely, ADM [16]. Another reason for this blend is that DoDAF, in comparison with other AF such as Zachman, 4+1 Views, FEAF, among others, do not support a capability model design approach directly. Hence DoDAF and TOGAF terminology are both used to carry out this paper's procedure: a capability analysis in the architecture through the requirement model, capability models (CV), and the business views represented by the operational models (OV) in the architecture.

The proper system behavior is connected to the requirement fulfillment stipulated by the stakeholders. The MC techniques seek to verify the requirement fulfillment using the behavior models (see Figure 4). For this purpose, it uses state machine (SM) diagrams. SM is a useful graphical tool to describe the dynamic system behavior from an architectural point of view. The OV models represent the system behavior; OV-6b (State Transition Description) is an example.

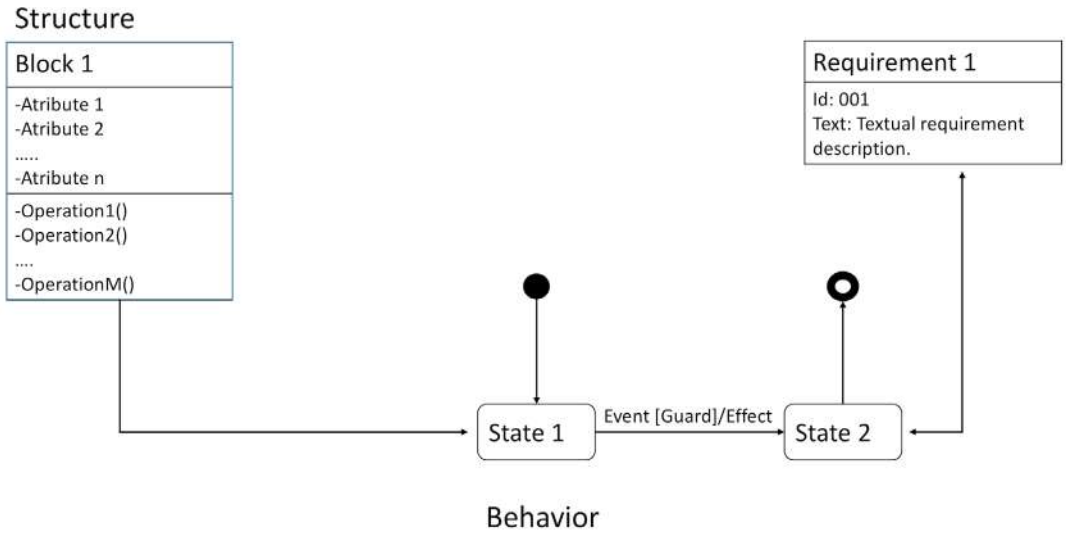

Figure 4 Structure, behavior, and design requirements

2.2 Case study

A Custom Implants Design System (CIDS) [9] is presented here. It uses some software packages: BioCAD, CAD, and CAE to model digital volumes and verify by simulation, as well as a 3D printer through rapid prototyping (RP) of Osteosynthesis implants. See Figure 5.

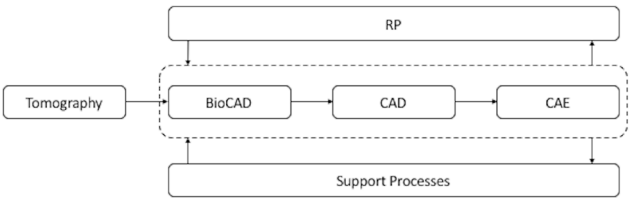

Figure 5 General perspective to the custom implants design system (CIDS)

From the system architectural point of view, the OV-6b model is taken as a sample to analyze the system behavior, see Figure 6.

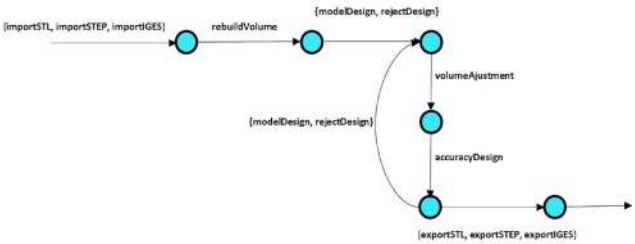

Figure 6 LTS fragment, CAD tool states in the CIDS system

OV-6b is a Statechart (SC) diagram used to describe the detailed sequencing of activities or work flow in the business process. It is useful for describing critical sequencing of behaviors and timing of operational activities that cannot be adequately described in the OV-5b Operational Activity Model. It could be defined as a tuple: $SC=\{S,Var,G,E,Edges\}$, where S is the states set, E is the set of events, Var is the state variables set, G is the guards set, and $Edges$ is the transitions set.

Although UML Statechart diagrams count on their own semantics and syntax, a system behavior graphic is represented here. The SC diagrams need to be transformed into an executable model with a mathematical formality.

Transition Systems (TS) are used here. They consist of a set of states and transitions such as labels denoted by actions and one initial state, to represent the behavior and communication between components in a concurrent system (see Figure 7). TS is a tuple $(S,Act,\rightarrow,I,AP,L)$:

1) S is a set of states,
2) Act is a set of actions
3) $\rightarrow \subseteq S \times Act \times S$ is a transition relation,
4) $I \subseteq S$ is a set of initial states,
5) AP is a set of atomic propositions, and
6) $L: S \rightarrow 2^{AP}$ is a labeling function.

Statechart diagrams and transition systems models have common elements: set of states S, events E, guards G, transition *Edges* represented in TS as the actions Act, and the transition relations → In contrast, The SC diagram does not contain the atomic propositions AP nor the labeling function L. The AP are defined as hidden actions in the system related to the requirements. See Table 1.

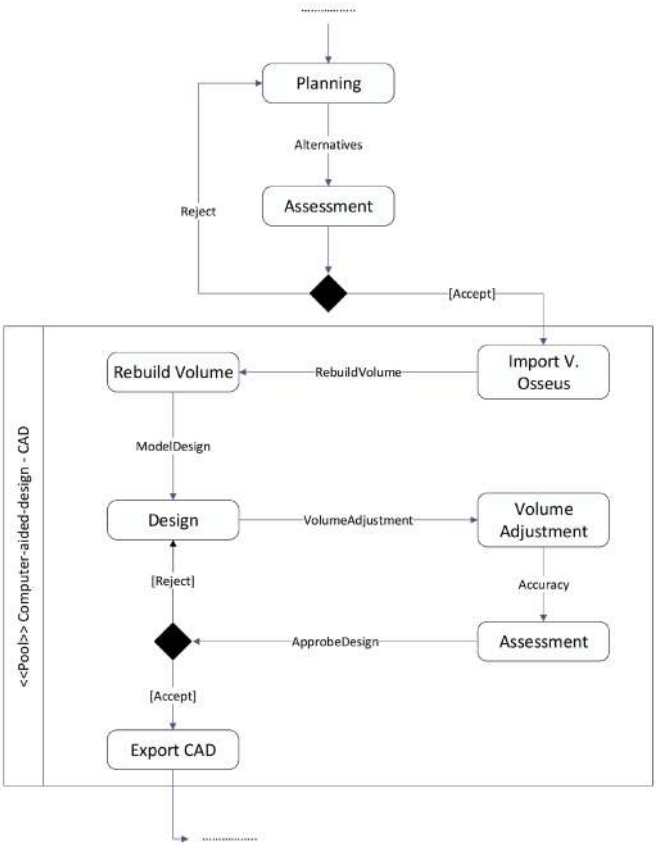

Figure 7 OV-6b model fragment, Statechart diagram fragment for CAD tool in the CIDS system

Table 1 Requirements list for the CAD tool obtained from the requirements model and atomic propositions obtained from OV-6a model

Code	Requirement	Atomic proposition
RCd1	CAD process must design orthopedic custom devices proposed by the specialist in each case study.	
RCd1.1	CAD should allow to import 3D bone models in IGES, STEP, or STL formats obtained from a BioCAD tool.	importSTL importSTEP importIGES alternatives
RCd1.2	CAD should allow to export 3D volumes in IGES, STEP, or STL format.	exportSTL exportSTEP exportIGES accuracyDesing
RCd1.3	CAD should save a historical record of 3D models reconstruction operations, allowing regression.	rebuildVolume importSTL importSTEP importIGES
RCd1.4	CAD should allow parameters edition in model construction operations.	modelDesign regectDesign rebuildVolume
RCd1.5	CAD should develop technical blueprints by using standard rules.	modelDesign rebuildVolume

The labeling function L creates a relation between a state $s_i \in S$ and a proposition $p_j \in AP$ (see Table 2). The AP and L are used to create a relation between system requirements and behavior models. Then, before transforming the SC diagram into a TS, the system requirements need to be represented in labels. In turn, they represent specific actions obtained from the $OV - 6a$ model (Operational rules model), which is an activity diagram that contains the detailed set of operations, activities, and guards that specify each state and satisfy each requirement. In summary, labeled function L allows to execute the TS in system requirements terms.

Table 2 L function deployment

State	L(State)
Import V. Osseous	{importSTL, importSTEP, importIGES}
Rebuild Volume	{rebuildVolume}
Design	{ modelDesign, regectDesign}
Volume Adjustment	{volumeAjustment}
Assessment	{accuracyDesing}
Export CAD	{exportSTL, exportSTEP, exportIGES}

2.3. Linear Temporal Logic Specifications (LTLs)

A TS execution is a sequence of actions (2^{AP}) that can be executed starting with an initial state I. A $TS=(S,Act,\rightarrow, I,AP,L)$ transits $TS'=(S,Act,\rightarrow,I',AP,L)$ by an action a denoted as $TS \xrightarrow{a} TS' \leftrightarrow (I,a,I') \in \rightarrow$. The L function in the LTS $ts=(S,Act,\rightarrow,I)$ takes a set of AP that come from activities related to system requirements and transform the Labeled Transition System LTS in $TS=(S,Act,\rightarrow, I,AP,L)$. Here the system behavior is labeled with a set of actions that are directly related to requirements. Such system behavior can be evaluated with the TS execution, specifically, L function.

An LTL formula is a mathematical language for describing linear-time properties; it is a widely used logic for expressing properties of programs viewed as sets of executions, and it is inductively defined by using Boolean and temporal operators × (next) and U (until), given an AP:

1) $p_i \in AP$ is a formula

2) If φ and ψ are formulas, then $\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi, X\varphi, \varphi U \psi$ are also.

An LTL formula interpretation is an infinite word $w = x_0\, x_1\, x_2 \cdots$ over the set 2^{AP}. Then for each time instant in the system execution, there is a subset of AP that is active. Being w_i the word w starting with x_i that comes from the TS execution, the LTL semantics is defined as follow. See Figure 8.

1) $w \vDash p \; if \; p \in x_0$, $To \;\; p \in AP$
2) $w \vDash \varphi \vee \psi \; if \; (w \vDash \varphi) \vee (w \vDash \psi)$
3) $w \vDash \neg\varphi \; if \; \neg(w \vDash \varphi)$
4) $w \vDash \varphi \wedge \psi \; if \; (w \vDash \varphi) \wedge (w \vDash \psi)$
5) $w \vDash X\varphi \; if \; w_i \vDash \varphi$
6) $w \vDash \varphi U \psi \; if \; \exists\, i \geq 0 \mid (w_i \vDash \psi) \wedge \forall\, 0 \leq j \leq i, \; (w_i \vDash \varphi)$

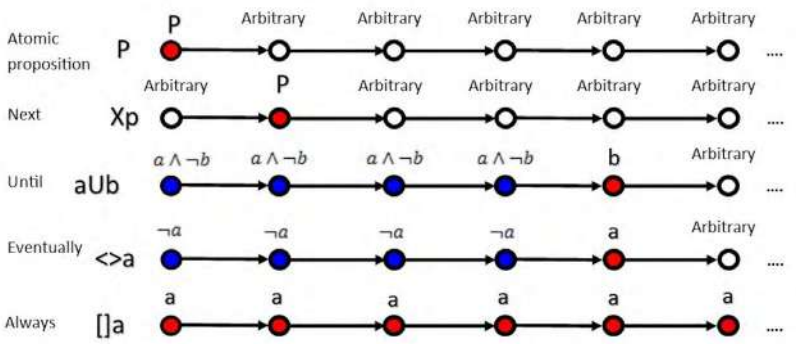

Figure 8 Intuitive view for a LTL

There are additional operators proposed by Giannakopoulou in [15]: *"true* $\equiv \varphi \vee \neg\varphi$*", "false* $\equiv \neg true$*"*, the Boolean operator $\Longrightarrow$ that is defined as $\varphi \Longrightarrow \psi \equiv \neg\varphi \vee \psi$, and the temporal operators <> (Eventually), [] (Always), W (Weak until) defined in terms of the main temporal operators: $<> \varphi \equiv true\, U\varphi$, $[]\varphi \equiv \neg F \neg\varphi$, y $\varphi W \psi \equiv ((\varphi U \psi) \vee G\varphi)$.

2.4. Use of transition systems to verify LTLs formulas

The LTL verification basic scheme is based on the use of Büchi automatons (BA) [18, 19]. A BA is a 5-tuple $B = <Q, \Sigma, \delta, q_0, F>$, where Q is a finite set of states, Σ is a finite set of labels, $\delta \subseteq Q \times \Sigma \times Q$ is a labeled transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepted states.

$A\ B$ execution in a finite word $<a_0\, a_1\, a_2 \cdots>$ over Σ is an infinite word $<s_0\, s_1\, s_2 \cdots>$ over Q, *so that* $s_0 = q_0$ and $\forall\, i \in N$, $(s_i, a_i, s_{i+1}) \in \delta$. If any elements in F occur infinitely, an execution is accepted and an infinite word w over Σ is accepted by the automaton B if there are any executions from B in w.

For an LTL formula over a set of AP, we formulated a BA to accept words over 2^{AP} that *satisfy*φ. A finite state system

is verified by an LTL specification φ by calculating the intersection between the system and the BA ($\neg\varphi$). The TS satisfy φ if the intersection does not accept words.

LTL functions in AP terms is designed here to try to verify the requirements in the systems behavior models. The set of operators used are shown in Table 3.

Table 3 Operators to formulate LTL formulas

Unitary operators	Binary operators
[] Always (G)	U Until
< > Eventually (F)	W Weak until
X Next	&& Logic operator AND
! Logic negation	|| Logic operator OR
	Implicación
	Equivalencia

An LTL formula F is designed for each requirement. A model-checking tool is used to verify φ (see Table 4). An LTSA analyzer is employed here. When a requirement is validated by using an LTL formula, one of these three different results is obtained:

1) The property (requirement) is fulfilled.

2) The property (requirement) is not fulfilled and a counter example is shown.

3) The test generates an inconclusive result.

Table 4 LTL formulas for the CAD requirements

Requirement	LTL formula
RCd1.1	assert Rcd11= <>((alternatives)->(importSTL || importIGES || importSTEP))
RCd1.2	<>((accuracyDesing)->((exportSTL || exportSTEP || exportGES) W (importSTL || importIGES || importSTEP)))
RCd1.3	<>((importSTL || importIGES || importSTEP) -> (rebuildVolume))
RCd1.4	(<>(rebuildVolume ->(({ modelDesign W regectDesign)))-> modelDesign)
RCd1.5	<>(rebuildVolume ->(modelDesign))

2.5. The problem of requirement verification

Requirement elicitation is the most effective phase of systems development processes. Such elicitation aims to correctly meet the stakeholders' requirements [20] so that the systems models design is allowed. However, as we can see in the Figure 9, a simple requirement verification in the systems models can only ensure that the stakeholders' needs are fulfilled, but it cannot ensure that the stakeholders' intentions are achieved with designed models.

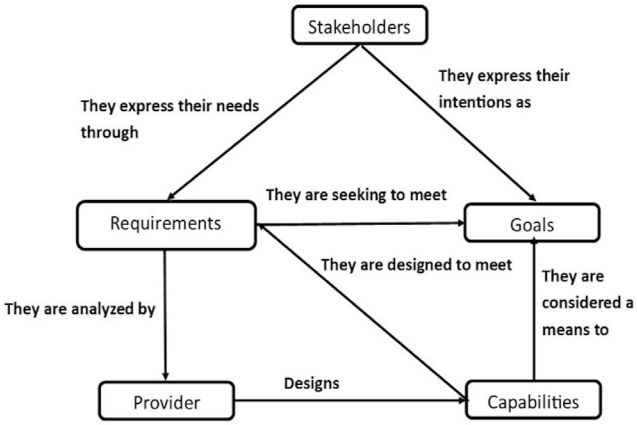

Figure 9 Relationship between requirements and capabilities [21]

3. Capability analysis by using a model-checking scheme

A system capability represents the ability to perform certain actions or outcomes through a set of controllable and measurable faculties, features, functions, processes, or services. From an architectural point of view, we model capabilities to meet the stakeholder requirements and organizational goals. In order to meet a set of requirements: $R=\{r_1, r_2,\cdots, r_m\}$, a set of capabilities $C=\{c_1, c_2, ..., c_n\}$ is designed. In the architecture, capabilities C are represented as a set of behavioral models that deploy them in terms of operational activities.

In this article, a capability verification approach is proposed by using a model-checking scheme. Such approach is based on the next assumptions:

1) A designed capability tries to meet one or many requirements.

2) A requirement is a condition or capability needed by a user to solve a problem or achieve an objective.

3) Capabilities provide an environment to meet the organizational goals.

In short, these are the general steps to carry out the capability analysis (see Figure 10):
1) Relate capabilities and requirements.
2) Validate the requirements by using a model-checking approach.
3) Analyze and categorize results in capabilities terms.

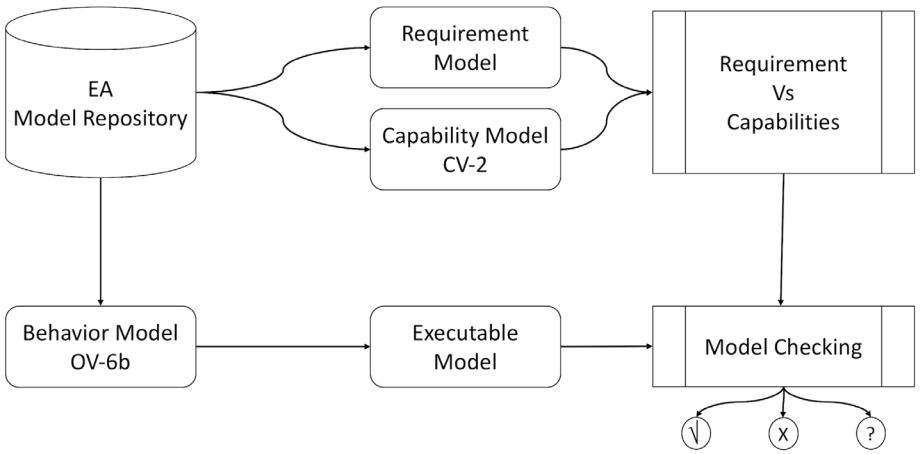

Figure 10 Methodological scheme for capability analysis by using a model-checking technique

3.1. Capabilities and requirements

The first assumption previously mentioned states that a capability is designed to meet one or many requirements. Nevertheless, a capability is a formal system component and a requirement is an informal stakeholder need (see Table 1 and Table 5).

An efficient and organized technique to translate the "Stakeholder voice" into the "Engineering voice" or designers is the process of quality function deploy (QFD) [22]. It is a useful tool to associate qualitative and ambiguous requirements with systems attributes. For the purpose of this article, an adaptation of QFD and the house of quality HOQ have been employed to relate the requirements to specific properties in the system.

Table 5 Capabilities list obtained from the CV-2 model

Code	Capability
C1	Planning design
C2	Share information
C3	Knowledge acquisition
C4	Knowledge Transfer
C5	Product evaluation
C6	Technology appropriation
C7	Digital reconstruction
C8	Implant modeling
C9	Structural simulation
C10	Surgical simulation
C11	Interoperability
C12	Prototyping

The properties of interest are the system capabilities. The HOQ methodology denotes the relations between requirements and capabilities as strong ⓪ (9), average ○ (3), and weak Δ (1). The relation between capabilities is defined as positive + and negative – correlations. As the name implies, a positive correlation means that a positive capability achievement has a positive impact over the capability related. A negative correlation, on the other hand, means that a capability positive achievement has a negative impact over the capability related (see Figure 11).

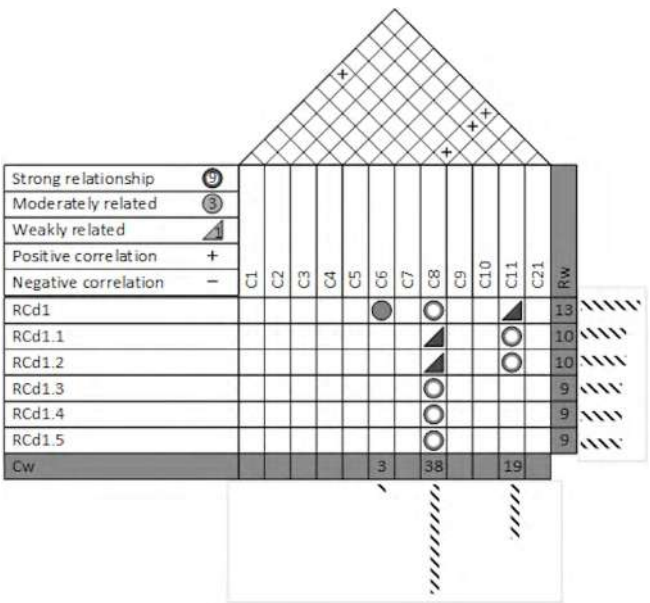

Figure 11 Capability and requirements relationship for the CAD tool

A weight factor vector C_w for capabilities and R_w for the requirements can be obtained from the relations. The vector $C_{R,i}$ to i=1,2,⋯n, where n is the number of capabilities in the system,

is obtained from Table 5 $C_{R,i}$ contains all the requirements (strong, average, and weak) related to the capability C_i. See an example in Table 6. Additionally, the correlation matrix C_c that contains the correlation between capabilities can also be obtained. (See the HOQ roof in Figure 11).

Table 6 Requirements related with the capability C_8 (Implant modeling)

$C_{R,8}$	
RCd1	Strong
RCd1.1	Weak
RCd1.2	Weak
RCd1.3	Strong
RCd1.4	Strong
RCd1.5	Strong

3.2. Capability analysis

To carry out the capability analysis, it is necessary to observe the models in Figure 6, since it represents the designed capabilities to meet the requirements that need the LTL formulas, as shown in Table 4.

From the $C_{R,i}$ vector that contains the requirements for the capability C_i, three additional vectors are denoted: $R_{s,i}$, $R_{a,i}$, $R_{l,i} \in R$, where $R_{s,i} \cap R_{a,i} \cap R_{l,i} = \emptyset$. That represents the strong, average, and low requirements related to the capability C_i.

In addition to the LTL formulas related to the requirements $R_{s,i}$, $R_{a,i}$, $R_{l,i}$, the $\varphi_{s,i}$, $\varphi_{a,i}$, $\varphi_{l,i}$ vectors are formulated and they contain the respective set of requirements (strong, average, and low). Being ξ the behavioral model used, the LTSA tool proceed to verify if the LTL formulas $\varphi_{s,i} \Vdash \xi$, $\varphi_{a,i} \Vdash \xi$, $\varphi_{l,i} \Vdash \xi$ satisfy the behavior model.

When the set of properties $\varphi_{s,i}$, $\varphi_{a,i}$, and $\varphi_{l,i}$ are tested, the vectors $O_{s,i}$, $O_{a,i}$, and $O_{l,i}$ contain the respective testing results, see Table 7.

In order to determine if a capability is achieved, the next variables are defined: N_i: the number of properties to verify the capability C_i in ξ, K_i, L_i, $T_i \in \mathbb{N}$: the number of properties evaluated in $O_{s,i}$, $O_{a,i}$ and $O_{l,i}$. $\alpha_{s,i}$, $\beta_{s,i}$ and $\gamma_{s,i}$: the positive, negative, and inconclusive results obtained for the strongly related properties $\varphi_{s,i}$.

In the same way, $\alpha_{a,i}$, $\beta_{a,i}$ and $\gamma_{a,i}$ are defined as the average related properties, and $\alpha_{l,i}$, $\beta_{l,i}$ and $\gamma_{l,i}$ as the weakly related properties. The capability analysis also needs the Eqs (1-3) since they indicate the number of properties verified for the strong, average, and low requirements related to the capability C_i, and the Eq. (4) represents the total number of requirements related to a particular capability.

Table 7 Capability testing report for capability

Subset	Result	
	Property ($\varphi_{s,8}$)	Output
$O_{s,8}$	assert RCd1.3= <>((importSTL || importIGES || importSTEP) -> (rebuildVolume))	✓
	assert RCd1.4= (<>(rebuildVolume ->(({ modelDesign W regectDesign)))-> modelDesign)	✓
	assert RCd1.5= <>(rebuildVolume -> (modelDesign))	✓
	Property ($\varphi_{s,8}$)	Output
$O_{a,8}$		
	Property ($\varphi_{s,8}$)	Output
$O_{l,8}$	assert Rcd1.1= <>((alternatives)->(importSTL || importIGES || importSTEP))	✓
	assert RCd1.2= <>((accuracyDesing)->((exportSTL || exportSTEP || exportGES) W (importSTL || importIGES || importSTEP)))	✓

$$\alpha_{s,i} + \beta_{s,i} + \gamma_{s,i} = K_i \quad (1)$$

$$\alpha_{a,i} + \beta_{a,i} + \gamma_{a,i} = L_i \quad (2)$$

$$\alpha_{l,i} + \beta_{l,i} + \gamma_{l,i} = T_i \quad (3)$$

$$K_i + L_i + T_i = N_i \quad (4)$$

For a particular set $O_{s,i}$, $O_{a,i}$, or $O_{l,i}$, there are six possible results when the model-checking process is carried out:

1) All the properties are achieved to the capability C_i.
2) Some properties are not achieved and the counter examples are shown.
3) None of the properties are fulfilled nor are the counter examples shown.
4) Some results are not conclusive.
5) The entire set of properties does not have conclusive results.
6) Some properties are achieved.

In a capability model checking verification, it is possible to obtain a combination of results when the three sets of properties $O_{s,i}$,$O_{a,i}$, and $O_{l,i}$ are being taken into account. To summarize, the combined results are presented in Table 8.

Table 8 Result analysis for the capability

Rule	Of_j	Om_j	Od_j
i	$\sum\alpha_f = k$	$\sum\alpha_m = l$	$\sum\alpha_d = t$
ii∧iv	$\sum\beta_f + \gamma_f = k$	$\sum\beta_m + \gamma_m = l$	$\sum\beta_d + \gamma_d = t$
iii	$\sum\beta_f = k$	$\sum\beta_m = l$	$\sum\beta_d = t$
v	$\sum\gamma_f = k$	$\sum\gamma_m = l$	$\sum\gamma_d = t$
ii∧iv∧vi	$\sum\alpha_f + \beta_f + \gamma_f = k$	$\sum\alpha_m + \beta_m + \gamma_m = l$	$\sum\alpha_d + \beta_d + \gamma_d = k$
iv∧vi	$\sum\alpha_f + \gamma_f = k$	$\sum\alpha_m + \gamma_m = l$	$\sum\alpha_d + \gamma_d = t$

According to Table 8, only one from the six options is selected as an output for the $O_{s,i}$, $O_{a,i}$, and $O_{l,i}$ sets. It is proposed to assign a possible value for each possible result, see Table 9.

Table 9 Value assignment proposal for the results in $O_{s,i}$, $O_{a,i}$, and $O_{l,i}$

	Of_j	Om_j	Od_j
i	3	3	3
ii∧iv	-2	-2	-1
iii	-3	-3	-3
v	0	0	0
ii∧iv∧vi	-1	0	1
iv∧vi	1	2	2

When the requirements associated with a particular capability are verified, a general value that summarizes the capability fulfillment is needed, see Eq. (5). $\chi \in \{-3,3\}$. It represents a certainty value associated with the requirements fulfillment in the capability C_i (see Table 10).

$$x=celling\left(\frac{Of_j+Om_j+Od_j}{3}\right) \quad (5)$$

Table 10 Capability compliance certainty levels

Certainty value	Result interpretation
-3	There is a high certainty of non-compliance.
-2	There is an average certainty of non-compliance.
-1	There is a low certainty of non-compliance.
0	There is no certainty of compliance or noncompliance.
1	There is a low certainty of compliance.
2	There is an average certainty of compliance.
3	There is a high certainty of compliance.

3.3. Result analysis

For this particular case study –analysis of the capability associated with the CAD tool–, it is possible to obtain a compliance analysis and indicator. See Table 11.

$\chi_{t,i} = C_{w,i} \times \chi_i$ denotes the compliance indicator for a particular capability, $T_i = \sum C_{w,i}$ the weight for the entire capabilities, $T_c = \sum \chi_{t,i}$ the compliance indicator for the entire capabilities, and $K_f = T_i \times 3$ the domain in which $\chi_{t,i}$ exists. $\chi_{t,i} \in \{-3\times C_{w,i}, -3 \times C_{w,i}\}$ defines the compliance value for each capability; negative values mean low expectations, positive values mean high expectations, and null values mean that nothing can be said about the capability achievement. The capability integration $T_c \in \{-K_f, K_f\}$ shows a measurement of the entire capabilities achievement. The values that are closer to K_f indicate that the capabilities are well incorporated, but the values closer to $-K_f$ indicate that the capabilities incorporation by the system is poor. Finally, the values that are closer to zero (0) indicate that no capability analysis could correctly be performed.

Table 11 Compliance analysis and indicator

Capability	$C_{w,i}$	Impact							Measurement
		-3	-2	-1	0	1	2	3	$X_{t,i}$
C_8	38							×	114
T_i	38							T_c:	114
								$K_f = T_i \times 3$	114

Figure 12 represents the compliance indicator to analyze the implications of the capability analysis.

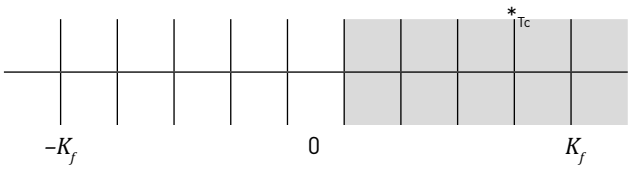

Figure 12 Graphical representation of the compliance indicator

Besides, the capabilities affected by an analysis can be obtained from the correlation matrix Cc. Particularly, if the capabilities C_1, C_9, C_{11}, and C_{12} could be affected.

4. Analysis and conclusions

With each capability compliance indicator, a notion on how the system general goal are met can be obtained at an early phase, as well as whether those goals could be achieved with the corresponding designed capabilities.

Particularly, the $\chi_{t,i}$ indicator measures three important aspects. Firstly, the capability weight $C_{w,i}$ indicates how important the capability is in relation to the stakeholder requirements. Secondly, the capability compliance certainty indicates how well designed a capability is. Thirdly, the compliance indicator domain $[-K_f, K_f]$ indicates the interval in which $\chi_{t,i}$ exists.

A $\chi_{t,i}$ value less than zero is an indicator of a wrong capability design. In contrast, a $\chi_{t,i}$ value greater than zero is an indicator of a good capability design. A $\chi_{t,i}$ value close to zero indicates that the capability design must be thoroughly checked. Finally, a $\chi_{t,i}$ value close to K_f or $-K_f$ indicates that capability designed is well conceived or an entire design change is needed, with a high certainty.

When the analysis includes more than one capability, the same analysis is carried out by using the T_c values that indicate how well integrated and designed the capabilities are. It is also important to take into account that T_c is more related to system goals than $\chi_{t,i}$ values.

Using the capability as an evaluation item of a fault detection model instead of directly using requirements allowing to detect problems at systems design early stages. This type of analysis is associated with the systems' goals rather than stakeholder needs. So it is useful for fault insertions reduction at early stages and correct systems goals design.

5. Acknowledgment

This research was supported by the Administrative Department of Science, Technology and Innovation (*COLCIENCIAS*); it has been funded by *Francisco José de Caldas* Doctoral Program Scholarship # 511. We thank Anna Shchiptsova who greatly assisted the research and the International Institute for Applied Systems Analysis (*IIASA*) that provided insight and expertise in the Advanced Systems Analysis area.

6. References

1. C. Baier and J. P. Katoen, *Principles of model checking.* Cambridge, USA: MIT Press, 2008.
2. E. Karch, *The Software Crisis: A Brief Look at How Rework Shaped the Evolution of Software Methodologies*, 2011. [Online]. Available: https://blogs.msdn.microsoft.com/karchworld_identity/2011/04/04/the-software-crisis-a-brief-look-at-how-rework-shaped-the-evolution-of-software-methodolgies. Accessed on: May 25, 2017.
3. V. Lima *et al.*, "Formal verification and validation of UML 2.0 sequence diagrams using source and destination of messages," *Electron. Notes Theor. Comput. Sci.*, vol. 254, pp. 143–160, 2009.
4. M. E. Beato, M. Barrio, C. E. Cuesta, and P. de la Fuente, "UML automatic verification tool with formal methods," *Electron. Notes Theor. Comput. Sci.*, vol. 127, no. 4, pp. 3–16, 2005.
5. S. Anwer and N. Ikram, "Goal oriented requirement engineering: A critical study of techniques," in *13th Asia Pacific. Software Engineering Conference* (APSEC), Bangalore, India, 2006, pp. 121–130.
6. IEEE, *Systems and software engineering - Vocabulary*, IEEE Standard 24765, 2010.
7. F. Dandashi, R. Siegers, J. Jones, and T. Blevins, *The Open Group Architecture Framework (TOGAF) and the US Department of Defense Architecture Framework (DoDAF)*, 2006. [Online]. Available: https://www.mitre.org/sites/default/files/pdf/06_0987.pdf. Accessed on: May 25, 2017.
8. K. Griendling and D. N. Mavris, "Development of a dodaf-based executable architecting approach to analyze system-of-systems alternatives," in *IEEE Aerospace Conference*, Big Sky, MT, USA, 2011, pp. 1-15.
9. S. Sharma, J. Conduit, and S. R. Hill, "Organisational capabilities for customer participation in health care service innovation," *Australas. Mark. J.*, vol. 22, no. 3, pp. 179–188, 2014.
10. G. P. Castro, "Evaluación de un modelo de integración de herramientas software dirigido al sector biomédico-ortopédico," Undergraduate thesis, Univ. Industrial de Santander, Bucaramanga Colombia, 2014.
11. A. Josey *et al.*, *TOGAF Version 9.1 - A Pocket Guide.* 1st ed. Van Haren Publishing, 2011.
12. Z. G. Tao, Y. F. Luo, C. X. Chen, and M. Z. Wang, and F. Ni, "Enterprise application architecture development based on DoDAF and TOGAF," *Enterprise Information Systems*, vol. 11, no. 5, pp. 627-651, 2017.
13. E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer Programming*, vol. 2, no. 3, pp. 241-266, 1982.
14. J. P. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR," in *International Symposium on Programming,* Turin, Italy, 1982, pp. 337–351.
15. D. Giannakopoulou, "Model checking for concurrent software architectures," Ph.D. dissertation, Imperial College of Science, London, UK, 1999.
16. T. J. Blevins, J. Spencer, and F. Waskiewicz, *TOGAF ADM and MDA® The Power of Synergy*, The Open Group, 2004. [Online]. Available: http://www.opengroup.org/cio/MDA-ADM/. Accessed on: May 25, 2017.
17. S. Friedenthal, A. Moore, and R. Steiner, "OMG Systems Modeling Language (OMG SysML™) Tutorial," *INCOSE International Symposium,* vol. 18, no. 1, pp. 1731–1862, 2008.
18. M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *1st Symposium on Logic in Computer Science* (LICS), Cambridge, MA, USA, 1986, pp. 322–331.
19. J. Magee, J. Kramer, R. Chatley, S. Uchitel, and H. Foster, *LTSA - Labelled Transition System*. [Online]. Available: http://www.doc.ic.ac.uk/ltsa. Accessed on: Jan. 1, 2017.
20. D. Pandey, U. Suman, and A. K. Ramani, "An effective requirement engineering process model for software development and requirements management," in *2nd International Conference on Advances in Recent Technologies in Communication and Computing* (ARTCom), Kottayam, India, 2010, pp. 287–291.
21. G. A. Ricardo and J. Borbinha, *Blog Post: Requirements and Capabilities*, 2013. [Online]. Available: http://www.timbusproject.net/portal/blogs-news-items-etc/timbus-blogs/196-requirements-and-capabilities/. Accessed on: May 16, 2017.
22. K. A. Griendling, "Architect: the architecture-based technology evaluation and capability tradeoff method," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, USA, 2011.