Formal verification of safety control system based on GHENESYS NET

Rodrigo Cesar Ferrarezi, Reinaldo Squillante Júnior, Jeferson A. L. Souza, Diolino J. Dos Santos Filho, José Reinaldo Silva, Paulo Eigi Miyagi, Lucas Antonio Moscato

Abstract**—**Due to the high complexity of the actual Productive Systems, the current industrial standards, and the possible negative impacts on the human being, on the environment and on equipment in case of faults, the development of control solutions that are both secure and stable – as some systems have to operate nonstop – is much demanded. In this context, the development of safety control systems which simultaneously present high reliability and availability is required. The concepts of SIS, according to experts, may be one solution to these problems. Due the complexity of these systems, project mistakes are expected during their development and thus, validation and verification processes became an imperative – as well as a normative requirement – before the actual deployment of the control software on site. One of the most outstanding system verification techniques is the Model Checking, which performs an exhaustive search on the state space of an event driven system and checks some specific properties written in temporal logic. The GHENeSys environment will be used as computational tool, as it provides a complete solution for modelling and verifying systems based on the GHENeSys network. The proposed methodology will then be applied to the development of a SIS control system to be implemented on a flexible manufacture system, which simulates assembly and handling of parts.

Keywords**—**GHENeSys nets, Model Checking, Safety Instrumented System, Verification.

I. Introduction

The growing demand on cost and quality of products and services, the highly competitive market with several players, the increasing hardware storage capacity, processing power and networks speeds, and above all, the concern with the environment, the foundation of all current suitability policies caused an implementation of more complex control systems in the most diverse areas, from the production of consumer products to services [1].

The increasing implementation of processes automation, mandatory for costs reductions and quality improvements, key factors to the survival of a company in a highly competitive market, induced an ever increasing complexity of the control systems required for these systems [2] [3]

The authors would like to thank the Brazilian governmental agencies CNPq, FAPESP, and CAPES for their financial support to this work.

All authors are with Polytechnic School – Department of Mechatronics Engineering and Mechanical Systems - University of São Paulo - São Paulo, Brazil (phone: +55 11 3091-5337 ; e-mail: rferrarezi@usp.br, reinaldo.squillante@usp.br, jeferson.souza@usp.br, diolinos@usp.br, reinaldo@usp.br, pemiyagi@usp.br, lamoscat@usp.br).

Being the control software increasingly complex, and the quality requirements more and more severe, there is a demand for more detailed and concise specification as well as a better control of the development process. It is also required a deeper understanding of the system to be controlled, including details regarding all relevant sub-systems and furthermore, how several system interacts and communicates with each other and with the environment, as the behavior of an interconnected system depends not only of its internal variables, but also of external events originated from the surrounds of the system [4] [5].

Additionally, any industrial system, as modern and innovative as it can be, still may pose serious risks to equipment, to operators and to the environment, in the event of a fault failing to be diagnosed and treated correctly [6]. Although many studies have been presented for diagnosis and treatment of faults, accidents still occur. The main problem is that there is no zero risk in process industries since: (i) physical devices do not have zero risk of fault, (ii) human operators do not have zero risk of error and (iii) there is no computational system that can predict all the reachable states by the system [7].

According to experts, the concepts of safety instrumented systems (SIS), is one solution to these types of issues. They strongly recommend the implementation of layers of risk reduction based on control systems organized hierarchically in order to manage risks by either preventing or mitigating faults, bringing the process to a safe state. In this sense, some safety standards such as IEC 61508 [8], IEC 61511 [9] among others, guide different activities related with a SIS Safety Life Cycle (SLC), such as design, installation, operation, maintenance, tests and others [10][11].

On this context, the processes of understanding, specifying, modelling and validating these systems became a highly complex task, resulting in great hardships on their development. Due all this, project mistakes are associated with the development of these systems and thus, validation and verification processes became an imperative before the actual deployment of the control software on the actual plant [3] [4]. Besides the obvious necessity of verifying and validating critical systems, these activities are required by the safety standard IEC 61511 [9] as part of the safety program development cycle, also known as “V-model.

Model Checking is a verification technique for finite state concurrent systems, and thanks to this restriction, the verification processes can be performed semi-automatically, being human interaction only needed for the analysis of the results. The basic procedure performs an exhaustive search of the space state of an event driven system, verifying properties specified from propositions described using some temporal logic. Given enough time and computational power, the procedure will always finish with a positive or negative result, in case of a negative result; a counterexample is given by the system, helping the designer to find the source of error [12].

In this work we propose the first steps towards the development of a framework for the modelling and formal verification of SIS control programs based on the IEC 61511 standard. On the framework, we expect to propose methods, techniques and systematics to comply with all phases of the "V model" according to the IEC 51511 standard. With the complete framework we expect to aid the control engineers to develop SIS control programs in a structured and well defined way as well as fulfilling the requirements of the IEC 51511 standard.

The GHENeSys environment will be used as computational tool modelling and verification processes, the environment is a proposal of the DesignLab from USP and was originally designed as unified approach to cover several types of Petri Nets as well as its extensions, support for timed nets was implemented later on.

This paper is organized as follows: Section 2 presents the main concepts of Model Checking, GHENeSys nets and TCTL. Section 3 presents the SIS Control System Modelling proposal. Section 4, presents the application example. Section 5 presents the conclusion. References are presented thereafter.

II. MAIN CONCEPTS

A. *Model checking*

The Model Checking technique is composed by the following main tasks [12]:

– Modelling: First the system is converted to a formalism accepted by the chosen verification tool.
– Specification: Before the verification process, it is necessary to list the required system properties. These specifications must also be supplied in some kind of formalism. Usually temporal logic is used to specify the system behavior.
– Verification – Usually the verification process is performed automatically by the tool, except by the results analysis – in case of errors being found. In this case, the tool will supply counterexamples for the verified property, helping the designer finding the source of error on the system.

Concurrent systems can frequently interact with their environments and usually are operating non-stop, therefore, these systems cannot be properly modelled by their input output behavior. The first feature of these systems that must be taken into account is the state. A state can be defined as an instantaneous description of the system, containing the value of every system variable in a single instant. Also it is important to understand how the states change as result of some action of the system. This change can be described as the state of the system before and after the some action, this pair of states determines a transition of the system [12].

In order to represent the behavior of concurrent systems several types of used graphs might be used, among them we have Kripke structures [12], automata [13], Petri Nets [14] and its extensions, as the GHENeSys nets [15].

There are many different types of concurrent systems (synchronous and asynchronous systems, sequential systems, parallel process, etc.) and due to this diversity, it is necessary to adopt unifying formalism in which these systems can be represented regardless of its type. For such representations will be used first order logic formulas, which are able to represent a great variety of systems [12].

In order to write specifications to describe the properties of a concurrent system it is necessary to define a set of atomic propositionsAP. Such propositions have the form $\mathrm{v} = \mathrm{d}$ where $\mathrm{v} \in \mathrm{V}$ and $\mathrm{d} \in \mathrm{D}$, an atomic proposition $\mathrm{v} = \mathrm{d}$ is said to be true in a state s if $s(\mathrm{v}) = \mathrm{d}$[12].

Temporal logic was proved to be very useful for specifying concurrent systems due to its capacity to describe the ordering of events without introducing time explicitly, and thanks to this feature, it was possible to develop completely automated verification algorithms.

B. *Timed Computation Tree Logic (TCTL)*

TCTL [16] was proposed as an extension of the CTL (Computation Tree Logic) proposed in [12], for the interpretation of temporal formulas over computational trees for systems modelled by temporized graphs. TCTL can be defined semantically related to a structure $\mathcal{M} = (\mathcal{S}, \mu, \mathrm{f})$, where $\mathcal{S}$ is the set of states, $\mu : \mathcal{S} \rightarrow 2^{\mathrm{AP}}$ the labelling function which labels each state with the set of atomic prepositions that hold on this state and f the mapping that assigns for each $\mathrm{s} \in \mathcal{S}$ a set of s-paths through $\mathcal{S}$ that obey the closure properties [16].

The formula ϕ of TCTL can be inductively defined as [16]:
$\phi ::= \mathrm{p}\ |\mathrm{false}|\neg\phi\ |\phi_1 \rightarrow \phi_2|\phi_1 \wedge \phi_2\ |\exists[\phi_1\ \mathrm{U}_{\sim c}\phi_2]|\forall[\phi_1 \mathrm{U}_{\sim c}\phi_2]$, where $\mathrm{p} \in \mathrm{AP}$ and $\mathrm{c} \in \mathbb{N}$.

TCTL formulas are composed of path quantifiers and temporal operators. There are two path quantifiers:

– $\forall$ (for all computation paths)
– $\exists$ (for some computation path)

Quantifiers are used to specify if from some state, if all paths or just some paths must have some propriety. Temporal operators are used to describe the properties of a path belonging to some computational tree and were defined by adding timing limitations to the classical CTL operators [16]:

– $\Diamond_{\sim c}\ \phi = \mathrm{true}\ \mathrm{U}_{\sim c}\phi$ (ϕ must hold eventually in some state of the computational path for $\sim$c time units);
– $\Box_{\sim c}\phi$ (ϕ must hold in all states of the computational path for $\sim$c time units);
– $\phi_1\ \mathrm{U}_{\sim c}\phi_2$ (ϕ_2 must hold in some state and ϕ_1 must hod in all previous states of the computational path for $\sim$c time units).

Where $\sim$ may represent any of the following binary operators $<, \leq, =, >, \geq$. It is important to note that TCTL, as opposed to CTL, does not defines the temporal operator that requires that some property must hold on the next state, because as the time is considered dense, by definition, there is not only one next state [16].

Given a TCTL structure $\mathcal{M} = (\mathcal{S}, \mu, f)$ and a state $s \in \mathcal{S}$, a TCTL formula ϕ holds $(\mathcal{M}, s) \vDash \phi$ if [16]:

– $s \vDash p$ if $p \in \mu(s)$

– $s \nvDash$ false;

– $s \vDash (\phi_1 \rightarrow \phi_2)$ if $s \nvDash \phi_1$ or $s \nvDash \phi_2$;

– $s \vDash \exists(\phi_1 \mathrm{U}_{\sim c} \phi_2)$ if for some $\rho \in f(s)$, for some $t \sim c$, $\rho(t) \vDash \phi_2$, and for all $0 \leq t' < t, \rho(t') \vDash \phi_1$.

– $s \vDash \forall(\phi_1 \mathrm{U}_{\sim c} \phi_2)$ if for each $\rho \in f(s)$, for some $t \sim c$, $\rho(t) \vDash \phi_2$, and for all $0 \leq t' < t, \rho(t') \vDash \phi_1$.

C. *GHENeSys environment*

A SIS control system can be seen as an event driven system and presents functional characteristics as asynchronism, possibility of reset, parallelism, concurrence, etc., thus this class of system can be classified as a discrete event system (DES), and thus be modelled through Petri Nets [14] [17] and its extensions.

The GHENeSys environment was developed as an extended Petri net with object orientation and abstraction mechanisms defined through hierarchy concepts, which are included as well as synthesis mechanisms that are implemented through a structured approach supported by the encapsulation introduced by the use of objects [15].

The GHENeSys environment is being developed with the goal of representing, in a unified way, classical Petri Nets, its extensions defined on the ISO/IEC 15909 standard, as well as High Level Petri Nets. The GHENeSys environment is composed of the following basic modules. The GHENeSys nets, the Editor tool, the simulation module and the verification tool [18].

The GHENeSys environment implements also several concepts to aid the modeling process, such as: Pseudoboxes that allow the modelling of the exchange of information between different parts of the system; Hierarchy that allows the encapsulation of subnets without losing any properties by using macro elements; The representation of non-deterministic time durations, where a set of time intervals can be defined for each transition.

The GHENeSys net is the tuple $G = (L, A, F, K, \Pi, C_0, \tau)$, where:

– $L = B \cup P$ is the set of places, which can be boxes or pseudoboxes;

– A are the activities, or active elements;

– $F \subseteq (L \times A \rightarrow \mathbb{N}) \cup (A \times L \rightarrow \mathbb{N})$ is the flux relation;

– $K : L \rightarrow \mathbb{N}^+$ is the capacity function;

– $\Pi : (B \cup A) \rightarrow \{0,1\}$ is the function that identify the macro elements or the hierarchy;

– $C_0 = \{(l, \sigma_j) | l \in L, \sigma_j \in \mathbb{R}^+ | l | < K(l)\}$ is the set of initial marks;

– $\tau : (B \cup A) \rightarrow \{\mathbb{Q}^+, \mathbb{Q}^+ \cup \{\infty\}\}$ is the function that maps the dense time intervals for each element.

The set of markings is the pair (l, σ_j) with $l \in L$, defining which place each token can be found and σ_j defining for how long this token will remain in place. The time measurement is globally synced and updated after each transition.

The GHENeSys verification tool performs the formal verification of real time concurrent systems modelled as GHENeSys net through Model Checking [12] techniques. The space state is constructed using the enumerative approach based on the state class [19] concept. The tool has options to build SCG, SSCG and CSCG state graph types. Checked properties are specified through TCTL [20].

The GHENeSys environment will be used in this work due to several reasons: (i) Due characteristics of the SIS, the amount of checked properties can be very large, so it might be desirable that the space state is generated through the enumerative approach instead of being generated "on the fly" several times. As the space state generation is done in exponential space and time, and the verification of a property is performed in polynomial time, if the space state is already constructed. (ii) The use of the dense time approach, as several SIS properties are time dependent. (iii) PNML [21] is implemented as the default transfer format, thanks to that, the interchange of information between the GHENeSys tools is possible, as well as with external tools that support the standardized format. (iv) The environment allows that all modelling and verification tanks to be performed without the need of external modules or tools. (v) The space state generation and the specification of the tested properties are done on the same tool.

D. *Requirements for safety control programs*

According to [9] and [22], safety control programs must be developed according to the development cycle proposed by the IEC 61511 standard as shown on Fig. 1and using modularity concepts.

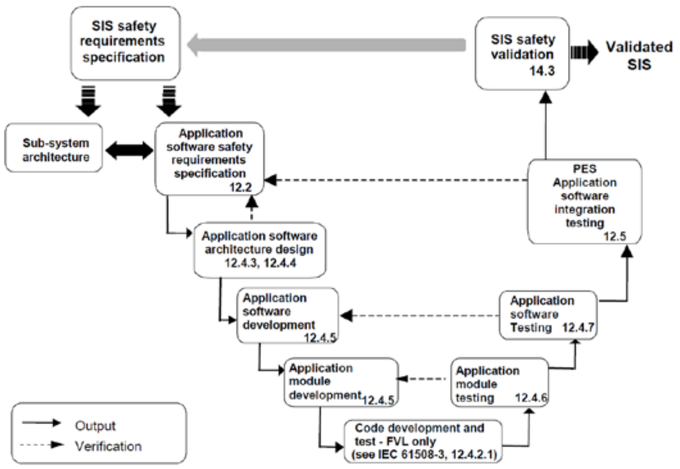

SIS safety requirements specification
SIS safety validation 14.3
Validated SIS
Sub-system architecture
Application software safety requirements specification 12.2
PES Application software integration testing 12.5
Application software architecture design 12.4.3, 12.4.4
Application software development 12.4.5
Application software Testing 12.4.7
Application module development 12.4.5
Application module testing 12.4.6
Code development and test - FVL only (see IEC 61508-3, 12.4.2.1)
Output
Verification

Fig. 1. Safety program development cycle or "V-model"

The development cycle is the combination of several phases ranging from the requirements analysis to the formal verification of the entire control program. The development phases, located on the left, also include steps related with the description of the operation of each SIS module, the choice of methods and tools to aid the development, the development of the control program and its modules integration and the control code development. The verification phases, located on the right, include the formal verification of the modules and their integration and the final tests of the control program and hardware integration.

III. SIS CONTROL PROGRAM MODELING

The initial step towards the proposed framework has already been done on the previous section. There, the GHENeSys environment was chosen as the modelling and verification tool, as well as the choice was justified as required by the standard as part of the second phase of the development cycle.

Through the concepts of modularity, we can break the SIS control program in two parts. The prevention module is responsible to detect dangerous events represented as critical faults and deploy the suitable treatment to degenerate the system leading it to a safe state. The mitigation module is responsible to detect the effects of a fault not being treated correctly – or even not being detected by the prevention module – and deploy the suitable treatment to degenerate the system and to extinguish the effects before they disseminate to other parts of the plant. The definition of the main control program modules and thus the high level control program architecture are the second part of the second phase of the development cycle.

The development of the framework will be performed according to the Model Based Design (MbD) approach [3]. Although this approach is not referred in the IEC 61511 standard, as according to the standard, all control programs are directly developed in an implementation language. The standard requires that a final validation shall be carried out by the end of the development cycle, and modeling is one of the recommended tools for validation. Thus by adopting the MbD, the framework will be not only complying with the standard but also improving the modularity of the SIS control programs.

The third and fourth phases of the development cycle are related with de control program development. On these phases must be chosen methodologies for the development of the prevention and mitigation modules. These methodologies must be based on formal models and must have been proposed according the requirements of the IEC 61508 and IEC 61511 standards.

The prevention module development methodology must first be able to define which faults the SIS will treat. This definition can be made through HAZOP reports, cause-effect matrix or other applicable technique. With knowledge of the faults, formal methods to discover the causal relations leading to each fault, that is, how – by which sensors – each fault can be detected must be presented. The methodology then must be able to propose actions to deal with each fault, through the actuation of some component – such as control valves – and/or the shutdown of an endangered component.

Now a mitigation module development methodology must be chosen. The same methodology as used on prevention can be adopted; as well as other methodology can be adopted, as long as the same criteria – as described on the previous paragraph – are used.

As opposite as the prevention activity, the mitigation activity does not need to define the treated faults through documents or reports. The mitigation activity shall treat the prevention activity faults, that is, the system will mitigate the consequences of the prevention system not being able to lead the plant or process to a safe state.

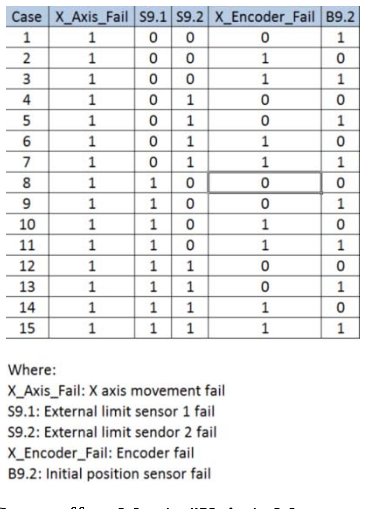
The sixth and seventh phases of the development cycle are related with the formal verification of the models. Due to the Model Based approach, the verifications are performed on the control program models and not on the control code. All properties specified no natural language shall be translated to TCTL according to the patterns proposed in [23].

IV. APPLICATION EXAMPLE

Now we will present a simple application example of the development of a prevention SIS control program according with the IEC 61511 phases already covered by the presented framework. We will begin the application example by choosing the methodology to develop the prevention SIS module, then the methodology will be applied and the resulting control program model will be verified.

The systematic proposed in [24] was chosen. Briefly, the faults that will be treated by the prevention SIS are extracted from the HAZOP report. The detection models are generated from cause-effect matrixes, where those matrixes are converted on Bayesian Networks, which are finally converted on Petri Nets. The treatment models, that is, the actuators related with each treated fault are generated from the HAZOP report.

The systematic was then applied to a flexible manufacturing system prototype. This prototype is composed by three manufacture cells: feeder, inspection and assembly. The cells are connected by a conveyor belt transportation system. On this example, the prevention SIS control system was developed for the assembly cell. This cell is composed of a three axis manipulator robot [24].



Case	X_Axis_Fail	S9.1	S9.2	X_Encoder_Fail	B9.2
1	1	0	0	0	1
2	1	0	0	1	0
3	1	0	0	1	1
4	1	0	1	0	0
5	1	0	1	0	1
6	1	0	1	1	0
7	1	0	1	1	1
8	1	1	0	0	0
9	1	1	0	0	1
10	1	1	0	1	0
11	1	1	0	1	1
12	1	1	1	0	0
13	1	1	1	0	1
14	1	1	1	1	0
15	1	1	1	1	1

Where:
X_Axis_Fail: X axis movement fail
S9.1: External limit sensor 1 fail
S9.2: External limit sendor 2 fail
X_Encoder_Fail: Encoder fail
B9.2: Initial position sensor fail

Fig. 2. Cause-effect Matrix "X Axis Movement Fault"

The first step of the systematic was the construction of the cause-effect matrix together with the definition of which fault will be treated by the SIS. On Fig. 2 is displayed the cause-effect matrix for 15 cases of “X axis movement fault” on the manipulator robot. For each case the combination of the four sensors that can detect this type of fault is presented.

On the next step, the data Fig. 2was inputted into the proper algorithms for construction of the Bayesian Network. The resulting Bayesian Network for the diagnostic model was then converted on the GHENeSys net presented on Fig. 4 .The grayed places displayed on the models are pseudo-boxes. These boxes carry the marking information of their master elements. The master element can be identified by the name of each pseudo-box.

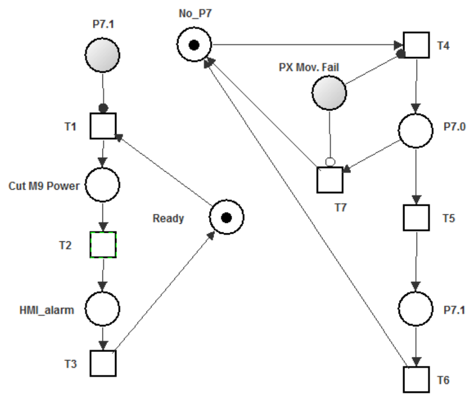
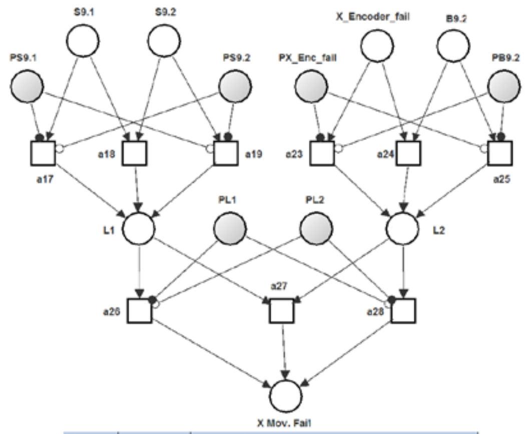


Fig. 3. GHENeSys net coordination model (left) and treatment model (right)

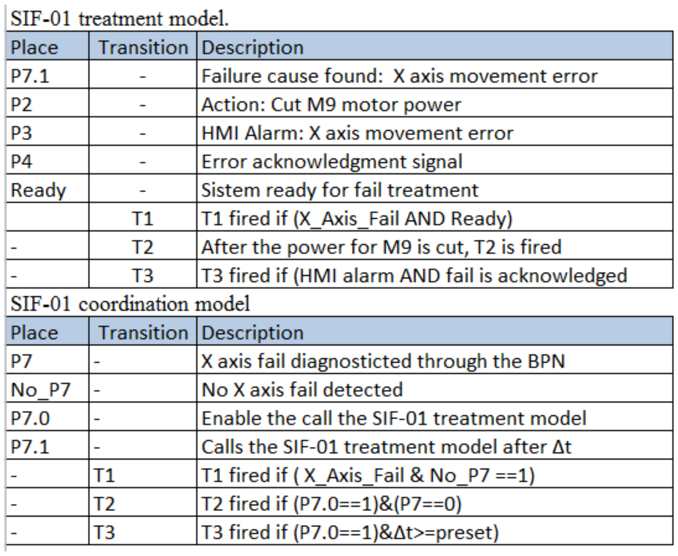
On the final step, the Safety Instrumented Function (SIF) for the manipulator robot “X” axis is determined based on the risk analysis HAZOP, and then the SIF treatment and coordination models were constructed as displayedFig. 3.The function of each transition and place used on the treatment and coordination models is explained on Fig. 5.



Place	Transition	Description
S9.1	-	"X" axis limit sensor 1
S9.2	-	"X" axis limit sensor 2
X_Encod	-	"X" axis encoder fail signal
B9.2	-	"X" axis position sensor
A1	-	Intermediate Place 1
A2	-	Intermediate Place 1
X_Axis_F	-	"X" axis movement fail diagnostic
-	T1	Logical (S9.1 OR S9.2) transition
-	T2	Logical (X_Encoder_Fail OR B9.2) transition
-	T3	Logical (A1 OR A2) transition

Fig. 4. Diagnostic model for “X axis movement fault”

A spurious events filter was implemented on the coordination model. This filter prevents the activation of the treatment model during a preset time, thus avoiding the degeneration of the plant and the costs associated with restating the plant in case of spurious readings from some sensor.



SIF-01 treatment model.

Place	Transition	Description
P7.1	-	Failure cause found: X axis movement error
P2	-	Action: Cut M9 motor power
P3	-	HMI Alarm: X axis movement error
P4	-	Error acknowledgment signal
Ready	-	Sistem ready for fail treatment
	T1	T1 fired if (X_Axis_Fail AND Ready)
-	T2	After the power for M9 is cut, T2 is fired
-	T3	T3 fired if (HMI alarm AND fail is acknowledged

SIF-01 coordination model

Place	Transition	Description
P7	-	X axis fail diagnosticted through the BPN
No_P7	-	No X axis fail detected
P7.0	-	Enable the call the SIF-01 treatment model
P7.1	-	Calls the SIF-01 treatment model after Δt
-	T1	T1 fired if (X_Axis_Fail & No_P7 ==1)
-	T2	T2 fired if (P7.0==1)&(P7==0)
-	T3	T3 fired if (P7.0==1)&Δt>=preset)

Fig. 5. Models elements descriptions

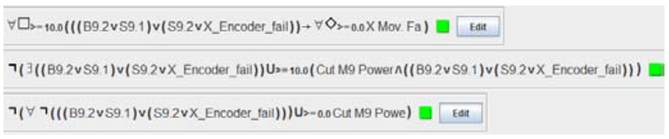
Currently there is no template or standard to determine a set of the system properties to be tested, thus, these properties usually are chosen by the control engineers based on previous knowledge and expertise [3] .

Table I. Checked Properties

1	If any of the sensors B9.2, S9.1, S9.2 or X_Encoder_Fail are trigged for longer than the pre-set time the treatment model is called.
	$\forall\Box_{\geq 10}\big((B9.2 \vee S9.1 \vee S9.2 \vee X\_Encoder\_Fail) \rightarrow \forall\,\Diamond_{\geq 0}\,(\mathrm{X\ Mov.\ Fail})\big)$
2	The motor remain turned off until no sensor is detecting the fault anymore
	$\neg\exists\Big((B9.2 \vee S9.1 \vee S9.2 \vee X\_Encoder\_Fail)\ U_{\geq 10}\big((\mathrm{Cut\ M9\ Power}) \wedge (B9.2 \vee S9.1 \vee S9.2 \vee X\_Encoder\_Fail)\big)\Big)$
3	Motors are not turned off until a fault is detected by the sensors
	$\neg\forall\big(\neg(B9.2 \vee S9.1 \vee S9.2 \vee X\_Encoder\_Fail)\ U_{\geq 0}(\mathrm{Cut\ M9\ Power})\big)$

Thus, we propose do extract the basic properties from the SIFs descriptions, through that we have the main properties that the system was designed to fulfil. Besides the properties the system must fulfil, as we are working with safety systems, it is also imperative to check if the system reaches undesirable – or unsafe – states. So we have the following natural language properties and their respective translated TCTL propositions onTable I.

The verification tool displays the results in a colored square besides the formula: (i) green if the formula holds, or (ii), red if the formula does not hold. On Fig. 6all tested formulas are presented with their verification results on a extract of the verification tool output window. All formulas were verified with satisfactory results.



```
∀□>=10.0(((B9.2∨S9.1)∨(S9.2∨X_Encoder_fail))→∀◇>=0.0X Mov. Fa) 🟩
¬(∃((B9.2∨S9.1)∨(S9.2∨X_Encoder_fail))U>=10.0(Cut M9 Power∧((B9.2∨S9.1)∨(S9.2∨X_Encoder_fail))) 🟩
¬(∀¬(((B9.2∨S9.1)∨(S9.2∨X_Encoder_fail)))U>=0.0Cut M9 Powe) 🟩
```

Fig. 6. TCTL Propositions and verification results

V. CONCLUSIONS

In this work, the first steps towards a framework for modeling and verifying SIS control programs were presented, the framework is based on the safety software development cycle from the IEC 61511 standard associated with the model based design approach. As demanded by the IEC 61511 standard, tools and methodologies to comply with some phases of development cycle were chosen, the initial high level control architecture was proposed, guidelines to choose the methodologies for developing the prevention and mitigation activities were also proposed. Also guidelines for TCTL propositions mapping from natural language properties – which we consider one of the most difficult tasks when using model checking techniques – were chosen.

SIS control program, as the one developed on this work, are critical to the systems they protect, as they responsible for the identification and treatment of critical faults. These faults, if not treated might lead to severe accidents and the loss of human lives. Frameworks as the one introduced on the present work, became crucial in order to enable these systems to be correctly interpreted and developed allowing the SIS to present a lesser probability of faults.

The next steps on the development of the framework might include more detailed modules architecture and functionalities, as well as systematics for the refinement of the high level modules; Methodologies for automatic isomorphic transformation of the models in IEC 61131-3 code; Methodologies for the integration of the prevention and mitigation modules, as well as modules for the treatment of several faults; And finally, systematics to verify the integrated SIS models and the study the relation between its modules.

REFERENCES

[1] M. Bani Younis and G. Frey, "Formalization of Existing PLC Programs: A survey," , Kaiserslautern, 2003.

[2] Leonardo Rodrigues Sampaio, *Validação Visual de Programas Ladder Baseada em Modelos*. Campina Grande: Universidade Federal de Campina Grande, 2011.

[3] Mauro Mazzolini, Alessandro Brusaferri, and Emanuele Carpanzano, "An Integrated Framework for Model-based Design and Verification of discrete Automation Solutions," in *Proceedings 2011 9th IEEE International Conference on Industrial Informatics*, Milan, 2011, pp. 545-550.

[4] Michel Diaz, *Petri Nets - Fundamental Models, Verification and Applications*. London: John Wiley & Sons, 2009.

[5] Peter Hoffmann, Reimar Schumann, Talal M.A Maksoud, and Giuliano C. Premier, "Virtual Commissioning of Manufacturing Systems," in *24th European Conference on Modelling and Simulation*, Kuala Lumpur, Malaysia, 2010, pp. 175-181.

[6] M. Sallak, C. Simon, and J.-F. Aubry, "A Fuzzy Probabilistic Approach for Determining Safety Integrity Level," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 1, pp. 239-248, 2008.

[7] Reinaldo Squillante Jr., Diolino J. Santos Filho, Jeferson A. L. de Souza, Paulo E. Miyagi, and Fabrício Junqueira, "Safety in Supervisory Control for Critical Systems," in *Technological Innovation for the Internet of Things*, Costa de Caparica, 2012, pp. 261-270.

[8] IEC, "IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems," International Electrotechnical Commission, Geneva, Switzerland, 2010.

[9] IEC, "IEC 61511 - Safety instrumented systems for the process industry sector," International Electrotechnical Commission, Geneva, 2003.

[10] Mary Ann Lundteigen and Marvin Rausand, "Architectural constraints in IEC 61508: Do they have the intended effect?," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 520–525, 2009.

[11] Laihua Fang and Lijun Wei and Ji Liu Zongzhi Wu, "Design and Development of Safety Instrumented System," in *Proceedings of the IEEE International Conference on Automation and Logistics*, Qingdao, 2008, pp. 2685 - 2690.

[12] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model Cheking*, 1st ed. Cambridge: MIT Press, 1999.

[13] Rajeev Alur and David L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183-235, 1994.

[14] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[15] Pedro M. Gonzalez del Foyo and José Reinaldo Silva, "Towards a unified view of Petri nets and object oriented modeling," in *In 17th International Congress in Mechanical Engineering*, São Paulo, 2003, pp. 518-524.

[16] Rajeev Alur, Costas Courcoubetis, and David Dill, "Model-Checking in Dense Real-time," *Information and Computation*, vol. 104, no. 1, pp. 2-34, 1993.

[17] Richard Zurawski and MengChu Zhou , "Petri nets and industrial applications: a tutorial," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 6, pp. 567–583, 1994.

[18] Pedro M. G. del Foyo, A. S. P. José Miralles, and José Reinaldo Silva, "UM VERIFICADOR FORMAL EFICIENTE PARA SISTEMAS DE TEMPO REAL," in *X SBAI – Simpósio Brasileiro de Automação Inteligente*, vol. X, São João del-Rei, 2011, pp. 1220-1225.

[19] Bernard Berthomieu and Miguel Menasche, "An Enumerative Approach For Analyzing Time Petri Nets," in *Proceedings IFIP*, Paris, 1983, pp. 41-46.

[20] Rajeev Alur, Costas A. Courcoubetis, and David L. Dill, "Model-checking for real-time systems," in *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, Philadelphia, 1990, pp. 414-425.

[21] ISO/IEC, "Software and Systems Engineering - High-level Petri Nets, Part 2: Transfer Format, International Standard WD ISO/IEC 15909. Wd version 0.9.0," 2005.

[22] Alois Mayr, Reinhold Plösch, and Matthias Saft, "Towards an Operational Safety Standard for Software - Modelling IEC 61508 Part 3," in *18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, Las Vegas, 2011, pp. 97-104.

[23] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett, "Property Specification Patterns for Finite-state Verication," in *Proceedings of 2nd Workshop on Formal Methods in Software Practice*, Clearwater Beach, 1998, pp. 7-15.

[24] Reinaldo Squillante Júnior, Diolino Jose Santos Filho, Fabricio Junqueira, and Paulo Eigi Miyagi, "Development of Control Systems for Safety Instrumented Systems," *IEEE (Revista IEEE America Latina) Latin America Transactions*, vol. 9, no. 4, pp. 451-457, 2011.