A Snapshot of AltaRica

Antoine Rauzy
Norwegian University of Science and Technology

Abstract

AltaRica is an object-oriented modeling language dedicated to probabilistic risk and safety analyses. It is a representative of the so-called model-based approach in reliability engineering. Since its version 3.0, it is developed by the non-profit AltaRica Association[1], which develops jointy the associated AltaRicaWizard modeling environment.

This note aims at providing a short introduction to the concepts, methods and tools, as well as the history and the applications of the AltaRica technology.

Characteristics

Although AltaRica involves concepts borrowed from object-oriented programming languages such as C++ or Python, and prototype-oriented programming languages such as JavaScript, it differs from these languages in a fundamental respect: AltaRica is a modeling language rather than a programming language. AltaRica models belongs to the general class of stochastic discrete event systems[1]. Technically, the underlying mathematical framework of AltaRica is the notion of guarded transition systems[2]. Once designed and validated, an AltaRica model is compiled into a guarded transition system. This guarded transition system which can then be assessed by different tools: stepwise simulator, compiler to fault trees, compiler to Markov chains, sequence generator, stochastic simulator, model-checker... All these tools contribute eventually, each with its own features, to assess the reliability of the system under study by calculating various probabilistic risk indicators and by extracting critical accident scenarios.

AltaRica 3.0, the last version of the language, can be described by the following equation.

$$\text{AltaRica} = \text{GTS} + \text{S2ML}^2$$

GTS stands for "Guarded Transition Systems", and S2ML stands for "System Structure Modeling Language", a versatile set of model structuring constructs stemmed from object-oriented and prototype-oriented programming languages.

AltaRica implements the "S2ML+X" paradigm[4] that relies on the idea that any behavioral modeling language consists of two parts: a mathematical framework in which the behavior is actually described (the X), and a set of constructs (functions, records, classes, prototypes...) to architect models (S2ML). The choice of the mathematical framework depends on which properties of the system the analyst is interested in. Examples of such frameworks are ordinary differential equations for Matlab/Simulink[5] and Modelica[6], Mealy machines for Lustre and Scade[7], and indeed guarded transition systems for AltaRica. Structuring constructs are independent to a large extent of the mathematical framework. S2ML is generic in this sense that it can be applied to all types of behavioral modeling used in systems engineering.

History

The design of AltaRica started at the end of the nineties at the computer science department of Bordeaux University (LaBRI). The rational for the creation of a new modeling language was to overcome difficulties encountered by safety analysts (in avionic, nuclear, automotive and oil and gas indutries) with "classical" modeling formalisms such as fault trees, Markov chains or stochastic Petri nets. These formalisms lack actually either of expressive power, or of structuring constructs, or both. The first scientific articles about the language were published from 1998 to 2000[8;9;10;11]. The original version of the language relied of three

[1] www.altarica-association.org

[2] This equation echoes the title of the famous book "Data structures + algorithms = programs"[3].

technologies: finite state automata that were extensively studied by the LaBRI's team working of the formal methods for software verification [12], structured programming taking inspiration of the modeling language Lustre [7], and constraint programming [13]. This last technology, although elegant and powerful, proved to be hard to be made efficient in practice. Constraint resolution was too computationally expensive to scale on industrial size systems. The LaBRI team went on working however on this original version, mainly for educational purposes, improving tools over the years [14;15;16;17].

A first turn has been therefore taken with the design of a data-flow version of the language [18;19]. In AltaRica data-flow, variables are updated by propagating values in a fixed order. This order is determined at compile time, from the annotations given in the model. AltaRica data-flow raised a significant academic and industrial interest. Integrated modeling environments have been developed for the language: Cecilia OCAS by Dassault Aviation, Simfia v2 by EADS-Apsys and Safety Designer by Dassault Systèmes (this latter tool was initially a clone of Cecilia OCAS, but evolved separately afterward). Successful industrial applications have been realized [20;21;22;23]. For example, AltaRica data-flow was used to certify the flight control system of the aircraft Falcon 7X (Dassault Aviation). A number of PhD theses were also dedicated to the language and its use in various contexts [24;25;26;27;28;29;30;31]. In a word, AltaRica Data-Flow reached scientific and industrial maturity. It is still daily used for a wide variety of applications.

Experience showed however that AltaRica data-flow could be improved in several ways, hence justifying to seriously rework the language. This rework gave eventually raise to AltaRica 3.0 [32;33] which improves AltaRica data-flow into several directions. The syntax of AltaRica 3.0 is closer to Modelica than to AltaRica data-flow, so to facilitate bridges between multiphysics modeling and simulation and probabilistic risk and safety analyses. Object-oriented and prototype-oriented structuring constructs have been assembled so to give the language a versatile and coherent set of structuring constructs (S2ML), which probably the most complete of all existing behavioral modeling languages. Moreover, AltaRica 3.0 semantics has been reinforced (via GTS), which opens new opportunities in terms of assessment of models.

Guarded Transition Systems

Guarded transition systems belong to the family mathematical models of computation gathered under the generic term of (stochastic) finite-state machines or (stochastic) finite-state automata. They have been introduced in 2008 [2] and later refined [34].

To illustrate ideas behind guarded transition systems, consider a motor pump that is normally in stand-by, that can be started on demand and stopped when there is no more demand. Assume morever that this pump may fail when in operation, with a certain failure rate λ, and that it can also fail on-demand with a certain probability γ, Assume finally that the pump can be repaired, with a certain mean time to repair τ.

We can then represent the behavior of this pump by means of a (stochastic) finite state automaton pictured in Fig. 1.

From outside, the motor pump can be seen as a black box with an input flow of liquid `in`, an input flow of information `demand` and an output flow of liquid `out`, i.e. as a transfer function that given the values `in` and `demand` calculates the value `out`. In the framework on reliability studies, the behavior of systems must be abstracted out to avoid the combinatorial explosion of situations to look at. Flows are thus typically abstracted as Boolean values, true interpreted as the presence of the flow and false as its absence.

The equation linking `in` and `demand` to `out` cannot be written directly since the motor pump has an internal state. Namely, we can consider that the pump can be in three states: `STANDBY`, `WORKING` or `FAILED`. On the figure, states are represented as rounded rectangle. The output flow `out` takes the value true if and only if the pump is working and the input flow `in` is true (hence the equation on the right hand side of the figure above).

A fundamental abstraction made by finite state automata consists in considering that the system under study can change of state only under the occurrence of an event. In between two occurrences of events, nothing changes. Occurrences of events are described by means of transitions, represented as arrows on the figure. In guarded transition systems, a transition is labeled with an event, has a certain pre-condition called the guard of the transition and a certain effect called the action of the transition. For instance, the event `failure` can only occur in the state `WORKING`. Its effect is to make the pump pass from this state to the state `FAILED`. The event `start` can occur if the pump is the state `STANDBY` and if

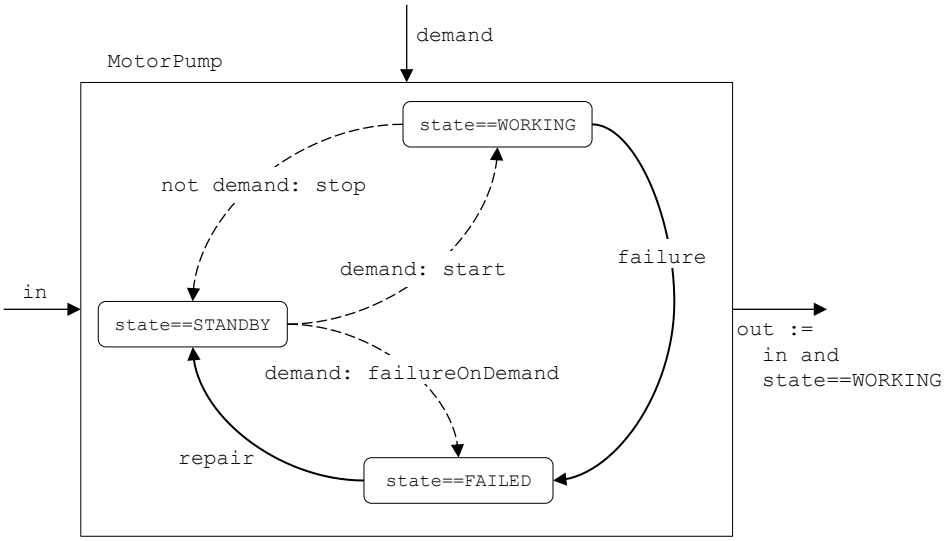

Figure 1: Model of a motor operated pump

the input flow `demand` is true. Its effect is to make the pump pass from the state `STANDBY` to the state `WORKING`. And so on.

Now, some changes of states may take time, while some other happen as soon as they are possible. For instance, a failure takes a certain time before occurring, while the pump is started as soon as needed (at least at the level of abstraction of reliability models). Guarded transition systems associate delays with events, and thus transitions. These delays can be either deterministic as for the event `start`, or stochastic as for the event `failure`. On the figure, deterministic delays are represented by dashed arrows while stochastic ones are represented by plain arrows.

To finish, transitions can be in competition in a state. For instance, the transition `stop` is in competition with the transition `failure` in the state `WORKING`. This competition is however not a real one as the transition `stop` is immediately fired (performed) when the input flow `demand` ceases to be true. A real competition occurs between the transitions `start` and `failureOnDemand` in the state `STANDBY`. Both are fired immediately when the input flow `demand` becomes true. In guarded transition systems, it is possible to associate a probability of occurrence to each transition in competition, namely γ to `failureOnDemand` and $1 - \gamma$ to `start` in our example.

Eventually, the AltaRica code for the guarded transition system we sketched is given in Fig. 2. The motor pump is represented as a `block`, i.e. as a container for basic elements. The block declares four variables: a state variable `_state` that takes its value in the domain (set of symbolic constants) `MotorPumpState`, and three Boolean flow variables `demand`, `in`, and `out`. Initially, `_state` takes the value `STANDBY`. The transfer function is represented by means of the assertion. Assertions tell how to calculate the values of output flow variables from the values input flow variables and the values of state variables.

The block `MotorPump` declares also five events and as many transitions. Guards of transitions are Boolean conditions on state and flow variables. Actions of transitions modify the values of state variables. Events are associated with delays and possibly expectations (which are used to calculate probabilities of occurrence of transitions in competition). The description of both delays and expectations may involve parameters.

System Structure Modeling Language

In general, systems under study are not made of a single, simple components as the above motor operated pump. Rather, they consists of a network of such components, interacting and organized in a hierarchical way. In other words, systems are architected.

To reflect the architecture of the system in the model, one needs dedicated constructs. This is where S2ML (system structure modeling language) comes into the play. S2ML emerged first as the set of structuring constructs for AltaRica 3.0. Then, it has been studied on its own[35;4]. As of today, S2ML

```
domain MotorPumpState {STANDBY, WORKING, FAILED}

block MotorPump
    MotorPumpState _state (init = STANDBY);
    Boolean demand, in, out (reset = false);
    event start (delay = Dirac(0), expectation=gamma);
    event failureOnDemand (delay = Dirac(0), expectation=1-gamma);
    event stop (delay = Dirac(0));
    event failure (delay = exponential(lambda));
    event repair (delay = exponential(1/tau));
    parameter Real lambda = 1.0e-4;
    parameter Real tau = 8;
    parameter Real gamma = 0.02;
    transition
        start: demand and _state==STANDBY -> _state := WORKING;
        failureOnDemand: demand and _state==STANDBY -> _state := FAILED;
        stop: not demand and _state==WORKING -> _state := STANDBY;
        failure: _state==WORKING -> _state := FAILED;
        repair: _state==FAILED -> _state := STANDBY;
    assertion
        out := in and _state==WORKING;
end
```

Figure 2: AltaRica model implementing the guarded transition system pictured in Fig. 1

gathers in a coherent way a versatile set of structuring constructs stemmed from object-oriented and prototype-oriented programming[36;37].

S2ML consists of height key concepts: the concepts of port, connection, container, prototype, class, cloning, instantiation, inheritance and aggregation.

Ports: Ports are basic modeling elements such as domains, state and flow variables, events and parameters.

Connections: Connections are relations amongst ports such as definitions of domains, probability distributions associated with state variables, definitions of parameters, transitions and assertions.

Containers: Containers gather declarations of ports and connections as well as of other containers forming nested hierarchies. The block describing the motor pump is such a container.

Prototype and cloning: Prototypes are individual containers. In AltaRica, blocks are prototypes. When the system under study contains two or more similar parts, it would be both tedious and error prone to duplicate the code. Cloning provides a solution to this problem: the first part is described, then the others are just obtained by cloning the first one, see Fig. 3 (upper part) for an illustration.

The dot notation makes it possible to access to ports declared in nested blocks, as illustrated by the assertion in the above code.

Classes and Instances: It is often the case that some modeling components are reused from model to model. It is then possible to create libraries of on-the-shelf modeling components and to instantiate them into model.

Classes are used for this purpose. They are just containers (or prototypes) defined outside the model. Instances of classes, also called objects, are clones of these containers defined outside the model, see Fig. 3 (lower part) for an illustration.

Inheritance: Inheritance is the mechanism by which one specializes modeling component. If a derived component D inherits from a base component B, all modeling elements of B are reproduced in D. D may declare additional ports, connections and containers.

Aggregation: Cloning and instantiation create components inside components. The inner component is part of the outer one. There are cases however where a component is used by another one, without being part of this component. This mechanism is reflected into model by a mechanism called aggregation.

```
block System
    // ...
    block Line1
        // description of line 1
    end
    clones Line1 as Line2;
    // ...
    assertion
        out := Line1.out or Line2.out;
end
```

```
class MotorPump
    // description of the motor pump.
end

block System
    // ...
    MotorPump P1; // 1st instance
    MotorPump P2; // 2nd instance
    // ...
end
```

Figure 3: Illustrations of the use S2ML constructs

Aggregation is extremely useful to project a functional architecture onto a physical one, and to represent so-called functional chains[38].

AltaRica 3.0 involves a few other constructs, such as a powerful mechanism to synchronize events. The essential has however been presented above.

Adding S2ML on top of a mathematical framework (GTS in the case of AltaRica), makes it possible to pass automatically and at no cost from the model as designed, which reflects the architecture of the system under study, to the model as assessed from which calculations of indicators and simulations can be performed efficiently. The transformation preserves the semantics of the models and is reversible for the most part: results of calculations and simulations are directly interpretable in the model as designed.

A recent trend in the AltaRica community is the design of modeling patterns[33]. Patterns are pervasive in engineering. They have been developed for instance in the field of technical system architecture[39], as well as in software engineering[40]. They prove to be extremely useful in reliability engineering as well, as they ease considerably the design and the maintenance of models. They are also a tool for risk analysts to communicate about the models they develop and share.

Tooling and Applications

In industrial practice, AltaRica models are used for four main purposes (the list below is however non limitative):

- To develop a common understanding, amongst stakeholders, about how the system under study works and may fail;
- To formally ensure that system under study is safe enough to be operated;
- To optimize maintenance policies;
- To assess the average production (or loss of production) of units subject to failures, malfunctions, human errors...

As these applications require different types of simulations and calculations, several tools have been developed, including:

- Stepwise simulators that make it possible to play interactively scenarios of evolution.
- Compilers to lower level modeling formalisms, primarily fault trees but also Markov chains, so to benefit of existing efficient assessment algorithms.
- Stochastic simulators that make it possible to calculate a wide range of risk indicators.

– Sequence generators and model-checkers that make it possible to validate models and to extract critical scenarios of failure verifying various properties.

References

[1] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems.* Springer, New-York, NY, USA, 2008.

[2] Antoine Rauzy. Guarded transition systems: a new states/events formalism for reliability studies. *Journal of Risk and Reliability*, 222(4):495–505, 2008.

[3] Niklaus Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, Upper Saddle River, New Jersey 07458, USA, 1976.

[4] Antoine Rauzy and Cecilia Haskins. Foundations for model-based systems engineering and model-based safety assessment. *Journal of Systems Engineering*, 22:146–155, 2019.

[5] Harold Klee and Randal Allen. *Simulation of Dynamic Systems with MATLAB and Simulink.* CRC Press, Boca Raton, FL 33431, USA, February 2011.

[6] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach.* Wiley-IEEE Press, Hoboken, NJ 07030-5774, USA, 2015.

[7] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[8] Jean-Pierre Signoret, Sylvain Lajeunesse, Gérald Point, Philippe Thomas, Alain Griffault, and Antoine Rauzy. The Altarica Language. In Lydersen, Hansen, and Sandtorv, editors, *Proceedings of European Safety and Reliability Association Conference, ESREL'98*, pages 1327–1334, Trondheim, Norway, June 1998. Balkerna, Rotterdam.

[9] Gérald Point and Antoine Rauzy. AltaRica: Constraint automata as a description language. *Journal Européen des Systèmes Automatisés*, 33(8–9):1033–1052, 1999.

[10] André Arnold, Alain Griffault, Gérald Point, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 34:109–124, 2000.

[11] Gérald Point. *AltaRica: Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement.* Thèse de doctorat, LaBRI – Université Bordeaux I, January 2000.

[12] André Arnold. *Finite Transition Systems.* C.A.R Hoare. Prentice Hall, Upper Saddle River, New Jersey, USA, 1994.

[13] Pascal van Hentenryck. *Constraint Satisfaction in Logic Programming.* The MIT Press, Cambridge, MA, USA, March 1989.

[14] Aymeric Vincent. *Conception et réalisation d'un vérificateur de modèles AltaRica.* Thèse de doctorat, Université de Bordeaux, December 2003.

[15] Alain Griffault, Gérald Point, and Aymeric Vincent. Vérification formelle des modèles AltaRica. In Hermès, editor, *Actes du Congrès de maîtrise des risques et de sûreté de fonctionnement, Lambda-Mu'14*, Bourges (France), October 2004.

[16] Alain Griffault and Aymeric Vincent. The mec 5 model-checker. In *Proceedings of the 16th International Conference on Computed Aided Verification (CAV 2004)*, volume 3114 of *Lectures Notes in Computer Science*, pages 488–491, Boston, MA, USA, July 2004. Springer Verlag.

[17] Fabien Kuntz, Stéphanie Gaudan, Christian Sannino, Éric Laurent, Alain Griffault, and Gérald Point. Model-based diagnosis for avionics systems using minimal cuts. In *Proceedings of 22th Worshop on Principles of Diagnosis (DX-2011)*, pages 138–145, Murnau, Germany, October 2011.

[18] Antoine Rauzy. Modes Automata and their Compilation into Fault Trees. *Reliability Engineering and System Safety*, 78(1):1–12, October 2002.

[19] Marie Boiteau, Yves Dutuit, Antoine Rauzy, and Jean-Pierre Signoret. The altarica data-flow language in use: Assessment of production availability of a multistates system. *Reliability Engineering and System Safety*, 91(7):747–755, July 2006.

[20] Romain Bernard, Jean-Jacques Aubert, Pierre Bieber, Christophe Merlini, and Sylvain Metge. Experiments in model-based safety analysis: flight controls. In Jean-Marc Faure, editor, *Proceedings of IFAC workshop on Dependable Control of Discrete Systems*, pages 43–48, Cachan, France, June 2007. Curran Associates, Inc.

[21] Pierre Bieber, Jean-Paul Blanquart, Guy Durrieu, David Lesens, Jocelyn Lucotte, Frédéric Tardy, Michel Turin, Christel Seguin, and Eric Conquet. Integration of formal fault analysis in assert: Case studies and lessons learnt. In *Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008*, Toulouse, France, January 2008. SIA (electronic proceedings). code R-2008-01-2B04.

[22] Xavier Quayzin and Emmanuel Arbaretier. Performance modeling of a surveillance mission. In *Proceedings of the Annual Reliability and Maintainability Symposium, RAMS'2009*, pages 206–211, Fort Worth, Texas, USA, January 2009. IEEE.

[23] Romain Bernard, Sylvain Metge, François Pouzolz, Pierre Bieber, Alain Griffault, and Marc Zeitoun. Altarica refinement for heterogeneous granularity model analysis. In *Actes du congrès Lambda-Mu'16*, page 2B, Avignon, France, October 2008. IMdR (actes électroniques).

[24] Claire Pagetti. *Extension temps réel du langage AltaRica.* Thèse de doctorat, École Centrale de Nantes et de l'Université de Nantes, 2004.

[25] Christophe Kehren. *Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement.* Thèse de doctorat, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace (SUPAERO), Toulouse, France, 2005.

[26] Minh Thang Khuu. *Contribution à l'accélération de la simulation stochastique sur des modèles AltaRica Data Flow.* Thèse de doctorat, Université de la Méditerranée (Aix-Marseille II), 2008.

[27] Sophie Humbert. *Déclinaison d'exigences de sécurité, du niveau système vers le niveau logiciel, assistée par des modèles formels.* Thèse de doctorat, Université de Bordeaux I, April 2008.

[28] Laurent Sagaspe. *Allocation sûre dans les systèmes aéronautiques : modèlisation, vérification et génération.* Thèse de doctorat, Université de Bordeaux, December 2008.

[29] Romain Bernard. *Analyse de Sécurité multi-systèmes.* Thèse de doctorat, Université de Bordeaux, November 2009.

[30] Romain Adeline. *Méthodes pour la validation de modèles formels pour la Sûreté de Fonctionnement et extension aux problèmes multi-physique.* Thèse de doctorat, Université de Toulouse, March 2011.

[31] Timothy Kombe. *Modélisation de la propagation des fautes dans les systèmes de production.* Thèse de doctorat, Institut National des Sciences Appliquées de Lyon et Ecole Nationale Supérieure Polytechnique de Yaoundé, 2011.

[32] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. *AltaRica 3.0: language specification.* AltaRica Association, 2017.

[33] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. Altarica 3.0 in 10 modeling patterns. *International Journal of Critical Computer-Based Systems*, 9(1–2):133–165, 2019.

[34] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. Altarica 3.0 assertions: the why and the wherefore. *Journal of Risk and Reliability*, 231(6):691—-700, September 2017.

[35] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. From models of structures to structures of models. In *IEEE International Symposium on Systems Engineering (ISSE 2018)*, Roma, Italy, October 2018. IEEE. Best paper award.

[36] Mauricio Abadi and Luca Cardelli. *A Theory of Objects.* Springer-Verlag, New-York, USA, 1998.

[37] James Noble, Antero Taivalsaari, and Ivan Moore. *Prototype-Based Programming: Concepts, Languages and Applications.* Springer-Verlag, Berlin and Heidelberg, Germany, 1999.

[38] Jean-Luc Voirin. Method and tools for constrained system architecting. In *Proceedings 18th Annual International Symposium of the International Council on Systems Engineering (INCOSE 2008)*, pages 775–789, Utrecht, The Netherlands, June 2008. Curran Associates, Inc.

[39] Mark W. Maier. *The Art of Systems Architecting.* CRC Press, Boca Raton, FL 33431, USA, 2009.

[40] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison-Wesley professional computing series. Addison-Wesley, Boston, MA 02116, USA, October 1994.