State of the Art of the S2C Project

Warning: This State of the Art has been mainly done at the beginning of the project in 2019. It doesn't reflect the situation in 2023.

DATE: 07/04/2022

Summary

This deliverable aims at providing the state of the art of the project. One part addresses consistency between systems engineering and safety and a second part addresses the state of the art for current MBSA methodologies and tools & MBSA dissemination.

Author(s)	*Function(s) & name(s)*	*IRT SystemX* *IRT Saint Exupéry*	 *S. Delavault* [illegible]
Checker(s)	*Function(s) & name(s)*	*WP Leaders*	*S. Guilmeau* [illegible]
Approver	*Function & name*	*Project Leader*	*J. Perrin* J. Perrin [illegible]

Evolutions

Version	Date	Modified §	Modification summary	Modified by
0	05/02/2020	All	Creation	All
1	01/07/2020	All	Completion of state of the art. Remarks from Partner on V0 Internal IRT remarks	All
2	13/10/2021	Refer to margin marks	Completion of state of the art. Remarks from Partner on V1 Internal IRT remarks	All
3	07/04/2022	All §6.3	Update of table contents Typo correction Headers: Update of header with the new IRT Saint Exupéry logo §6.3: • Update to add new MBSA use case from S2C members Update of the return of experience	Simon Delavault

Table of contents

List of Tables

List of Figures

List of symbols, acronyms and abbreviations

AADL:	Architecture Analysis and Design Language
ACTL:	Action Computation Tree Logic
API:	Application Programming Interface
CCA:	Common Cause Analysis
CTL:	Computation Tree Logic
DAL:	Design assurance level
DL:	Description Logics
ESACS:	Enhanced Safety Assessment for Complex Systems
FC:	Failure Conditions
FHA:	Functional Hazard Analysis
FMEA:	Failure Modes and Effects Analysis
FMECA:	Failure Mode Effects and Criticality Analysis
FMI:	Functional Mock-up Interface
FMU:	Functional Mock-up Unit
FTA:	Fault Tree Analysis
INCOSE:	International Council on Systems Engineering
ISAAC:	Improvement of Safety Activities on Aeronautical Complex Systems
ISML:	Intention-Specific Modeling Languages
LTL:	Linear Temporal Logic
MBSA:	Model-Based Safety Analysis
MBSE:	Model-Based Systems Engineering
OMG:	Object Management Group
OSLC:	Open Services for Lifecycle Collaboration
OWL:	Web Ontology Language
P&O:	People and Organization
PDES:	Product Data Exchange Specification
PLM:	Product Lifecycle Management
PSSA:	Preliminary System Safety Assessment
RCA:	Root Cause Analysis
RDF:	Resource Description Framework
REST:	REpresentational State Transfer
SIL:	Safety Integrity Level
SSA:	System Safety Assessment
STEP:	Standard for the Exchange of Product Model Data
TGG:	Triple Graph Grammars
UML:	Unified Modeling Language
ZRA:	zonal risk assessment

1 Introduction

1.1 Context of the project

The S2C project was born thanks to a strong industrial need to improve current development processes and to ensure consistency between safety analyses and system design, which today requires a significant effort.

Indeed, the development of systems like aircrafts is made difficult by two major factors: the inherent complexity of such system and the complexity of the organization required to build and maintain them. A large number of stakeholders are involved throughout the lifecycle. They have various concerns that are not necessarily consistent with each other. Each stakeholder has its own viewpoints and descriptions, focused on his concerns and based on the use of dedicated languages, formalisms, and tools. These viewpoints are highly interrelated, which produce overlaps. Clearly, these overlaps and all elements mentioned above may possibly introduce inconsistencies.

In the S2C project, we will focus on the interactions between two particular disciplines: systems engineering and safety analyses. The challenge is to develop methodologies to manage inconsistencies and to maintain a coherent state over the development cycle. These methodologies should be validated through a representative case study supporting by tools demonstrator in order to define tooling specification.

The main objectives are to:

- Improve confidence in safety analysis by aligning and maintaining consistency with system definition models.
- Provide safety specialists with more efficient means to understand and analyze complex systems.
- Facilitate the use of model-based approaches in compliance with the requirements of certification bodies (for the MBSA approach, there are no requirements issued by certification bodies).
- Reduce the number of iterations between system definition and related safety analyses, in order to limit costs and delays, by enabling efficient MBSE/MBSA interfaces or MBSE/FTA interfaces.
- Control changes during the product development cycle; reduce risks due to redesign.

In order to address these issues, the project was divided into 4 work packages:

- WP1: Definition of a global process for ensuring and maintaining system/safety consistency all along the life cycle
- WP2: Methods and means of implementing and maintaining system/safety consistency for the integrated system and system levels (horizontal links)
- WP3: Methods and means of setting and maintaining consistency of safety data between different system levels (vertical links)
- WP4: Modeling methodology for safety

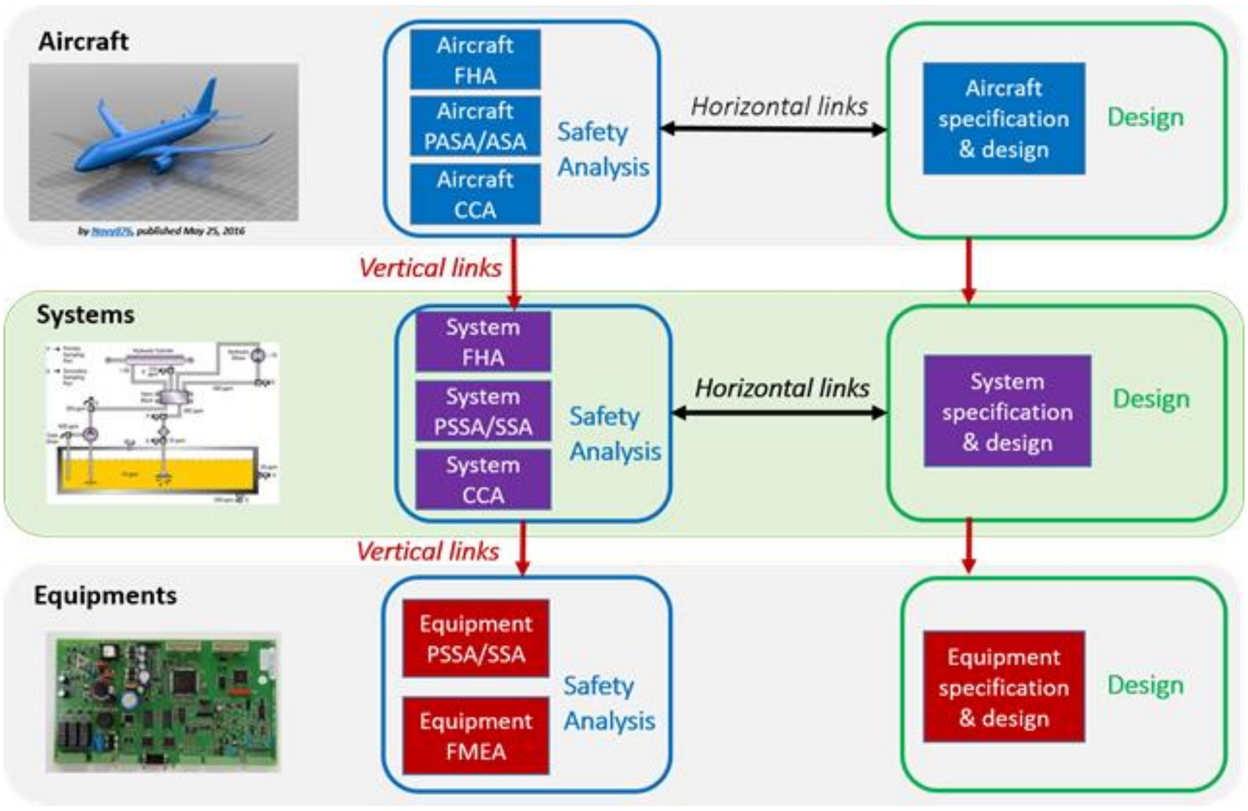

Figure 1: Positionning the activities of the project

The S2C project is mainly focused on the aeronautics domain.

In this context, we have to take into account certification requirements and constraints of aircraft systems in order to meet the project objectives listed above. The Safety global process and approach, deemed to answer to certification requirements and contraints, is the Guidelines for Development of Civil Aircraft and Systems [ref SAE ARP4754 rev.A / Eurocae ED-79A]. We will use also the Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment [ref SAE ARP4761 /--]. These processes support certification of aircraft systems .

1.2 Purpose and organization of the document

This deliverable aims at providing the state of the art of the project scope. One part addresses consistency between systems engineering and safety, through the following paragraphs:

- Key concepts (§2)
- Theoretical activities that have to be led to manage consistency (§3)
- Some existing consistency methods (§4)
- Tools and Platforms to answer consistency issue (§5)
- Synthesis of methods and tools for consistency management in S2C (§6)

The other part addresses the state of the art for current MBSA methodologies and tools & MBSA dissemination, through § 7 and Appendix A.

1.3 Referenced documents

1.3.1 S2C referenced documents

Title	Reference
S2C Glossary	NT-2016-168
S2C 3DExperience experiment	IRT Saint Exupéry NT-S085L02-015 IRT SystemX ISX-S2C-DOC-351
S2C Lot 4 – Modelling Guide and Validation Report	IRT Saint Exupéry LIV-S085L01-001 IRT SystemX ISX-S2C-LIV-1001

1.3.2 External referenced documents

Title	Reference
[ARP4754A/ED-79A]	ARP4754 Rev A Issue 1996-11, Revised 2010-12 : Guidelines for Development of Civil Aircraft and Systems
[ARP4754B/ED-79B]	ARP4754 Rev B Issue 1996-11, Revised 2022-04 : Guidelines for Development of Civil Aircraft and Systems
[ARP4761]	Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
[MOISE]	LIV-S-014-S2.21-61-457 (issue V1) from the IRT Saint Exupery document base. This document is accessible by any employee of finished MOISE member and by the restricted team of S2C.
[ISC]	ISX-IS-ISC-LIV-0310-v6.1-L1.1 from the IRT SystemX. Collaborative Platform Requirements (in French).

1.3.3 Referenced publications

A. Arnold, A. G. (2000). he AltaRica formalism for describing concurrent systems. *IOS Press Inf. 40, issue 2-3*, 109-124.

A.Rauzy. (2008). Guarded Transition Systems: a new States/Events Formalism for Reliability Studies. *Journal of Risk and Reliability*, 495--505.

Adeline, R. (2011). *Méthodes pour la validation de modèles formels pour la sûreté de fonctionnement et extension aux problèmes multi-physiques.* Ph.D. dissertation, Toulouse, ISAE.

AFNORNFX60-500. (s.d.). *Terminologie relative à la fiabilité, maintenabilité, disponibilité.*

AFNORNFX60-503. (s.d.). *Introduction à la disponibilité.*

Alain Griffault, A. V. (2004). *The Mec 5 model-checker. Computer Aided Verification.* Boston, United States.

AP233SE website. (s.d.). Consulté le 08 24, 2016, sur http://homepages.nildram.co.uk/~esukpc20/exff2005_05/ap233/

B. Kaiser, P. L. (2003). A new component concept for fault trees. *8th Australian workshop on Safety critical systems and software.*

Basole, R. C., Qamar, A., Park, H., Paredis, C. J., & McGinnis, L. F. (2015). Visual Analytics for Early-Phase Complex Engineered System Design Support. *IEEE Computer Graphics and Applications, 35*, 41-51.

Batteux M, P. T. (2018). ALTARICA WIZARD: AN INTEGRATED MODELING AND SIMULATION ENVIRONMENT FOR ALTARICA 3.0. *Congrés Lambda Mu 21 « Maîtrise des risques et transformation numérique : opportunités et menaces ».* Reims.

Batteux M., P. T. (2017). AltaRica 3.0 assertions: The whys and wherefores. *Proceedings of the Institution of Mechanical Engineers , Part O: Journal of Risk and Reliability*.

Batteux, M., Prosvirnova, T., & Rauzy, A. (2019). Model Synchronization: A Formal Framework for the Management of Heterogeneous Models.

Benjamin Aupetit, M. B.-M. (2015). Improving performance of the {AltaRica} 3.0 stochastic simulator. *Proceedings of Safety and Reliability of Complex Engineered Systems: ESREL 2015.*

Birolini, A. (2010). Reliability Engineering: Theory and Practice.

Bitetti, L., De Ferluc, R., Mailland, D., Gregoris, G., & Capogna, F. (2019). Model Based Approach for RAMS Analyses in the Space Domain with Capella Open-Source Tool.

Boehm, B., & In, H. (1996). Identifying quality-requirement conflicts. *Proceedings of the Second International Conference on Requirements Engineering*, (pp. 218-).

Boiteau, M. D.-P. (2006). he AltaRica Data-Flow language in use: Assessment of Production Availability of a MultiStates System. *Reliability Engineering and System Safety*, 747-755.

Bouissou M. (2005). Automated Dependability Analysis of Complex Systems with the KB3 Workbench: the Experience of EDF R&D.

Bouissou M, V. N. (1991). Knowledge Modelling and Reliability Processing: Presentation of the FIGARO Language and Associated Tools. *SAFECOMP'91.* Trondheim.

Broy, M., & Ştefănescu, G. (2001). The algebra of stream processing functions. *Theoretical Computer Science, 258*, 99-129. doi:https://doi.org/10.1016/S0304-3975(99)00322-9

Cambridge online. (s.d.). Récupéré sur https://dictionary.cambridge.org/fr/dictionnaire/anglais/inconsistency

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal, 45*, 621-645.

Deschamps, F. (2014). Definition and application of a troubleshooting methodology on complex electronic hardware. *9e Congrès de Maîtrise des Risques et Sûreté de Fonctionnement.* Dijon.

Dominique Riera, J. A. (2018). Et si on faisait du retour d'expérience sur la testabilité ? *ongrès Lambda Mu 21 « Maîtrise des risques et transformation numérique : opportunités et menaces ».* Reims.

Easterbrook, S. (1991, 9). Handling Conflict between Domain Descriptions with Computer-Supported Negotiation. *Knowl. Acquis., 3*, 255–289. doi:10.1016/1042-8143(91)90007-A

Easterbrook, S., Finkelstein, A., Kramer, J., & Nuseibeh, B. (1994). Coordinating distributed ViewPoints: the Anatomy of a Consistency Check. *Concurrent Engineering, 2*, 209-222. doi:10.1177/1063293X9400200307

Egyed, A. (2011). Automatically Detecting and Tracking Inconsistencies in Software Design Models. *IEEE Transactions on Software Engineering, 37*, 188-204.

Egyed, A., Zeman, K., Hehenberger, P., & Demuth, A. (2018). Maintaining Consistency across Engineering Artifacts. *Computer, 51*, 28-35.

Ehrig, M., & Sure, Y. (2004). Ontology Mapping - An Integrated Approach. (pp. 76-91). Springer Verlag.

Emmanuel Clement, D. R. (2018). ANALYSE DES PERFORMANCES DE DIAGNOSTIC D'UN SYSTEME AU MOYEN D'UNE MODELISATION ALTARICA 3.0. *Congrès Lambda Mu 21 « Maîtrise des risques et transformation numérique : opportunités et menaces ».* Reims.

EXPRESS. (s.d.). Récupéré sur Logiciel eXpress de DSI internationnal: https://www.dsiintl.com/products/express/

Extendsim. (s.d.). Récupéré sur Power tool for simulation: https://extendsim.com/

Feldmann, S. (2019). Diagnosis and Resolution of Inconsistencies in Heterogeneous Models of Automated Production Systems.

Feldmann, S., Herzig, S. J., Kernschmidt, K., Wolfenstetter, T., Kammerl, D., Qamar, A., . . . Vogel-Heuser, B. (2015). Towards effective management of inconsistencies in model-based engineering of automated production systems. *IFAC-PapersOnLine, 28*(3), 916-923. Récupéré sur http://dx.doi.org/10.1016/j.ifacol.2015.06.200

Finkelstein, A. C., Gabbay, D., Hunter, A., Kramer, J., & Nuseibeh, B. (1994, 8). Inconsistency Handling in Multiperspective Specifications. *IEEE Trans. Softw. Eng., 20*, 569–578. doi:10.1109/32.310667

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., & Goedicke, M. (1992). Viewpoints: A Framework For Integrating Multiple Perpsectives In System Development. *International Journal of Software Engineering and Knowledge Engineering, 02*, 31-57. doi:10.1142/S0218194092000038

Gibb, A. G., & Isack, F. (2001). Client drivers for construction projects: implications for standardization. *Engineering Construction and Architectural Management, 8*, 46-58. doi:10.1046/j.1365-232x.2001.00184.x

Giese, H., & Wagner, R. (2009). From model transformation to incremental bidirectional model synchronization. *Software & Systems Modeling, 8*, 21-43.

Glabbeek, R. J., & Weijland, W. P. (1996, 5). Branching Time and Abstraction in Bisimulation Semantics. *J. ACM, 43*, 555–600. doi:10.1145/233551.233556

Gudemann, M. &. (2010). A Framework for Qualitative and Quantitative Formal Model-Based Safety Analysis. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*.

Harel, D., & Rumpe, B. (2004, 10). Meaningful Modeling: What's the Semantics of "Semantics"? *Computer, 37*, 64–72. doi:10.1109/MC.2004.172

Hehenberger, P., Egyed, A., & Zeman, K. (2010). Consistency checking of mechatronic design models. *Proceedings of the ASME Design Engineering Technical Conference, 3*(PARTS A AND B), 1141-1148.

Heitmeyer, C. L., Jeffords, R. D., & Labaw, B. G. (1996, 7). Automated Consistency Checking of Requirements Specifications. *ACM Trans. Softw. Eng. Methodol., 5*, 231–261. doi:10.1145/234426.234431

Herzig, S. J. (2015). *A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering.* Ph.D. dissertation, Georgia Institute of Technology.

Herzig, S. J., & Paredis, C. J. (2014). A Conceptual Basis for Inconsistency Management in Model-based Systems Engineering. *Procedia CIRP, 21*, 52-57. doi:https://doi.org/10.1016/j.procir.2014.03.192

Herzig, S. J., Qamar, A., & Paredis, C. J. (2014). An Approach to Identifying Inconsistencies in Model-based Systems Engineering. *Procedia Computer Science, 28*, 354-362. doi:https://doi.org/10.1016/j.procs.2014.03.044

Hoare, C. A. (1978, 8). Communicating Sequential Processes. *Commun. ACM, 21*, 666–677. doi:10.1145/359576.359585

INCOSE. (2010). Integrating Systems Engineering Information with AP233.

ISO 10303-233:2012. (s.d.). Consulté le 08 24, 2016, sur http://www.iso.org/iso/catalogue_detail.htm?csnumber=55257

ISO 10303-239:2005. (s.d.). Consulté le 08 24, 2016, sur http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=38310

ISO/IEC/IEEE 24765 Systems and software engineering — Vocabulary. (2017, 9). *ISO/IEC/IEEE 24765:2017*, 1-522.

ISO/IEC/IEEE Systems and software engineering -- Architecture description. (2011, 12). *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, 1-46.

ISO15026. (1998). *Technologies de l'information — Niveaux d'intégrité du système et du logiciel.*

ISO15288. (2015, 5). *ISO/IEC/IEEE 15288:2015 Systems and software engineering — System life cycle processes.*

J., J., Chilenski, & Kerstetter, M. S. (s.d.). SAVI AFE 61S1 Report - Summary Final Report.

J., J., Chilenski, & Ward, D. T. (s.d.). SAVI AFE 61 Report - Summary Final Report.

Jackson, M. (1997, 1). The Meaning of Requirements. *Ann. Softw. Eng., 3*, 5–21.

Kabir, S., Aslansefat, K., Sorokos, I., Papadopoulos, Y., & Gheraibia, Y. (2019). A Conceptual Framework to Incorporate Complex Basic Events in HiP-HOPS.

Kehren, C. (2005, 12). *Systems architectures formal safety patterns.* Theses, Ecole nationale superieure de l'aeronautique et de l'espace. Récupéré sur https://tel.archives-ouvertes.fr/tel-00011496

Khuu, M. (2008). *Contribution à l'accélération de la simulation stochastique sur des modèles AltaRica Data Flow.* Université de la Méditerranée (Aix-Marseille II).

Königs, A. (2005). Model Transformation with Triple Graph Grammars. *IN MODEL TRANSFORMATIONS IN PRACTICE SATELLITE WORKSHOP OF MODELS 2005, MONTEGO.*

Lefebvre, A. (2014). *Contribution à l'amélioration de la testabilité et du diagnostic de systèmes complexes : Application aux systèmes avioniques.*

Legendre, A. (2017). *Ingénierie système et Sûreté de fonctionnement : Méthodologie de synchronisation des modèles d'architecture système et d'analyse de risques.* Ph.D. dissertation.

Legendre, A., Lanusse, A., & Rauzy, A. (2019). System Engineering and Dependability: Methodology of Model Synchronization between System Architecture Models and Risk Analysis. *INSIGHT, 22*(4), 28-30.

Lisagor, O., Kelly, T., & Niu, R. (2011). Model-based safety assessment: Review of the discipline and its challenges. *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, (pp. 625-632).

Lisagor, O., Sun, L., & Kelly, T. (2010). The illusion of method: Challenges of model-based safety assessment. *28th international system safety conference (ISSC).*

Lynch, N. A., & Tuttle, M. R. (1989). An introduction to input/output automata. *CWI Quarterly, 2*, 219-246.

Machrouh, J., Blanquart, J.-P., Baufreton, P., Boulanger, J.-L., Delseny, H., Gassino, J., . . . Deleuze, G. (2012, 2). A cross-domain comparison of software development assurance standards. *Embedded Real Time Software and Systems (ERTS2012).* Toulouse. Récupéré sur https://hal.archives-ouvertes.fr/hal-02171454

Marco Filax, T. G. (2014). Short & tutorial proceedings of the 4th international symposium on model based safety assessment. *IMBSA 2014.*

Maxime Monnin, B. I. (2007). Méthodologie pour l'évaluation de la disponibilité opérationnelle des systèmes d'armes en présence de défaillance, de dommage et de régénération. *2èmes Journées Doctorales / Journées Nationales MACS.* Reims.

Meng Huixing, B. A. (2016). Production performance of an offshore system by applying AltaRica 3.0. *Lambda-Mu 20th, IMdR, Institut pour la Maîtrise des risques.* Saint-Malo.

Mens, T., Van Der Straeten, R., & D'Hondt, M. (2006). Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis. Dans O. Nierstrasz, J. Whittle, D. Harel, & G. Reggio (Éd.), *Model Driven Engineering Languages and Systems* (pp. 200-214). Berlin: Springer Berlin Heidelberg.

Meštrović, A., & Meštrovi´c, A. (2014). *Semantic Matching Using Concept Lattice Croatian speech technologies View project Semantic Matching Using Concept Lattice.*

Michel Batteux, T. P. (2018). From Models of Structures to Structures of Models. *4th IEEE International Symposium on Systems Engineering.* Rome, Italy.

Milner, R. (1982). *A Calculus of Communicating Systems.* Berlin, Heidelberg: Springer-Verlag.

NASA. (2017). *NASA Systems Engineering Handbook - NASA SP-2016-6105 Rev2: Design Test Integrate Fly.* CreateSpace Independent Publishing Platform. Récupéré sur https://books.google.fr/books?id=VqabtAEACAAJ

Nuseibeh, B., Easterbrook, S., & Russo, A. (2000). Leveraging inconsistency in software development. *Computer, 33*(4), 24-29.

OASIS. (s.d.). *PLCS FAQ*. Récupéré sur https://www.oasis-open.org/committees/plcs/faq.php

OMG. (2018). Semantics of a Foundational Subset for Executable UML Models (fUML™), v1.4. Récupéré sur http://www.omg.org/spec/FUML/1.2.1/

Open-Altarica. (s.d.). *Logiciel Open-Altarica*. Récupéré sur https://www.openaltarica.fr/

Papadopoulos, Y., Walker, M., & Kabir, S. (2018). Dynamic system safety analysis in HiP-HOPS with Petri Nets and Bayesian.

Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., . . . Lien, R. (2011). Engineering Failure Analysis & Design Optimisation with HiP-HOPS.

Pasquire, C. L., & Gibb, A. (2011, 11). Considerations for assessing the benefits of standardisation and pre-assembly in construction.

Pierre Bieber, J.-P. B. (2008). Integration of formal fault analysis in ASSERT: Case studies and lessons learnt. *Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008.* Toulouse.

Pierre-Antoine Brameret, A. R.-M. (2015). Automated generation of partial Markov chain from high level descriptions. *Reliability Engineering and System Safety*, 179--187.

Pollari, G., & Shaw, N. (2017, January). System Architecture Virtual Integration (SAVI) Project: IntermodelError Checking and Consistency Reviewand Demonstration. *INCOSE international workshop*.

Product Life Cycle Support. (s.d.). Consulté le 08 24, 2016, sur http://www.plcs.org/ap239/

Prosvirnova, T. &. (2015). Automated generation of minimal cut sets from AltaRica 3.0 models. *International Journal of Critical Computer-Based Systems.*

Rauzy, A. (2002). Mode automata and their compilationinto fault trees. *Reliability Engineering and SystemSafety*.

Rauzy, A. (2004). An experimental study on iterativemethods to compute transient solutions of large markovmodels. *Reliability Engineering & System Safety*.

Redman, D. (2014). Importance of Consistency Checking in the SAVI Virtual Integration Process (VIP).

Redman, D. (s.d.). The System Architecture Virtual Integration Program (SAVI).

Rieke, J., Dorociak, R., Sudmann, O., Gausemeier, J., & Schäfer, W. (2012, 1). Management of Cross-Domain Model Consistency for Behavior models of Mechatronic Systems. *Proceedings of the International Design Conference – DESIGN.*

Romain Bernard, J.-J. A. (2007). *Experiments in model-based safety analysis: flight controls, Proceedings of IFAC workshop on Dependable Control of Discrete Systems.* Cachan.

S3000L. (2014). *International pocedure specification for Logistic Support Analysis (LSA).* ASD (AeroSpace and Defence Industries Association of Europe), AIA (Aerospace Industries Association).

Spanoudakis, G., & Finkelstein, A. (1997). Reconciling requirements: a method for managing interference, inconsistency and conflict. *Annals of Software Engineering, 3*, 433-457.

Spanoudakis, G., & Zisman, A. (2001). Inconsistency Management in Software Engineering: Survey and Open Research Issues. 329-380.

Sylvain Breton, P. L.-C. (2018). SYSTEM-ANALYST -UN OUTIL MBSA POUR L'ANALYSE DES RISQUES, LIBRE DE DIFFUSION ET COMPATIBLE AVEC ARBRE-ANALYSTE ET OPEN- ALTARICA. *Congrès Lambda Mu 21, « Maîtrise des risques et transformation numérique : opportunités et menaces ».* Reims.

T.Prosvirnova, M. P.-A.-M. (2013). The AltaRica 3.0 project for Model-Based Safety Assessment. York.

TEICHTEIL-KÖNIGSBUCH, F. I. (2011). Lazy forward-chaining methods for probabilistic model-checking. *Advances in Safety, Reliability and Risk Management.*

Van Der Straeten, R., & D'Hondt, M. (2006). Model Refactorings through Rule-Based Inconsistency Resolution. *Proceedings of the 2006 ACM Symposium on Applied Computing* (pp. 1210–1217). New York, NY, USA: Association for Computing Machinery. doi:10.1145/1141277.1141564

Van Der Straeten, R., Mens, T., Simmonds, J., & Jonckers, V. (2003). Using Description Logic to Maintain Consistency between UML Models. Dans P. Stevens, J. Whittle, & G. Booch (Éd.), *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications* (pp. 326-340). Berlin: Springer Berlin Heidelberg.

Ward, D. T., Redman, D. A., & Lewis, B. A. (2013). An Approach to Integration of Complex Systems: The SAVI Virtual Integration Process. *Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology* (pp. 43–46). New York, NY, USA: Association for Computing Machinery. doi:10.1145/2527269.2527275

Wouters, L., & Gervais, M. (2011, 8). xOWL: An Executable Modeling Language for Domain Experts. *2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, (pp. 215-224). doi:10.1109/EDOC.2011.13

Wouters, L., Creff, S., Bella, E. E., & Koudri, A. (2017, 4). Collaborative systems engineering: Issues challenges. *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, (pp. 486-491). doi:10.1109/CSCWD.2017.8066742

Wouters, L., Creff, S., Bella, E. E., & Koudri, A. (2017, 12). Towards Semantic-Aware Collaborations in Systems Engineering. *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, (pp. 719-724). doi:10.1109/APSEC.2017.92

Xavier Quayzin, E. A. (2009). Performance Modeling of a Surveillance Mission. *Proceedings of the Annual Reliability and Maintainability Symposium, RAMS'2009.* Fort Worth.

Y. Papadopoulos, J. A. (1999). Hierarchically Performed Hazard Origin and Propagation Studies. *International Conference on Computer Safety, Reliability, and Security (SafeComp).*

Xavier de Bossoreille, Pierre Darfeuil, Frédéric Deschamp, Christophe Frazza, Jean Gauthier, Mathilde Machin, Tatiana Prosvirnova, Christel Seguin , Estelle Saez; (2021). Altarica strategies for modelling failure propagation in dynamic systems, Lambda Mu 2023

2 Key concepts

This chapter gives definitions of some key concepts for the project. We estimated that it is important to have a common and shared understanding of these concepts. The two concepts selected so far are independence and inconsistency. The aim is to enrich this chapter as the project progresses. What is independence from the ARP point of view?

Both ARP4754A and 4761 are defining the notion of “independence” in a quite similar way. The ARP4761 definition of the independence is:

INDEPENDENCE: (1) A design concept which ensures that the failure of one item does not cause a failure of another item. (Derived from JAA AMJ 25.1309.) (2) Separation of responsibilities that assures the accomplishment of objective evaluation.

The ARP4754A gives the following definition:

INDEPENDENCE: 1. A concept that minimizes the likelihood of common mode errors and cascade failures between aircraft/system functions or items, 2. Separation of responsibilities that assures the accomplishment of objective evaluation e.g. validation activities not performed solely by the developer of the requirement of a system or item.

The ARP4754B introduces the MBSA but does not modify the definition of independence.

These definitions includes two concerns: the first one is related to the product and more specifically the architecture of the product, the second one implies organizational constraints on the designing and development process.

In the S2C Project, the focus is on the second concern. Indeed, we will consider in WP1 and WP2 studies two different actors (Safety and System Engineer) and we will describe the best process to ensure System and Safety consistency based on these different actors. The responsibilities are clearly shared and an organizational aspect of independence is directly taken into account. We consider although in S2C project that system engineer and safety analyst will work on different artefacts (model, analysis …) deducted and constructed from the same root requirements but without any direct and implicit relation between them. Common pitfalls of such an approach is the analysis of a former version of design or the misunderstanding of the design. This approach probably reinforces inconsistencies risk but there is no doubt that it is compliant with independence requirement.

Concerning the first concern (architecture independence), it will be addressed as part of the safety analysis to be performed (WP4).

2.1 What is inconsistency?

Based on the Cambridge dictionary definition (Cambridge online, s.d.), inconsistency is defined as “a situation in which two things do not match and are opposed”.

(Spanoudakis & Zisman, Inconsistency Management in Software Engineering: Survey and Open Research Issues, 2001) defines it as “a state in which two or more **overlapping elements** of different software models **make assertions** about aspects of the system they describe which are **not jointly satisfiable**”

(Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015) defines it as “a state of conflict in which, under an interpretation, two or more related statements are accepted as true, that cannot jointly be true.”

These definitions of inconsistency remain to be instantiated and specified for the S2C project, but we can note that the two last definitions are widely used in the literature as a reference.

Moreover, some literature points out notions of source or cause of inconsistencies, in particular:

- **Incompleteness:** emphasizes the fact that a part is present on one side and absent on the other side whereas it should be present;
- **Ambiguity**: concerns the multiplicity of possible interpretations, e.g an MBSE model cannot be correctly interpreted that can lead to an error of MBSA modeling;
- **Contradiction**: discrepancy between MBSE and MBSA parts that reflects something different;
- **Incompatibility**: non agreement to MBSE and/or MBSA methods development rules or to certification recommendations which could imply inconsistency between the two models;

In order to deal with these inconsistencies, it is firstly necessary to be able to qualify them. This means identifying the dimensions related to consistency activities and the types of inconsistencies. The literature resulting from work on consistency engineering provides some answers to these needs to qualify inconsistencies.

2.1.1 Dimensions of consistency

Identifying the dimensions of the inconsistencies enables the classification of these inconsistencies in order to know how they can be managed afterwards. (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015) sums up a number of dimensions issued from various other works .

Horizontal / Vertical

Abstraction levels relationships (e.g. vertical axes could mean a "refinement" relationship, and horizontal axes could mean a relationship between System Engineering and discipline engineering (e.g. Safety)

Horizontal dimension refers to consistency between different models at the same level of abstraction, and expressing different viewpoints. Whereas vertical dimension deals with the consistency of the models in different levels of abstraction and that must be consistent (Generally, this dimension addresses the consistency of models that are refined from a higher to a lower level of abstraction).

These dimensions are closely linked to the issues addressed by S2C. The horizontal dimension, which deal with consistency between the MBSE and MBSA models, will be addressed in lot 2. While the vertical dimension addresses the problem of lot 3, which is how to ensure consistency between the MBSA/FTA models of the different systemic levels.

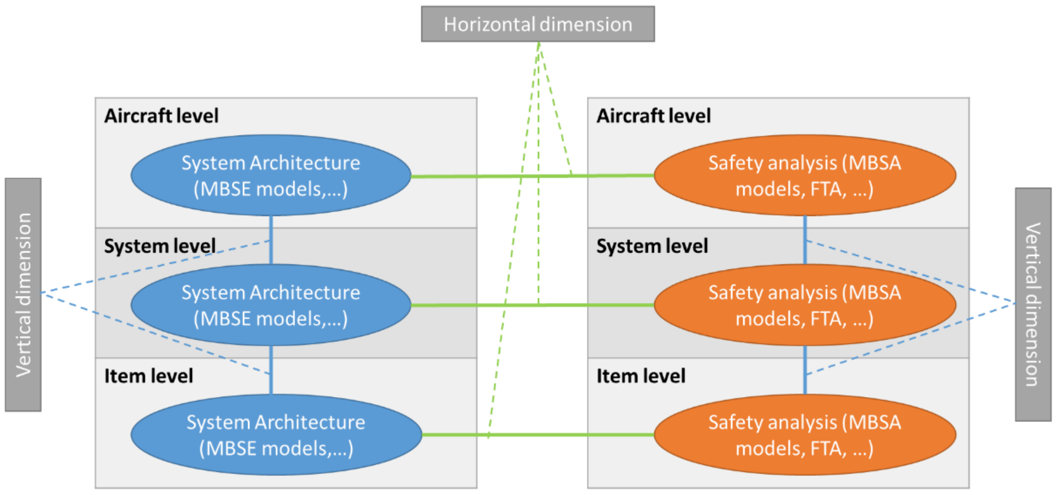
Horizontal dimension
Aircraft level
System Architecture (MBSE models,...)
System level
System Architecture (MBSE models,...)
Item level
System Architecture (MBSE models,...)
Aircraft level
Safety analysis (MBSA models, FTA, ...)
System level
Safety analysis (MBSA models, FTA, ...)
Item level
Safety analysis (MBSA models, FTA, ...)
Vertical dimension
Vertical dimension

Figure 2 : Horizontal / vertical dimension concept

Syntactic / Semantic is about abstract and/or concrete syntax conformance and semantic alignment

Designing a complex system like aircrafts needs collaboration between different stakeholders. These stakeholders build models using different tools, and each tool has a specific syntax and semantics. The challenge is that some or all of these models should be consistent. Except that having a common syntax (saying things in the same way) and aligning semantics is not an easy thing to do.

(Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015) affirms that syntax usually facilitates communication. Except that a syntax that may seem "obvious" to one person, is not necessarily obvious to another person or to a computer. The problem is here, we need the same semantic interpretation (the correct understanding) of the syntactic structure to be consistent. The semantic interpretation is especially complicated because sometimes two semantically identical things can be expressed in two different ways syntactically: e.g. “01/17/1986" and “The third Friday in the first month of the year 1986" refer to the same date, but are syntactically very different. One solution to address this problematic of syntactic and semantic consistency is the use of meta-modeling. A meta-model can help to have a syntactic representation of a semantic domain and semantic mapping.

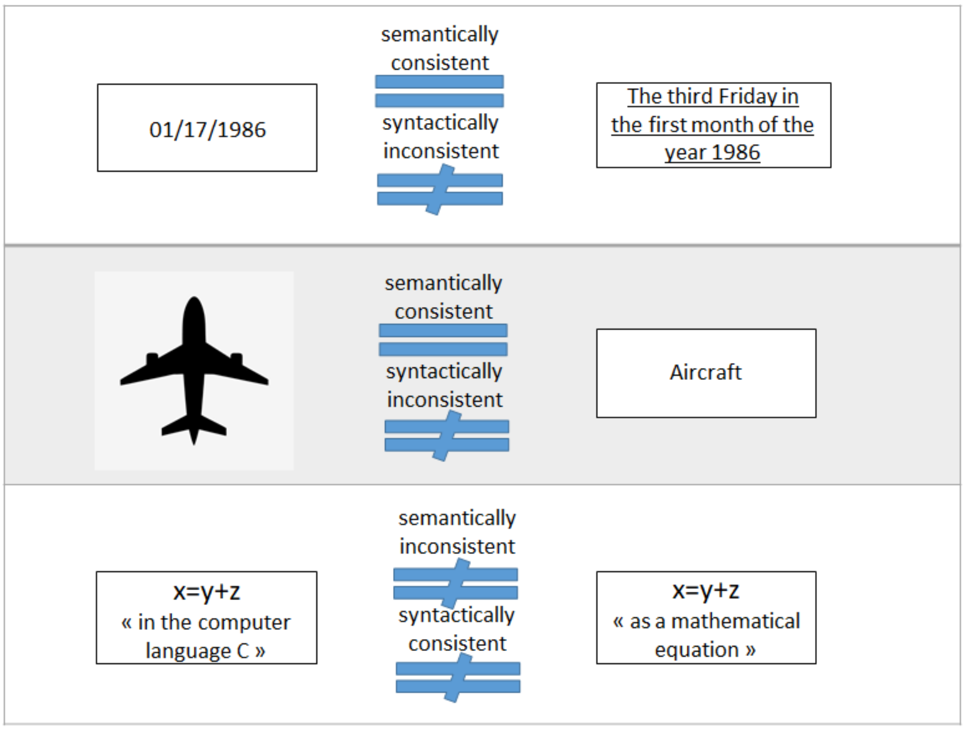
01/17/1986
semantically consistent
syntactically inconsistent
The third Friday in the first month of the year 1986
semantically consistent
syntactically inconsistent
Aircraft
x=y+z « in the computer language C »
semantically inconsistent
syntactically consistent
x=y+z « as a mathematical equation »

Figure 3 : Syntactic / Semantic concept

Structural / Behavioral

With the terms structural and behavioral consistency, either structural or behavioral aspects of models are described (Van Der Straeten, Mens, Simmonds, & Jonckers, 2003). For instance, structural inconsistencies may refer to missing components in one of the models (System or Safety), whereas behavioral inconsistencies arise if incompatible behavior description are observed between the system’s model and the safety analysis.

The major part of the work towards inconsistency management in the literature focuses on the structural consistency, cf. Chapter 4.3.

To compare behavioral models in a meaningful manner, one needs to assume that these models have semantics (therefore well defined, explicit and shared). Semantics are not only useful in defining behavioral

aspects (Harel & Rumpe, 2004): both behavior and structure need unambiguous semantics. Behavioral semantics are typically much harder to define. This is because deciding upon a specific semantic domain of discourse requires deciding upon the kinds of things we want our language to express. For example, standard semantic domains used for describing behavioral semantics are trace semantics (Hoare, 1978), input/output relations (Lynch & Tuttle, 1989), and streams and stream processing functions (Broy & Ştefănescu, 2001).
In the consistency literature, few works focused on behavioral checking, mainly: the behavioral incompatibility description in the software domain in UML by van Der Straeten et al. (Van Der Straeten, Mens, Simmonds, & Jonckers, 2003), the work of Rieke et al. (Rieke, Dorociak, Sudmann, Gausemeier, & Schäfer, 2012) on managing consistency between behavioral models of the mechatronic system.
Dealing with the notion of consistency, one need may be to assume some equivalence notion (when are two models the same?).
Communities working on formal methods have proposed a wide variety of equivalence notions e.g., trace equivalence, bisimulation (Milner, 1982) (two models may be identical under trace equivalence but are different when considering stronger notions of equivalence), branching bisimulation (Glabbeek & Weijland, 1996), etc. In these works, researchers have worked on different notions of equivalence for FSM and timed-automaton (e.g., trace equivalence, bi-simulation, branching bi-simulation, listed above), not considering stochastic timed automaton.

Besides, in the safety context, Kehren (Kehren, 2005) worked towards the comparison of three notions of equivalence relation to identify compatibility between architectural patterns, namely bisimulation, simulation equivalence, and simulation preorder. The author concludes that the stronger the abstraction is, the less it preserves the properties of the initial model. Thus, if the bisimulation strongly preserves CTL (Computation Tree Logic), the simulation only weakly preserves ACTL (Action Computation Tree Logic). As far as simulation equivalence is concerned, it strongly preserves ACTL and therefore LTL (Linear Temporal Logic). The bisimulation seems to be too restrictive in terms of abstraction and the simulation too weak in terms of preserving properties. In this work, simulation equivalence is therefore chosen to validate the substitution of a real architecture by a safety architecture pattern. This one, although weaker than the bisimulation, strongly preserves the logic while admitting a correct level of abstraction.

Evolution - Version management: concerning consistency management during all the life cycle of the product

- Change management
- Version management
- Iterative management

In S2C, we are interested in change and version management. Consistency will be analyzed and addressed on versions identified of models.

2.1.2 Types of inconsistencies

Orthogonal to the dimensions of inconsistencies introduced before, different types of inconsistencies can be found in the literature. A classification, taken from (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015), is reproduced in Figure 4: Classification of inconsistency types illustrated as a type definition hierarchy using a simplified class diagram syntax .

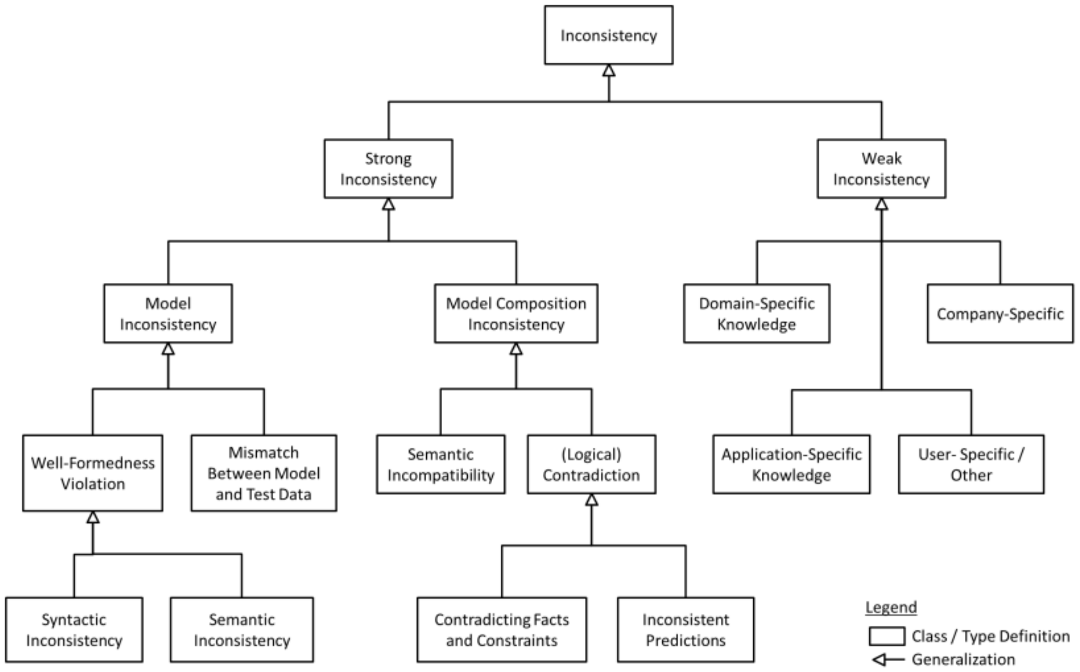

Figure 4: Classification of inconsistency types illustrated as a type definition hierarchy using a simplified class diagram syntax (Herzig, 2015)

A distinction is made by the author between **strong** inconsistencies and **weak** inconsistencies:

- Strong inconsistencies have a profound effect on the value of both a set of models and the artifact intended to be described, since it is impossible for the described system to exist in any logical (or rational) world.
- In contrast to this, a violation of a best practice, guideline or style guide has an impact on the value of the models and the system, but it is not necessarily impossible for the system to be deployed.

Note that the author consider all guidelines and standards to be weak inconsistencies, in contrast, in the aeronautic context, norms and standards do not only provide guidelines but also rules to be fulfilled, and therefore should be considered as strong inconsistencies.

Considering consistency in a unique (single) model, this work summarizes concerns in different domains. For example, **well-formedness**, description identity, application domain, development compatibility and development process compliance rules are, according to Spanoudakis and Zisman (Spanoudakis & Zisman, 2001), important. **Well-formedness** rules are classically fulfilled by the models to be legitimate models of the respective modeling language (i.e. model conformance to its definition, i.e. the metamodel, with potentially additional constraints, like expressed in OCL for example). Well-formedness is concerned with a **unique model** (cf. Section 4.2.1.1). Well-formedness violations can be both **syntactic** and **semantic** in nature.

On the opposite, **composing models** (cf. multiple models approach in Section 4.2.1.2) can reveal **semantic incompatibilities** or **logic contradiction** between models. Contradicting statements are a specialization of logical contradictions, in that two propositions are compared. Related to this are the class of **inconsistent predictions**: given identical prior beliefs and observations, as well as identical causal assumptions, predictions are inconsistent if they yield a different result.

Semantic incompatibility from the Figure refers to the issue of incompatible interpretations. That is, it is impossible to construct a consistent interpretation when forming the composition of two or more models.

Considering rules, introduced before, application domain rules refer to relations that must hold between individuals in the specific application domain. Finally, development process compliance rules describe practices, guidelines or standards that need to be followed by the models (cf. Section 4.1).

What is considered as an inconsistency here is the violation of a standard practice, best practice, convention or guideline. Guidelines and best practices are important for a variety of reasons: for instance, for capturing expert knowledge and the readability and maintainability of development artifacts (e.g., coding or style guidelines, and part numbers). Instead of defining classes related to style guide violations and violations of best practices, these classes of inconsistencies are abstracted further and the classes of **domain**-, **application**, **company**- and **user**-**specific** inconsistencies are suggested.

Contrary to some classifications from the related literature, **model evolution inconsistencies** are not taken into account in Figure 4: Classification of inconsistency types illustrated as a type definition hierarchy using a simplified class diagram syntax . The rationale for this is that only the state of the most up-to-date information should be considered when checking for inconsistencies. **Process-related inconsistencies** were also not made explicit, since these are considered inconsistencies with respect to a particular model of the process.

Finally, other types can be found in the literature, bringing precisions about the concerns of the approach, e.g. the SAVI integration effort (more details in Section 4.4.6) identifies at least six types of consistency (Ward, Redman, & Lewis, 2013): (1) interface consistency, (2) compositional consistency, (3) constraint consistency, (4) behavioral consistency, (5) version consistency, and (6) verification consistency.

3 Consistency management – theoretical concepts

Inconsistency management provides an alternative view on the problem of consistency management, where the goal is to either prove the consistency of a set of models, or to ensure that inconsistencies are never introduced (e.g., by enforcing constraints during construction of a model).

In this chapter, we will list all the activities required to achieve an efficient consistency management throughout the development cycle of a system.
These activities are inspired from the work of (Spanoudakis & Zisman, Inconsistency Management in Software Engineering: Survey and Open Research Issues, 2001) on consistency management in software engineering “Inconsistency Management in Software Engineering: Survey and Open Research Issues”. They are grouped according to four macro activities which are:

1. Linking engineering artifacts (Matching and mapping)
2. Detection of inconsistencies
3. Diagnosis of inconsistencies
4. Handling inconsistencies

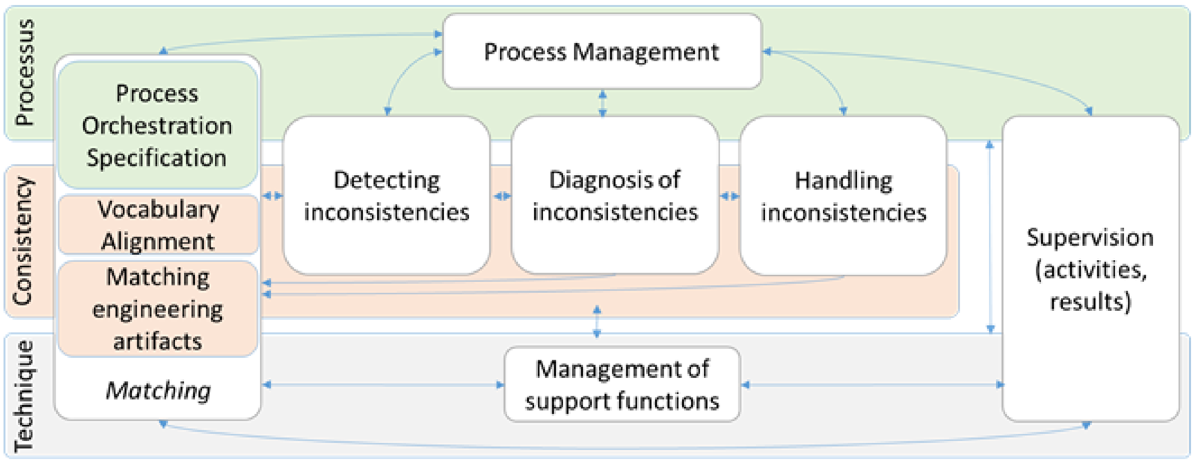
Processus
Consistency
Technique
Process Management
Process Orchestration Specification
Vocabulary Alignment
Matching engineering artifacts
Matching
Detecting inconsistencies
Diagnosis of inconsistencies
Handling inconsistencies
Supervision (activities, results)
Management of support functions

Figure 5: Orchestration of consistency activities

3.1 Linking engineering artifacts (Matching and mapping)

The linking engineering artifacts activity consists in identifying, specifying and referencing the links between engineering domains and the alignment between their processes. Matching is the process that produces correspondences between artefacts; a mapping is a relationship, i.e., a constraint that must hold between their respective instances. For the mappings to be generated, a fundamental requirement are the matches between the elements of the schemas. These matches can be either generated automatically through a matching process, or can be manually provided by an expert user.

More globally, this activity concerns the semantic overlaps between engineering artifacts.

(Meštrović & Meštrovi´c, 2014) presents in the following figure an example of what is meant by semantic overlaps.

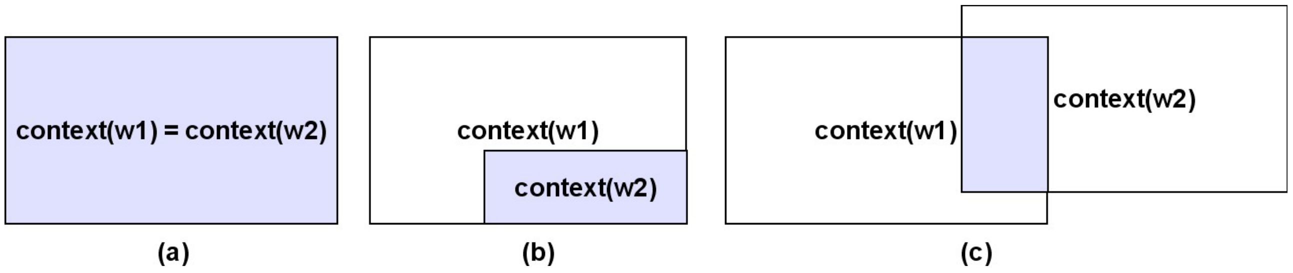

Figure 6: Cases of semantic overlapping

Figure 6 shows three possible relations between two words (w1, w2) that have semantic overlapping. In the first case (a) word w1 can replace word w2 in any context and vice versa. In the second case (b) word w2 can be replaced with word w1 in any context, but not vice versa since word w1 has a more general meaning. In the third case (c) both words w1 and w2 have some additional meaning and therefore can be replaced with each other only in certain contexts. These cases actually reflect the relationships between two synonyms or hyperonym and hyponym described as relations of equivalence, less specific and more specific respectively.

Semantic overlaps between models or part of models (that consider multiple viewpoints) that express views on the system under study, exist when the models "incorporate elements which refer to common aspects of the system under development" (Spanoudakis & Zisman, 2001), i.e. that they represent different views of the same information. It is generally achieved by formalizing dependency links between these artifacts that have been constructed in isolated way. The objective is to identify these overlaps that allows to connect initially disjointed spaces whose internal consistency is assumed. All the newly relationship could imply the highlighting of potential inconsistencies.

More generally, this activity of matching can be broken down into three facets (knowledge, engineering artifacts, and processes). In detail, the breakdown is as follows:

- Alignment of business vocabularies: defining a vocabulary shared by the domains / stakeholders involved in the collaboration, and linking it to the specific lexicons. Needs range from the definition of thesauri and synonymous dictionaries to more formal ontological representations and associated alignments.
- Matching and mapping engineering artifacts at different levels:
  - Alignment of the languages used in the business lines: choice of common formalisms, and/or mapping between formalisms,
  - Matching of collaborative artifacts: mapping models of different natures and granularities,
  - Specification of inter-business matching rules: elicitation and formalization.
- Specification of the orchestration of the engineering processes:
  - Defining the phasing (activities) of engineering process
  - Defining the interactions (criteria for stopping modeling, interfaces)
  - Defining the synchronization (the time of verification of consistency).

3.2 Detection of inconsistencies

The objective of this activity is to detect contradictions between assertions concerning engineering artifacts (engineering data), process or any other elements. External constraints or rules to be fulfilled can be defined, and thus be checked for this purpose.

This activity can be broken down into:

- Detection of contradictions between assertions, with two main approaches:
  - Expert inspection (consistency review), potentially collaborative,
  - Tool-based analyses (e. g. model checking, rule checks).
- Detection of deviations in the implementation and execution of process orchestration (control of compliance with procedures, etc.) that could raise a suspicion of inconsistency on artifacts.

3.3 Diagnosis of inconsistencies

Once inconsistencies are detected (detection step - previous section), an analysis of them is required. This is the objective of the activity of diagnosing inconsistencies. This activity is divided into three main tasks:

- Identification of the source of the inconsistency (locate the inconsistency);
- Identification of the cause of the inconsistency;
- Identification of the impact of the inconsistency.

3.4 Handling inconsistencies

The handling of inconsistencies is considered a central activity of inconsistency management. This activity concerns:

- the identification of possible actions to address an inconsistency,
- the assessment of the costs and benefits that would result from the application of each of these actions,
- the assessment of the risks that would result from not resolving the inconsistency,
- the arbitration of the actions to be carried out,
- the implementation of actions (e.g. reconciliation of artifacts)

Transverse to these phases, there are technical fact management and version management associated with this processing management associated with this processing.

4 Consistency methods and methodologies

This chapter reviews academic and industrial works to address consistency in a model-driven and SE/SA context.

The core problem systems engineering in a complex heterogeneous environment is concerned with is managing design artefacts inconsistency in order to retain the eventual **correctness** of the product, cf. def. in (ISO/IEC/IEEE 24765 Systems and software engineering — Vocabulary, 2017) . Therefore, the correctness of the product is approximated with the correctness of its descriptive models (e.g. design artifacts). Inconsistency management argues that *carrying out the engineering process and design artifacts consistently helps achieving the correct product* and increases the *efficiency of the process*.

In this context, consistency is about sharing and relating knowledge adequately of the system under study and organize the processes efficiently.

First, one way to prevent inconsistencies is to consider sharing and alignment of knowledges and practices (vocabulary and rules) through standards, norms and guidelines (Section 4.1). On another side, in a model-based context, different approaches are observed to represent viewpoints as expressed in (ISO/IEC/IEEE Systems and software engineering -- Architecture description, 2011), i.e. the unique model approach or the heterogeneous models reconciliation (Section 0), which introduces variability in the methods to "assemble" the models and raises different issues considering consistency. Besides, considering methods and tools to help manage inconsistency, different kind of concerns and therefore approaches can be found in the literature, see Section 4.3. Next, a focus is made on the consistency between systems engineering and safety analysis, formalized as MBSA (Section 4.4) or fault trees (Section 4.5). The vertical dimension is then explored between two systemic levels of safety analysis (Section 4.6).

Finally, a comparison of these methods is provided in Section 0.

Left apart from this chapter is the human inspection, generally formalized as a **review**, e.g. which can be performed by a pair (NASA, 2017), seen in a collaborative modeling context [ISC], or dedicated to consistency as described in Section 4.3.1.3.

4.1 *Towards standards, norms and guidelines*

First, it is a question of defining a knowledge, vocabulary and rules shared by the domains / stakeholders of the different spaces/communities/teams. One way to prevent inconsistencies is to consider sharing and alignment of knowledges through standards, norms and guidelines (best practice, convention, etc.), and therefore reduce the disparity among stakeholders. This section provides an overview of standards and guidelines related to our context: model-based, systems and safety engineering.

Standardization is the wide use of components, parts, procedures, or processes in which there is regularity, repetition, and a successful practice and predictability (Gibb & Isack, 2001), (Pasquire & Gibb, 2011). Some of the items can be standard by their nature (generic standardization) or as assigned by the legislation of a country (national standardization). Both clients and suppliers may have standard processes or products.

Systems engineering and processes

A plethora of norms and standards exists on the domain, Figure 7 shows a representation of the norms considered at the ISO sub-comity ISO JTC1/SC7 considering Software and Systems engineering.

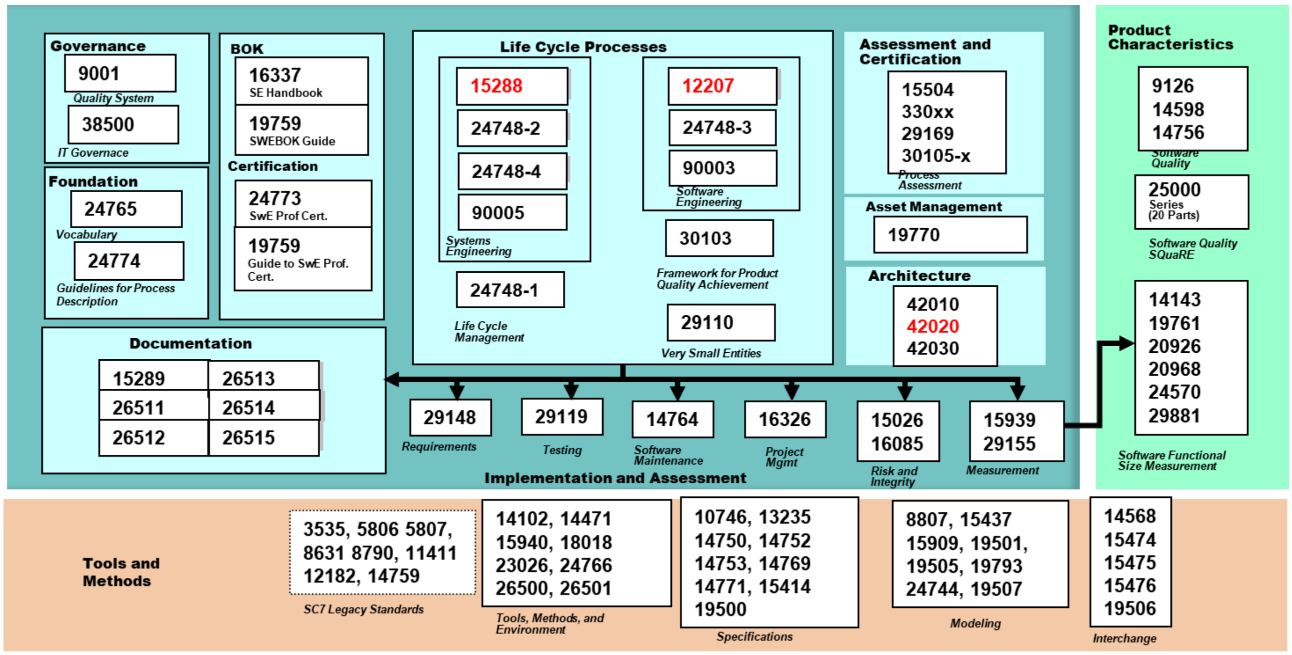

Figure 7: Norms of the sub comity ISO JTC1/SC7 – Software and Systems engineering

On the Systems engineering and process side, the ISO/IEC/IEEE 15288:2015 (ISO15288, 2015) establishes a common framework of process descriptions for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology from an engineering viewpoint. These processes can be applied at any level in the hierarchy of a system's structure. From this point of view, Safety is considered as a concern impacting technical processes.

This standard groups the activities that may be performed during the life cycle of a system into four process groups as depicted in Figure 8:

i) Agreement Processes: acquisition and supply processes;

ii) Organizational Project-Enabling Processes: concerned with ensuring that the resources needed to enable the project to meet the needs and expectations of the organization’s interested parties are met;

iii) Project Processes: concerned with managing the resources and assets allocated by organization management and with applying them to fulfil the agreements into which the organization or organizations enter;

iv) Technical Processes: concerned with technical actions throughout the life cycle. They transform the needs of stakeholders first into a product and then, by applying that product, provide a sustainable service, when and where needed in order to achieve customer satisfaction.

Each of the life cycle processes within those groups is described in terms of its purpose and desired outcomes and list activities and tasks which need to be performed to achieve those outcomes.

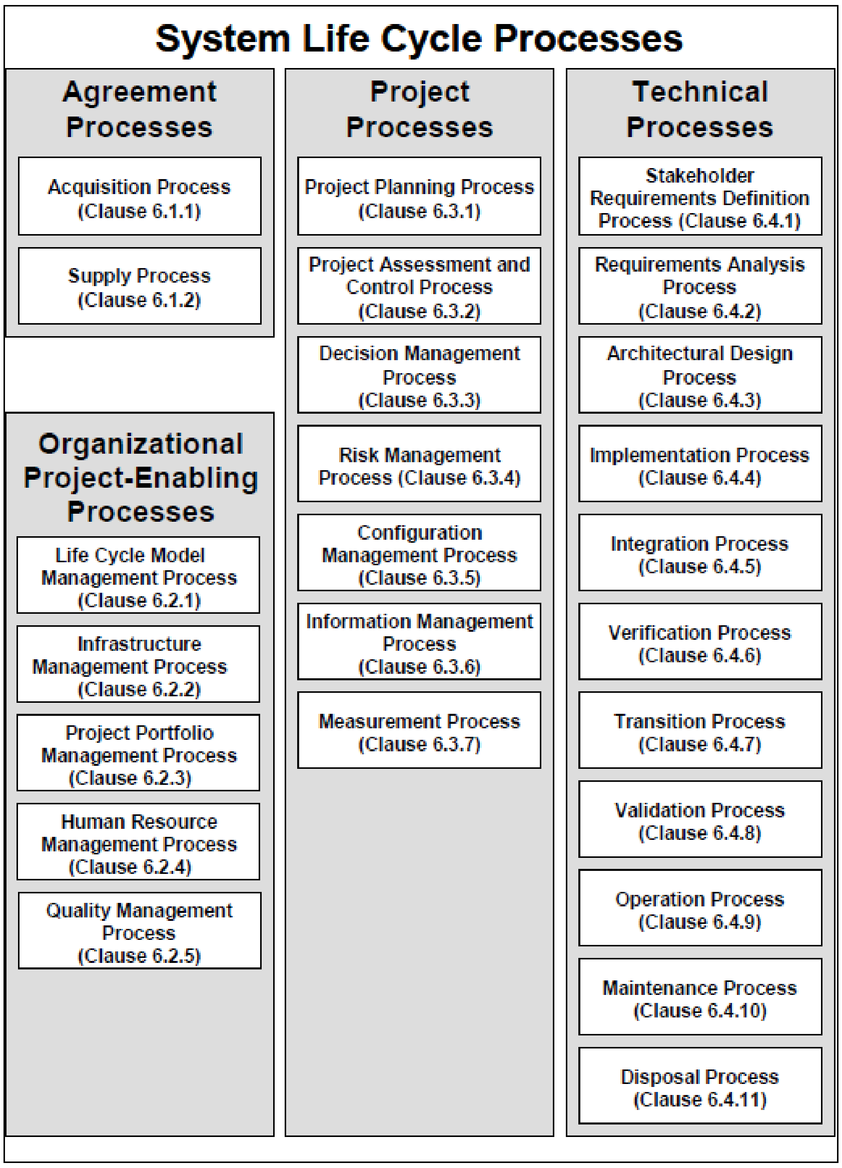
System Life Cycle Processes
Agreement Processes
Acquisition Process (Clause 6.1.1)
Supply Process (Clause 6.1.2)
Organizational Project-Enabling Processes
Life Cycle Model Management Process (Clause 6.2.1)
Infrastructure Management Process (Clause 6.2.2)
Project Portfolio Management Process (Clause 6.2.3)
Human Resource Management Process (Clause 6.2.4)
Quality Management Process (Clause 6.2.5)
Project Processes
Project Planning Process (Clause 6.3.1)
Project Assessment and Control Process (Clause 6.3.2)
Decision Management Process (Clause 6.3.3)
Risk Management Process (Clause 6.3.4)
Configuration Management Process (Clause 6.3.5)
Information Management Process (Clause 6.3.6)
Measurement Process (Clause 6.3.7)
Technical Processes
Stakeholder Requirements Definition Process (Clause 6.4.1)
Requirements Analysis Process (Clause 6.4.2)
Architectural Design Process (Clause 6.4.3)
Implementation Process (Clause 6.4.4)
Integration Process (Clause 6.4.5)
Verification Process (Clause 6.4.6)
Transition Process (Clause 6.4.7)
Validation Process (Clause 6.4.8)
Operation Process (Clause 6.4.9)
Maintenance Process (Clause 6.4.10)
Disposal Process (Clause 6.4.11)

Figure 8: ISO/IEC/IEEE 15288 processes overview

Data Exchange Standard

On the interoperability and exchange side, the ISO organization started a STEP (Standard for the Exchange of Product Model Data) project in 1984 for Product modeling of industrial data. The actual designation of the STEP standard is: ISO 10303 Industrial Automation Systems - Product Data Representation and Exchange.

STEP aims are to provide a mechanism that is capable of describing product data throughout the life cycle of a product, independent from any particular system and suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

The AP233 standard (ISO 10303-233:2012; AP233SE website, s.d.), which is the AP for Systems Engineering, was developed since 1996 by many organizations (OMG, OASIS, INCOSE, ISO, and PDES) including the involvement of industrial companies, tool-vendors, and research centers.

The scope of AP233 has been summarized in (INCOSE, 2010) as follows:

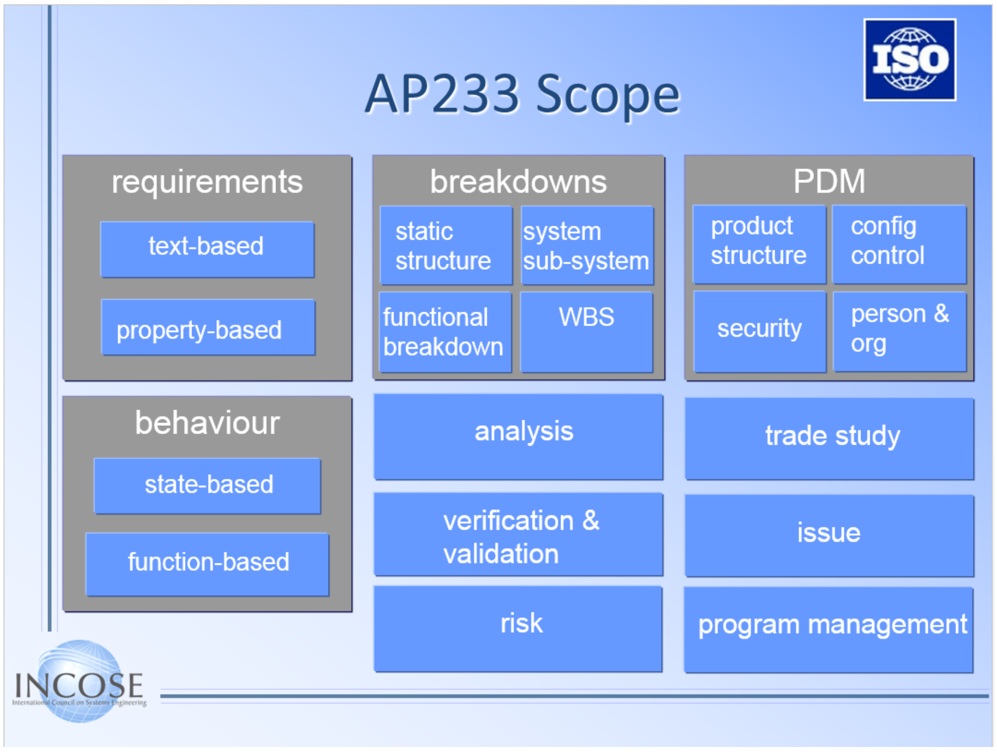

Figure 9: ISO 10303-233 Scope

AP233 uses many elements from other AP information models as Configuration Controlled Design (AP203), Engineering Analysis (AP209 E2) and Product Lifecycle Support (AP239/PLCS).

For information, this last one, AP239 standard (ISO 10303-239:2005; Product Life Cycle Support, s.d.; OASIS, s.d.) is the one sharing a significant number of modules with AP233 as it covers:

- Identification and composition of a product design from a support viewpoint,
- Definition of documents and their applicability to products and support activities,
- Identification and composition of realized products,
- Configuration management activities, over the complete life cycle,
- Properties, states and behavior of products
- Activities required to sustain product function,
- Resources needed to perform such activities,
- Planning and scheduling of such activities,
- Capture of feedback on the performance of such activities, including the resources used,
- Capture of feedback on the usage and condition of a realized product,
- Definition of the support environment in terms of support equipment, people, organizations, skills, experience and facilities,
- Definition of classes of product, activities, states, people, organizations and resources.

Safety Standards and guidelines in the project context

Concerning safety, the work by (Machrouh, et al., 2012) provides a cross-domain comparison of software development assurance standards, indeed that of civil aviation (aeronautics' DO-178/ED-12), automotive (ISO 26262), space (ECSS-Q-ST-80C), process automation (IEC 61508), nuclear (IEC 60880) and railway (EN 50128).

Efforts are provided by various groups to align standards concerning safety. Let's cite the IEC 63187 - Framework Safety for Defense applications with CD in late 2020, and IS in mid-2023.

Dealing with guideline, obvious in the domain, the Aerospace Recommended Practice established industry practices to achieve the compliance with regulatory requirements. It have been developed over many years to ensure a common and shared approach in order to provide an acceptable level of Safety for civil aircraft. The panel is composed of: the Safety Assessment Process Guidelines & Methods (ARP 4761), the Aircraft & System Development Processes (ARP 4754 / ED-79), the Safety Assessment of Aircraft in Commercial Service (ARP 5150 / 5151), the Guidelines for Integrated Modular Avionics (DO-297/ED-124), the Electronic Hardware Development Life-Cycle (DO-254 / ED-80), and the Software Development Life-Cycle (DO-178B/ED-12B).

4.2 Models and viewpoints

Stakeholders have concerns, e.g. system design or safety analysis, that can be framed by specific viewpoints or express on the same one. The last case is exemplified by the ESACS and ISAAC approach (Lisagor, Kelly, & Niu, 2011), where some design models expressed in specified languages can be extended with safety component "injections". Although it is not clear whether the extended model is then used by the design or that the two models evolves in parallel, the viewpoint is here unique, defined in the same tool.

Considering distinct viewpoints, in a model-based context, different ways are observed to represent viewpoints as expressed in (ISO/IEC/IEEE Systems and software engineering -- Architecture description, 2011). Figure 10 provides a representation of concepts extracted from the standard.

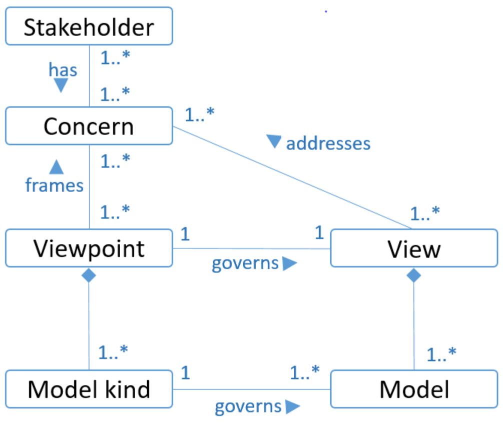

Figure 10: Concerns, viewpoints and models in the ISO/IEC/IEEE 42010

On the one hand, viewpoints can be translated into layers on top of a main structure, e.g. from a technical concern the Sirius approach on top of the EMF. Which results in the Systems Engineering context in Capella, that defines Systems Engineering as the core structure, and the other technical processes (engineering specialties) as viewpoints. This approach can lead to difficulties when a viewpoint expressed as a layer needs to modify the core structure.

On the other hand, viewpoints can be traduced into distinct models. With this approach, which allows freedom in expressing each viewpoint, the key point become the reconciliation phase (model composition or merge). Indeed, these models can be heterogeneous (the literature consider heterogeneity between diagrams kinds into a language, between languages, between technological spaces and between technical spaces – i.e. disparate models; each one having is specific issues), can be distributed, and can therefore rely on different languages and tools. The reconciliation phase in this second approach can be facilitated by the use of standards, norms and guidelines (cf. Section 4.1) to reduce the “distance” between the viewpoints.

Some key points to distinguish these two approaches are:

- The moment when the viewpoints are reconciled, and consequently,
- The constraints imposed to the stakeholders to express their concerns (more or less freedom).

In the following, considering model-based approaches in the SE/SA context, these two modeling approaches are described:

- “Unique model” approach: Building a model relying on the same structure used for both purposes, the core model being defined by the Systems Engineering. This model can be addressed with a single viewpoint (model “injection” extension) or by multiple ones (layers on top of the main structure);
- “Multi-models” approach: build two models more independently (one per concern).

4.2.1.1 Unique model approach

One intuitive way to ease consistency management between two engineering domains is to enrich an existing model with information enabling to study the same system in another domain. In the following, we call this approach “unique model” and we illustrate it on in the case of system design and safety analysis consistency. We distinguish two possible unique model approaches:

- The base model used is a simulation design model. It is extended with failure modes (i.e. safety information). Typically, in an electrical model simulating voltage from 0V to 5V, a failure mode is added, which results in applying a 0V voltage. The values used in the model are the values that can be measured in the system. This approach is presented and discussed in (Lisagor, Kelly, & Niu, 2011).
- The base model used is an architecture model without systematic behavior definition. It is enriched with a safety layer is added to define a safety behavior in each model element. Typically, an electrical wire enabling voltage is assigned to an enumerated domain {nominal, erroneous, lost} modeling the deviation from the nominal behavior rather than the actual values that could be measured in the system. Moreover, safety computations use only the safety behavior and disregard the original SE behavior, if one is defined. In (Papadopoulos, et al., 2011) a MBSA model defined from the structure of a Simulink model is presented, and more recently (Bitetti, De Ferluc, Mailland, Gregoris, & Capogna, 2019) presents a Capella model enriched with failure rates to enable dependability computations. More related works can be found in Chapter 1, Section 4.1 of (Legendre, Ingénierie système et Sûreté de fonctionnement : Méthodologie de synchronisation des modèles d’architecture système et d’analyse de risques, 2017).

In the following, we focus the unique model with a safety layer. Figure 11: Overview of unique model approach illustrates it: an “unique model” gathers information used for architecture description and safety analysis. As a consequence, both SE architect and SA specialist contribute to the same model. The language used for this model is based on MBSE with some extension to support safety analysis. Typically, some properties, specific to dysfunctional behavior are added to the model elements. For example, failure modes are added to physical elements.

From the unique model, a MBSA model is generated to be processed by MBSA tools and thus to get cutsets and probabilities. This intermediate MBSA model is automatically generated and is not associated to any modeling activities. From a process point of view, it is not a model but an intermediate technical artifact.

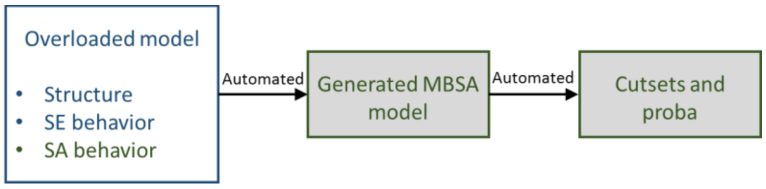

Figure 11: Overview of unique model approach

As the model structure is completely shared, this approach does not raise any structural consistency issue. Nevertheless, each structural element has two different behaviors, one from the SE point of view, and the other for SA analysis. The consistency between the SE and SA behaviors should be ensured.

A unique model intents to cover at the same time the needs and concerns of both architecture and safety points of view. In particular, the model addresses the expressiveness requirements of both architecture and safety. This may be complicated to achieve from a tooling point of view. Moreover, this raise some organizational and cultural issues when a tool is imposed to different teams with great experience in other tools. To overcome this difficulty, an approach with two models is possible.

4.2.1.2 Multi-models approach

On the other side, SE and SA viewpoints can be expressed in distinct models. This approach is called “multi-models” approach. Mainly, these models are built from a tooling point of view, and therefore activity dependencies, more independently, i.e. that they are not based on a common structure imposed by one of the parties.

To illustrate it in the MBSE/MBSA context, a simple view of the approach is illustrated in Figure 12: Overview of two-model approach: one MBSE model and one MBSA model that require a method to keep them consistent, both from structural and behavioral points of view.

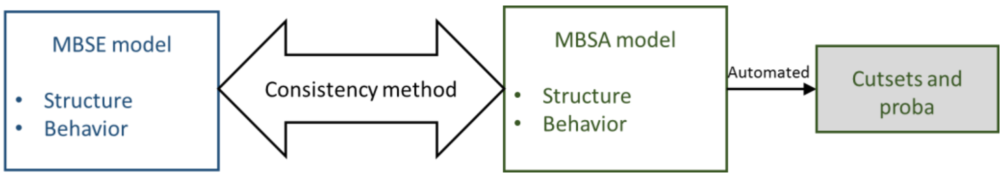

Figure 12: Overview of two-model approach

In the multi-models approach, the MBSA model is the support of modeling activities such as assumptions, validation, etc. and thus is fully considered as a model in the process.

Having a distinct structure for MBSE model and MBSA models is a source of inconsistencies to manage. Nevertheless, it also enables to have different structures for both models. For example, the bleed air flow normally runs from engines to air conditioning system. In case of failures, and only in case of failures, the air flow may run reversely. As this behavior is only dysfunctional, due to the different concerns of systems engineering and safety, the backward flow may not be mentioned in the MBSE model whereas it is necessary in MBSA model. It results in a directed flow in MBSE model and a bidirectionnal flow (or two directed flows) in MBSA model, i.e., a structural difference[1]. This difference is not an inconsistency, as it represents the same system part, from different point of views. Thus, the possibility to have distinct (and possibly different structures) is crucial to separate concerns and have a complete safety model.

[1] Note that in this example, a backflow protection may be necessary and then shall be modeled in both MBSE and MBSA domains.

4.2.1.3 Qualification and synthesis

Between the two "extreme" approaches presented in previous sections, intermediate approaches are possible. We first present a simple or simplistic comparison of these approaches before deepen it.

"Table 1: Some properties of unique model and two models approaches" presents some properties of both approaches. An intermediate approach would exhibit a different combination of properties: a unique model copied in two files[2] with different writing rights is a possible example; a common structure language with the possibility to define different structures for SE and SA (raising structure consistency issues) is another one.

Table 1: Some properties of unique model and two models approaches

	Unique model	Two distinct models
Storage (if stored as file)	1 file	2 files
Writing rights	SE and SA have writing rights on the whole model	SE and SA and only writing rights on their respective files
Language used for structure description	Common to SE and SA	Usually different for SE and SA
Structure description	Completely identical for SE and SA	May be different for SE and SA

It is unclear whether the unique model approach is compliant with independence requirement mentioned in Section 0. A unique model approach would limit the capability to address the specific concerns of respective concerns of system and safety modeling, as structural difference is complicated to manage in approach. On the contrary, unique model approaches eases version comparison and checking that both safety and architecture cover the same system scope without oversight (i.e., completeness).

During development of complex systems in complex organizational context, systems and safety teams cannot work on only one model file without possibility of duplicate it to work on different branches and version. As a consequence, the extreme form of unique approach (presented in Table 1: Some properties of unique model and two models approaches) with only 1 file is not applicable. Indeed, SE and SA models have different life cycles. In particular in some development phases, the SA analyst works based on a "baseline" of MBSE model, i.e., a frozen version, that is not up to date to the latest changes done by SE architect. This desynchronized work is required because a (safety) analysis takes some time to be performed. During this time, any external change is not desired. As a consequence, it is mandatory to have a duplication of the MBSE model (as separate files, or as versions or as branches in a version management tool) and manage some issues of consistency, including structural consistency. This "light unique model approach" with several branches requires to manage both structural and behavioral consistency between branches. For example, performing version merging is necessary in some steps.

Table 2 summarizes some characteristics of both approaches. Concerning the unique model, the benefits are clearly on the reconciliation phase (and consistency facilities), as it imposes constraints through the tooling which, as a result, induces constraints on the sequencing of the activities and the freedom in expressing the safety analysis. It is not clear whether the unique model approach is compliant with independence requirement mentioned in Section 0. Concerning the multi-models approach, the benefits of the independence between the realizations are at the expense of difficulties in the reconciliation phase.

Table 2: Some characteristics, benefits and difficulties of the approaches

[2] For simplification of the explanation, and amplification of the problems, the description do not consider data-bases, a current approach in the industry.

	Unique model	Two distinct models
Activity dependencies	System design before safety analysis	No constraints
Independence (ARP)	To be confirmed	Yes
Reconciliation between System and Safety viewpoints	Reconciliation de facto, done when defining the language of the tool	Reconciliation to do, Needs support (method and tools)

4.2.1.4 Approaches to models integration and viewpoints reconciliations

In the case of multi-models, one can distinguish the type of approaches to integrate these models, which can schematically be summed up in:

- Model Mappings and Linking Support: the models are linked among each other's through mappings or trace links for example, see Figure 13 part a);
- Integrated Modeling Languages and Formats: the models are projected onto a common formalism (e.g. language, metamodel, ontology), see Figure 13 part b);
- A combination of the two, the projected/abstracted models are linked among each other's through mappings or trace links for example, see Figure 13 part c);

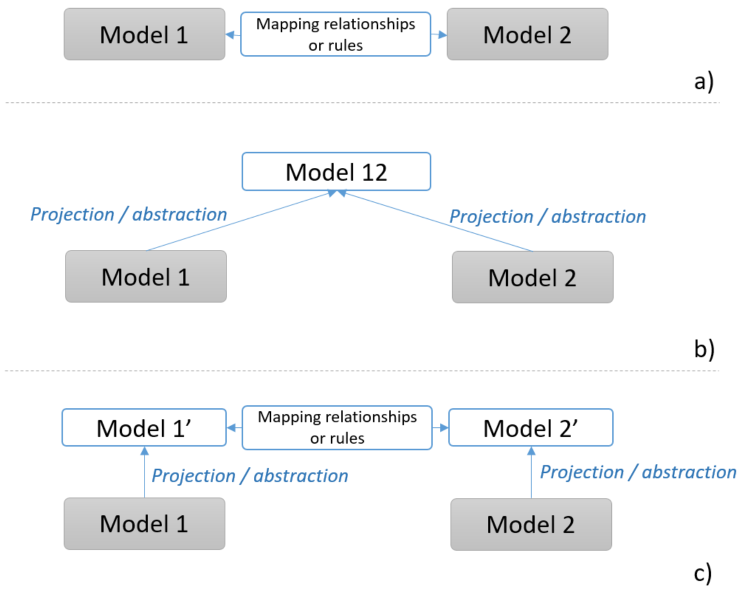

Figure 13: A simplified view of approaches to heterogeneous models integration

Considering projection and abstractions of models, several approaches exist:

1. Projection onto a metamodel (i.e. building a metamodel pivot and the associated transformations), an example in our immediate context is the Teepee/Interop approach (more details in Chapter 5);
2. Projection onto a shared ontology, an example in our immediate context is the KUBIK/Languages Federation approach (more details in Section 4.4.4);
3. Projection onto a dedicated language, an example in our immediate context is the S2ML approach (more details in Section 4.4.3).

Considering mapping relationships and/or rules, approaches in our immediate context are:

1. Using traceability links (with a pre-defined typology of links), cf. the 3DExperience approach of Dassault Aviation in Section 4.4.5;
2. Defining a dedicated mapping language, cf. the MOISE approach in Section 4.4.1.

For additional bibliographical references on the subject, we refer to a recent thesis from Feldman (Feldmann, Diagnosis and Resolution of Inconsistencies in Heterogeneous Models of Automated Production Systems, 2019) that summarizes in chapter 4.1 a related work towards approaches to Integrate Heterogeneous Models.

4.3 Consistency approaches for engineering models

This section introduces different approaches from the related literature, which provide concepts, methods and technics related to inconsistency management in various domains: software, systems engineering, production and mechatronics. In the following, first, are described approaches to help define correspondences between multiple models; then approaches to (Semi-) automate inconsistency management (for the detection and the identification activities); finally, existing management frameworks are described.

4.3.1 Approaches to build relations between multiple models

Models defined separately by different teams with different concerns in mind, expressing viewpoints on the system under study, have to be combined in some way. This section present some literature oriented towards helping (providing automation) the users and conceivers to perform such a task.

As noted in (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015), much research related to the detection of semantic overlap (which may be partial) of models has been conducted in a variety of domains. And that however, the fully automated inference of such model relationships remains an open challenge.

In the related literature, four distinct classes of approaches define correspondences between models (with a focus on overlap identification) can be identified: the exploitation of representation conventions; the use of a unifying, domain-spanning ontology; the human inspection; and similarity analysis.

4.3.1.1 Exploitation of Representation Conventions

The exploitation of (syntactic) representation conventions is the "oldest [sic] and most commonly used form of detecting a model overlap" (Spanoudakis & Zisman, Inconsistency Management in Software Engineering: Survey and Open Research Issues, 2001). Approaches in this class are typically based on (syntactic) unification algorithms. These approaches mainly applied on models (instance level) can also be applied at the definition level.

Generally, unification algorithms perform a syntactic matching between terms, some approaches are introduced:

- ***predicate matching***, based on symbolic equality of the predicates, as employed by Finkelstein et al. in (Finkelstein, Gabbay, Hunter, Kramer, & Nuseibeh, 1994) for instance, when working with distributed databases of logical expressions. Similar work is conducted by Easterbrook et al. in (Easterbrook, Finkelstein, Kramer, & Nuseibeh, 1994);
- overlap identification based purely on ***name matching*** is employed for consistency checking of requirements using model checking techniques in (Heitmeyer, Jeffords, & Labaw, 1996);
- utilizing the **correspondences** defined as part of specifying model (Czarnecki & Helsen, 2006) (definition of a **syntactic correspondence** between elements of a meta-model), or **graph**

transformations (Königs, 2005) (e.g. *triple graph grammars* – TGGs -, which also define syntactic correspondences among (meta-)model elements).

4.3.1.2 Use of Unifying, Domain-Spanning Ontologies

Using a common, shared ontology is a second approach to enabling the identification of overlap between models. This approach requires the authors of the models to tag model elements with elements from a common ontology shared among all stakeholders. Examples are (Boehm & In, 1996), or Hehenberger et al. in (Hehenberger, Egyed, & Zeman, 2010) for the specific case of developing mechatronic systems.

4.3.1.3 Human Inspection

A third approach to identifying overlap is the use of human inspection (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015). Typically, approaches designed for identifying model overlap using human inspection methods implement some forms of aid to the stakeholder responsible for identifying inconsistencies (e.g., through visual aids, or various ways of representing the models). Examples are:

- the Synoptic system presented in (Easterbrook S. , 1991), which investigates various methods including the provision of visual aids for browsing models, graphical selection of elements from two partial models, and the recording of overlap;
- similar approaches are provided in (Jackson, 1997).

As highlight in (Spanoudakis & Zisman, Inconsistency Management in Software Engineering: Survey and Open Research Issues, 2001), identifying overlap using human inspection can be very exact, but is also highly labor intensive and time consuming; particularly when developing highly complex systems using (expectedly) highly interrelated models.

However, there are a number of ongoing research efforts, particularly geared towards visualizing and enabling the analysis of large data sets (such as sets of heterogeneous models), more details in (Basole, Qamar, Park, Paredis, & McGinnis, 2015).

Within a development process, this human inspection can be formalized by reviews (and more particularly consistency reviews), that act on different granularity of models, on different pre-defined scope to set up and justify the consistency state of the models combination (status of the dependencies, completeness and correctness of the consistency analysis).

4.3.1.4 Similarity Analysis

Equivalences, common elements, or relations between models (or views in a model) may be detected using similarity analysis. Similarity analysis exploits the fact that modeling languages incorporate constructs which imply or strongly suggest the existence of certain semantic overlap relations (Spanoudakis & Zisman, Inconsistency Management in Software Engineering: Survey and Open Research Issues, 2001). For instance, the fact that two model elements carry the same name does not entail their semantic equivalence, but merely serves as evidence to suggest their equivalence.

As noted by (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015), similarity analysis is one of the very few techniques where abductive and inductive inference is employed rather than relying on deductive reasoning. However, most approaches using similarity analysis use ad-hoc methods for measuring similarity (e.g., weighted sums, arbitrary scores, incorporation of weak assumptions), hence putting their general applicability in model-based approaches in question.

Some examples are:

- In (Spanoudakis & Finkelstein, 1997), the authors analyze structural similarity of models to identify overlap based on the weighted bipartite graph matching problem;

- Similarity analysis for semantic overlap identification has also been used in research related to ontology matching and database schema matching. For instance, Ehrig and Sure use a series of similarity rules (e.g., equal names, equal types, equal sub-concepts) and calculate a weighted similarity score from these (Ehrig & Sure, 2004).
- (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015), where a probabilistic learning approach support in semi-automatically identifying semantic overlaps between models is presented.

4.3.2 Approaches to (Semi-)Automated Inconsistency Management

Detection and identification of inconsistencies may be helped by a tooling support. As summarized recently by Feldman (Feldmann, Diagnosis and Resolution of Inconsistencies in Heterogeneous Models of Automated Production Systems, 2019), approaches to (Semi-)Automated Inconsistency Management can broadly be classified into three different categories:

Approaches that make use of logical reasoning and theorem proving aim at a formal knowledge base in terms of a set of axioms, from which the (in-)consistency of a set of models can be formally concluded. A seminal work from Finkelstein et al. (Finkelstein, Gabbay, Hunter, Kramer, & Nuseibeh, 1994) make use of first-order logic to diagnose inconsistencies within multi-view software models such as class diagrams, sequence diagrams, etc. In order to capture the links between the different entities within the models, they manually add statements to the models. Whether an inconsistency occurs or not is inferred by an automated theorem prover. Another example is the work by Van Der Straeten et al. (Van Der Straeten, Mens, Simmonds, & Jonckers, 2003), where a UML profile is introduced to express both consistency between models within the same version (horizontal consistency) and between different versions of the same model (evolution consistency). By means of Description Logics (DLs), different types of consistency can be checked within the UML profile, e.g., structural inconsistencies between class, sequence and class diagrams such as classless instances or dangling references as well as incompatible behavior definitions between state charts and sequence diagrams.

Rule- and pattern-based inconsistency management approaches, where (in-)consistency rules and patterns are used to describe the circumstances under which a model is (in-)consistent. Similarly to approaches above, rule-based inconsistency management aims at applying a rule base that describes either the sufficient conditions that a model must satisfy for it to be considered consistent (Hehenberger, Egyed, & Zeman, 2010) – that is, rules are used as *positive constraints* (e.g. Egyed et al. (Egyed, Zeman, Hehenberger, & Demuth, 2018))– or as *negative constraints(e.g. Mens et al.* (Mens, Van Der Straeten, & D'Hondt, 2006)*)*, which represent the sufficient conditions that indicate an inconsistency (Herzig, Qamar, & Paredis, 2014). Examples of pattern-based Approaches are: i) Herzig et al. (Herzig & Paredis, 2014), (Herzig, Qamar, & Paredis, 2014), that propose a framework for the diagnosis of inconsistencies, in which graph patterns are used to define inconsistency diagnosis rules, or ii) the ontological integration approach from Wouters et al. (Wouters, Creff, Bella, & Koudri, 2017) described in more details in Section 4.4.4. These approaches relies on Semantic Web Technologies, the latter providing an extension to behavioral considerations. Behaviors can be represented using a wide variety of ways, such as state machines, activities, processes and algorithms. In xOWL, behaviors are represented as algorithmic structures containing actions executed on the ontologies. xOWL is inspired from the OMG fUML language (OMG, 2018), and implements an imperative paradigm enhanced with some features borrowed from the functional paradigm (the language is also able to manipulate lambda expressions). In a Systems of Systems engineering context, some experiments where performed to illustrate behavioral consistency at IRT SystemX between a state-machine (expressed in SystemArchitect NAFV3) and scenarios in Papyrus SysML. In the approach, the state machine (behavior definition) is interpreted as rules; scenarios are then run and their conformity checked, making possible to detect incompatible behavior definitions between state charts and sequence diagrams.

Model synchronization approaches, it is aimed at avoiding inconsistencies through synchronizations between the models under investigation. Contrary to logical reasoning and theorem proving approaches, in which a formal system needs to be formulated as the basis to manage inconsistencies, as well as rule- or pattern-based approaches, which make use of a flexible set of rules or patterns, model synchronizations aim at unidirectional or bidirectional transformations between the models involved in the engineering process. Consequently, in such approaches, transformation rules are formulated that capture, how entities in one model are related to entities in another model. One example for such synchronizational approaches is the concept of Giese et al. (Giese & Wagner, 2009), where the authors make use of so-called Triple Graph Grammars (TGGs) to specify bidirectional syn- chronizations between models. Another is the one from Rieke et al. (Rieke, Dorociak, Sudmann, Gausemeier, & Schäfer, 2012) with an extension towards managing consistency between behavioral models of the mechatronic system.

4.3.3 Existing Inconsistency Management Framework

S2C is designed to deal with the consistency between system engineering and safety. These two disciplines can rely on the use of models. It would therefore be interesting to see the works that has been carried out with the aim of addressing the coherence between different heterogeneous models.

What appears to be the first published conceptual frameworks related specifically to what has later become known as inconsistency management were developed by Finkelstein (Finkelstein, Kramer, Nuseibeh, Finkelstein, & Goedicke, 1992) (Finkelstein, Gabbay, Hunter, Kramer, & Nuseibeh, 1994) . In this work was introduced the Viewpoint-Oriented Systems Engineering (VOSE) framework, the core principle is the use of viewpoints to partition the system specification (e.g., functional hierarchy and system block diagram), development method (e.g., “top-down” and “bottom-up”) and formal representations used to express the system specifications.

Another work on a consistency management approach was carried out by (Nuseibeh, Easterbrook, & Russo, 2000). The framework defines several activities related to identify and solve inconsistencies. It can be noted that these activities are close to the activities we have listed above, which are the result of the work carried out by (Spanoudakis & Zisman, Inconsistency Management in Software Engineering: Survey and Open Research Issues, 2001).

The objective of this approach is to define rules for consistency in order to check consistency. Once an inconsistency has been detected, it is diagnosed, its cause identified, and the inconsistency is classified. Thereafter, the inconsistency is handled, which may result in the resolution (or “fixing”) of the inconsistency, or in tolerating it (e.g., if the cost of fixing exceeds the benefits). Tolerating can include ignoring and circumventing the inconsistency. Then, the last activity in the process is the monitoring of the consequences of a particular handling action.

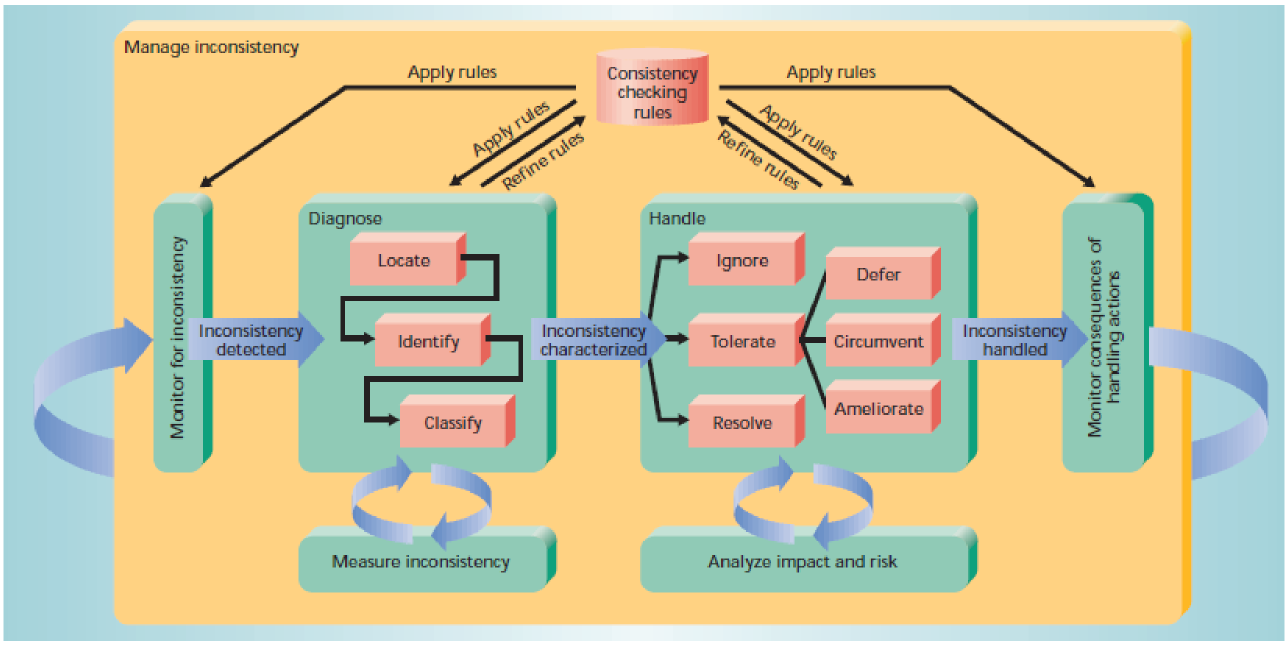

Figure 14: Framework for inconsistency management proposed by (Nuseibeh, Easterbrook, & Russo, 2000)

A later work presented in (Hehenberger, Egyed, & Zeman, 2010) addresses the need to have an automatic analysis of the consistency of a set of engineering data that are represented in the form of models. The application case was a mechatronic system. The objective was to make electronic models, control models, mechanical models and software engineering models consistent. For this purpose, an approach inspired from the software engineering has been developed.

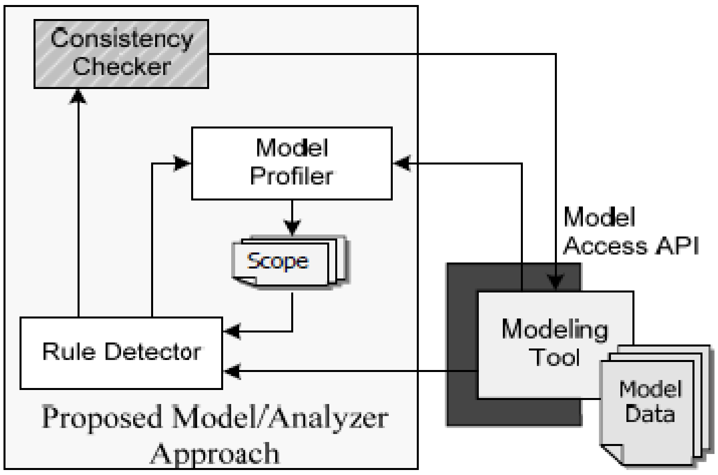

Figure 15: The proposed approach in (Hehenberger, Egyed, & Zeman, 2010)

Figure 15 shows the approach developed. It shows two possible paths of reasoning.

- The first path is from the “Modeling Tool” to the “Rule Detector”. In this "Rule Detector" we identify the set of consistency rules that we want to check. This module can also decide which subset of consistency rules has to be evaluated when the model changes. It then goes to the "Consistency Checker" which does the reasoning. We can consider this “Consistency Checker” as an inference engine. The results of this "Consistency Checker" are then reinjected in the "Modeling Tool". Nevertheless, it is unclear whether it modifies the model, give a consistent model for comparison or other feature

- ➢ The second path aims to address the consistency of a specific perimeter of a model. To do this, an intermediate brick between the "Modeling Tool" and the "Rule Detector" has been developed. This is what is called "Model Profiler" which allows us to determine a "Scope" and to play a set of rules adapted to this scope of the model.

This approach has been tooled and experimented on a mechatronic module. An ontology of this module has been developed in order to apply the approach.

A last related work is an approach presented in (Feldmann, et al., 2015). This approach is summarized in the following figure.

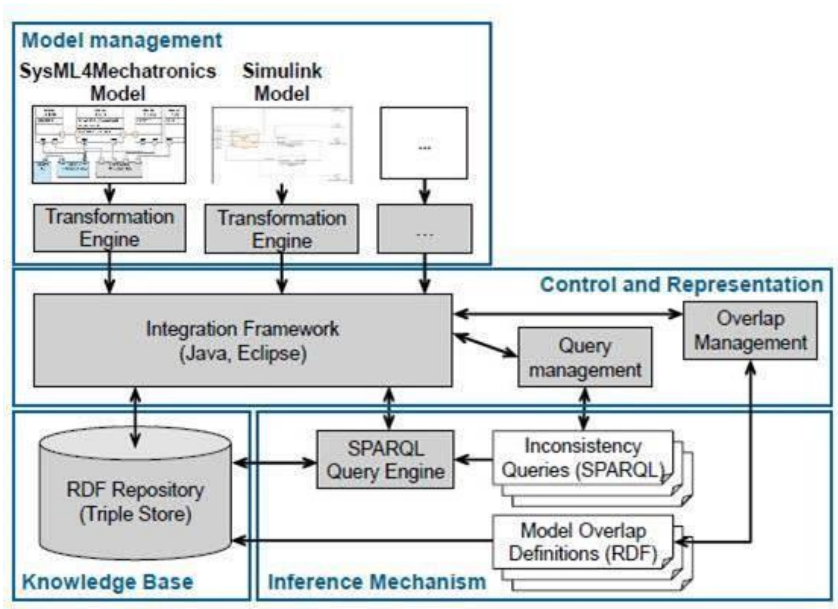

Figure 16: Basic architecture of the technology demonstrator in (Feldmann, et autres, 2015)

The model management brick allows a model transformation in order to move to an integrated technical space. It is a transformation to an RDF representation which is then stored in the triple store which represents the knowledge base. The inference mechanism brick is to identify inconsistencies in the knowledge base. And it is in the control and representation brick, the queries are managed and the results can then be interpreted and visualized for the user.

4.4 MBSE/MBSA consistency

This part is a review of practices of consistency between design and safety. The scope of this version is limited to models usable for the PSSA context, and more precisely, failure conditions (FC) assessment. For design, we consider MBSE models (and not description or requirements in natural language). For safety, we consider only failure propagation models (see definitions of Section 2). We mainly focus on the structure of models. The review is not exhaustive and focuses on the works of project members and recent works.

4.4.1 Capella safety viewpoint and coupling with Safety Architect

The Capella safety viewpoint is a Capella extension enabling model based safety analysis, based on a Capella MBSE model, with the Safety Architect tool (developed by All4Tec). In a co-engineering context, it supports also an iterative model based safety analysis workflow, thanks to a "diff" capability, enabling multiple safety analysis iterations, between Capella and Safety Architect. The objective of the coupling between Capella and Safety Architect tools is to ease the update of MBSA (Safety Architect) model following a new version of MBSE (Capella) model. The toolchain is an internal tool by both Thales and All4Tec.

Through the Capella safety viewpoint, the safety engineer identifies a critical path for each failure condition. This path can be defined in the functional architecture (it is a Capella functional chain), in the physical architecture (Capella physical path) or in both architectures linked by allocations (based on a functional chain). Elements involved in critical path, i.e., a subset of the MBSE model, are then exported to Safety Architect. Inside these elements, the safety engineer models failure modes and propagations, using Safety Architect. Then, the tool produces fault trees. When changes are done in the MBSE model, the toolchain highlights the modifications, enabling the safety engineer to efficiently update the safety model, without losing the added failure modes and propagations.

This method is a hybrid between unique model and two models approaches. Indeed, as safety analysis begins in the MBSE model, by defining critical paths, the independence between system design and safety analysis is partial. When safety analysis requires elements specific to dysfunctional behavior and absent from the MBSE model (such as the example of backflow in section 4.2.1.2), the addition of corresponding elements in the MBSA model is not taken into account, encouraging MBSE model modification for safety concerns.

The mapping between MBSE elements and MBSA elements is fully automated. If a textual justification is required, it is recorded and traced in a requirement management tool, enabling to link justification to a model or a model element.

4.4.2 MOISE MBSE/MBSA method

The MOISE project from IRT Saint Exupéry has defined a method for consistency between MBSE and MBSA models. The objective of the method is to assist safety model review by system architect. This method belongs fully to the category of “multi-model”. Dimensions considered are (cf. Section 2.1): horizontal, semantic, structural, and evolution.

Models integration and viewpoints reconciliations and approach to inconsistency management

The approach promotes the use of filtering from the original models and the use of limited transformations (like flows to ports, ..).

The method defines objects that are dedicated to consistency and called “synchronization points”. Each synchronization point links 0 or more MBSA model elements with 0 or more MBSE model elements. Thus, the method supports structural differences and their justification: the structural differences are not seen as inconsistencies provided that they are justified. Each synchronization point contains a justification and a consistency status.

The status of the synchronization point intends to keep trace of validation (done during the review) and to ease version management: when a new model version is released, the status of the synchronization points that are linked to updated model elements is changed to point out the need of re-validation of model elements.

The behavior is seen as attached to output ports and only formalized in MBSA model. The validation of behavior is part of the validation done in review.

This method is defined through a meta-model linking model element types and consistency objects. It has been applied through a table format. Though completeness and correctness rules have been defined, no tooling is offered to check them. The method is further described in [MOISE].

The MBSE model element types that are taken into account are: functional breakdown, functional ports, “functional to physical” allocations, physical elements and physical ports.

4.4.3 S2ML method

In (Legendre, Ingénierie système et Sûreté de fonctionnement : Méthodologie de synchronisation des modèles d’architecture système et d’analyse de risques, 2017), a method to assist interactions between several engineering domains is presented. The presented case study proposes a concrete application of the method for MBSE/MBSA consistency. The MBSE/MBSA method is deepen in (Batteux, Prosvirnova, & Rauzy, 2019). It fully belongs to the category of “multi-model”. Dimensions considered are (cf. Section 2.1): both horizontal and vertical, syntactic and semantic, and structural.

Models integration and viewpoints reconciliations and approach to inconsistency management

The two models are both transformed in a common formalism: the S2ML language that focuses on the structure of models and omits their behavior. Then the two transformed models are compared (as shown in Figure 17). If an MBSE element and an MBSA element have the same name they are considered as consistent. The detected inconsistencies (structural differences) highlighted. The user can either modify one of the two models to solve them or establish some manual matching between model elements. On top of 1-1 matching, it is possible to declare that 1 model element is consistent with N>1 model element or that 1 model element has no equivalent in the other model.

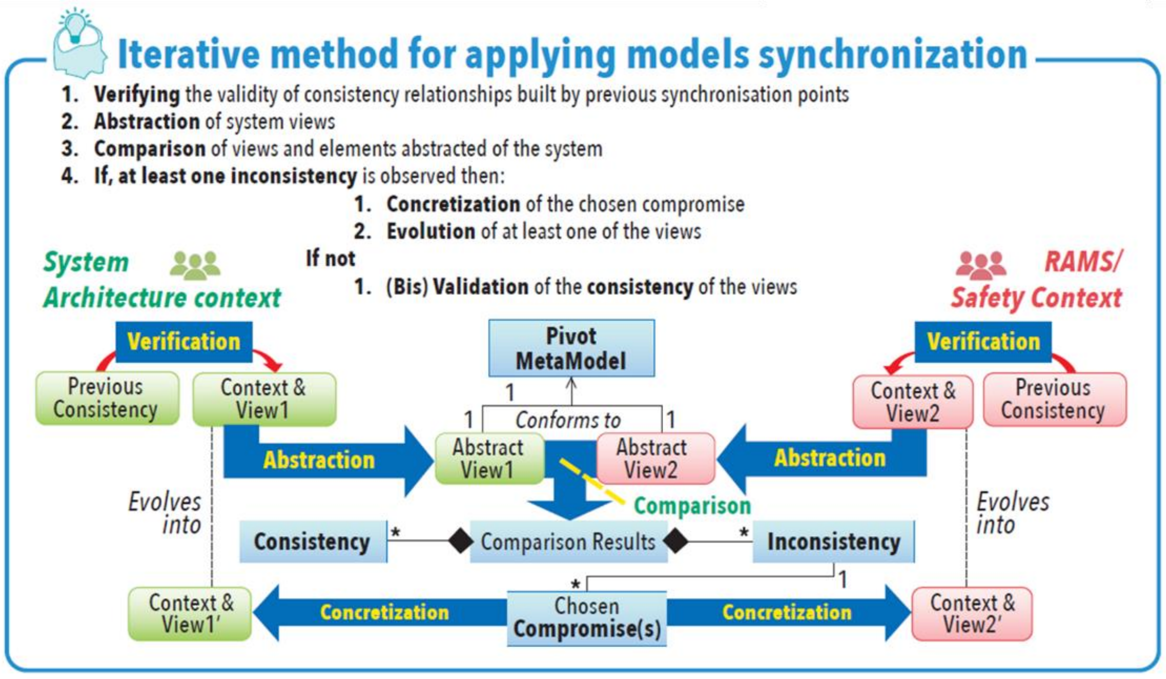
Iterative method for applying models synchronization
1. Verifying the validity of consistency relationships built by previous synchronisation points
2. Abstraction of system views
3. Comparison of views and elements abstracted of the system
4. If, at least one inconsistency is observed then:
1. Concretization of the chosen compromise
2. Evolution of at least one of the views
If not
1. (Bis) Validation of the consistency of the views
System Architecture context
RAMS/ Safety Context
Verification
Previous Consistency
Context & View1
Pivot MetaModel
Conforms to
Abstract View1
Abstract View2
Context & View2
Abstraction
Comparison
Evolves into
Consistency
Comparison Results
Inconsistency
Chosen Compromise(s)
Concretization
Context & View1'
Context & View2'

Figure 17: Overview of S2ML method from (Legendre, Lanusse, & Rauzy, System Engineering and Dependability: Methodology of Model Synchronization between System Architecture Models and Risk Analysis, 2019)

The case study for MBSE/MBSA consistency addresses the following MBSE model element types: use cases, functional breakdown, allocations, physical elements and physical ports.

With respect to the MOISE method, this method underlines the need to have a pivot language to compare MBSE and MBSA structure.

The tooling is limited to Excel report generation without safeguard on user inputs. As a consequence, highlight of structural difference is textual. Transformation of models are not tooled.

4.4.4 An ontological integration approach (KUBIK experiment)

This section describes an ontological integration approach based on the report of an experiment on a collaborative space solution for which a prototype tool was implemented at the IRT SystemX under the name of KUBIK, integrating the OpenAltaRica tool[3] for a demonstration in 2016 (illustration Figure 18 – right). Dimensions considered are (cf. Section 2.1): both horizontal and vertical, semantic, structural and behavioral, and evolution.

Models integration and viewpoints reconciliations

The approach relies on a digital collaborative space based on the federation of modeling languages (Wouters, Creff, Bella, & Koudri, 2017), i.e. heterogeneous languages handled by the different engineering and business domains , as illustrated in Figure 18 – left. The collaborative space, aims to meet the challenges of collaborative systems engineering (Wouters, Creff, Bella, & Koudri, 2017), i.e.: Building the Collaboration, around a shared Vocabulary; Building the Collaboration, and Orchestration; Controlled Exposition of the artefacts; Preserve consistency with Domain Rules; and allow Networks of Collaborations.

[3] https://www.openaltarica.fr/

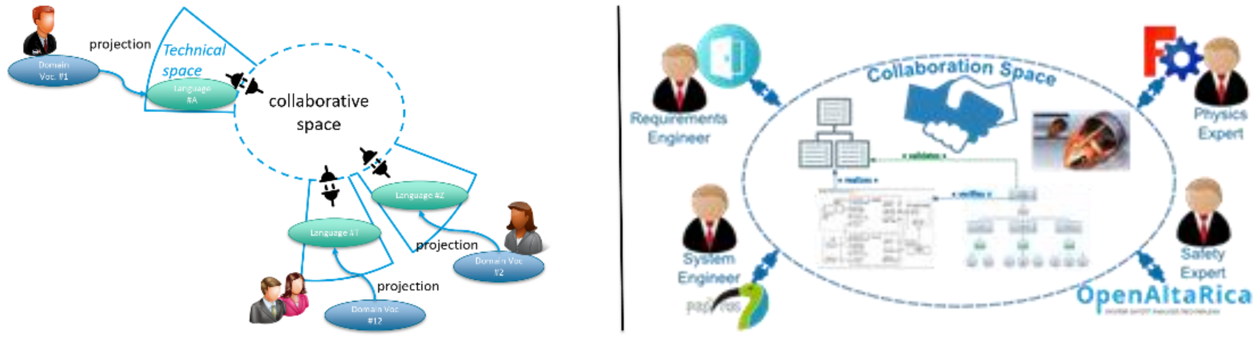

Figure 18: Collaborative space principles (left) – A collaboration space example including Safety concerns

The heterogeneity of the business domains, as well as the heterogeneity of the languages supporting them, results in difficulties concerning:

1. the explanation of the business domains (Domains Voc. in Figure 18 – left side),
2. the elicitation of the projections of the business domains in the languages (in the technical spaces),
3. as well as the semantic alignment in the collaborative space (according to the distance between these business domains).

The collaboration space (cf. Figure 18 – right side) is an application integration platform that allows different engineering and business domains, via their tools, to share and exchange data during the systems development cycle.

This space is built around the modeling languages federation (Wouters, Creff, Bella, & Koudri, Towards Semantic-Aware Collaborations in Systems Engineering, 2017), which from a language and MBSE point of view, proposes to reduce the languages to a "homogeneous" representation of the information (projection onto an ontological support (Wouters & Gervais, xOWL: An Executable Modeling Language for Domain Experts, 2011)), assuming a possible translation in a common logic. The semantic projection captures the modeling intention: capture of the usage of the language by its denotation, formalization of Intention-Specific Modeling Languages - ISML, and ontology mapping techniques. Figure 19 illustrates these principles where two stakeholders collaborates via two modeling tools through projections onto the collaborative space (semantic reasoning, at the center of the Figure).

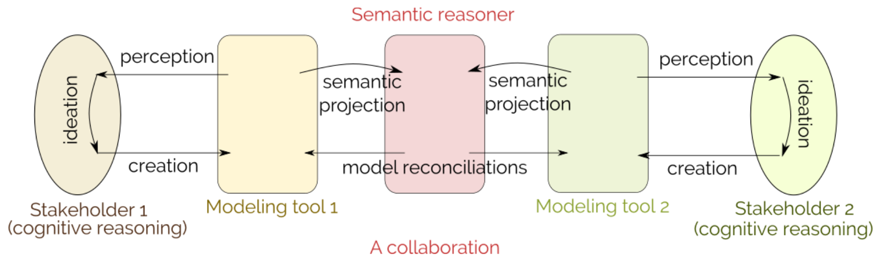

Figure 19: Two stakeholders collaborating via modeling tools

In the experiment illustrated Figure 18 – right, Requirements and their criticality level where considered in the Doors tool, the MBSE model was represented in Papyrus tool (activity diagrams where used to represent the organic architecture), MBSA related model was performed in Open Altarica tool, and some alternative physical descriptions in FreeCAD to allow trade-offs. Considering the MBSE/MBSA part, note that only structural modeling elements were projected onto the ontological space (at the physical modeling level). One important observation of this experiment was a higher complexity in the projection mapping between MBSE and MBSA compared to MBSE and other engineering specialties, enforcing our belief on the fact that automatic transformations between the two domains is costly and not a solution.

Approach to inconsistency management

The current approach is related to pattern-based automation ones (cf. Section 4.3.2). Following a semantic-web approach, the inconsistency detection is based on native services:

- specification of rules on top of SPARQL Protocol and RDF Query Language (SPARQL), for the purpose of specifying inconsistency diagnosis patterns and of matching them against the models, e.g. in our example rules

are set to check whether any critical function (e.g. SIL4 or DAL A) is covered by the safety analysis to ensure risk mitigation, and that the cut orders are acceptable,
- inference of new information based on rules,
- queries and impact analysis, combined with artifact versioning.

4.4.5 Dassault Aviation approach (3DExperience)

This section describes the tooled approach followed by Dassault Aviation, based on the philosophy of the 3DExperience platform (details on the platform can be found in [S2C 3DExperience experiment] and in Chapter 5).

Dimensions considered are (cf. Section 2.1): both horizontal and vertical, semantic, structural, and evolution.

Models integration and viewpoints reconciliations

The levels of collaboration defined by the 3DExperience approach are shown in Figure 20 as follows:

0. interoperability (import/export of files),
1. federation (addition of navigation services, traceability and review),
2. integration (life cycle, configuration, P&O, simulation - e. g. FMU, research,…).

As introduced in Section 4.2.1.4, the approach falls into the category: Using traceability links (with a pre-defined typology of links). Indeed, "System Traceability “ (SysTRA) allows to access external modeling content build with authoring tools, and provides traceability services (links, coverage analysis, etc.).

On top of SysTRA, using its facilities, Dassault Aviation has developed a bridge to the MBSA tool Cecilia, making it possible to expose MBSA elements and relate them to MBSE elements. We do not have more details about it.

Traceability and reviews for consistency purpose are then performed.

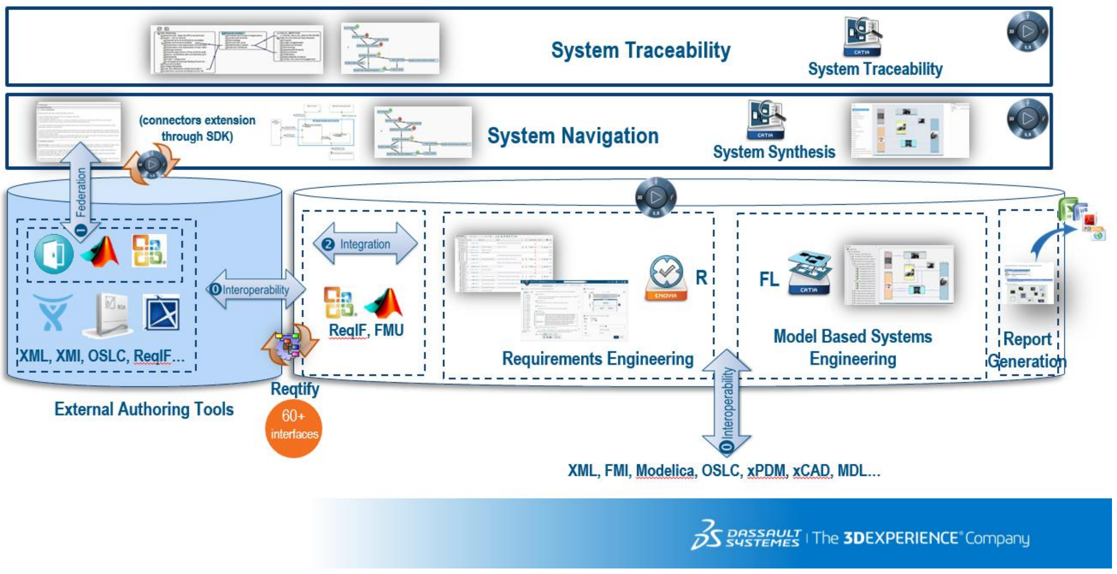

Figure 20 - Overview of Systems Traceability, extracted from Dassault Systèmes support (version R2016x)
In version of the platform (R2018x) used for the experimentation, the System Synthesis App is not available to the user, all access are made via SysTRA

4.4.6 SAVI approach

The System Architecture Virtual Integration (SAVI) program (J., Chilenski, & Ward, SAVI AFE 61 Report - Summary Final Report) is a multi-year program intended to implement the capability to virtually integrate complex hardware/software systems before designs are committed to physical form. The SAVI program is led by the Aerospace Vehicle Systems Institute (AVSI), a cooperative research environment.

SAVI aims at providing a methodology for managing the exponentially increasing complexity of modern aerospace systems. The following descriptions are based on publicly available documents provided by the institute.

Dimensions considered are (cf. Section 2.1): both horizontal and vertical, syntactic and semantic, structural, and evolution.

Models integration and viewpoints reconciliations

As indicated in the docs, the SAVI program is an attempt to provide a solution to increasingly complex hardware and software used in critical aerospace systems posing integration problems nearing the limits of complexity effectively handled by the current system acquisition process. The scope is broad. Figure 21 illustrates two of the *core elements* of SAVI, the Model Data Exchange Layer and the Model Repositories, and how they interact with other elements to allow **virtual integration** of a system of systems.

The SAVI Program is developing the definitions for the Model Repository (the data structure needed for information storage and analysis) and the Model Data Exchange Layer (the data transformations needed for information interchange) These components enable multiple business entities to exchange diverse forms of information utilizing multiple tools SAVI's model oriented approach encourages frequent and early system validations (virtual integrations) to help reduce rework, and thereby cycle-time and cost. Tools chains (integration of multiple tools) is built to enable new and more effective integration checks and analyses.

Found in (J., Chilenski, & Kerstetter, SAVI AFE 61S1 Report - Summary Final Report), SAVI explores the use of standards (e.g. ISO 10303-239, STEP AP-239, Data Exchange Specification – DEX), but also inter-model consistency checking seems to rely on ontological approaches like the one presented in (Herzig & Paredis, A Conceptual Basis for Inconsistency Management in Model-based Systems Engineering, 2014) or (Herzig, Qamar, & Paredis, An Approach to Identifying Inconsistencies in Model-based Systems Engineering, 2014).

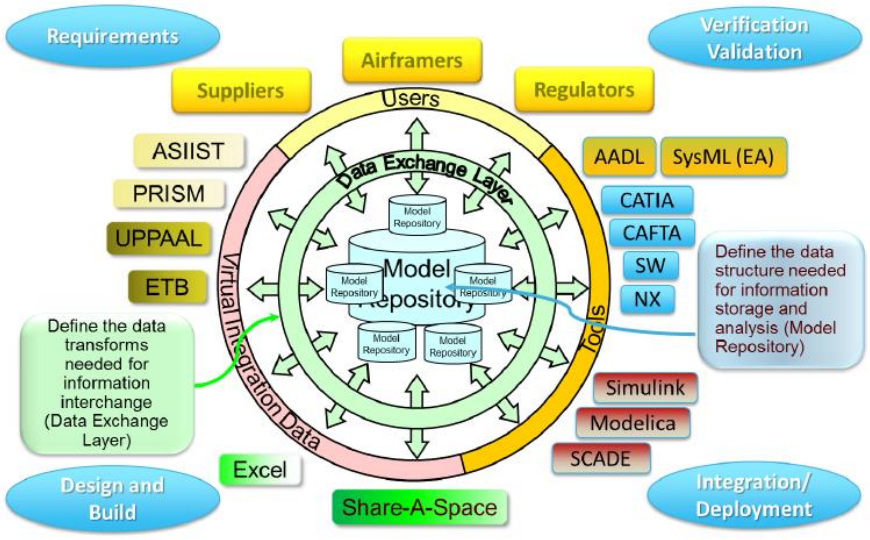

Figure 21 - Overview of the SAVI program approach

Approach to inconsistency management

As found in (Redman, Importance of Consistency Checking in the SAVI Virtual Integration Process (VIP), 2014), the virtual integration is an early and continuous integrated analysis, with the following characteristics:

- Proof-based (consistency checked – but not all with formal models);
- Component-based (hierarchical models);
- Model-based (annotated models).

Illustrates the SAVI consistency checking process in the “Discovery & Processing” activity, based on rules and tolerances (Pollari & Shaw, 2017). Few details are provided in public documents. The Figure highlights that consistency between two models exists when the dependence relations between those two models are satisfied (Some dependence relations can be detected automatically, some tools are using patterns to assist; some dependence relations will – always- require manual identification). Fidelity of consistency is proportional to the effort put into consistency modeling. Two levels of verification are defined: i) consistency checks at a fine dependency level, and ii) verification check at the global level (conformance to higher level checks – rules).

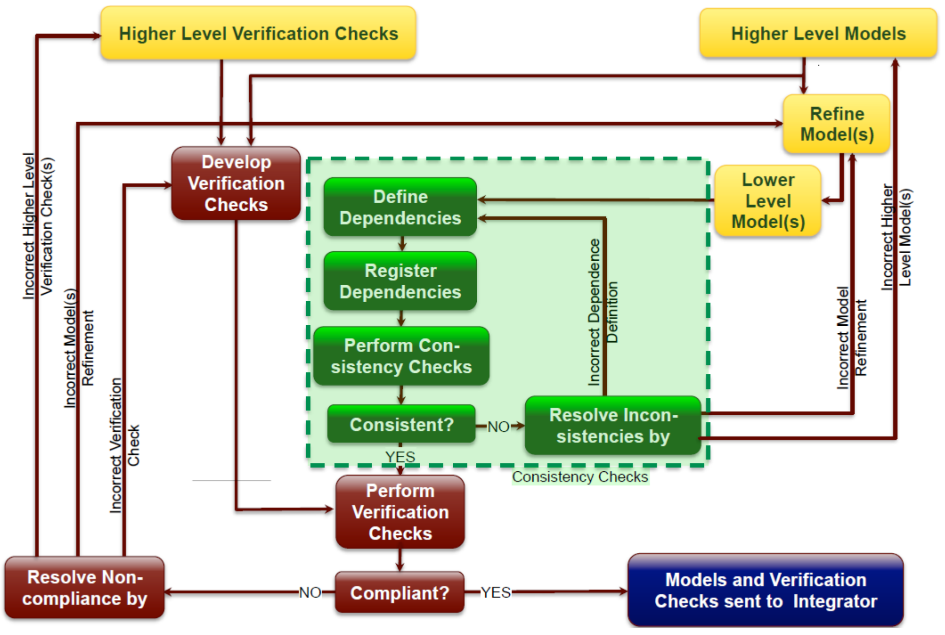
Higher Level Verification Checks
Higher Level Models
Refine Model(s)
Develop Verification Checks
Define Dependencies
Lower Level Model(s)
Register Dependencies
Perform Consistency Checks
Consistent?
NO
Resolve Inconsistencies by
YES
Consistency Checks
Incorrect Dependence Definition
Incorrect Model Refinement
Incorrect Higher Level Model(s)
Incorrect Higher Level Verification Check(s)
Incorrect Model(s) Refinement
Incorrect Verification Check
Perform Verification Checks
Resolve Non-compliance by
NO
Compliant?
YES
Models and Verification Checks sent to Integrator

Figure 22 - Overview of the SAVI consistency checking process

Work towards Safety

Final reports and presentations cited before define the focus for Safety within the SAVI V 1.0A, as:

- Exercised in V. 1.0A FTA, FHA, FMECA, PSSA
- Identified, but not exercised: RCA, CCA, SSA

Taken from (J., Chilenski, & Ward, SAVI AFE 61 Report - Summary Final Report): “The process so far includes generation of the fault tree analysis (FTA) based upon an assumed set of hazards presented as inputs to the process using a spreadsheet-based Functional Hazard Analysis (FHA). A Failure Modes and Effects Analysis (FMEA) has been demonstrated but a full-blown failure modes, effects, and criticality analysis (FMECA) has not been demonstrated. Other safety analyses like the common mode analysis (CMA), the zonal safety analysis (ZSA), and evaluation of a system's development or design assurance level (DAL) have not yet been demonstrated from the SAVI model set”.

Relying on information found in (Redman, The System Architecture Virtual Integration Program (SAVI)) and illustrated by Figure 23, the systems and software architecture is build using the AADL formalism, a unique model approach (as described in Section 4.2.1.1), with transformations to safety analysis spaces. No more details are provided.

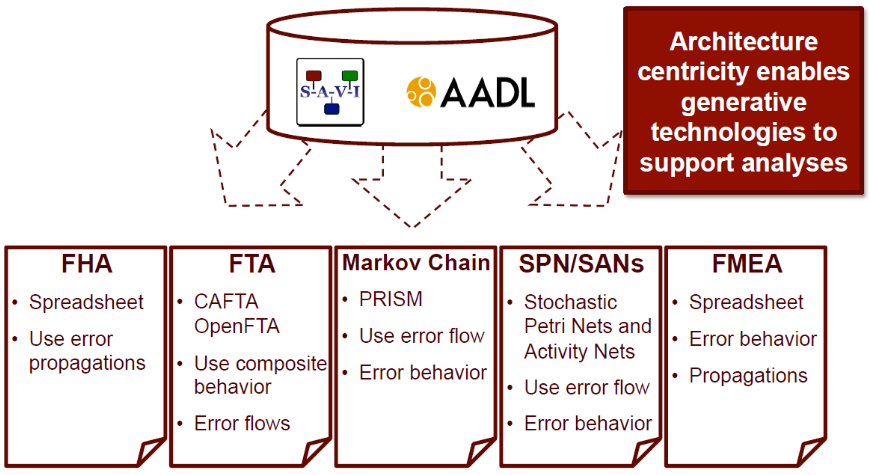

Figure 23 - Overview of the SAVI Safety environment

4.5 MBSE/FTA consistency

To ensure consistency between MBSE and FTA, in the end of comprehensive review, two possible approaches have been identified:

- Building a fault tree following a method that keeps trace of MBSE breakdown and add traceability between on the one hand leafs and gates of the tree and on the other hand elements from MBSE model. E.g., in [MOISE] appendix B about Medini.
- Decomposing fault tree based on MBSE model breakdown. This approach can be seen in Component Fault Trees and Safety Architect (alone without Capella connector).

Note that the difference between unique model or multi-model approaches is still efficient when discussing MBSE/FTA: Medini and Safety Architect can be classified as “multi-model” methods whereas CFT defines an unique model.

In this case, dimensions considered are (cf. Section 2.1): both horizontal, and structural.

4.6 SA/SA consistency

This section will describe in further version the existing approaches to ensure consistency between the different safety analyses.

Dimensions considered are (cf. Section 2.1): vertical, structural, and evolution.

4.7 Comparison of methods

The organization of this comparison follows the consistency activities described in Chapter 3: The definition of correspondences between artefacts (Heterogeneous models considerations, Definition of correspondences between artefacts), the detection of inconsistencies, the diagnosis of inconsistencies, and Inconsistency handling.

First, the kind of **modeling approach** is reported, i.e. if the approach considers multiple models or not (cf. unique vs. multi-model distinction in Section 0). In the case of heterogeneity, one may distinguish the **kind of approaches to integrate** these models (cf. Section 4.2.1.4), which can be:

- Integrated Modelling Languages and Formats: the models are projected onto a common formalism (e.g. language, metamodel, ontology);
- Model Mappings and Linking Support: the models are linked (without projection, abstraction or filtering) among each other’s through mappings or tracelinks for example.

Then, considering **the definition of correspondences between artefacts**, the matching part (to identify overlaps or to identify links between artefacts) can be manual or be focused on providing automation support, i.e.:

- Help (semi-automation/automation);
- Manual

As soon as, these correspondences are made is also listed, which can be: i) when defining the language (or building the collaboration tool, i.e. at metamodeling level), ii) when building the collaboration, iii) when considering the models during a project.

Next, considering the **detection of inconsistencies**, two concerns are revealed: i) the kind of approach followed and if it provides automation, and ii) the **inconsistency type** considered (structural or behavioral, see Section 2.1). The approaches found in the literature are: *visual inspection* (review), or based on some framework (*automation*): Logical Reasoning and Theorem Proving; Rule- and Pattern-Based approaches; Model Synchronizations (more details in Section 4.3.2).

For the last two activities, the **diagnosis of inconsistencies** (identifying/classifying of located inconsistencies) and Inconsistency handling (Ignore, tolerate, resolve), are identified the following characteristics: whether the approach considers the activity, if it is manual, and what service if considered when existing (e.g. for resolving the inconsistency, a modification of one/many models can be considered in a way).

“” provides a synthesis of the methods in light of the above characteristics. Methods considered in the MBSE/MBSA scope are the following (detailed in Section 4.4)[4]: Capella Safety viewpoint, MOISE method, S2ML method, Ontological integration approach (identified as KUBIK for short in the table), and DA approach (3DExp). Some other approaches from domains like Systems Engineering, Software or mechatronics are also included to highlight distinct approaches (details in Section 4.3).

[4] The SAVI approach is not considered as we have less details

Methods	**Models considerations** Approach	**Definition of correspondences between artefacts** [When], automation	**Detection of inconsistencies** Approaches and automation	**Detection of inconsistencies** Approaches and automation	**Diagnosis of inconsistencies** Considered / service (if relevant)	**Inconsistency handling** Considered / service (if relevant)
Thales/All4Tec Section 4.4.1	Unique model with viewpoints	[Conceiver: When building the language + User: When filling the model] Manual	Manual (review) + Automation (Well-formedness conformance checking)	-	Not considered	Direct modification of the model
MOISE Section 4.4.2	Multi-models Mappings	[User: When defining correspondences between models - CPs] Manual	Manual (review)+ Automation (Well-formedness conformance checking)	Structural	Not considered	Tolerate (justifications)
S2ML Section 4.4.3	Multi-models Integration via Languages	[Conceiver: When building the collaboration + User: When defining correspondences between models] Manual + Semi-automation (Comparison on names and structure)	Automation (Diff checking), Locating	Structural	?	Concretization phase (resolve) or justification (i/t/r)
KUBIK Section 4.4.4	Multi-models Integration via Languages + Mappings	[Conceiver: When building the language federation for the collaboration+ User: When defining correspondences between models] Manual	Automation (Rule- and Pattern-Based approaches) Locating	Structural + some behavioral	Rule reasoning Impact analysis service Identifying (labeled rule)	Not considered

DA approach (3DExp) Section 4.4.5	Multi-models Mappings	[User: When defining correspondences between models] Manual	Manual (review)	?	Not considered	Not considered
Herzig[5]	Multi-models Integration via Languages + Mappings	[Conceiver: When building the language projection+ User: When defining correspondences between models] Manual	Automation (Rule- and Pattern-Based approaches) Locating	Structural	Rule reasoning identify/classify	Not considered
Van Der Straeten[6]	None	[Conceiver: When building the language] -	Logical reasoning (specification) Locating	Structural + some behavioral	Logical reasoning (specification) identify	Rule-based inconsistency resolution
Giese et al.[7]	Multi-models Integration via Languages	[Conceiver: When building the languages integration]	Automation (synchronization-based approach)	Structural	model synchronization identify	Resolution (bidirectional transformation technique of triple graph grammars)
Rieke et al.[8]	Multi-models Integration via Languages	[Conceiver: When building the languages integration]	Automation (synchronization-based approach)	Structural + some behavioral	model synchronization identify	Resolution (incremental model synchronization)

[5] (Herzig, A Bayesian Learning Approach To Inconsistency Identification in Model-Based Systems Engineering, 2015)
[6] (Van Der Straeten & D'Hondt, Model Refactorings through Rule-Based Inconsistency Resolution, 2006), (Van Der Straeten, Mens, Simmonds, & Jonckers, 2003)
[7] (Giese & Wagner, 2009)
[8] (Rieke, Dorociak, Sudmann, Gausemeier, & Schäfer, 2012)

In conclusion, some points can be highlighted:

- The approaches are conceptually and technically (underlying framework) different;
- Consistency activities are not fully covered, efforts are specifically made on relating artefacts and detection inconsistencies (less on diagnostics and resolving, which can require some substantial efforts);
- Some approaches tend to provide automation to matching and inconsistency detection;
- Few considerations of behavioral consistency checking (hard problem in itself).

All the methods are missing focus on prevention of inconsistencies to appears, i.e. activities related to defining and establishing the collaboration (building a shared vocabulary, a shared vision of the collaboration, and an alignment of the process - more details in Section 3.1).

Approaches could be observed in terms of:

- Impact on existing practices (additional workload needed to implement the method, impact on existing processes);
- Efforts/ Benefits for certification purpose;
- Comprehensibility;
- Scalability and performance.

5 How Tools/Platforms answer consistency issue

In this section, we will present the way the different existing platforms support consistency issue. The purpose is here to list all functionalities of the platforms in relation with consistency needs: traceability aids, consistency rules and verification, review capabilities, …

The evaluation of the tools was based on three categories

- Generic services that position the context of use of each tool
- Services that support and facilitate collaboration
- Services that respond to the different consistency activities:
    - Linking engineering artifacts (Matching and Mapping)
    - Detection of inconsistencies
    - Diagnosis of inconsistencies
    - Handling of inconsistencies

The tools we investigated are:

- **3DX**: it's a commercial tool with a fully integrated and equipped collaboration context, linked to a Product Lifecycle Management (PLM) platform. In this context, the platform is widely used for the management, piloting and various engineering tasks. For our evaluation needs, we relied on the services offered by version 2016x

- **SECollab**: it’s a commercial tool presenting a collaborative context based on a platform for specific use (the collaborative review) and connected to dedicated tools for the use of certain services (management of actions or change requests for the review for example). The version evaluated here is the V3.1.1

- **Teepee**: a dedicated integration management platform, which is a research prototype, built to face heterogeneity of data, methods and tools to provide unambiguous communication in the so-called Extended Enterprise. The platform provides digital continuity along the development process and relies on Interoperability principles, publication and federated data controlled exposition.

The following table represents an overview of the consistency services offered by the 3 platforms 3DX, SECollab and Teepee. We have organized this table according to 3 main categories of services:

- Generic services that position the context and purpose of each tool
- Services that support and facilitate collaboration
- Services that respond to the different consistency activities listed in chapter 3

	3DX	SECollab	Teepee
Generics			
Plateform kind	PLM platform + regrouping several heterogeneous apps	Specific review plateform	Heterogeneity management (integration)
Licence	commercial	commercial	Free for S2C members for research use, commercial for other uses

Collaboration			
Data Integration kind	Import/export of PLM object + direct bridges to tools	Import/export of files+ OSLC tool connectors	Direct bridges to tools or connection from files Distributed REST API
Model rendering and view	Copy of the rendering and view of the original tool	Copy of the rendering and view of the original tool	Rendering as defined by viewpoint definition Context specific view (autolayout)
Technological stacks	PLM objects (oracle based)+ specific (proprietary) RDF support	neo4j	REST API (format csv)
Connectors list (limitated to SE/SA)	Native applications and Several connectors (listed in the figure below). These include: • DOORS • ReqIF • RFLP • System Architect • Rhapsody • Magic Draw • Cecilia • Office	• DOORS • MEGA • System Architect • Rhapsody • Capella • MagicDraw • Enterprise Architect • Office	Limited to defined viewpoints: • Capella • MagicDraw through TeamWorkCloud (block method) • MagicDraw through TeamWorkCloud (activity method) • Modelio through Constellation • Excel • SimfiaNeo
Possibility to develop new connectors (sdk)	Yes (and without software editor's intervention)	Yes (and without software editor's intervention)	Yes (and without software editor's intervention)
Consistency activities			
Linking engineering artifacts (Matching and Mapping)			
Alignment of business vocabularies	No native service available	No native service available	No native service available
• **Alignment of the languages used in the business lines: choice of common formalisms, and/or mapping between formalisms,** • **Matching of collaborative artefacts** • **Specification of inter-business matching rules**	• No native service available • No native service available • No native service available	• No native service available • No native service available • No native service available	• Dedicated viewpoints defined as a common vocabulary and provided in the REST API. Mapping between formalisms and common vocabulary is done in the connector. • No native service available • No native service available

Specification of the orchestration (or choreography) of the engineering processes	Project Management App (activities and workflows)	No native service available	No native service available
Definition of correspondences between artefacts (mapping)	Traceability service (links) App: SysTra	Traceability service (links)	Traceability service (alias)
Detection of inconsistencies			
Detection of contradictions between assertions (e.g. with definition and execution of consistency rules)	No native service available + Possible implementation (semantic browser) but still not easy to use by non-expert user	Yes but still not easy to use by non-expert user	No (hardcoded in the tool)
Detection of contradictions between assertions with expert inspection (consistency review)	Yes (Collaborative review with the use of SysTra and review path)	Yes (Native collaborative review)	Yes (Review of the result of integration)
Detection of deviations in the implementation and execution of process orchestration (control of compliance with procedures, etc.)	Validation and approbation routes (on activities, product lifecycles) – no deviation allowed	No native service available	No native service available
Diff functionality (two versions of the same model)	Yes (textual and graphical)	Yes (textual and graphical)	No native service available
Diagnosis of inconsistencies			
Identification of the source of the inconsistency	No native service available (but we can use the diff as a solution)	No native service available (but we can use the diff as a solution)	No native service available
Identification of the cause of the inconsistency	No service available	No service available	No native service available
Identification of the impact of the inconsistency	No native service available (but traceability link can help to identify the impact)	No native service available (but traceability link can help to identify the impact)	No native service available
Handling of inconsistencies			
Data synchronization between private space (specialized modeling tool) and public space (platform)	No (the edition is done in the tool of speciality and then published to the platform)	No (the edition is done in the tool of speciality and then published to the platform)	No (the edition is done in the tool of speciality and then published to the platform)
Versioning	Yes	Yes	Yes
Technical facts management	Yes	Yes, connection with JIRA	No native service available

Table 3: Comparison of tools

As a conclusion, it can be observed that:

- Methods and tools focus on dealing with the heterogeneity of viewpoints and data with different approaches,
- The tools are built for distinct purposes and are technically different (framework to assemble the model), and services available to the users,
- Native services to address all consistency activities are missing (assistance and automation), more specifically to help detect and diagnose inconsistencies:
  - Initials steps of managing the information is generally performed (Linking engineering artifacts, providing integration, views, …), but
  - Assistance and automation (tooling) approaches can bring benefits to inconsistency management activities like: matching, detection, diagnosis and handling.

6 MBSA methodologies and tools

6.1 Key definitions and Concepts

This section gives elements to understand the MBSA state of the art, fundamental concepts but also vocabulary details.

6.1.1 Definitions

Artifact:

Term from the Unified Modeling Language utilization for "the specification of a physical piece of information that is used or produced"…."by deployment and operation of a system." [9]

This term is used to characterize a type of node in AltaRica modeling language.

(Minimal) Cut set

A cut set (also called failure set in ARP4761) is a set of failures that brings the system to reach a failure condition.

A minimal cut set is minimal, in the sense that, if a failure is removed from the set, the failure condition is not reached.

In models, failures are modeled by events, a cut set is then a set of events.

Failure logic modeling (FLM):

This approach is based on the notion of failure modes and failure mode flows. This safety analysis method is also build using the development of formal and semi-formal languages to enable specification, composition, and analysis of system failure behaviour. Their 'topology' closely resembles that of typical design models. Overall, FLMs can be seen as a 'modular' variant of the fault trees. It's a not fully automated process, indeed Failure Logic Models (FLMs) are manually constructed by the safety engineers. The FLM approach focuses on faults rather than constructing a model of a system dynamics.

The term and definition of FLM are taken from (Lisagor, Sun, & Kelly, 2010) and (Papadopoulos, et al., 2011). Most MBSA methods of S2C members do not fulfill this definition of FLM, nor other approaches defined by Lisagor. That is the reason why, we propose a new definition that cover these MBSA methods:

Failure Propagation Model (FPM):

The safety process captures, using a modeling language, the architecture, the failure mode models, the safety-relevant functional behavior (e.g., reconfigurations, monitoring, etc.); failure condition formulae and possibly additional data depending on the analysis objective.

The resulting analytical model is called a "Failure Propagation Model" [From ARP4761 rev A DRAFT]. This model can be used to compute quantitative and/or qualitative safety indicators.

Failure propagation models are, in some way or another, computable. Several mathematical formalisms are used in dependability analysis to support the computations, mainly Markov chains, Petri nets or finite state machines, each with different variants. These types of underlying formalisms are important for the qualification of computation tools.

Functional and Dysfunctional modeling (FDM):

The purpose of this definition is to define the project approach and to provide a framework for project members' use MBSA. This MBSA approach is based on the notion of modes and flows in order to model system dysfunctional and functional* behavior. Modeled functional behaviors are a subset of functional behavior selected because of their impact on the safety analysis. These models 'topology' closely resembles that of typical design models modeling selected functional and dysfunctional behaviors. These models support a sound and complete computation of the causes of feared events.

(*it deals with nominal behavior abstracted).

[9] https://en.wikipedia.org/wiki/Artifact_(UML)

MBSA (Model Based Safety Analysis):

A technique that models system content and behavior in order to provide safety analysis results. MBSA employs an analytical model called a Failure Propagation Model (FPM) [From ARP4761 rev A DRAFT].

(System) Modelling:

A method to represent systems in a conceptual model that defines the structure and behaviour of a system. The main objective is to obtain a standardised representation of a current definition of the system as a base to support the work of the development process.

Modeling language:

"A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. A modeling language can be graphical or textual." [10]

Resource:

"An entity that is utilized or consumed during the execution of a process.

Note 1: Resources may include diverse entities such as personnel, facilities, capital equipment, tools, and utilities such as power, water, fuel and communication infrastructures.

Note2: Resources may be reusable, renewable or consumable." (definition from (ISO15288, 2015))

[10] https://en.wikipedia.org/wiki/Modeling_language

6.1.2 Basic concepts

Deterministic or non-determinist computation:

"A **deterministic algorithm** is an algorithm which, given a particular input, will always produce the same output"[11]. To compute results (cutset, sequence, probabilities) from a MBSA model, we can use either a deterministic algorithm or a non-deterministic algorithm, typically Monte-Carlo simulation (see definition below).

Deterministic or non-deterministic model:

A model is deterministic if, given a set of events (according to models, a set with a defined sequence or a set with timed instants for each event), the state of the model is determined. In other words, the outcome of a transition from one model state to another is only determined by the event name for all model runs. Figure 24: Example of deterministic model (left) and non-deterministic model (right) Figure 24 shows an example of non-determinism: from a given state (A), the same event (E1) may lead to different states (B or C).

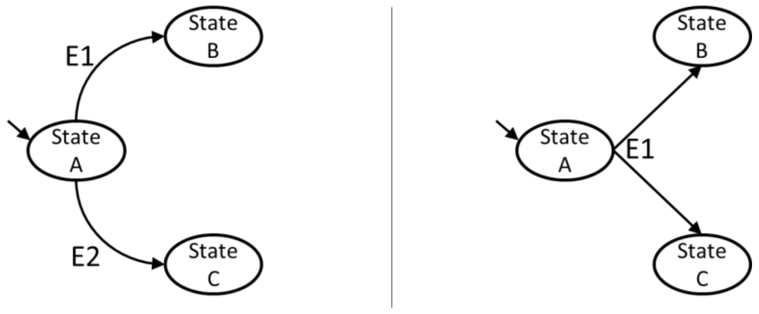

Figure 24: Example of deterministic model (left) and non-deterministic model (right)

Let note that the determinism of models is not strongly related to the determinism of computation. A deterministic model can be computed in a non-deterministic manner and reversely.

Mathematical formalisms:

A mathematical formalism is a form that enables to solve mathematically the problem defined by the model. In general, the model, expressed in modeling language, is transformed in a mathematical formalism before computation.

Failure propagation models are, in one way or another, computable. Several mathematical formalisms are used in dependability analysis to support the computations, mainly Boolean equations, Markov chains, Petri nets or finite state machines, each with different variants. The modeling engineer needs only the knowledge of the modeling language and its semantics. The knowledge of the detailed formalism is required to develop the computation engine. Even though one can use those different formalisms without knowing in details their mathematical justification, it is necessary to understand how to use them properly. In addition, the knowledge of underlying formalisms becomes compulsory in the framework of the qualification of computation tools.

In the following of the document, we are mainly interested in the type of results and the modeling aspects available to the safety engineer that are in some extent independent from the underlying computational framework.

To be completed: based on ONERA input, a description of main mathematical formalisms will be available in a future version.

Model run / trace:

A model run (also called "trace") is a sequence of states of the model (a state of the model is a combination of values of state variables). From one state to a second one, a transition occurs.

[11] https://en.wikipedia.org/wiki/Deterministic_algorithm

Monte Carlo:

"Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle." [12]

Static or dynamic MBSA modeling:

Let consider an MBSA model that is an asynchronous discrete-event model. By asynchronous, we mean that 2 events cannot occur at the same time.

Such a model is called **static** if the occurrence order of the events has no influence on the resulting state. For example, in Figure 25, we consider a model with two possible events E1 and E2. Both sequences (E1, E2) and (E2, E1) result in the same state (State D). The model is static if it verifies this property for all possible sequences in the model.

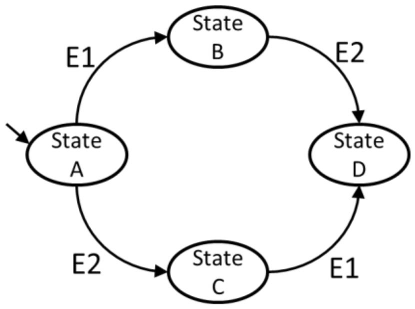

Figure 25: Example of static model

In the same context of MBSA asynchronous discrete-event model, a model is called **dynamic** if it exists at least one couple of sequences that are constituted with the same events and result in different states. For example, in Figure 26, the sequences (E1, E2) and (E2, E1) are constituted with the same events and result in different states: (E1, E2) results in State E and (E2, E1) results in State F.

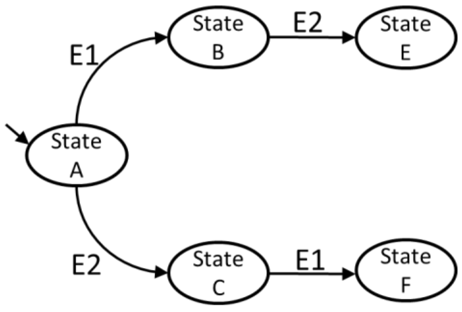

Figure 26: Example of dynamic model

This definition addresses overall model behavior. It does not allow to easily deciding whether a given model is static or dynamic (more precisely it would require to use model-checking techniques).

Consequently, a practical definition of static and dynamic is sometimes used: a model is static if it has a very simple state machine and a behavior mainly encoded in assertions. This definition is not well formed, there is no expert's consensus about it and is a working topic of S2C project. Nonetheless, it introduces the important distinction between the "behavioral staticity" and the "syntactical staticity". A model can contain complicated state machines (syntactically dynamic) and, once compute for a given feared event, exhibit only sequences that can be gather in cutsets (behaviorally static). The syntactic staticity is important for computation. The computation algorithm used for (syntactically) static models is the Boolean equations and is faster than any algorithm computing (syntactically) dynamic models.

[12] https://en.wikipedia.org/wiki/Monte_Carlo_method

To sum up, behavioral staticity is theoretical and well defined, whereas syntactical staticity has to be defined and has operational importance.

6.1.3 An introduction to AltaRica DataFlow

This section presents briefly and simply the basics of AltaRica DataFlow, with shortcuts and simplifications to ease understanding by beginners in AltaRica. They match industrial practices in a graphical editor, following the "Functional and dysfunctional modeling" as defined in Section 6.1. For a complete, precise and academic presentation of AltaRica, please refer to Section 6.4.2.1.

AltaRica DataFlow is a modeling language. The main element of the language is a "node". A node can model a component, a function or any intended artifact. Nodes have input/output ports that can be linked to each other. A node can contain sub-nodes, then it is called hierarchical, otherwise it is atomic. Figure 27 illustrates these concepts.

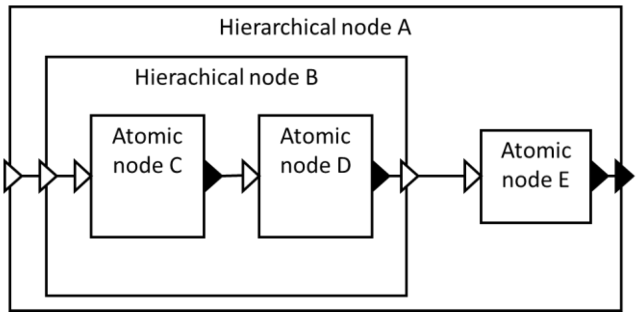

Figure 27: Composition of nodes

As illustrated in Figure 28, an atomic node contains:

- One state machine, defined by state variables and transitions. Each transition is associated to an event. When a transition is triggered, the state changes, i.e., the value of one or several state variables change.
- One assertion, modeling the dependency of output to inputs and state variables. The assertion aims to assign to each output ports a value according to the value of input ports and state variables. A common practice is to define one assignment per output port, resulting in the common abuse of language that one assertion is one assignment (instead of the set of assignments of the node). As a whole, assertions model the propagation of the failure modes (or more generally information) throughout the model.

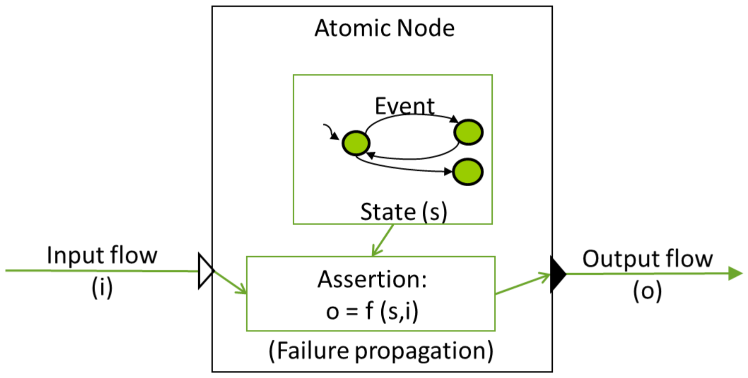

Figure 28: Atomic node in AltaRica

As a basic example, one can consider the atomic node shown by Figure 29. In the initial state defined by "stateVar = nominal", the event "failure" is enabled (i.e., it is triggerable). Let consider that the input value of the flow is "nominal". Then the output is nominal at initial state. When the event "failure" is triggered, the "stateVar" value changes to "failed". Following the assertion, the output is then set to "failed", propagating the failure out of the node.

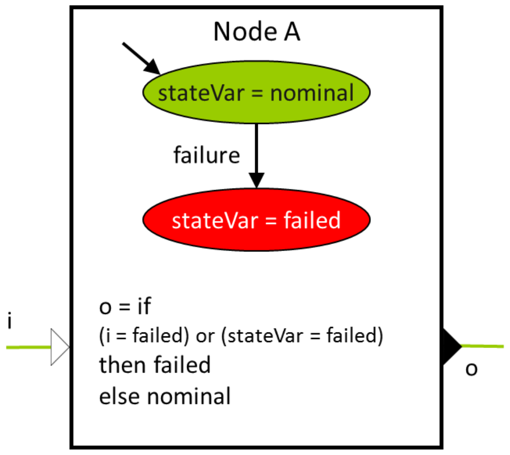

Figure 29: A basic AltaRica atomic node (description view)

Consider now that this first node “A” is linked to a node “B” that has the state machine and assertion shown in Figure 30. The output of “A” is propagated to the input of node “B”. Following the assertion of “B”, its output is set to “failed” and propagation continues in the downstream nodes.

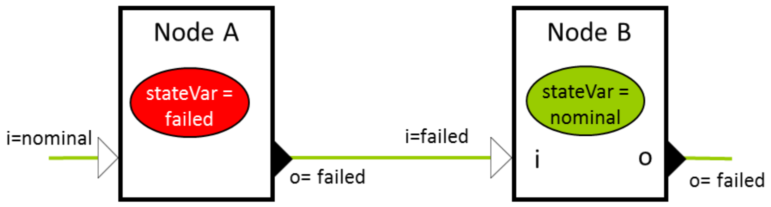

Figure 30: A basic AlataRica model (simulation view)
The input of node “A” is set as nominal for the sake of simplicity of the example. In a model, it should be linked to an upstream node.

The objective of MBSA is to generate cut sets or sequences to perform a safety analysis of a failure condition. The failure condition is defined in the model as an observer, as illustrated in Figure 31.

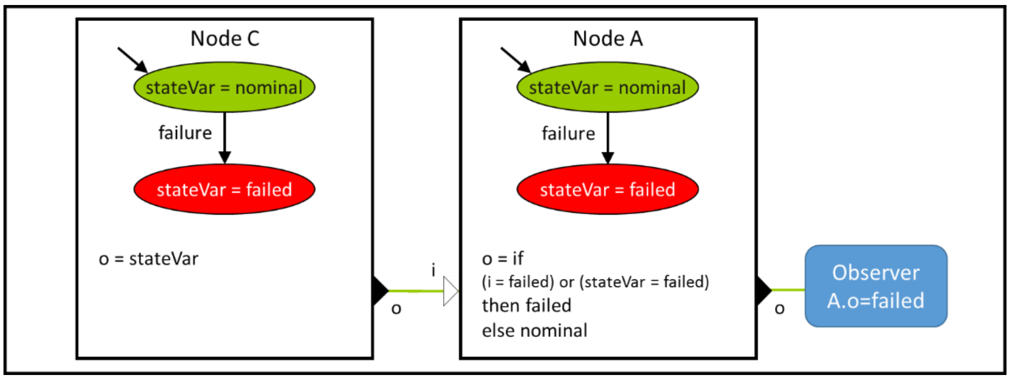

Figure 31: Example of a model with an observer

To understand how a model behaves as a whole, consider Figure 32. All computations are based on this reformulation of the model where:

- state variables values are combined to give overall model states;
- transitions are NOT combined with one another (asynchronous assumption);
- decomposition in nodes is flattened (more generally, all structures, as defined in Section 6.4.1.2, are flattened);
- assertions are expressed only in states variables. For example, note the observer is expressed in terms in state variables.

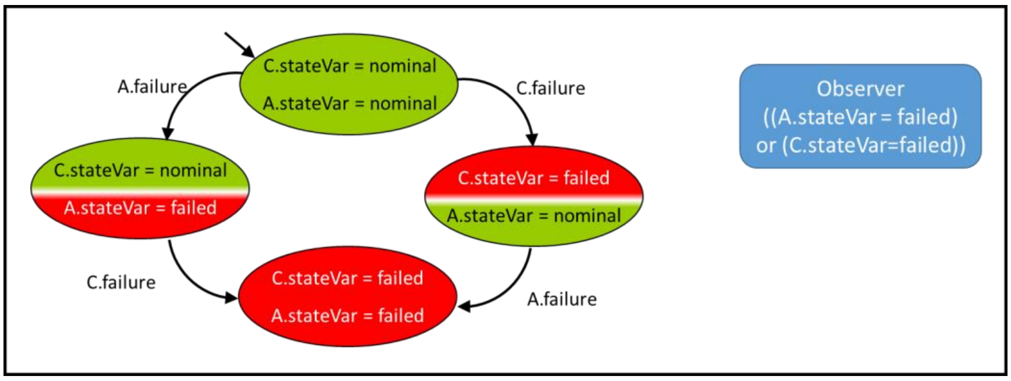

Figure 32: Reformulated version of the model defined by Figure 31

To compute cut sets from this reformulated model, we search for sets of events that lead, from the initial state to states where the observer is true (i.e., failure condition is reached).

The basic example follows the "Functional and dysfunctional modeling" approach, as shown by the explicit modeling of the nominal state. In this simple example, there is no need to model some reconfigurations but they can be taken into account for more complex systems.

Modeling in AltaRica involves three viewpoints:

- The description view (Figure 29), or edition view, that enables to build the model
- The simulation view (Figure 30) that enables to execute a scenario in the model, mainly in the objective to validate that the model represents the system of interest. A simulation step is defined by the occurring of an event, the resulting propagation through flows and assertion is regarded as instantaneous.
- The result view shows the cut sets or sequences generated from the model, enabling to assess the safety of the system of interest. Depending of the model complexity, the model is static or dynamic, and different results are computed: cut sets for static models, sequences for dynamic models.

Of course, models are usually more complex than this simple example. The following points are an illustration of this complexity:

- A variable can have more than 2 possible values ("nominal" and "failed")
- Different variables can have different sets of possible values
- An atomic node contains more state variables and more transitions
- The condition to trigger an event can take into account input variables the condition to trigger the transition "failure" is "stateVar = nominal", so no input variable is taken into account
- An event is associated to probability laws
- Two events defined in different nodes can be synchronized, i.e. they occur at the same time. This enables communication between nodes in another manner than flows between nodes. This is the only exception to the asynchronous assumption.
- The result view shows the cutsets or sequences generated from the model, enabling to assess the safety of the system of interest. Depending of the model complexity, the model is static or dynamic, and different results are computed: cutsets for static models, sequences for dynamic models.

In general, the expressiveness of AltaRica, notably of state machines, enables to model functional behavior (required for safety analysis). This capacity is essential to support the "Dysfunctional and functional modeling" defined in Section 6.1.1.

6.2 Introducing dependability using models

S2C project focuses on safety. This section intent is to provide a more global vision of modelling potential by focusing on dependability other aspects.

The propagation modeling approach characteristic gives the opportunity to explore different attributes of dependability.

This section provides a list of attributes covered by dependability, and selects the ones where it is relevant to use an MBSA type model. Then an overview of topics with associated tools is provided in order to provide some illustration of applications.

6.2.1 Relevant parts of dependability

We can find dependability in multiple communities. We choose, in this section to present dependability in the framework of LSA and its multi-domain standard [S3000L]. LSA is wider than dependability, as a consequence, we limit the scope to dependability and subjects that are related to dependability models.

Dependability is considered to be covered by following concepts: Reliability, Availability, Maintainability, Safety and Testability. This list is completed by attributes from the list of "considerations" influencing and justifying the design in [S3000L].

Selected considerations, with definition extracted from (S3000L, 2014) are listed hereafter:

- **Availability**
  "Availability is the measure of the degree to which an item is an operable and ready-for-use state at the start of a mission or operation, when the mission or operation is called for at an unknown time.
  Reliability, maintainability and supportability all contribute to the availability of the product."
  Several estimators of availability can be calculated such as inherent availability, i.e., foreseen availability regarding the system itself, achieved availability, i.e., the observed availability during operations and operational availability, i.e., foreseen availability regarding the system as operated within repair and logistic context (refer to (S3000L, 2014) for full definitions).
  The last one is the most difficult to be estimated as takes into consideration time to repair and reliability of course, but also preventive maintenance, repair team availability, stocks …
  The complex models have the most interest for operational availability. This attribute is detailed in §6.2.3.
- **Reliability**
  "Reliability is a prime driver of support resources. It is related to the duration or probability of failure-free performance of a product under stated conditions, or the probability that an item can perform its intended function for a specific interval under stated conditions. ".
  Modeling of availability is close from reliability ones. Hence, reliability is analyzed as a part of availability domain. Most of time, the reliability is expressed by MTBF (Mean time between failure).
- **Maintainability**
  "Maintainability is the measure of the ability of an item to be retained in or restored to a specified condition, when maintenance is performed by personnel having specified skill levels, using prescribed procedures and resources, at each prescribed level of maintenance and repair."
  The main director of maintainability is the Mean Time To Repair (MTTR) which is the main indicator. Models, as considered in this document, are not considered to be used often for this limited purpose. Most of the time, maintainability, during the design phase, is performed using Maintenance Task Analysis as defined in [S3000L]. MTTR is estimated by analysis of mechanical design models (such as CATIA models), plans or maintenance procedure. Formally, there are models (like Markov) but for the moment, their use in industrial context is very infrequent.
- **Supportability**
  "Supportability is the measure of the degree to which all resources required to operate and maintain the Product are provided in sufficient quantity. Supportability encompasses all elements of ILS support resources such as technical information, support equipment, spare or personnel".
  This attribute is considered, in this state-of-the-art document, as a contributor to operational availability in order to have a model close to reality constraints. Hence this topic is covered by availability subject.
- **Testability**
  "Testability is a design characteristic which allows the status (operable, inoperable, or degraded) of an item and the location of any faults/failures within the item to be confidently determined in a timely fashion".
  This attribute leads to several tests and application of MBSA type tools in order to assess the level of testability of an equipment, system or system of system. At present, no prevalent industrial solution is simple to use.
  You will find more details in §6.2.4 with tool example and publications.
- **Cost effectiveness**
  "The design and the choices involved in design have a cost implication for the LCC (Life Cycle Costs) , which can be broken into design/development, production/procurement, operational and disposal costs. Comparative analysis between different alternatives and costs are necessary to balance availability and LCC."

This attribute is linked with availability models, indeed to judge of the cost effectiveness it is necessary to check balances between system effectiveness mainly availability) and LCC. This attribute will be addressed during availability analysis.

- **Safety** from (ISO15026, 1998)

  "The expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered."

  Note this section will not deal with safety as it is addressed in the ones following.

Based on this preview of the attributes of dependability, Availability and Testability will be addressed in §6.2.3 and §6.2.4.

6.2.2 Model usages of S2C members

The contributors of S2C are using advanced models in several dependability activities. The table below provides an overview of the type of applications per domain.

Table 4 : Dependability using models

	Domain	Types of applications
Dassault-Aviation	Aeronautics Civil, business and military aviation,	Operational availability
Safran Helicopter engines	Aeronautics, missile and railway	To be completed
Liebherr-Aerospace Toulouse	Aeronautics	To be completed
Thales SA	Aeronautics, space, transport, digital identity and security, defense and security	To be completed
APSYS	Aeronautics, space, naval, automotive, air traffic management, energy, nuclear and industry	• Safety • Reliability • Availability • Supportability/Costs effectiveness with Simfia, SimfiaNeo and Simlog tools
LGM	Aeronautics, railway, naval, defense, energy, telecommunications, air traffic	• Operational availability (GRIF, Extend) • Testability (eXpress) • Mission reliability (GRIF, tools from OAR platform)
ONERA	Aerospace and defense	• Reliability • Operational availability • Testability
DGA	Aeronautics, aerospace, naval, terrestrial, drones, missiles, air traffic control, testing facilities, systems of systems for civil and military applications	• Safety (80%) • "Operational effectiveness" (10%) • Accident investigation (10%)

The “types of applications”, in Table 4, is not exhaustive. When the applications, associated methods and tools will be provided, the next sections will be completed with member’s information.

6.2.3 Modelling for operational availability analysis

Usually, the type of availability modeling is based on Reliability block Diagram, Markov chain and Petri nets use.

The concept of availability implies three other concepts:

- Reliability which deals with the frequency of failures,
- Maintainability which characterizes maintenance times,
- Logistics which deals with all the material, personnel and spare resources, documentation and maintenance policy implemented.

The following graph (extract from Standard (AFNORNFX60-500) translated) highlights the intrinsic availability and the operational availability:

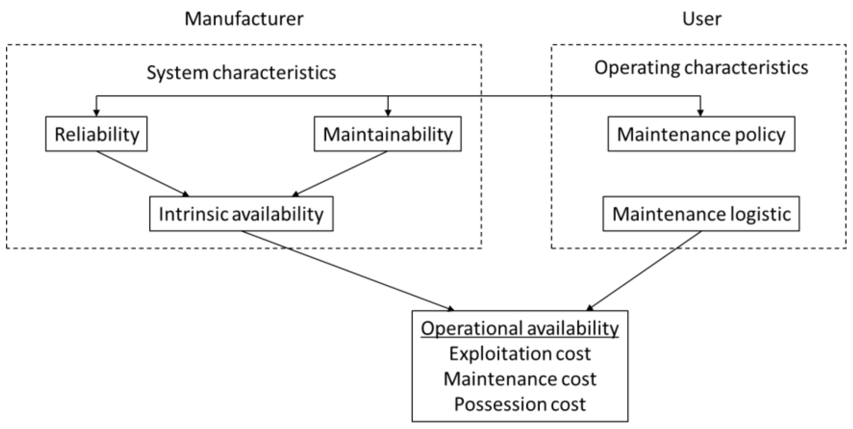

Figure 33 : Intrinsic availability and the operational availability

The (AFNORNFX60-503) standard gives as definition:

- Intrinsic: qualifies a value determined under the supposedly ideal maintenance and operating conditions.
- Operational: qualifies a value determined under the given maintenance and operating conditions.

An example of the modeling approach is detailed in (Maxime Monnin, 2007) by presenting methodological principles of a system modeling integrating failures, damages and regeneration (recovery) for the evaluation of the availability in operational mission. Modeling is supported by the formalism of Stochastic Activity Networks.

Petri nets are a typical method to perform operational availability analysis. AltaRica is also used to perform availability analysis in (Meng Huixing, 2016) using stochastic tool from (Open-Altarica, s.d.). This study shows the application of the AltaRica 3.0 modeling language for assessing the production availability of a Floating Production Storage and Offloading system (FPSO) in order to identify the sensitive parameters and crucial components with respect to the production availability.

Software originally from production simulation using flow could be used for more complex simulation (for example (Extendsim).

The table below presents a summary of methods and tools used for availability application. The objective of this part is to provide an overview of the project member’s methods and tools.

Table 5 : Availability models

Method	Examples of tools (not exhaustive, examples only)	Comments
Analytical Formula	Pencil and Paper Excel	Simple to use Limited to theorical case Analytical computation
Reliability Block Diagram	Excel Blocksim from ReliaSoft BFiab from GRIF RBD from RAM Commander	Simple to use Tools ensure computation Analytical computation
Reliability Block Diagram with simulation (Monte Carlo)	Blocksim from ReliaSoft RBD from BQR RFiab from GRIF	Simple presentation as classic reliability block diagram but taking into account more parameters as repair team, spare management,, etc…. Allows to take complex stock management, and repair teams availability into consideration. Stochastic computation
Markov	Markov from RAM Commander Markov from GRIF Markov from BQR	Combinatorial explosion Stochastic computation, Analytical calculation possible for simple cases
Stochastic Petri nets	Petri from GRIF CPN Tools from Aarhus University	Good adaptability and flexibility in use Stochastic computation Interactive HMI associated with a stepper

6.2.4 Modelling for testability analysis

Testability usual requirements are listed from (Dominique Riera, 2018) and (Deschamps, 2014):

- Coverage rate (% of failure covered),
- Location rate (% of ambiguity).

The coverage rate represents the ratio between the number of failures detectable by a given set of tests and the total number of potential system failures. It's also possible to define the coverage rate by considering the failure rate of components. The location rate represents the capacity of the test to identify the part or the sub-assembly failed down.

A basic method to perform such an analysis is testability FMECA. Specific methods to consider test propagation are also used.
In order to determine the coverage rate, the tests are graphically modeled by selecting the monitored flow (available with (EXPRESS) for example).
A dependency model is so created by adding in all the functions of the components and then building the functional relationships between the functions of the design to represent how function flows through the design. Failure modes are then added to the components to define how the components and their associated functions will fail. The dependency model once created then can allow for accurately defining the tests that will be realized.
This activity can be performed with dedicated tool (Deschamps, 2014) or using more generic software like (EXPRESS).
Some experiments have been performed with AltaRica. Testability performances (coverage and location rates) can be computed using tools from (Open-Altarica, s.d.) according to (Sylvain Breton, 2018). This ability is used to perform analysis on diagnosis with AltaRica in (Dominique Riera, 2018).

Different tests have been performed using AltaRica with other tools to address testability but no published literature has been found.
Other approaches using Bayesian networks are sometimes used for testability (see (Lefebvre, 2014)). They allow to adapt the model results using the in-service data in order to improve the ability to identify the right part to be "removed and replaced".

Bayesian networks are a commonly used model to describe causal relationships between variables to calculate conditional probabilities.. A Bayesian networks is generally described like an acyclic graph whose nodes represent random variables and whose arcs represent probabilistic relationships. They can thus be used to model the causal relationships between variables, events or phenomena physical to make a diagnosis.

The table below presents a synthesis of methods and tools used for testability.

Table 6 : Testability models

Method	Tools (not exhaustive, examples only)	Comments
Testability FMECA	Excel FMECA tool from generic dependability tool (for example RAM Commander)	Simple implementation Allow to estimate test coverage rate, location rate. Risk: not exhaustive approach.
Flow tool	eXpress from DSI international [EXPRESS]	Allow to estimate main testability performance indicators Optimization of tests in regards of duration, cost, …
AltaRica	Cecilia Ocas	Several experiments performed to use AltaRica language in order to perform some testability analysis.

6.3 Using MBSA propagation models in industry

This section focuses on the S2C members feedbacks about MBSA failure propagation models (as defined in §6.1.1) S2C members come from different domain and have very different concerns. They use MBSA with different levels of maturity and expectations. Our intent is to provide an overview of their different viewpoints and lessons learnt.

Even though all dependability attributes may be alluded this section focuses more specifically on safety.

6.3.1.1 History

This section presents a quick history of the failure propagation MBSA in aeronautical developments from the S2C member's point of view. It shows that this type of safety analyses have been developed since the nineties and have driven the interest of a very large range of academics and industrials.

Three timelines are provided hereunder.

The first is focused on AltaRica languages and associated tools (**Figure 34 : AltaRica propagation - Selected dates about languages and tools)**, the second presents an overview of R&D projects (Figure 35) and the last one is dedicated to the operational use and certification aspects of MBSA (Figure 36Figure 36).

These timelines are not exhaustive and could be completed with additional information from S2C project members.

As a complement of

Figure 34 : AltaRica propagation - Selected dates about languages and tools the list below provides a synthesis of the tools presented in the timeline with details:

- FIABEX: Software workshop for modeling and analyzing the reliability of industrial systems. It has been developed by DATA CEP, for the industrial club FIABEX which brought together the largest French companies.

- Cecilia OCAS: Dassault tool based on AlatRica modeling. The last version of this tool is now called Cecilia only (without OCAS).
- RAMSES: Airbus tool for the analysis of safety models of systems.
- Sofia: First MBSA APSYS tool based on Unix OS.
- SimFia: MBSA APSYS tool based on Windows OS (Sophia evolution).
- SimFia (V2): Evolution of SimFia with integration AltaRica Dataflow.
- SImFiaNeo : Evolution on SimFia V2 with user interface and post-processing enhanced

In addition, the Figure 34 also shows that the development of languages around AltaRica has been driven by the academic sector and more precisely by the following laboratories:

- the LaBRI (Laboratoire Bordelais de Recherche en Informatique) for the first version of the AltaRica langage.
- the IML (Institut de Mathématiques de Luminy) for the AltaRica DataFlow.
- the LIX (Laboratoire d'informatique de l'École polytechnique) for the AltaRica 3.0 version.

To support the understanding of the **Figure 35 : AltaRica propagation in industry – Overview of R&D projects (limited to S2C members)t**he legend “R&D about MBSA” can be precised as “Research and development dealing with the MBSA itself”. For instance it can be an assessment of MBSA by comparison with FTA model.

Moreover, in order to precise the information provided in this figure several precisions maybe brought. First it can be added that the DGA R&D methods discussed from 2010 to today are not only for aircrafts, but in various domains such as missile, satellite, air traffic management, system of systems and also on multiple types of analyzes such as vulnerability, security, human factors.

In addition, for Airbus defense and space, the period 2005-2012 served to apprehend classic MBSA method on R&D contexts by trying to extend to modeling and analysis of the dynamics of fault tolerance mechanisms (FDIR, Fault Detection, Isolation and Recovery). This started by mixing AADL and UML with respectively OSATE and Rhapsody tools. Then this continued on AADL (with OSATE) associated with timed automates for the dynamic properties' demonstration by model-checking (Kermeta use for model transformation). There has also been work on an association between AltaRica (with Cecilia OCAS) and Luster (Scade), in a study that focused on hybrid modeling (Linear Hybrid Automata, Hytech).

To complete **Figure 36 : AltaRica propagation in industry – Overview of operational developments and certification** projects**,** it is important to outline that Dassault aviation is the only industrial to have successfully achieved aeronautical certification of a system using the MBSA in 2007. This was done for the Falcon7X fuel system and digital flight control system. Thanks to this experience and according to exchanges already carried out in 2002 with the European and American authorities, Dassault continues to be a major actor for MBSA by participating in the redaction of ARP4761 A which includes the MBSA approach.

Eventually, regarding the operational projects of Airbus defense and space, it can precise that since the pilot project, there have been other operational activities according to punctual projects needs. Generally the tool used have been Simfia.

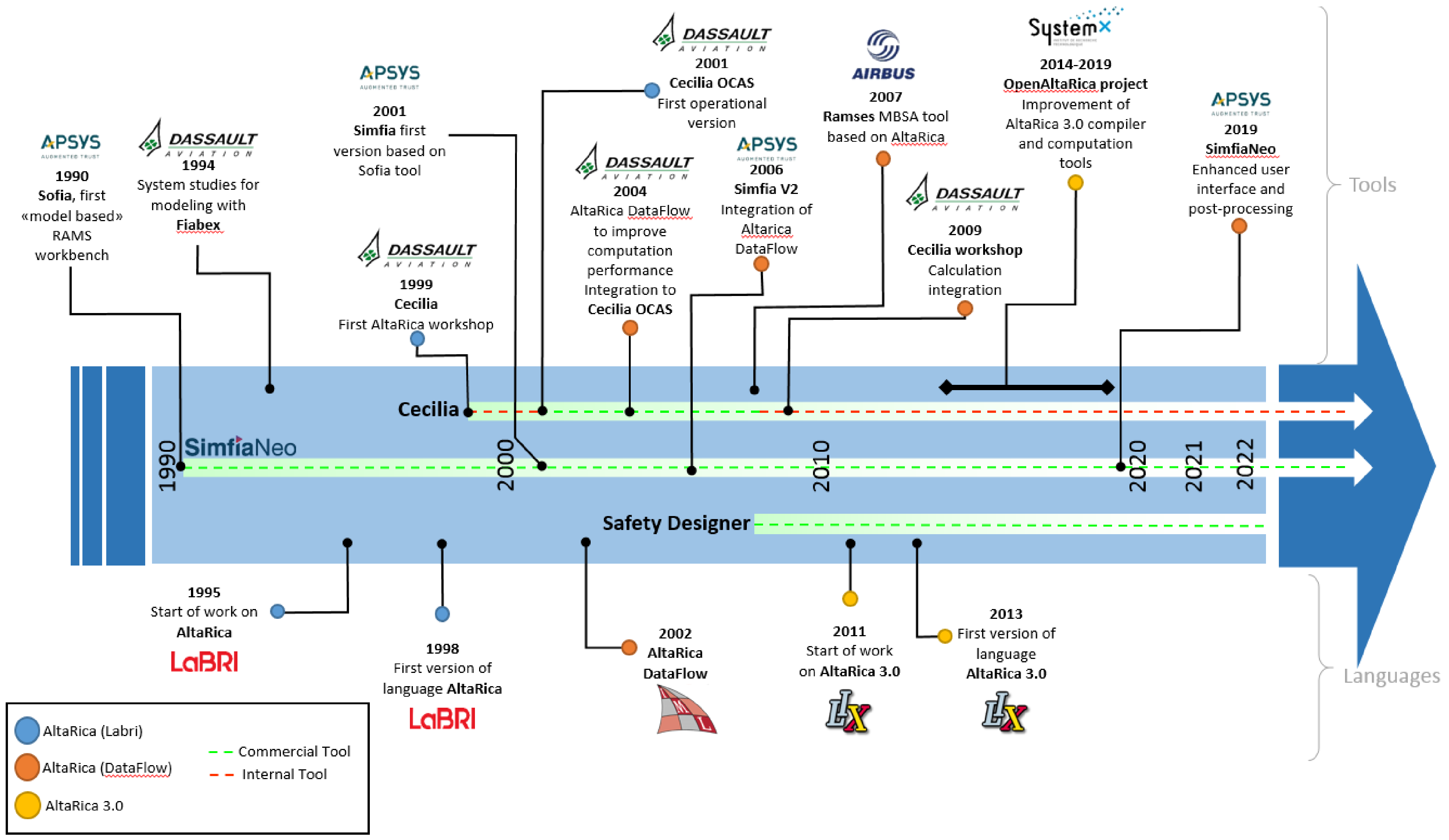
1990
Sofia, first «model based» RAMS workbench
1994
System studies for modeling with Fiabex
2001
Simfia first version based on Sofia tool
1999
Cecilia
First AltaRica workshop
2001
Cecilia OCAS
First operational version
2004
AltaRica DataFlow to improve computation performance
Integration to Cecilia OCAS
2006
Simfia V2
Integration of Altarica DataFlow
2007
Ramses MBSA tool based on AltaRica
2009
Cecilia workshop
Calculation integration
2014-2019
OpenAltaRica project
Improvement of AltaRica 3.0 compiler and computation tools
2019
SimfiaNeo
Enhanced user interface and post-processing
Tools
Cecilia
SimfiaNeo
Safety Designer
1990
2000
2010
2020
2021
2022
1995
Start of work on AltaRica
LaBRI
1998
First version of language AltaRica
LaBRI
2002
AltaRica DataFlow
2011
Start of work on AltaRica 3.0
2013
First version of language AltaRica 3.0
Languages
● AltaRica (Labri)
● AltaRica (DataFlow)
● AltaRica 3.0
Commercial Tool
Internal Tool

Figure 34 : AltaRica propagation - Selected dates about languages and tools

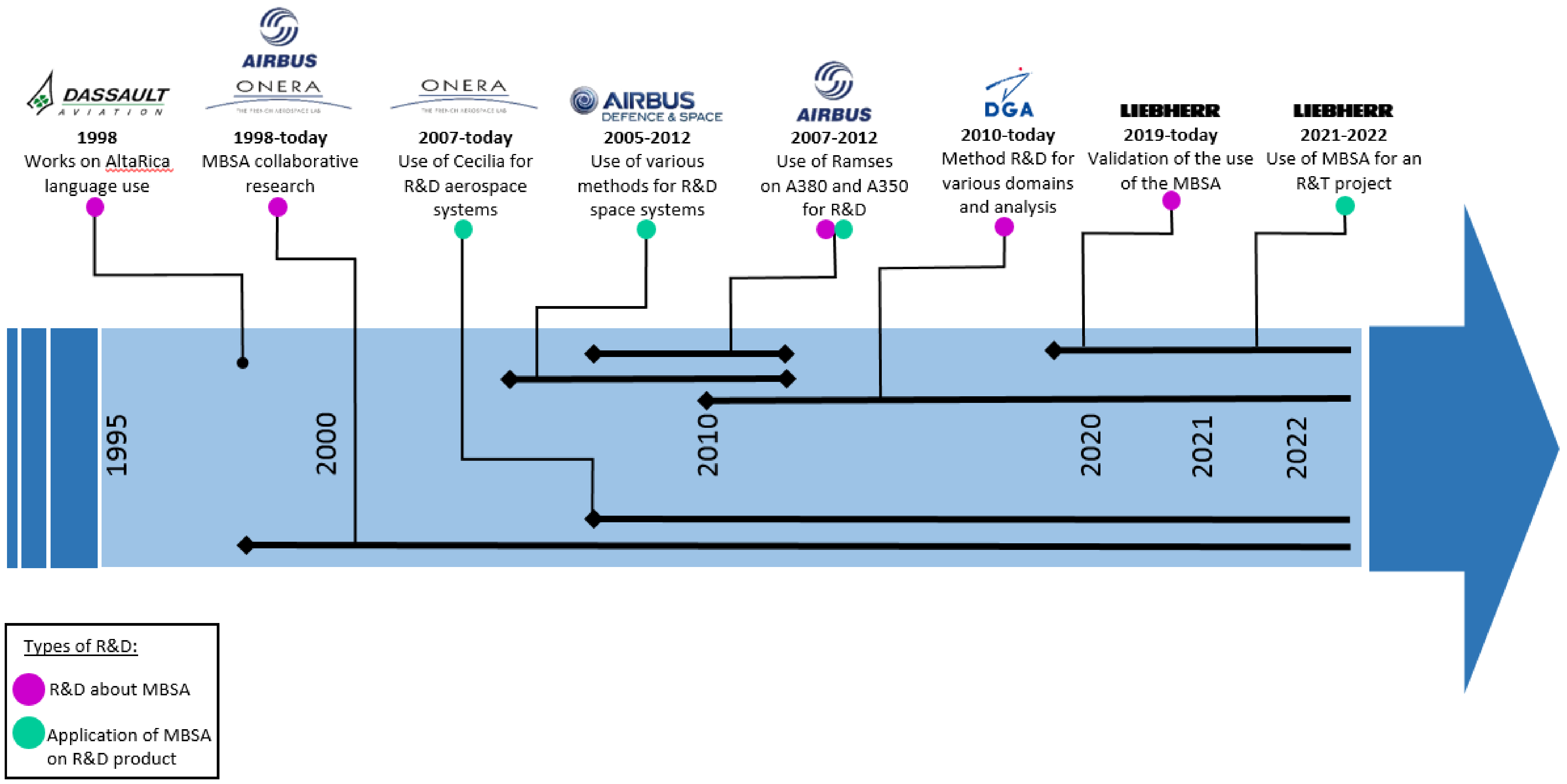
DASSAULT AVIATION
1998
Works on AltaRica language use
AIRBUS ONERA
1998-today
MBSA collaborative research
ONERA
2007-today
Use of Cecilia for R&D aerospace systems
AIRBUS DEFENCE & SPACE
2005-2012
Use of various methods for R&D space systems
AIRBUS
2007-2012
Use of Ramses on A380 and A350 for R&D
DGA
2010-today
Method R&D for various domains and analysis
LIEBHERR
2019-today
Validation of the use of the MBSA
LIEBHERR
2021-2022
Use of MBSA for an R&T project
1995
2000
2010
2020
2021
2022
Types of R&D:
R&D about MBSA
Application of MBSA on R&D product

Figure 35 : AltaRica propagation in industry – Overview of R&D projects (limited to S2C members)

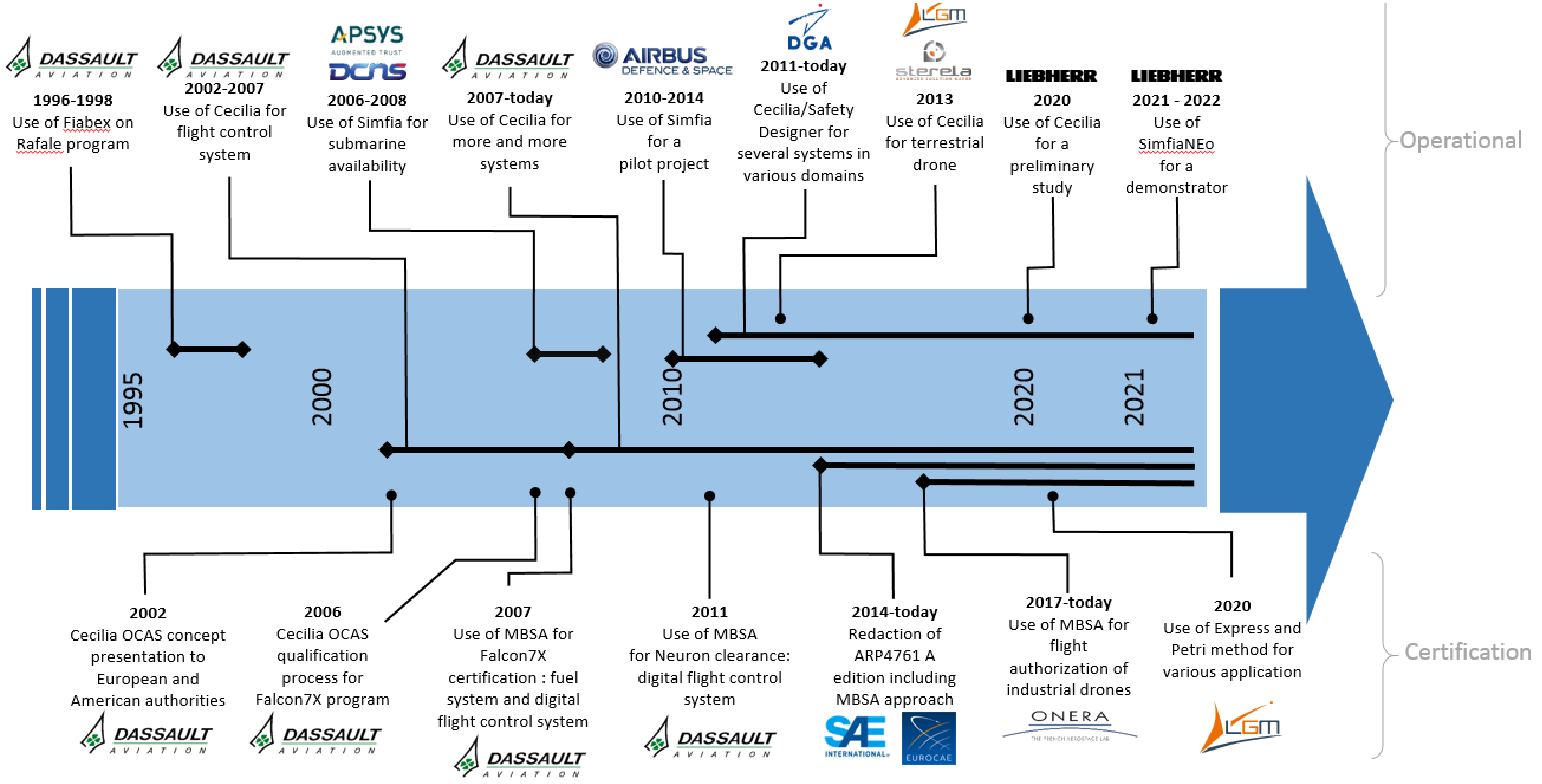
1996-1998
Use of Fiabex on Rafale program
2002-2007
Use of Cecilia for flight control system
2006-2008
Use of Simfia for submarine availability
2007-today
Use of Cecilia for more and more systems
2010-2014
Use of Simfia for a pilot project
2011-today
Use of Cecilia/Safety Designer for several systems in various domains
2013
Use of Cecilia for terrestrial drone
2020
Use of Cecilia for a preliminary study
2021 - 2022
Use of SimfiaNEo for a demonstrator
Operational
1995
2000
2010
2020
2021
2002
Cecilia OCAS concept presentation to European and American authorities
2006
Cecilia OCAS qualification process for Falcon7X program
2007
Use of MBSA for Falcon7X certification : fuel system and digital flight control system
2011
Use of MBSA for Neuron clearance: digital flight control system
2014-today
Redaction of ARP4761 A edition including MBSA approach
2017-today
Use of MBSA for flight authorization of industrial drones
2020
Use of Express and Petri method for various application
Certification

Figure 36 : AltaRica propagation in industry – Overview of operational developments and certification projects

6.3.1.2 Members, domain and activities

The table below presents the S2C members who have shared their experience and feedbacks about MBSA propagation models. In particular, it gives an overview of the different domain and activities they are involved in. One can note that all the participants of our survey are actors of the aeronautic industry even though they may have a larger field of interest.

Note that additional contributions are expected and will be added in further versions.

Table 7 : Members domain and activities

	Description	Domain	Activity of involved members
Dassault-Aviation	Aircraft manufacturer	Aeronautics Civil, business and military aviation,	Development, activities for certification and manufacturing
SAFRAN HELICOPTERE ENGINES	Engines manufacturer for helicopters and planes.	Aeronautics, missile and railway	Development, activities for certification and manufacturing
LIEBHERR Aerospace – Toulouse	Air System manufacturer	Aeronautics	Development, activities for certification and manufacturing
THALES SA	Electronic systems manufacturers	Aeronautics, space, transport, digital identity and security, defense and security	Development, activities for certification and manufacturing
APSYS	Consulting MBSA tool development	Aeronautics, space, defense, naval, automotive, air traffic management, energy, nuclear and industry	Development and certification activities for customers MBSA tool development Training in methods and tool
LGM	Consulting	Aeronautics, space, railway, naval, defense, energy and nuclear telecommunications air traffic, infrastructure , building, automotive	Development and certification activities for customers and training Safety and availability studies for military applications Testability for military products. Availability studies for stadium opening roof
ONERA	Studies and Research institute	Aeronautics, space and defense	Research, consultancy service
DGA	Research and expertise. French Defense Procurement Agency	Aeronautics, space, naval, terrestrial, drones, missiles, air traffic control, testing facilities, systems of systems for civil & military applications	Certification/Qualification, Safety analyses Test center for the qualification of military and civil aeronautical platforms and their components

6.3.1.3 The use of MBSA by our members

This section provides a synthesis of the use of propagation models by S2C members. It highlights the fact that MBSA is used mostly in R&T, but also for operational developments and to support aeronautical certification (Dassault Aviation). It is used in different domains such as aeronautics, rail transport or naval industry. In addition, systems analyzed can have very different specificities or characteristics: they can be mostly electrical or involving physical flows as well as mechanical constraints. To provide generic MBSA solutions these different characteristics need to be taken into account.

Table 8 : Members MBSA usage

	Description of MBSA use	Context of MBSA use	Domain of MBSA use	Some information about the systems analyzed using MBSA
Dassault Aviation	Safety analyses for certification of several systems This approach is used for all new civilian and military programs. * In certification: Flight controls * In Development: Flight Controls, Multi-System Approach	Operational and certification in aeronautics	Civil, military and business aviation	Flight control system : Size: around 2000 constituents Types of information: digital, physical and resources
Safran Helicopter Engines	Evaluation and evaluation (two dedicated PhD): * 2005-2008: Declination of system requirements towards software (Sophie HUMBERT). * 2008-2011: Multi-physical modeling and validation of models (Romain ADELINE)	R&T evaluation	Aeronautics	Types of information: digital, mechanical, physical and resources *To be completed*
LIEBHERR Aerospace - Toulouse	Evaluation of MBSA interest for safety analyses and certification support	R&T evaluation Feasibility demonstrator	Aeronautics	Types of information: digital, mechanical, physical (air) and resources
THALES SA	Part of the development but not used for certification	Operational (outside of aeronautics)	Aeronautics, defense and space	*To be completed*
APSYS	Support of customer activities Studies for customers Rise in terms of skill: MBSA tool development and released	Operational (outside of aeronautics) R&T	Automotive, aeronautics, naval, industry, space	Mainly logical systems for aeronautics, logical and physical systems for industry

	Description of MBSA use	Context of MBSA use	Domain of MBSA use	Some information about the systems analyzed using MBSA
LGM	Support of customer activities	Operational (30%) but not for certification / Intern (50%) / R&T extern (20%) Internally for R&T and benchmarking	Telecommunication, naval, Land transport, defense, Air traffic, energy, Railway, Aeronautics, Defense, automotive	Aeronautical system Robot (operational, functional, physical up to electrical components), Military applications, Construction sector.
ONERA	Internal research and support of customer Validation &Verification activities and trade-offs	Operational but not for certification R&T	Aeronautics, space and defense	CdV elec, Hydrau, Elec, Motor control Drone and ground station Network of external functions providing the services Networks of technical functions performing external functions "Equipment": networks of physical components performing the technical functions
DGA	Support of safety analyses On request of projects, mainly on critical systems (in the security sense) Support for specification Support for qualification and certification activities of new systems produced by major manufacturers Support for modifications and renovations. Support for the BEA analyses (Bureau d'enquêtes et d'analyses pour la sécurité de l'aviation civile).	Operational and R&T (exploration of new areas of application)	All civil and military fields (aeronautics, land, naval, space, air traffic control, missile, test facility, drone, etc.)	Types of information: high level information; zonal information, digital, and resources

6.3.1.4 Different kind of systems, analyses, and developments phases

This section focuses on the analyses performed using the MBSA propagation models. Note that in the table below we did not limit the analyses to safety analyses in order to give a larger overview. Furthermore, it is also interesting to outline that in order to perform quantitative and qualitative analyses members mainly used the MBSA models we defined as functional and dysfunctional models in Section 6.1.1.

As industrial members need quantitative results to complete their safety analyses, they use the MBSA to compute probabilities while the DGA or ONERA privileged the qualitative results and the expressiveness of the models. In all cases cut sets and sequences need post processing (manually or automatically) in order to be analyzed.

Whether models are dynamic or static, quantitative results are considered satisfactory. Nevertheless some members such Dassault Aviation tend to privilege static models and during Safran Helicopter Engines evaluation the quantitative computation or allocation have been discarded when using dynamics models due to their excess of conservatism.

Table 9 : MBSA for analysis

	The analyses performed using MBSA models	System level *	Development phases
Dassault Aviation	Operational quantitative and qualitative safety analysis, maintenance	System and multi-systems	V&V activities and trade-offs
Safran Helicopter Engines	Declination of system requirements towards software and safety Safety analysis No quantitative analysis or allocation Evaluation of dispatch analysis	System	Preliminary design activities
LIEBHERR Aerospace - Toulouse	Evaluation of quantitative and qualitative analysis	System	Preliminary design activities R&T trade-offs
THALES SA	PSSA support	System	Preliminary design activities
APSYS	Safety analysis quantitative and qualitative (including FMEA) Availability	System	V&V activities and trade-offs
LGM	Safety , testability, availability, Simulation	System and system of systems	V&V activities and trade-offs
ONERA	Safety analysis quantitative and qualitative, with a majority of quantitative analyses Reliability, availability, testability	System, multi-systems and system of systems	V&V activities and trade-offs
DGA	Mainly qualitative safety analysis : DAL allocation Analysis of Common Causes (common modes, particular risks, zonal analyzes) Principle of FAIL SAFE	Systems, Integrated systems, System of systems	Specification, certification/qualification, modification

6.3.1.5 MBSA models exploited by our members

As a generality the MBSA output exploited by members are the cut sets or sequences according to whether they chose static or dynamic modeling (see Section 0 for details and definitions).

The table below details the MBSA tools used to support members the safety analyses, the formalism they use as well as their computation type.

Table 10 : MBSA tools usage

	Static or dynamic modelling	Language and tool	Formalism and type of computation
Dassault Aviation	Static	AltaRica Cecilia	State machines and boolean equations Computation by simulation and analytical (BDD)
Safran Helicopter Engines	Dynamic	AltaRica Cecilia	State machines and boolean equations Computation by simulation and analytical (BDD)
LIEBHERR Aerospace - Toulouse	Dynamic	AltaRica Cecilia AltaRica SimfiaNeo	State machines and boolean equations Computation by simulation and analytical (BDD)
THALES SA	Static	Safety Architect	Boolean equations (TBC)
APSYS	Static and dynamic	AltaRica SimfiaNeo	State machines and boolean equations Computation by simulation
LGM	Static and Dynamic	AltaRica SimfiaNeo eXpress Petri	State machines and boolean equations. Safety assessment Computation by simulation
ONERA	Mostly Dynamic	AltaRica Cecilia	State machines and boolean equations Computation by simulation and analytical (BDD)
DGA	Static and Dynamic	AltaRica Cecilia	State machines and boolean equations Computation by simulation and analytical (BDD)

6.3.1.6 Return of Experience

The table below provides an overview of S2C member's feedback about the strengths and weaknesses of the MBSA.

Globally MBSA has been experimented as a way to communicate efficiently, to improve complexity handling and favor collaborative work. MBSA models are considered as easier to maintain than fault trees and an efficient way to improve the capitalization of information.

On the other hand, the MBSA requires investment in training and in deployment of a customized methodology. More specifically some modelling difficulties can be encountered and guidance and skilled are required to solve those technical blocking points (in particular to handle the "looped systems"). Eventually some members outlined their wish to improve the post-processing of the models output and to include them in current documentation.

Table 11 : Members MBSA experiences

	Assests of the MBSA	Weaknessess
Dassault Aviation	Produced studies are consistent and maintainable Update easier than with other methods Appropriate approach for complex or highly integrated systems	Difficulties to model looped systems.
DGA	Easier to master the complexity of systems Collaborative work and capitalization Great communication tool	Fairly high entry ticket (adaptation of working methods, training in the tool and building its library of components) Language limitations (looped system and dynamic system)
ONERA	Explicit capture of knowledge useful for analysis dysfunctional (failures) and functional hypotheses (monitoring logic, isolation, etc.) Static (topologies) and dynamic dependencies (reconfiguration, cascade of events, etc.) Component-based modeling for complex cases Progressiveness of the construction and debug of the model Reuse of standardizable components Overall behavior emerging from the composition Automated calculations of event propagations in a dependency network Simulation of failure cases for validation of the model Calculation of combinations of events causing failure cases Representation and graphical simulation for information sharing View of the propagation paths of failures in the network More precise and more homogeneous analysis results	Methodology to acquire to be operational quickly Debug of the emerging behaviors of the composition (ex : oscillation = model error or system error?) Time-consuming graphic modelling for large models : currently there is no tool to assist the placement of boxes and wires to avoid unreadable spaghetti dishes No simultaneous update of several library components Post-processing limitations
Safran Helicopter Engines	Consistent view for system architectures and safety modeling more easy than with fault trees; Simulation of fault propagation Easier communication with other specialists	Difficulties to compute the sequences Difficulties to perform efficient probabilities allocation from cutsets or sequences Difficulties to know how to integrate the MBSA analyses in official documentation and in the ARP process for certification Difficulty to integrate the models : Intellectual property issues.

	Assests of the MBSA	Weaknessess
Liebherr aerospace	Efficient to share and trace the system description understanding between system and safety specialist Greatly ease the updates for local modification (failure rates, control logics) Integration of MBSA analyses in official document is easier since ARP process has been updated with MBSA approach (ARP.4754B)	Necessity to create a customized methodology which is time consuming. Need of an MBSA specialist in order to handle blocking modeling issues
THALES SA	Better consistency between engineering artifacts (System & Safety) (support-tooled) Easier impact analysis (tooled) Sharing by System & Safety engineers of a single base of concepts & language of modeling (from Capella-Arcadia) Support tooled for System / Safety Co-Engineering (Coupling with dedicated external tool (FMEA, FTA))	Life cycles, modeling objectives, Model-Based adoption speed & maturation of the different models between System Engineering & Safety (Desynchronization, granularity) → Need to consolidate the Co engineering methodology
APSYS	Enable modeling complex systems or functions whose dysfunctional behavior is difficult to characterize with conventional approaches (FTA or DD) Ease update for local modification Enable sharing with system designers based on a common graphical formalism (model structure) and simulation scenarios.	Generate many results (cutsets or sequences) which requires post-processing Requires specific skills and tooling
LGM	Process architectures quickly for Trade-Offs Discussion around the model with integrator Modeling of different possible layers 1 single model possible for several CF Allows fault propagation analysis (no need to be part of the system in FMEA) Allows dynamic modeling (network) Adapted to deal with developments Coupling with optimization algorithms ("early design" approach)	Uncommon method and tooling Loop modeling Some modeling tricks are sometimes necessary Graphic representation (define a framework, a method) Probabilities computation Need of reassuring of the user and customers

6.3.1.7 Conclusions

It is worth outlining in this conclusion that the MBSA community is a small community and that S2C projects involves a significant number of its actors and recognized experts.

The overview provided in this section shows that expectations and level of expertise of S2C members greatly differ. Nevertheless, there is a strong will of the experts to disseminate their methodology. On the other hand, the main points preventing a better diffusion and integration of the MBSA in operational developments have also been identified.

In order to allow a larger use of the MBSA, S2C project will aim to provide adaptable generic methods and guidance for modelling. It will also support the safety activities within the development process as well as the certification.

Eventually it is also important to point that in order to fit the member’s choice and experience, S2C project will focus on the MBSA defined as Functional and Dysfunctional modeling in Section 6.1.1. The chosen modelling language will be AltaRica.

6.4 Review of modelling languages of propagation models for safety

The scope of this review matches with the scope of the project S2C. Consequently, it focuses on the Model Based System Analysis defined in **ARP4761A** guidelines. Note that at the time of writing **ARP4761A** has not been issued yet but has been released for review.

ARP4761A current version defines MBSA as the "[…] technique which models system content and behavior in order to provide safety analysis results. MBSA employs an analytical model called a Failure Propagation Model **(FPM)."**

We explain and analyze the different formalisms corresponding to this definition as well as the methods and tools solutions supporting those formalisms. The methods and tools analyzed targeted are the ones with enough maturity to be used in an industrial context, and suitable for aeronautics developments.

In the following, we review different languages supporting the MBSA modelling within S2C scope. To describe efficiently languages, Section 6.4.1 presents and defines some language characteristics, Section 6.4.2 introduces the different languages and finally Section 6.4.3 gives an overview and provides a comparison between languages.

The following modelling languages supporting the MBSA approach are considered:

- AltaRica (AltaRica LaBRI, AltaRica DataFlow, AltaRica 3.0)
- Figaro
- SAML
- HiP-HOPS
- Component Fault Trees
- GRIF version of Generalized Stochastic Petri Nets
- Safety Architect

6.4.1 Modeling language characteristics

Modeling languages enable capturing the knowledge about a system with more or less accuracy and flexibility. The modelling language characteristics have an impact on the problems one can solve as well as the efficiency of the modelling. In order to describe and to compare modelling languages and associated assessment tools we consider the following characteristics:

1. Expressive power of the language;
2. Structural constructs;
3. Autonomy regarding architecture/design models;
4. Graphical or textual modeling;
5. Processing algorithms and outputs;
6. Model verification and validation methods.

These characteristics are explained below.

6.4.1.1 Expressive power

The expressive power of a modeling language is its capacity to capture the system behavior with the needed level of discernment. It depends on the syntax and semantics of the language.

Modeling languages provide at least writing conventions called also the language *syntax*. A syntax defines the usable symbols (e.g. letters for textual languages) and predefined concepts (e.g. key words or constructs) to build system models. It also defines grammatical rules to combine symbols and predefined concepts into *well-formed system models, i.e. system models readable by tools and users*.

For instance, fault trees are a subset of well-formed formulae of the Boolean algebra. Leaves of the trees are atomic independent propositions. They are combined by constructs "and" and "or" into intermediate events until the top event. According to the grammatical rules, the "and" construct (gate) has two inputs and each input shall be connected to an event (leaf or intermediate event).

It is worth noting that *a syntax with high level constructs helps to write concise system models*. For instance, equivalent fault trees can be written only with gates "and" and "or". The gate "k/n" helps to encode with a unique gate a predefined combinations of several "and" and "or" gates.

Formal modeling languages have also a *semantics* to give an unambiguous meaning to the system models. A semantics defines the usable mathematical structures for interpreting system models as observation of the system behavior. It also defines the semantical rules to decide whether a given interpretation satisfies a system model, i.e. whether it represents one observable behavior of the system model. *Users and tools can have the same understanding of a well-formed system model if this model is satisfiable by at least one interpretation*. Unsatisfiable models lead to runtime errors or to wrong analyses.

For instance, the interpretation structures associated to the fault trees are snapshots of one system state during a simulation, i.e. the assignments of one value to each observable variables of the system. In this case, the observable variables of the system are the fault tree leaves, which represent atomic independent failure events. A leaf value is Boolean, i.e. "true" or "false". The leaf is "true" means that the corresponding failure event occurred before the system snapshot. A system cut-set is a set of leaves that makes the top event true when their assignment is true. The set of cut-sets of a top event represents all the interpretation structures that satisfies the top event.

It is worth noting that *system models are equivalent if and only if they are satisfied by the same interpretations*. For instance, a k/n gate of a system model can be replaced without ambiguity by any equivalent combination of "and" and "or" gates.

Usual interpretation structures used for failure propagation analysis are system states (like for fault trees) or traces (i.e. sequences of system states that characterize model runs) or automata that generate traces. For instance, AltaRica models can be interpreted over both automaton and traces. Rich interpretation structures help to observe more accurately the system behavior. *System model shall be built with a language expressive enough to capture the system behavior with the expected level of discernment*.

The equivalence of system models can be used to compare the expressive power of two modeling languages. Let L1 and L2 be two modeling languages that can be interpreted over the same class of structures. *L1 is at least as expressive as L2* if and only if, for each system model M2 of L2 there exist a system model M1 of L1 equivalent to M2 over the considered class of interpretation structures.

For instance, fault trees are interpreted over system states. Therefore, they can be interpreted over system traces reduced to a sequence of one state. It is possible to build for each fault tree an AltaRica model that is satisfied by the same reduced traces. It means that AltaRica has at least the same expressive power as fault trees. The reverse is false. The AltaRica models can discern the temporal orders between the several system states. The classical fault trees cannot discern such details of the system behavior.

More generally speaking, these concepts can be used to refine formal definitions given earlier.

Static system models discern only system states whereas *dynamic* system models can also discern system traces.

Determinist system models discern traces such that the next system state of a sequence is fully defined by the current state and the system input or event.

Failure logic modelling languages (FLM) discern only the occurrence of failure events and the system variables that describes the system error states.

Functional and Dysfunctional modeling (FDM) can discern not only the failure events and the error states but also the functional events and modes.

6.4.1.2 Structural constructs

The second characteristic is related to the ability of the language to organize models, to make them easily readable, maintainable and reusable.

The basic structural construct is a block, also called a prototype. A block is a container for variables, parameters and all the other modeling artifacts. The simplest structuring relation is the composition. A block may be composed of several other blocks. Classical safety analysis formalisms, such as Fault Trees and Reliability Block Diagrams, use only blocks and composition for structuring models.

In order to be able to reuse blocks, structured programming languages introduce the notions of class and instantiation of classes. A class is a reusable "on-the-shelf" block, which is stored in a library and can be reused everywhere in the model via instantiation.

In some cases, it is necessary to modify or to extend a modeling unit (a class or a block) without instantiation. It can be achieved via inheritance relation introduced in object-oriented programming languages. If a modeling unit A inherits from a modeling unit B, then A contains all the characteristics of B and adds some new characteristics.

There are cases where the same component is used in several places or to contribute to different functions of the system. In other words, a modeling unit is shared between several other modeling units. This kind of "uses" relation between modeling units is called aggregation.

In object- oriented programming languages, the reuse of modeling units is done by means of instantiation of classes. In modeling languages using only blocks (called prototype-oriented languages), the reuse of blocks is also possible. It is achieved via the notion of cloning. If a block A is a clone of a block B, then the block A has exactly the same characteristics as the block B.

To know more about structuring constructs of modeling languages see (Michel Batteux, 2018).

To summarize, there are the following constructs to organize and structure models:

- Two types of modeling units: block and class;
- Three structural relations: composition, inheritance and aggregation; and
- Two mechanisms making possible to reuse modeling elements: prototype/cloning and class/instantiation.

These constructs originate from programming languages (see the table below) and are exploited in different ways by MBSA formalisms.

Table 12 : Structural constructs

	Structuring paradigm	**Structural constructs**
1	*Block diagrams*	Blocks + composition
2	*Structured programming*	Classes + composition
3	*Object-oriented programming*	Classes + composition + inheritance
4	*Prototype-oriented and object-oriented programming*	Blocks + Classes + composition + inheritance + aggregation + cloning

Not only modeling units, but also flows may be structured to gather them (see Figure 37 for example). It requires the definition of a structured domain.

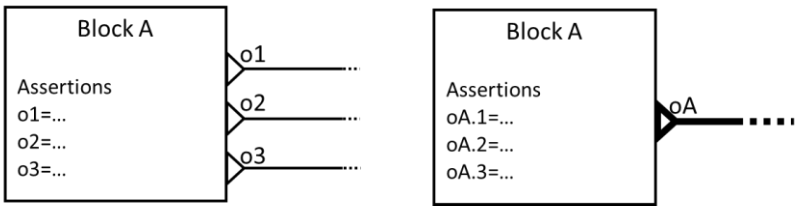

Figure 37: An example of simple flows (left hand) and structured flow (right-hand)

Languages structural construct can be more or less complex. Even though complexity is usually positive as it provides a large range of possibilities of maintainability and reusability, it is worth noting that it can also be difficult to handle, understand and read. For example, class structure enables to modify several blocks in one operation and thus ensure some internal consistency at cost of training people handling the model to know the concept of class and making less direct the modification of only one block.

6.4.1.3 Autonomy regarding architecture models

The third characteristics deals with the links between safety and design models. If the failure propagation model is autonomous, it is fully built and exploited independently from any architecture model.

Some MBSA languages are extensions of system architecture modeling languages and models are built by annotating design models. In this case, there is no autonomy.

There are MBSA languages that are domain specific languages. In this case, the creation of safety models is completely independent from the creation of system design models.

6.4.1.4 Graphical or textual modeling language

Modeling languages can be textual or graphical. Examples of graphical modeling languages are UML or SysML. Classical safety assessment formalisms, like Fault Trees and Reliability Block Diagrams, have both a graphical and a textual format.

In general, MBSA languages are textual. However, modeling tools propose different graphical views to represent, to edit and to animate failure propagation models. Contrary to full graphical formalisms such as fault trees, most MBSA languages are displayed graphically partially only. Graphical views offered by tools can be organize in two categories:

- Graphical support of text edition (with optionally graphical representation of model breakdown and state machine) without graphical representation of flows. The model is mainly represented as text and navigation in text is eased.
- Graphical support of flows. The model is mainly represented as a diagram with blocks and flows. Some subpart of the model are displayed textually.

Regarding description view, text edition is very efficient for experts whereas graphical edition is easier to learn, especially for safety engineers from industry. Regarding simulation view, it is a consensus that graphical representation of failure propagation through flows is a significant support.

6.4.1.5 Model processing algorithms and outputs

Different processing algorithms are available for Model Based Safety Assessment:

1. Boolean equations generation (see Appendix A);
2. Sequence generation (see Appendix A);
3. Markov chain generation;
4. Monte-Carlo simulation (see Appendix A);
5. Model-checking;
6. Probabilistic model-checking.

Boolean equations generation.

When it is possible (the input model is static), MBSA models are transformed into a set of Boolean equations. The generated Boolean equations are assessed by classical Fault Tree Assessment tools in order to calculate Minimal Cut Sets (MCS) and probabilistic indicators. Note that some dynamic models also can be transformed into Boolean equations. However, some information can be lost.

Sequence generation.

For dynamic models, the generation of Boolean equations is not advisable. Then the sequence generation is used. The output is critical sequences of failure events (in some cases the most probable).

Markov chains generation.

In some cases, when the input model is quite small and it obeys Markovian property, it is possible to transform it to continuous or discrete time Markov chains. Then the generated Markov chains are assessed in order to calculate probabilistic indicators.

Monte-Carlo simulation.

Monte-Carlo simulation consists in running a pseudo-random a set of executions (histories) of the model and making statistics on them. In case of MBSA, it is used to calculate probabilistic indicators.

Model-checking.

In general, model-checking is a method for checking whether a finite-state model of a system verifies a given set of properties. If the property is not satisfied, the model checker provide a counter example scenario. In case of MBSA, it enables to generate counter example sequences of failure events that lead to an unsafe system state.

Probabilistic model-checking.

Probabilistic model-checking is used for modeling and analysis of systems that exhibit probabilistic behavior. In case of MBSA, it enables to calculate probabilistic indicators such as the system reliability at a time instant.

6.4.1.6 Model and validation methods

In the following, we consider that the verification of a model ensures that the model complies with construction rules defined by a given modeling method. On the other hand, the validation activity of a model ensures that the model structure and behavior are valid relatively to what it represents.

Model verification

Model verification usually relies on syntactical analysis. Construction rules can restrict the use of the language, forbidding given structural constructs or given behavioral features such as synchronization in AltaRica. They can also prescript to use some features, such as to add comments to each model element.

Verification relies on simple techniques of language analysis and may be integrated in modeling tools.

Model validation

There are three main classes of tooled validation methods:

- Review of model elements,
- Stepwise or step by step simulation,
- Model checking.

The review of model elements is simple to perform and does not require any other tool on top of edition tools. It enables a validation of model elements separately from one another: it is local.

Stepwise simulation consists in testing failure scenarios by triggering events and visualizing each step of failure propagation (this propagation can be visualized graphically or textually, see Section 6.4.1.6). Stepwise simulation enables the modeling engineer to play some defined scenarios. Contrary to review, these scenarios may involve several model elements that are linked with one another. A scenario represents the behavior of the model in one precise case. If the simulated behavior is as expected, the model is validated against the simulated scenarios. When the set of simulated scenarios is sufficient, the engineer infers that the model is globally validated. By nature, simulation only gives access to a limited number of scenarios (within the whole set of scenarios possible in the model).

Model checking techniques aim at ensuring that the model complies with defined formal properties and can be used for model validation. By default, model checking applies to the whole set of scenarios possible in the model. Consequently, it can state that a precise behavior exists or does not exist in the model. It is worth noting that the use of model checking strongly depends on the ability to define formally the properties to be checked.

6.4.2 Modeling languages overview

6.4.2.1 AltaRica

AltaRica is a high-level textual formal domain specific modeling language dedicated to Safety Analysis. AltaRica is an event-centric language. The behavior of components is described by means of state machines. The state of a component is represented by variables (the so-called state variables) and their values. The changes of state are possible when, and only when, an event occurs. The occurrence of an event updates the values of the variables, by the firing of a transition: a triple <guard, event, action>, where a guard is a Boolean expression built over the variables and an action is an instruction which modifies the value of state variables.

AltaRica distinguishes two types of variables:

- State variables that can be modified only through the firing of transitions;
- Flow variables whose values are calculated the values of state variables thanks to a mechanism described by means of the so-called assertion. The assertion is executed after each transition firing. Flow variables are used to model information circulating between nodes of a model.

The behavior of components is described inside nodes. Nodes can be assembled into hierarchies, their input and output flow variables can be connected and their transitions can be synchronized. Nodes –can be stored in the libraries of reusable components and are reused by instantiation, like in structured programming languages.

AltaRica is an asynchronous language: only one transition can be fired at a time. However, it offers a mechanism to synchronize events. For example, common cause failures, shared repair crews, broadcasts can be represented by means of synchronizations.

There are three versions of AltaRica modeling language:

- AltaRica LaBRI, the first version of the language developed by LaBRI;
- AltaRica DataFlow, the second version of the language implemented in several industrial tools; and
- AltaRica 3.0 developed by AltaRica Association and implemented in the OpenAltaRica platform by AltaRica Association and IRT SystemX.

The main differences between the three versions can be divided into three parts:

- The semantics of the assertion (calculation of values of flow variables);
- The structural constructs of the language;
- The underlying model of time.

AltaRica LaBRI

The first version of the language has been created at the Computer Science Laboratory of Bordeaux (LaBRI) at the end of the nineties. The semantics of the first version of AltaRica is defined in terms of Constraint Automata (A. Arnold, 2000). In this version of the language, the assertion is a set of constraints. There is no input or output variables. Each time a transition is fired, first the action of the transition is executed to calculate the values of state variables, second a set of constraints (the assertion) is resolved to calculate the values of flow variables. Constraints have a large expressive power. However, in general, solving constraints involves multiple computation iterations and may be very resource consuming. A set of constraints may have several acceptable solutions resulting in non-deterministic model. In addition, there may be no solution. In this case, the initial model is incorrect. Modeling errors in the assertion are detected during the execution of the model.

AltaRica language permits to specify priorities between events. Priorities are locally defined in a node by a partial order on events. In a given configuration, if two events are enabled then only the one having the highest priority can actually occur. If two events are incomparable with respect to the specification (same priority or no defined priority) then they are both possible, that means that the transitions labelled by these events are enabled.

The first version of AltaRica is supported by AltaRica studio, a workbench developed by LaBRI. Several assessment tools have been developed, such as model-checkers (Alain Griffault, 2004), a Fault Tree compiler and a generator of critical sequences of events.

AltaRica DataFlow

This first version of AltaRica, AltaRica LaBRI, was too resource consuming for industrial scale models (for instance, to perform stochastic simulation). Moreover, industrial models are mainly based on oriented flows of data, or energy that do not require the full expressiveness of constraints. To be able to assess industrial scale models, a second version, AltaRica DataFlow, has been created reducing the expressive power of the assertion (Boiteau, 2006). Its semantics is based on Mode Automata (Rauzy, Mode automata and their compilationinto fault trees, 2002). In this version of the language, the assertion is a set of DataFlow assignments. Each flow variable is assigned only once in the model and there is no circular definitions. The order of the execution of the DataFlow assignments is calculated only once during the compilation of the model. When a transition is fired, first, the value of state variables is calculated; second, the value of the flow variables is calculated by executing the DataFlow assignments only once, which is more efficient than resolving constraints. In addition, modeling errors in the assertion are detected during the compilation of the model.

In AltaRica DataFlow, two types of events are introduced: stochastic and deterministic. Stochastic events are associated with probability distributions (for example, exponential, Weibull, etc.). When a transition labelled by a stochastic event becomes enabled (that means that its guard is satisfied in a current configuration), then it is fired after a certain delay, calculated randomly according to the associated probability distribution. Deterministic events are associated with probability distribution named “Dirac(T)”, where $T \geq 0$. When a deterministic transition becomes enabled, it should be fired exactly after a delay T. A special case of deterministic transitions are immediate transitions, associated with Dirac(0), they should be fired as soon as their guards become true. There are cases, when several immediate transitions are enabled at the same time. These transitions may be synchronized if they should be fired exactly at the same time. It is also possible to add a priority to immediate (also called instantaneous) events. A priority is a positive number associated with an event. When several immediate transitions are enabled at the same time, they are fired according to the priorities of their events.

A set of efficient assessment tools has been developed for this version of the language, including a Fault Tree compiler (Rauzy, Mode automata and their compilationinto fault trees, 2002) , a generator of critical sequences of events, and a stepwise simulator. The definition of timed and stochastic semantics of AltaRica DataFlow made it possible to develop a compiler to Markov chains (Rauzy, An experimental study on iterativemethods to compute transient solutions of large markovmodels, 2004), a probabilistic model-checker (TEICHTEIL-KÖNIGSBUCH, 2011) and a Monte Carlo simulator (Khuu, 2008).

AltaRica DataFlow is used as a description language of several industrial modeling tools:

- Cecilia OCAS (Dassault Aviation), and
- Simfia V3 and SimfiaNeo (Apsys).

Many industrial scale experiments have been conducted with this version of the language [(Romain Bernard, 2007), (Pierre Bieber, 2008), (Xavier Quayzin, 2009)].

AltaRica 3.0

In 2012, the third version of AltaRica has been created, called AltaRica 3.0 (T.Prosvirnova, 2013). The idea for this version of the language is:

- to enrich AltaRica DataFlow with new constructs to structure, and to organize models;
- to define new semantics for the assertion to support circular definitions, and
- to stay compatible with AltaRica DataFlow (semantic of execution except priority)

Along with nodes (called classes in this version of the language) and composition relation, AltaRica 3.0 implements inheritance and aggregation relations. It also introduces the concept of block and cloning (another way to reuse modeling elements). AltaRica 3.0 unifies structural constructs coming from object-oriented paradigm of programming languages: prototype-based and class-based.

The semantics of AltaRica 3.0 is defined in terms of Guarded Transition Systems [(A.Rauzy, 2008), (Batteux M., 2017)]. To be able to model easily some kind of looped systems, for instance networks or electrical systems, AltaRica 3.0 introduces the concept of bidirectional assignment. If x and y are variables, *x :=: y* is a bidirectional assignment, which is equivalent to two assignments: *x:=y, y:=x*. The assertion is an instruction, where each variable may be defined in several assignments and there may be circular definitions. After each transition firing, first the action of the transition is executed to calculate the value of state variables, second, the value of flow variables is calculated by fix point solving of the assertion.

It is possible to create DataFlow models with AltaRica 3.0. In this case, the complexity of calculations is the same as for AltaRica DataFlow. In general, fix point solving of the assertion is more resource consuming than calculation of DataFlow assignments, but less resource consuming than resolving constraints used by AltaRica LaBRI. Modeling errors of incorrect assertions are detected during the execution of the model.

Like in AltaRica DataFlow, there are also two types of events: stochastic and deterministic. The concept of priority for immediate transitions does not exist. Only the concept of expectation, associated with immediate events remains. It corresponds in a certain way to the probability of occurrence of the event. When several transitions $t_1, t_2, \ldots, t_n$, labelled by the events $e_1, e_2, \ldots, e_n$, are enabled at the same time, the probability of each transition to be fired is calculated as follows:

$$P(t_i) = \frac{expectation(e_i)}{\sum_{j=1}^{n} expectation\ (e_j)}$$

AltaRica 3.0 also introduces other types of probability distributions, which may be associated with events:

- A uniform distribution, and
- A user defined distribution (described by a set of points).

AltaRica 3.0 is supported by AltaRica wizard developed by the project OpenAltaRica. A set of assessment tools has been developed, as a joint effort of AltaRica Association and IRT SystemX, including:

- AltaRica Wizard, a textual user interface for AltaRica 3.0 projects management (Batteux M, 2018),
- A stepwise simulator,
- A compiler to Boolean equations (Prosvirnova, 2015), and

- A stochastic simulator (Benjamin Aupetit, 2015).

A prototype of a generator of Markov chains also has been developed (Pierre-Antoine Brameret, 2015).

Differences between the three versions of AltaRica are summarized in the following table:

Table 13 : The three versions of Altarica

		AltaRica LaBRI	AltaRica DataFlow	AltaRica 3.0
Assertion	**Computation**	Constraints solving	DataFlow assignments solving	Fixpoint solving
	Bidirectional flows	Yes	No	Yes
	Computational cost	High	Low	Middle
	Errors detection	During the execution of the model	During the compilation of the model	During the execution of the model
Structural constructs		- Class, - Composition, - Class instantiation	- Class, - Composition, - Class instantiation	- Class and block, - Composition, inheritance, aggregation, - Block cloning, class instantiation
Timed/stochastic models		- Stochastic: Exponential, Weibul, etc. - Determistic: Dirac(T) - Priority	- Stochastic: Exponential, Weibull, etc. - Deterministic: Dirac(T) - Priority	- Stochastic: Exponential, Weibull, etc. - Stochastic: Uniform, user defined distribution. - Deterministic: Dirac(T)

6.4.2.2 Figaro

Figaro (Bouissou M V. N., 1991) is a domain specific textual modeling language dedicated to dependability assessment of complex systems developed by EDF R&D. It combines object-orientation language features, like inheritance and hierarchical representation, and first order production rules:

- interaction rules to model the propagation of instantaneous effects and
- occurrence rules, yielding a list of events that may happen in a state of the system and that have a particular semantics related to time.

The semantics of the language is defined in terms of stochastic processes.

Figaro is used as a description language to create libraries of reusable components called knowledge bases for KB3 (Bouissou M, 2005), a workbench developed by EDF R&D to automatically perform systems dependability assessment. In the KB3 workbench, graphical representations are associated to textual models.

KB3 workbench provides different assessment tools, including Boolean equation generation, Monte-Carlo simulation, Markov chain generation and quantification and generation of critical sequences of events (in some cases the most probable).

6.4.2.3 SAML (Safety Analysis Modelling Language)

SAML (Safety Analysis Modeling Language) (Gudemann, 2010) is a formal textual domain specific modeling language. The special feature of SAML is that it is a synchronous language. A SAML model is expressed in terms of finite stochastic state automata. Automata are all executed in discrete time steps with parallel composition. The semantics of a SAML model is defined as Markov decision process.

VECS (Marco Filax, 2014) is a design and verification environment focused on SAML models. It provides a model editor and model analysis tools: a stepwise simulator and translators to the input languages of the probabilistic model checker

PRISM and the symbolic model checker NuSMV. Analysis with PRISM model checker enables to compute probabilistic indicators, such as reliability or availability at a given instant of time. Analysis with NuSMV model checker provides minimal cut sets or sequences of events leading from an initial state to a failure state defined in the model.

6.4.2.4 CFT

This section will be completed in next revision.

6.4.2.5 HiP-HOPS

HiP-HOPS (Hierarchically Performed Hazard Origin & Propagation Studies) (Papadopoulos, et al., 2011) is a modeling language and a processing tool for dependability analysis, mainly availability and safety. The tool has an architecture optimization feature (with respect to availability and cost) that is out of the scope of this document.

To use HiP-HOPS, the first requirement is to have a detailed design model in SimulationX or Simulink (with a nominal behavior and continuous physics as level of representativeness). The system breakdown is reused and components are annotated with dysfunctional behavior, with a logical level of representativeness. Each component has

- Failure modes, modeling its internal failures (contrary to AltaRica, the transitions between failure modes are not specified)
- Inputs, propagating failure modes from upstream components
- Outputs that propagate downstream failure modes that are function of internal failure modes and input failure modes (use of AND and OR operators)

Each component is then equivalent in expressiveness to a “mini fault-tree” in the sense it combines only failures and Boolean operators.

The HiP-HOPS tool is able to generate a fault tree by exploring backwards failure propagation (resolving Boolean equations). Resulting cutsets are then outputted in a so-called FMEA table that takes into account multiple failures. Probability of feared event at system level is computed based on cutsets. An extension to dynamic models with PAND (Priority-AND) and POR (Priority-OR) is available to get sequenced behavior.

Recent publications present extensions for dynamic modeling, either using “Priority-AND gates”[Papadopoulos2018] or Semi Markov processes[Papadopoulos2019] but these features are not documented in the user manual of the tool [HiP-HOPS2013]). Consequently, they are not considered in Section 6.4.3.

6.4.2.6 Underlying formalism of GRIF (Stochastic Petri Nets ++)

To be completed in next revision

6.4.2.7 Underlying formalism of Safety Architect (CFT)

Safety Architect relies on the original concepts of Interface Focused-FMEA (Y. Papadopoulos, 1999) and Component Fault Tree (CFT)) (B. Kaiser, 2003)

6.4.3 Languages characteristics summary

In the following, we compare MBSA formalisms presented in this section using modeling languages characteristics described in section 6.4.1.

6.4.3.1 Expressive power

The studied formalisms can be divided into 2 categories according to their expressive power:

- Combinatorial formalisms, describing the system only in terms of states;
- States/transitions formalisms, describing the system in terms of states and transitions.

Hip-HOPS and component Fault Trees are classified in the first category. They are based on Probabilistic Boolean Algebras. They can describe only static and deterministic models using Failure Logic Modeling (FLM) approach. Recent publications present extensions of HiP-HOPS taking into account dynamic modeling, either using “Priority-AND gates” [Papadopoulos2018] or Semi Markov processes [Papadopoulos2019].

Figaro, SAML, the underlying formalism of GRIF and all the versions of AltaRica are part of the second category. They are based on different types of transitions systems and enables to describe either static or dynamic models, deterministic or non-deterministic models, using FLM (Failure Logic Modeling) or FDM (Functional and Dysfunctional Modeling) approaches.

The comparison of expressive power of studied MBSA formalisms is summarized in the following table.

Table 14 : Summary of expressive power characteristic

Modeling formalism	Mathematical formalism	States/Transitions	FLM/FDM	Deterministic/non-deterministic	Static/Dynamic
AltaRica LaBRI	Constraint Automata	States and transitions	FLM or FDM	Deterministic or non-deterministic	Static or dynamic
AltaRica DataFlow	Mode Automata (or DataFlow Guarded Transition Systems)	States and transitions	FLM or FDM	Deterministic or non-deterministic	Static or dynamic
AltaRica 3.0	Guarded Transition Systems	States and transitions	FLM or FDM	Deterministic or non-deterministic	Static or dynamic
Figaro	Stochastic Processes	States and transitions	FLM or FDM	Deterministic or non-deterministic	Static or dynamic
SAML	Markov decision process	States and transitions	FLM or FDM	Deterministic or non-deterministic	Static or dynamic
HiP-HOPS	Probabilistic Boolean Algebra	States	FLM	Deterministic	Static
Component Fault Trees	Probabilistic Boolean Algebra	States	FLM	Deterministic	Static
GRIF	Generalized Stochastic Petri Nets	States and transitions	FLM or FDM	Deterministic or non-deterministic	Static or dynamic

6.4.3.2 Structural constructs

The following table summarizes the structural constructs of MBSA formalisms.

Table 15 : Summary of structural constructs

Modeling formalism	Flow structure	Block structure	
		Structuring paradigm	Structural constructs
AltaRica LaBRI	Yes	*Structured programming*	Classes + composition
AltaRica DataFlow	Yes	*Structured programming*	Classes + composition

AltaRica 3.0	Yes	*Prototype-oriented and object-oriented programming*	Blocks + Classes + composition + inheritance + aggregation + cloning
Figaro	No	*Object-oriented programming*	Classes + composition + inheritance
SAML	No	*Block diagrams*	Blocks + composition
HiP-HOPS	*To be completed*	*Block diagrams*	Blocks + composition
Component Fault Trees	*To be completed*	*Block diagrams*	Blocks + composition
GRIF	*To be completed*	To be completed	*To be completed*

HiP-HOPS, SAML and Component Fault Trees are based on Block diagrams, like classical safety analyses formalisms: Fault Trees and Reliability Block Diagrams.

AltaRica LaBRI, AltaRica DataFlow and the underlying formalism of GRIF have classes (called nodes in AltaRica LaBRI and AltaRica DataFlow) and composition relation.

Figaro is an object-oriented modeling language. In addition to classes and composition relation, it also implements inheritance relation.

The new version of AltaRica, AltaRica 3.0, includes structural constructs coming from object-oriented and prototype-oriented programming languages. It implements both modeling units (blocks and classes), the three relations (composition, inheritance and aggregation) and both ways to reuse models (class/instantiation and block/cloning). It offers a full range of ways to structure models.

6.4.3.3 Autonomy regarding architecture/design models and graphical/textual modeling

Figaro, SAML, the underlying modeling language of GRIF, component Fault Trees and all the versions of AltaRica are independent from design models domain specific languages dedicated to Safety analyses. HiP-HOPS models are quite different because they are created by annotating design models (SimulationX or Simulink).

In general, MBSA modeling languages are textual. It is the case of Figaro, SAML and all the versions of AltaRica. However, it is possible to associate graphical representations to textual models. For example, Integrated Modeling and Simulation Environments, like Cecilia OCAS or Simfia, propose different types of graphical views for AltaRica DataFlow models. KB3 workbench also provides graphical representations for Figaro models. Components Fault Trees and the underlying formalism of GRIF are graphical.

A special case is HiP-HOPS, which is both graphical and textual: annotations are textual, but the annotated models are graphical.

The following table summarizes the comparison.

Table 16 : Summary regarding autonomy and graphical characteristics

Modeling formalism	**Autonomous language**	**Graphical or textual**	**Graphical modeling facilities**

AltaRica LaBRI	Yes	Textual	Graphical support of text edition with graphical representation of model breakdown and state machine
AltaRica DataFlow	Yes	Textual	Graphical support of flows. Edition and simulation supported by Cecilia Workshop and SimfiaNeo.
AltaRica 3.0	Yes	Textual	Graphical support of text edition. Prototype of graphical visualization of model structure under development
Figaro	Yes	Textual	Graphical support of flows. Edition and simulation supported by KB3 tool
SAML	Yes	Textual	
HiP-HOPS	No	Graphical and textual	Textual annotations for Simulink or SimulationX design models
Component Fault Trees	Yes	Graphical	
GRIF	Yes	Graphical	Graphical support of flows

6.4.3.4 Model processing algorithms and outputs

Combinatorial formalisms, such as HiP-HOPS and Component Fault Trees, are transformed into Boolean equations. Recent publications present extensions of HiP-HOPS taking into account dynamic modeling.

AltaRica LaBRI is supported by AltaRica Studio, an open source tool, developed by LaBRI. It includes a prototype of a graphical front-end for ARC model checker. It can be downloaded from https://AltaRica.labri.fr/wp/?page_id=286. Boolean equations and sequences are generated using ARC model-checker.

Cecilia OCAS, developed by Dassault Aviation, and SimfiaNeo, developed by Airbus/Apsys, use AltaRica DataFlow as a description language. These tools provide graphical user interfaces for edition and stepwise simulation of AltaRica DataFlow models. Cecilia OCAS includes a Boolean equation and a sequence generator. SimfiaNeo also provides a Monte-Carlo simulator. SimfiaNeo is a commercially distributed modeling tool. A Markov chains generator and a probabilistic model-checker have been developed for AltaRica DataFlow by research teams. However, today they are not available for the large audience.

AltaRica 3.0 is at the core of the OpenAltaRica platform, developed by AltaRica Association and IRT SystemX. It is freely available for research and education purposes and can be downloaded from https://www.openAltaRica.fr/docs-downloads/. The OpenAltaRica platform is composed of several tools:

- AltaRica Wizard, a textual user interface for AltaRica 3.0 projects management,
- A stepwise simulator,
- A compiler to Boolean equations, and
- A stochastic simulator.

A sequence generator and a prototype of graphical visualization of model structure are under development. A prototype of a compiler to Markov chains has been developed; today it is not available for the large audience.

Figaro is supported by the KB3 platform developed by EDF Research and Development. It includes a graphical user interface and a set of tools: generators of Boolean equations, sequences and Markov chains, stepwise and Monte Carlo simulators.

SAML is supported by VECS, an Eclipse-based specification framework for the analysis of software-intensive systems. It includes an implementation of the SAML language and model transformations to verification engines NuSMV for model checking and PRISM for probabilistic model checking.

The underlying formalism of GRIF, developed by SATODEV, is an extended version of Generalized Stochastic Petri Nets. Models are assessed by Monte Carlo simulation.

The following table summarizes the assessment tools available for the studied MBSA formalisms. Available tools are marked:

- In green, if they are open source tools;
- In blue, if they are freely available for education and research purposes;
- In grey, if they are research prototypes not available for the large audience;
- In orange, if they are industrial or commercial tools.

Table 17 : Summary of processing algorithm

Modeling formalism	**Boolean equations**	**Sequences**	**Model-checking**	**Probabilistic model-checking**	**Monte-Carlo simulation**	**Markov chains**
AltaRica LaBRI	Yes	Yes	Yes	No	No	No
AltaRica DataFlow	Yes	Yes	Yes	Yes	Yes	Yes
AltaRica 3.0	Yes	Yes	No	No	Yes	Yes
Figaro	Yes	Yes	No	No	Yes	Yes
SAML	No	No	Yes	Yes	No	No
HiP-HOPS	Yes	No	No	No	No	No
Component Fault Trees	Yes	No	No	No	No	No
GRIF	No	No	No	No	Yes	No

6.5 AltaRica MBSA modelling methods

S2C aims at developing a shared methodology based on members experience and best practices. This section provides an overview of the AltaRica modeling methodologies developed by S2C members within their company. Our goal is to give a general survey of the key features of existing methodologies, provided as an organized list for each contributor.

6.5.1 Key features of ONERA AltaRica modeling method

This part deals with ONERA AltaRica modeling method main features. It does not provide an exhaustive description but outlines the main features of the approach.

Main principles

A general modeling and analysis methodology tuned according to the analysis goal and context:

- Specify the analysis goal: for example, model of the procedures of system operation to consolidate the classification of failure condition (FHA), model of the system technical functions to compute functional

failure sets and support the FDAL allocation, model of the physical architecture of the system to support PSSA or SSA.
- Specify the analysis context: available system documents, former reusable models of components or system, etc.
- Specify the model contents according to the analysis goal and context: failure conditions, basic entities of the model (for example, external function, technical function or physical component), failure/functioning modes local to entities or global ones, safety functions used to monitor hazard and reduced the risk.
- Reuse, adapt or develop new AltaRica library of models according to the analysis goal and context. Models are grouped into library depending on the type of propagated values. For instance, a model of reliability blocks propagates Boolean values (the block is working or not). The model of alarm function propagates three status values: alarm is ok, false or lost. The models of electrical components can propagate physical failure modes such as “short-circuit” and so on.
- Connect the basic components into a hierarchical structure that is as close as possible to the structure of reference specifications, for example, like BPMN models of procedures, like functional chains or like the physical architectures.
- Validate the model by simulation of reference scenarios.
- Compute and analyze the sequences of events or the cut-sets leading to the failure conditions.
- Reuse these outputs for DAL allocation (manually or automated with ONERA Dalculator) or for quantifying the probability of occurrence of the failure condition via fault tree tools.

Points of special importance

Experience returns enable a large capitalization of former models via libraries of family of reusable components.

Modeling patterns helps also to prevent modeling errors such as circular variable definition, oscillation of events …

Graphical conventions are systematically associated to the component of libraries to ease model reading without entering into the detail of the AltaRica code. They help also to see failure propagation paths inside the model.

Kind of models

- Models are dynamic because they incorporate the reconfigurations

Models output

- Sequence generated :
    - functional failure sets of operation or functional models;
    - physical failure sets from the system architecture models.
- FDAL allocation and requirement for functional segregation.
- Verification of the fail safe (resp. fail operational) criteria i.e. no single (no double) failure leads to an unsafe situation.
- Quantification of the probability of occurrence of failure conditions.

6.5.2 Key features of APSYS AltaRica modeling method

This part deals with APSYS AltaRica modelling method main feature. It does not provide an exhaustive description but outlines the main features of the approach.

Main principles

- Models for very different systems and study objectives, necessary to adapt to each context, no end-to-end method.
- Given the identification of feared events and indicators to study, modeling begins with the identification of one (or several) domain (s) to be used in most of the model (e.g. {ok, err, lost} or float number).

Points of special importance

- Modeling of physical architecture
- No reuse of bricks from a model to another (this method point may be updated in close future)

- Avoid using synchronization, because it endangers model readability and complexify model semantics
- Documentation inside each brick (description field of the brick)

Kind of models

- For safety: static or lowly dynamic models to compute cutsets or sequences
- For availability: highly dynamic models to compute availability rate through Monte Carlo computation

Models output

- Minimal cutset or sequences (with post-treatment support)
- Simulation mode: validation of dysfunctional and/or functional behavior, support to help cutset or sequence understanding
- Indicators computed with Monte Carlo simulations

6.5.3 Key features of DGA AltaRica modeling method

This part deals with DGA AltaRica modeling method main feature. It does not provide an exhaustive description but outlines the main features of the approach.

Main principles

- Used to validate the integration of several systems
- Each block, each flow, each failure of the model is specified
- Granularity : item for DAL allocation, unless an important logic stands underneath
- Three views : functional, organic and zonal
- Maximal use of the graphical tool in order to ease the validation and the maintainability
- Generic library which is the basis of each model
- Modeling rules followed to ease the re-use and the collaborative work
- Use of synchronizations to catch potential common modes as soon as possible

Points of special importance

- Modeling guidelines (why, when and how model)
- Formalized generic components library (specified, verified, use of modeling rules)
- Formalized models (clear concepts, identified objectives)

Kind of models

- Dynamic models (reconfigurations), few highly looped models
- Dysfunctional approach (except the reconfiguration part), trying to use also the models for a functional use (depending on the program needs)

Models output

- Functional failure sets generation for DAL allocation validation (automatic post-treatment)
- Minimal cut-sets generation for fail safe principle (no probability computation yet)
- Identification of common causes (common modes, particular risks, zonal analysis)
- Simulation mode used to validate the model and support the cut-sets understanding

Note: Functional failure sets and Minimal cut-sets are generated through Boolean equations (when possible) or sequence generation (for looped or time dependent models)

6.6 S2C Lot 4 Modeling Guide and Validation Report

The Modeling Guide and Validation Report proposed by S2C project, provides methodological guidance for MBSA using AltaRica Data flow modelling languages and related tools. It presents general principles as well as main identified difficulties that modellers can encounter. Our intent is to provide recommendable generic practices to use MBSA for the support of classical ARP4761A PSSAs and SSAs analyses.

The recommended practices are illustrated using the AltaRica Data Flow language supported by Cecilia OCAS (Dassault Aviation), SimfiaNeo (Apsys) tools and Open AltaRica.

7 MBSA Dissemination State of the art

This section provides a state of the art of Model Based Safety Analysis (MBSA) dissemination by presenting different type of formations, existing groups and conferences available.

7.1 Formations

The table below gives a presentation of the current panel of dissemination. For now, this document version is focused on information from actors in the Toulouse area.

Table 18 : Dissemination actions

	Audience	School, university	Level	Time/Content	Tools	Cost
SATODEV	All	N/A	All	12h	CECILIA V6	Paid
LGM	Internal employee (10 persons trained)	N/A	Safety engineer	2 days Lessons, theoretical exercises and practical exercises (e-learning type)	OAR SIMFIA CECILIA	Free
Polytechnic (ONERA)	All	N/A	All	1 day Safety and formal method	OAR	Paid
ONERA	Student	ENSEEIHT computing specialty	5th year	6h (4h lesson, 2h practical exercises)	OAR	Free
ONERA	Student	INSA electrical engineering	5th year	3h (1h lesson, 2h practical exercises)	CECILIA	Free
ONERA	Student	ISAE engineering and autonomous systems	5th year	10h (6h lesson, 4h practical exercises)	CECILIA	Free
ONERA	Student	ISAE System engineering and embedded systems	Master	5h (3h lesson, 2h practical exercises)	CECILIA	Free
EUROSAE (DGA)	All	N/A	All	Level 1 : 12h (9h lesson,3h theoretical exercises) Level 2 : 24h (9h lesson,3h theoretical exercises, 12h practical exercises)	CECILIA	Paid
DGA	Student	INSA engineering system	5th year	16h (4h lesson,4h theoretical exercises, 8h practical exercises)	CECILIA	Free

	Audience	School, university	Level	Time/Content	Tools	Cost
DGA	Internal employee (10 person/year since 4 years)	N/A	N/A	TBC	TBC	Free
APSYS	Student	Université Toulouse III - Paul Sabatier	Master/5th year	8h, practical exercices	SimfiaNeo	Free
APSYS	Student	ISAE-Supaero	Advanced master	20h, dependability theory and practice (availability and safety analyses)	SimfiaNeo	Paid
APSYS	Student	Hochschule Ruhr West	Bachelor	24h, practical exercices	SimfiaNeo	Free
APSYS	Custormer or internal employee	N/A	N/A	2 days, tool practice	SimfiaNeo	Paid

Table 19 : MBSA trainings available

(In 2015, EUROSAE (DGA) formed thirty experts from EASA. On opportunity, ONERA can give informal training to his study partners).

The synthesis table shows the existence of several channels of dissemination.

Two types of population are concerned. First, there's students and so future engineers in job market with MBSA skills. Then, some employees, with generally safety expertise, are formed to MBSA methodology. Nevertheless, we find that the time allowed to the formation seems to be low.

There does not seem to be a common method of training. Each member offers an adaptation that reflects their own experience or use.

This list is not exhaustive but reflect the status at the time of the document delivery.

7.2 Existing groups, conferences: industrials, academics

7.2.1 Working groups

At the time of writing, within ARP4761A WG63 and S18 working groups, there is a section dedicated to MBSA (Appendix Q).

Except from some specific dedicated days organized by the ONERA (French Aerospace), AFIS (Association Française d'Ingénierie Système) or GIFAS (Groupement des Industries Françaises Aéronautiques) there are no industrial or academic dedicated working group dedicated to MBSA.

7.2.2 Conferences

The following table lists the main conferences having MBSA dedicated sessions.

SafeComp	**International Conference on Computer Safety, Reliability and Security**
(every year)	SAFECOMP was established in 1979 by the European Workshop on Industrial Computer Systems. Since then, it has contributed to the progress of the state-of-the-art in dependable application of computers in safety-related and safety-critical systems. SAFECOMP has contributed to the progress of the state-of-the-art in dependable application of computers in safety-related and safety-critical systems.
IMBSA	**International Symposium on Model-Based Safety and Assessment**
(every year)	Latest developments in model-based engineering, formal techniques, probabilistic analyses and cutting edge optimization to address hard problems in the design of safe complex systems including software intensive and open cyber-physical systems. The objectives are to present experiences and tools, to share ideas, and to consolidate and grow the community
Esrel	**European Safety and Reliability Conference**
(every year)	The annual European Safety and Reliability Conference (ESREL) is an international conference under the auspices of the European Safety and Reliability Association (ESRA) .
lambda mu *(every 2 years)*	Congrès de maîtrise des risques et de sureté de fonctionnement de l'Institut de Maîtrise des Risque (IMdDR)
ERTS	Embedded Real Time Systems
(every 2 years)	Embedded computing platforms and networked systems (MBSE/MBSA)

Appendix A Principles of MBSA computation

This appendix describes the principles of 3 possible MBSA computations. It is intended to guide a modeling engineer to understand the overall computation working and to get him aware of the validity and limitations of results. Note that it will completed in a future revision of the document.

As a pre-requisite, the reader is expected to have a (small) experiment in MBSA model behavior or, otherwise, to have read the section "An introduction to AltaRica DataFlow". Three types of computations are briefly described in their principles, Monte Carlo simulation, Sequence generation and Boolean equation building and resolution. Section A.4 addresses the qualitative differences between the three computations.

A.1 Monte Carlo simulation

Objective: compute numerical indicators, such as average number of failures during the mission time, availability percentage, etc.

Model particularities: dynamic model, including stochastic and Dirac probability distributions.

The principle of Monte Carlo simulation is to generate a large number of random scenarios, to get a meaningful average value of a given numerical indicator.

More precisely, many clones of the model are generated and each clone lives a different story with events occurring at different times, randomly chosen. For each story, the indicator is computed. Once all stories are executed, the value of the meaningful indicator is computed as the average value on all the stories, as shown in Figure 38.

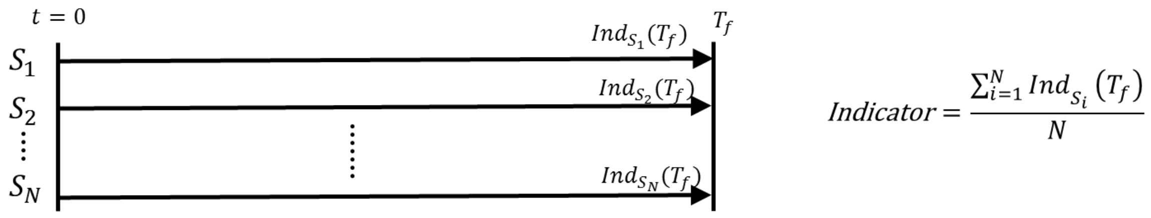

Figure 38: Principle of Monte Carlo simulation

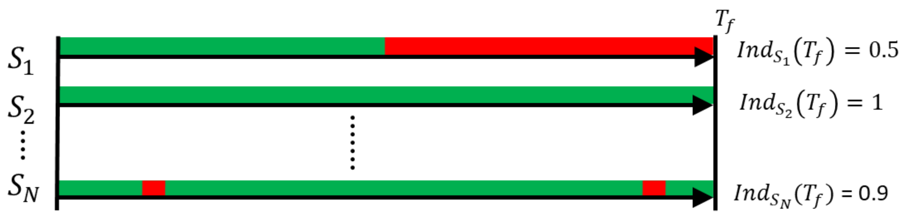

Figure 39: Example of Monte Carlo simulation for availability computation

Figure 39 shows an example for an availability indicator. According to events randomly fired, the system is available (depicted in green in the figure) or not (depicted in red). For each story, the ratio of the time when the system is available relatively to the total time is computed, enabling to compute the average availability.

A safety indicator could also be defined for instance to observe whether the final state of the model is a feared event or not. A reliability indicator could count the number of times the system fails.

Obviously, the occurrence of the events inside each story must follow the probability distributions assigned to events. This is managed by the choice of the firing dates of events. To simplify following explanations, please note that we do not distinguish transitions and events, considering that an event labels only one transition. Let now focus on the generation of theses firing dates.

Monte Carlo simulation is based on a pseudorandom number generator:

a function f such as for any $u_n \in]0,1[$, $u_{n+1} = f(u_n)$ and (u_n) is a periodic sequence with random terms inside one period. As the period is very high, the terms of (u_n) are considered as random numbers.

To launch simulation, a seed (a number in]0,1[) is either given by the user or generated by the computer. This seed is used as the initial term of the sequence, i.e., as u_0.

When launching a Monte Carlo simulation, firstly, the list of enabled stochastic events is determined and a term of the sequence (u_n) is associated to each event. An event is said enabled at time t if the guard of its transition is true at time t.

Let consider a model with Event1 (exponential λ_1) and Event2 (exponential λ_2), both initially enabled. Then association between terms of pseudo-sequence of events is:

- Event1 is associated to $u_1 = f(u_0)$
- Event2 is associated to $u_2 = f(u_1)$

For each event, according to its probability distribution, a firing date is computed, as represented in Figure 40, using the inverse of the integral of the probability distribution (this integral is named cumulative distribution function).

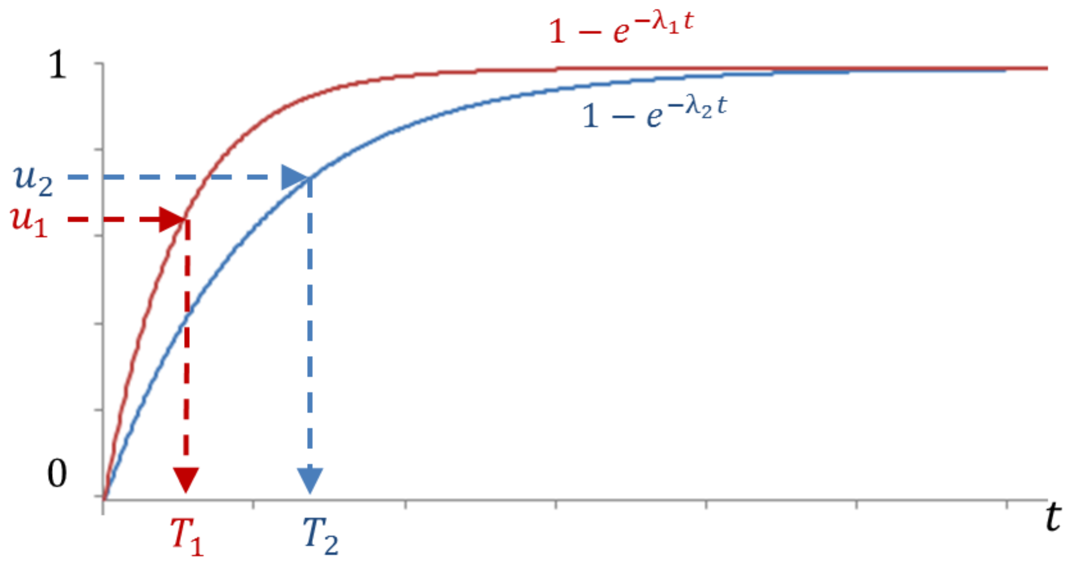

Figure 40: Computation of time of each stochastic event.
The computation is based on the inverse of cumulative distribution function (integral) of probability law

Finally, firing dates computed for stochastic events are compared. The event associated with the lowest firing date is triggered. In the example, $T_1 < T_2$. As a consequence, Event1 is triggered at time T_1, making the first step of the story. In the simulated model, the event has consequences: state changes, value propagation, triggered immediate Dirac events (i.e. Dirac with delay equal to 0). Then a new list of enabled events is determined and the process is redone to get a new step of the story.

If the model contains events with delayed Dirac (i.e., Dirac(T) with T>0), their delays are compared to the firing dates computed for stochastic events. If Event3 with Dirac(T_3) is added to the example, we compare the delay $T_3$to the computed firing dates $T_1$and T_2 and the event with the lowest time is triggered.

The result of a Monte Carlo simulation is a numerical indicator. Depending on tools, stories are available or not. For safety use, the number of occurrence of a feared situation can be transformed into a probability. Nevertheless, if stories are unavailable, no clue is given about the weakest points of the system.

A.2 Sequence generation

Objective: compute minimal sequences that result in a feared situation

Model particularities: dynamic model, including stochastic and Dirac probability distributions.

The principle of sequence generation is to examine each possible sequence in the model and to determine whether it results in the situation that must be observed, i.e. the feared situation.

Sequences are composed of one or several events. The order of a sequence is the number of events that compose the sequence.

As an example, the left part of Figure 41 shows a model composed of three nodes. For the sake of simplicity, no flows are considered between nodes and only events are necessary to the following explanations. In this model, transitions, labeled by the events A, B and D are initially enabled, whereas the transition labeled by C, is not enabled as long as B has not occurred. The right part of the figure shows all possible sequences in the model. In the initial state of the model, transitions, labeled by the events A, B and D, are enabled, resulting in 3 sequences of order 1. Considering the sequence (A), the possible next events are B and D: the sequence (A) results in two branches: (A, B) and (A, D). On the sequence, to continue the sequence (B), the event C is possible, resulting in 3 branches: (B, A), (B, C) and (B, D). The whole set of possible sequences is generated in the same way.

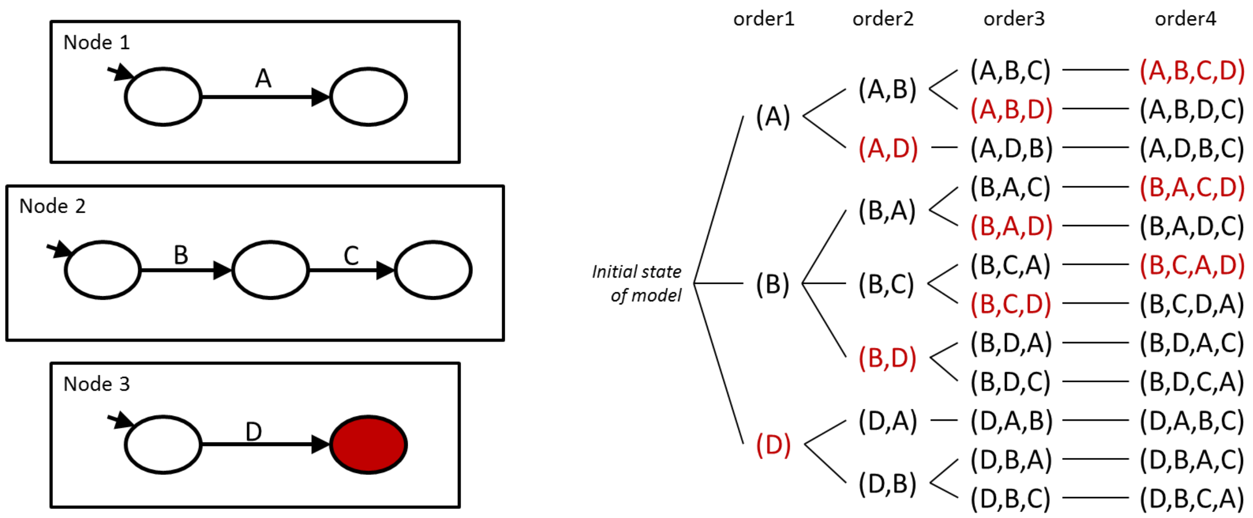

Figure 41: Example of the set of sequences (right part) possible in a given model (left part)

The objective of the computation is to find the sequences resulting in the observed situation. Consider that the observed situation in Figure 41 is the state resulting after the firing of the transition labeled by the event (then the observation depends only on the state of Node 3, which is a very simple observation). The red sequences result in this situation and are the results of the sequence exploration. Nevertheless, the safety analyst needs only minimal sequences. To get them, the computation filters resulted sequences and/or takes into account the resulted sequences during the space exploration.

For each minimal sequence, the probability is computed from the probability distribution of each event.

When industrially used, the number of events generates a space that is too large to be exhaustively explored. Then, the computation is usually limited to a given order of sequences. The rationale is that the higher is the order, the less probable is the sequence.

A.3 Boolean equation building and solving

To be completed.

A.4 Qualitative differences between computations

During sequence generation, the space is subject to combinatorial explosion, in function of the number of events. On the contrary, the Monte Carlo simulation computation is linear in number of events. In particular, models with numerous delayed Dirac are difficult to compute with sequence generation while Monte Carlo is well-fitted to them. If a model has periodic events, the Monte Carlo execution time is impacted by the lowest period (equivalent to a simulation step).

Sequence generation and Boolean equation methods are usually bounded to a maximum order, while, according to the number of stories, Monte Carlo results are bounded to a probability.

Rare events are not a problem for sequence generation or Boolean equation methods because probabilities are computed analytically. On the contrary, Monte Carlo simulation requires a high number of stories to make visible and to compute accurately the probability of rare events.

To be completed.

End of document