# A Self-certifiable Architecture for Critical Systems Powered by Probabilistic Logic Artificial Intelligence

**Chapter** · August 2019

DOI: 10.1007/978-3-030-26250-1_32

Raúl Mazo
ENSTA Bretagne
**119** PUBLICATIONS **1,102** CITATIONS
SEE PROFILE

Henrique Madeira
University of Coimbra
**236** PUBLICATIONS **4,402** CITATIONS
SEE PROFILE

Raul Barbosa
University of Coimbra
**59** PUBLICATIONS **294** CITATIONS
SEE PROFILE

Daniel Diaz
Université de Paris 1 Panthéon-Sorbonne
**108** PUBLICATIONS **2,034** CITATIONS
SEE PROFILE

# A Self-Certifiable Architecture for Critical Systems Powered by Probabilistic Logic Artificial Intelligence

Jacques Robin[1] ✉, Raul Mazo[1], Henrique Madeira[2], Raul Barbosa[2], Daniel Diaz[1] and Salvador Abreu[1, 3]

[1] Centre de Recherche en Informatique, Université Panthéon-Sorbonne, Paris, France
{jacques.robin, raul.mazo, daniel.diaz}@univ-paris1.fr
[2] Departamento de Engenharia Informatica, Universidade de Coimbra, Portugal
{henrique, rbarbosa}@dei.uc.pt
[3] Departamento de Informática, Universidade de Evora, Portugal
spa@uevora.pt

**Abstract.** We present a versatile architecture for AI-powered self-adaptive self-certifiable critical systems. It aims at supporting semi-automated low-cost re-certification for self-adaptive systems after each adaptation of their behavior to a persistent change in their operational environment throughout their lifecycle.

**Keywords:** AI certification, autonomic architecture, argumentation, rule-based constraint solving, probabilistic logic machine learning.

## 1 Introduction

Critical systems must be certified as dependable before being legally allowed to be deployed. Today, certification consists in a dialog between two dependability experts: the expert from an engineering institution seeking certification for a new system (*i.e. the engineer*) and the expert from an independent accredited certification body (*i.e. the auditor).* It is completed when the engineer presents evidence-based arguments convincing the auditor that the system conforms to the industry dependability standard. Both the standard and the conformance arguments are formulated in natural language [1]. These arguments are of two kinds: (1) arguments, such as safety cases, that the engineered system satisfies its critical dependability requirements up to the probability threshold prescribed by the standard and (2) arguments that the process followed to engineer the system conformed to the engineering process prescribed by the standard. Today, the requirements and implementation of a critical system do not change post-deployment. Its certification is thus only questioned after repeated catastrophic failures (*e.g.* the Boeing 737 MAX).

Introducing AI in critical systems to make them autonomous and self-adaptive to new contexts disrupts this assumption. The design space of possible self-adaptations may be open and thus no longer certifiable once and for all before deployment. This should be the case of critical systems using on-line machine learning for lifelong self-adaptation such as autonomous cars adapting to evolving smart road infra-structures, traffic safety regulations and cybersecurity threats. Even when the self-adaptation design space can be closed, it may still be too large and sparse to be both exhaustively and cost-effectively verified and certified before deployment. The alternative is to incrementally re-certify it following each major adaptation. This makes reducing the cost overhead of certification, through automation, an absolutely crucial issue.

In this paper, we propose a generic architecture addressing this issue. Its key idea is that a sufficiently versatile AI inference engine can be reused for a wide range of both (a) the application-specific reasoning tasks needed by a dependably autonomous critical system and (b) the application-*independent meta*-reasoning tasks needed to make such system additionally autonomic [2] in the sense of being self-adaptive, self-explainable, self-verifiable, self-argumentative and consequently largely self-certifiable. In the next section, we explain why a probabilistic constraint solving rule engine can provide the needed versatility. Then in section 3, we describe the various autonomous and autonomic reasoning tasks needed to make a critical system both self-adaptive and self-certifiable. Finally in section 4, we discuss the main limitations of our proposed architecture and engine and outline approaches to overcome them.

## 2 Probabilistic Logic Constraint Solving Rules

We propose to leverage in synergy the versatility of (a) constraint solving [3] and (b) probabilistic rule-based reasoning [4] to parsimoniously support both autonomous reasoning and autonomic meta-reasoning. Among the various formalisms proposed for probabilistic rule-based constraint solving, we choose to present *CHRiSM (CHance Rules in Statistical Modeling)* [5] in this paper, because it is conceptually simple, very expressive and has been shown to support the kind of legal argumentative reasoning [6] that is central to our certification automation proposal.

A CHRiSM solver is composed of a task-independent CHRiSM engine and a task-specific CHRiSM rule base. The engine solves a *Constraint Solving Problem (CSP)* [3] by applying the rules that gradually transform an initial constraint store representing the CSP into a final constraint store representing its solution.
The store is of logical form $\forall i, L_i \wedge_i c_j(L_j)$ where the $L_i$ are variable sets from a given mathematical domain and the $c_j$ are so-called *constraints, i.e.* relations that restrict the possible values that the $L_i$ can simultaneously take in their domain.

A CHRiSM rule base is a logical conjunction of two kinds of logical rules:

- Constraint *simplification* rules: $\forall i,j,k,l,m,L_m\ p{::}(\wedge_i g_i \Rightarrow (\wedge_j h_j \Leftrightarrow \vee_k (q_k{::}\wedge_l b_k^l)))$
- Constraint *propagation* rules: $\forall i,j,k,l,m,L_m\ p{::}(\wedge_i g_i \Rightarrow (\wedge_j h_j \Rightarrow \vee_k (q_k{::}\wedge_l b_k^l)))$

where the $g_i$, $h_j$, $b_k^l$ are *logical constraints* (respectively called the *guards*, *heads* and *bodies* of the rule) that may match those in the store, the $L_m$ are logical variables in $g_i$, $h_j$, or $b_k^l$, while $p$ and the $q_k$ are *arithmetic probability expressions* in [0,1]. These expressions may contain *random* variables $R_n$ in addition to some *logical* variables also appearing the $g_i$ or $h_j$, the latter allowing these expressions to depend on the result of rule guard evaluation and rule head matching. For each rule, $\sum q_k = 1$ and for each rule set sharing structurally matching $g_i$ and $h_j$, $\sum p = 1$.

When the current store entails $\wedge_i g_i$ and contains $\wedge_j h_j$ (modulo logical variable pattern matching) of such rule set, then one rule from the set is fired with probability $p$. If it is a simplification (*resp.* propagation) rule then $\wedge_l b_k^l$ *subsititutes* $\wedge_j h_j$ in the store with probability $q_k$, since, with that probability, it is logically equivalent to $\wedge_j h_j$ (*resp.* is *added* to the store with probability $q_k$ since it is logically implied by $\wedge_j h_j$).

A CHRiSM engine can perform three kinds of inferences:

- *solve($S_i$,$S_f$)* to compute the *most probable* solution $S_f$ for CSP $S_i$; if $S_i$ is exactly constrained, $S_f$ has the form $\wedge_n L_n=v_n$, assigning a single value to each variable; if $S_i$ is overconstrainted $S_f$ is *false*;
- *prob*($S_i \Leftrightarrow S_f$, *P*) to compute the probability *P* of $S_f$ being a solution for CSP $S_i$ ;
- *learn(E,R,D)* to machine learn, given a set *E* of example pairs ($S_i$,$S_f$) where $S_i$ is a CSP and $S_f$ one of its solutions, the probability distribution *D* of the random variables $R_i$ in the probability expressions of a CHRiSM rule base *R* using the *Expectation-Maximization* algorithm [7] initialized with the uniform distribution.

Any propositional or relational Bayes net can be represented by a semantically equivalent CHRiSM rule base [5]. For example, the classic alarm triggering Bayes net can be represented by the CHRiSM rule base:

*go* $\Rightarrow P_b$*::burglary(yes)* $\vee$ *(1-*$P_b$*)::burglary(no)*
*go* $\Rightarrow P_e$*::earthquake(yes)* $\vee$ *(1-*$P_e$*)::earthquake(no)*
*burglary(B)* $\wedge$ *earthquake(E)* $\Rightarrow P_a$*(B,E)::alarm(yes)* $\vee$ *(1-*$P_a$*(B,E))::alarm(no)*
$P_j$*(A)::(alarm(A)* $\Rightarrow$ *johncalls)*
$P_m$*(A)::(alarm(A)* $\Rightarrow$ *marycalls)*

where $P_b$, $P_e$, $P_a$, $P_j$ and $P_m$ are probabilities expressions. Given this rule base, the query *prob({go}* $\Leftrightarrow$ *{go, burglary(no), earthquake(yes), alarm(yes), marycalls}, P)* instantiates variable *P* with value $(1-P_b)*P_e*P_a(no,yes)*P_m(yes)$

A CHRiSM rule base without probability expressions is a *CHR*$^\vee$ *(Constraint Handling Rules with disjunctive bodies)* rule base [3]. With *CHR*$^\vee$ bodies being equiprobable, they are tried in writing order and backtracking is triggered when the current choice combination leads to a *false* store. CHR$^\vee$ subsumes the three main classes of rule-based formalisms: *term rewrite rules* (corresponding to CHR simplification rules), *production and business rules* (corresponding to CHR propagation rules) and *Constraint Logic Programming (CLP) rules* (which rule sets sharing the same head are equivalent to a single-head guardless CHR$^\vee$ simplification rule [3]). In addition, CHR$^\vee$ and CLP solvers have been successfully used to implement AI reasoning paradigms as diverse as ontological reasoning with description logics and frame logic, default reasoning, abduction, belief update, belief revision, natural language processing and optimization in addition to deductive constraint solving for which the approach was initially designed[1]. Therefore, AI components providing a critical system with any such reasoning capability can be uniformly implemented with the conceptual parsimony and built-in explainabilty of only applying the two kinds of rules shown above with their straightforward probabilistic or logical semantic readings.

## 3 An Architecture Supporting Self-Certification

Our proposed architecture for a *Self-Adaptive Self-Certifiable AI-Powered Critical System* is shown in Fig. 1 as a component diagram in the *Unified Modeling Language (UML)* standard (www.omg.org/spec/UML/2.5.1/). It composes:

[1] Since lack of space prevents us to insert all the relevant references in the bibliography of this short paper, see https://dtai.cs.kuleuven.be/CHR/biblio.shtml for a more complete one.

- A *Configurable Application Component* assembly (first left on 2nd row in Fig. 1) implementing the application, with *Deep Learned AI* components (second left of 2nd row) for fine-grained perception and actuator control and *Symbolic AI* components (first left of 3rd row) for explainable high-level cognition;
- A set of abstract *Probabilistic Rule-Based Constraint Solver* components (center of 3rd row) each one composed of a distinct project-specific CHRiSM *Rule Base* but all reusing the same project and industry independent CHRiSM *Rule Engine* component, which itself contains a *Rule Learner* (center of 2nd row) component to machine learn CHRiSM rules from examples of CSP with their solutions (CSPS);
- An industry-specific but project-independent *Standard Process Model* (right of 5th row) of the process to follow to engineer the critical system in order to certify it;
- An industry-independent *Process Enactment Trace Generator* component (right of 7th row) recording the interactions of all stakeholders with the tools used in the project to generate the *Process Effectively Enacted* (right of 6th row) during it;
- The set of *Context-Aware Critical Requirements* (left of 6th row) of the system;
- A *Context Monitor* component (left of bottom row) that maintains a runtime *Context Model* (just above it) [8] that includes flags for transient or persistent errors.
- A natural language certification *Document Generator* (right of next to bottom row).

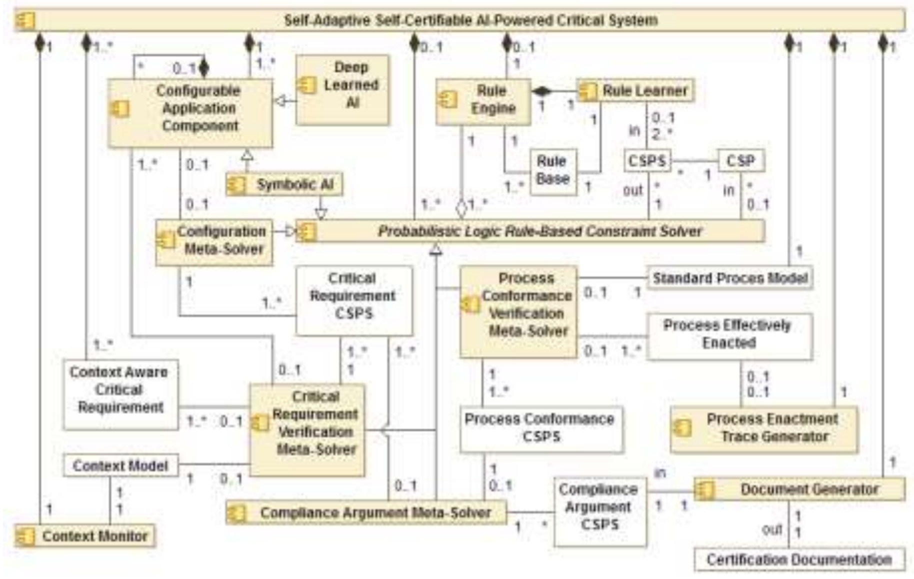
Self-Adaptive Self-Certifiable AI-Powered Critical System
Configurable Application Component
Deep Learned AI
Rule Engine
Rule Learner
Rule Base
CSPS
CSP
in
out
Symbolic AI
Configuration Meta-Solver
Probabilistic Logic Rule-Based Constraint Solver
Critical Requirement CSPS
Process Conformance Verification Meta-Solver
Standard Proces Model
Process Effectively Enacted
Context Aware Critical Requirement
Critical Requirement Verification Meta-Solver
Process Conformance CSPS
Process Enactment Trace Generator
Context Model
Compliance Argument CSPS
Document Generator
Compliance Argument Meta-Solver
Context Monitor
Certification Documentation

**Fig. 1.** Our proposed self-certifiable architecture for AI-powered critical systems

The abstract *Probabilistic Rule-Based Constraint Solver* components specialize into (a) system-specific symbolic AI components and (b) four meta-solvers, each providing a different system and industry independent autonomic capability to the system. The first of these meta-solvers is the *Critical Requirement Verification Meta-Solver* (center left of of 6th row). Taking as input constraints (a) the *Context-Aware Critical Requirements* of the system, (b) the current *Configurable Application Component* assembly and (c) the current *Context Model*, it verifies whether (b) still satisfies (a) in the context of (c), yielding as output the *Critical Requirement CSPS* (center left of 5th row). When this output is *false*, this triggers the second meta-solver, the *Configuration Meta-Solver* (left of 4th row) to infer a new *Configurable Application Component* assembly that satisfies the *Context-Aware Critical Requirements* in the new context. If the context change signals a fault, such automated reconfiguration can provide one form of fault-tolerance. Previous work [8] showed how rule-based constraint solving can automate context-aware requirement verification and reconfiguration.

The third meta-solver is the *Process Conformance Verification Meta-Solver* (center right of 5th row). Taking as input constraints (a) the *Process Effectively Enacted*, and (b) the *Standard Process Model*, it verifies whether (a) conforms to (b). Previous research [9] showed rule-based constraint solving can automate such verification. The last meta-solver is the *Compliance Argumentation Meta-Solver* (center of bottom row). Taking as input constraints the results of the first and third meta-solver, *the Critical Requirement CSPS* and *the Process Conformance CSPS* respectively, it outputs a *Compliance Argument* (right of bottom row) combining the evidence provided by both. Previous work [6] used a CHRiSM solver to build a legal argument likely to be accepted by a judge. The key idea for this last meta-solver is the similarity between this task and that of building a compliance argument likely to be accepted by a certification auditor. This last meta-solver can also be given as additional input a counter-argument to refute provided by the certification auditor. The *Document Generator* translates the logico-probabilistic *Compliance Argument* into a natural language *Certification Documentation.*

## 4 Discussion and Conclusion

In this paper, we proposed an architecture model for AI-powered critical systems allowing them to self-adapt and semi-automatically generate a certification documentation update after each adaption throughout its lifecycle. This approach reflects our position that introducing self-adaptation in critical systems will require abandoning the current one-shot certification process concluded at the development stage of systems engineering and switch to an iterative certification process spanning the whole lifecycle. This will make lowering the cost of certification through certification documentation automation a crucial issue. Our architecture addresses this issue by integrating in a unique, new synergy, architectural principles from component-based engineering, dynamic product line engineering [8], context-aware computing, autonomic computing, models-at-runtime, process-centered software engineering environment, together with AI technologies such as automated argumentation and rule-based, probabilistic constraint solving and machine learning.

We intend to evaluate the benefits of this architecture on the railway cybersecurity, AI-assisted medical imaging and industry 4.0 pilot case studies from the H2020 AI4EU project (www.ai4eu.eu/) which partially funds our research. We also intend to investigate how to overcome three limitations of the current CHRiSM engine: (1) its learning ability currently limited to learn probability parameters of rules which logical structure must be handcrafted, (2) its lacking of an interface with deep-learned AI components needed in *Cyber-Physical Systems (CPS)*, and (3) its Prolog implementation which is unpractical for real-time CPS. For these tree limitations we can leverage previous research for languages related to CHRiSM but slightly less expressive and investigate how to extend those approaches to the more general case of CHRiSM. For the first limitation, we can start from the various *logical rule structure learning* algorithms available for languages such as ProbLog and CP-Logic [4]. For the second limitation, we can start from the DeepProbLog [10] scheme to interface deep-learned reasoners with ProbLog. For the third limitation, we can start from the compiler of CHR to *Very high speed integrated circuit Hardware Description Language (VHDL)* [11] from which a fast, parallel hardware implementation can then be generated. An alternative is compiling CHRiSM to native code. An implementation of our architecture with these three CHRiSM extensions would provide a parsimoniously versatile automation framework to engineer the self-adaptive, self-certifying, machine learned, neuro-logico-probabilistic, hardware implemented, real-time AI needed by the incoming next generation of autonomous, dependability-critical CPS.

## References

1. J. Boulanger, Safety Management for Software-based Equipment, Wiley, 2013.
2. P. Lalanda, J. McCann and A. Diaconescu, Autonomic Computing: principles, design and implementation, Springer, Ed., 2013.
3. T. Frühwirth, Constraint Handling Rules, Cambridge University Press, 2009.
4. F. Riguzzi, Foundations of probabilistic logic programming: languages, semantics, inference and learning, Rivers Publishers, 2018.
5. J. Sneyers, M. Wannes and J. Vennekens, "CHRiSM: Chance Rules induce statistical models," in *Proceedings of the 6th International Workshop on Constraint Handling Rules*, Pasadena, CA, USA, 2009.
6. J. Sneyers, D. De Schreye and T. Frühwirth, "Probabilistic legal reasoning in CHRiSM," *Theory and Practice of Logic Programming (TPLP),* vol. 13, no. 4-5, 2013.
7. J. Sneyers, W. Meert, J. Vennekens, Y. Kameya and T. Sato, "CHR(PRISM)-based probabilistic logic learning," *Theory and Practice of Logic Programming,* vol. 10, 2010.
8. J. Muñoz, G. Tamura, I. Raicu, R. Mazo and C. Salinesi, "REFAS: A PLE approach for simulation of self-adaptive systems requirements," in *Proceedings of the 19th International Software Product Line Conference (SPLC'15),* Nashville, TN, USA, 2015.
9. M. Almeida da Silva, A. Mougenot, X. Blanc and R. Bendraou, "Towards Automated Inconsistency Handling in Design Models," in *22nd International Conference on Advanced Information Systems Engineering,* Hammamet, Tunisia, 2010.
10. R. Manhave, S. Dumancic, A. Kimmig, T. Demeester and L. De Raedt, "DeepProbLog: Deep Neural Probabilistic Programming," in *Proceedings of the 32nd Conference on Neural Information Processing (NeurIPS),* Montreal, Canada, 2018.
11. A. Triossi, S. Orlando, A. Raffaetá and T. Frühwirth, "Compiling CHR to parallel hardware," in *Proceedings of the 14th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming.,* Leuven, Belgium, 2012.