xFFBD: towards a formal yet functional modeling language for system designers

Bruno AIZIER
ENSTA Bretagne
2 rue François Verny
29806 Brest CEDEX 9 – France
Bruno.Aizier@ensta-bretagne.fr

Vincent CHAPURLAT
LGI2P
Parc scientifique Georges Besse
30035 Nîmes CEDEX 1 – France
Vincent.Chapurlat@mines-ales.fr

Stéphanie LIZY-DESTREZ
ISAE – SUPAERO
10 avenue Édouard Belin
31055 Toulouse CEDEX 4 – France
Stephanie.Lizy-Destrez@isae.fr

Daniel PRUN
ENAC - LII
7 avenue Édouard Belin - BP 54005
31055 Toulouse – France
Daniel.Prun@enac.fr

Charlotte SEIDNER
L'UNAM - University of Nantes - IRCCyN
1 rue de la Noë, 44300 Nantes – France
Charlotte.Seidner@univ-nantes.fr

Jean-Luc WIPPLER
LUCA Ingénierie
1 Chemin de Pechmirol, 31320 Mervilla – France
jlwippler@gmail.com

Abstract. Although the eFFBD formalism dates back to the 1990s (or even, in a simplified form, the 1950s), it seems that it is still not as much used by the Systems Engineering community as it could. Indeed, eFFBD is a modeling language focusing on functional paradigm *i.e.* allowing functional and behavioral modeling and reasoning about a system. Currently, it is often confronted or compared to other languages such as SysML for activity modeling (activity diagrams) based on object paradigm. This paper aims to demonstrate the interest and the potential advantages for systems designers, like most of the *discipline-oriented* designers to dispose of an enriched (conceptually and semantically) eFFBD modeling language called here xFFBD. This has to be a credible framework for modeling, communicating and reasoning about complex systems. After shortly recalling the history, the key concepts and capabilities of eFFBD, this paper compares eFFBD with other formalisms considered here as relevant for the study, Petri nets and SysML. Several leads are then identified and discussed in order to improve the eFFBD language and to provide a first draft version of xFFBD specification.

Functional and object paradigm

For many years now, systems designers involved in the development of complex systems have been essentially guided or interested by the *functional paradigm*. This paradigm may be summarized as "designing a system means to describe its functions, their organization in order to fulfill a given mission, and to gather all these functions in a coherent functional architecture, later allocated to a physical architecture[1]. This allows then to describe and to reason not only on *what the system must do* but also on *when and how it must do it i.e.* the resulting dynamic of the entire system of interest". Indeed, this dynamic or behavior is specified through the functions' dynamic (their execution duration, temporal hypothesis, synchroniza-

[1] "System architecture is the embodiment of concepts, and the allocation of physical/informational function to elements of form, and definition of interfaces among the elements and with the surrounding context" (Ed. Crawley – MIT)

tion rules…) and the control environment that is the usual control structures (loops, choices, etc.). In this way, (e)FFBD provide means and concepts relevant for functional and dynamic aspects modeling.

Since the last decade, OMG and INCOSE have promoted another way of modeling based on *object paradigm*. This paradigm particularly highlights the possibility to model a system by using encapsulation mechanisms (a set of common behaviors and knowledge is gathered into a class allowing then to describe a sub set of entities of the world), the communication between classes by using messages, the inheritance of behaviors between generic and more specific classes. So, the object paradigm cannot be strictly opposed to the functional one but has to be considered as different way of thinking a system. In real life, both are relevant for systems such as embedded systems and more largely software intensive systems. As an example, SysML (System Modeling Language) has gained a wide acceptance in the SE community as a general-purpose modeling language based on object paradigm. Based on UML, often deemed too software-centric, it is also a graphical language, organized around diagrams.

In the context of designing complex systems within a multidisciplinary team encompassing many engineering disciplines, at once fundamentals (mechanics, electronic, thermal …) and transversal (integrated logistics support, supply-chain, human factors …), the functional paradigm seems to be the most appropriate to build together, in a collaborative way, an optimal solution against expected various stakeholders goals to be achieve. Even if the object paradigm has truly demonstrated its potentiality, there is still a strong cultural gap for the disciplines mentioned above, that are largely founded on the functional paradigm, to adopt such a new paradigm for them. So, this paper intents to demonstrate the interest and the potential advantages for system design multidisciplinary teams to dispose of an enriched (conceptually and semantically) version of eFFBD modeling language called here xFFBD (x, as variable *x*, means “unknown” at this stage). The goal is here to propose and formalize a credible extended functional-oriented framework for modeling, communicating and reasoning about complex systems.

(e)FFBD position

A brief history of (e)FFBD in Systems Engineering

The FFBD (Functional Flow Block Diagram) language was first introduced in the late 1950s at TRW. It was not the very first process - or functional- modeling language (see Figure 1), yet it can be considered as the first modeling technique favored by the Systems Engineering community (Chesnut 1967). The FFBD language, focused on the control sequencing of the functions was for instance used by NASA to model the time sequence of space systems and flight missions. Indeed, this language provides system designers with an easy framework to describe the behavior of complex, distributed, hierarchical, concurrent and communicating systems.

In the 1990s, some enhancements were added to provide the designer with a way to model *flows*, whether they are input, output or triggering flows (FAA 2006, NASA 1995, NASA 2007). These *enhanced FFBD* (or eFFBD) can be described as discrete event systems and therefore be executed in simulation tools, thus providing some validation capability (Seidner *et al.* 2008, Seidner 2009). In the next section, the main characteristics of eFFBD are highlighted and discussed in contrast to some other modeling languages.

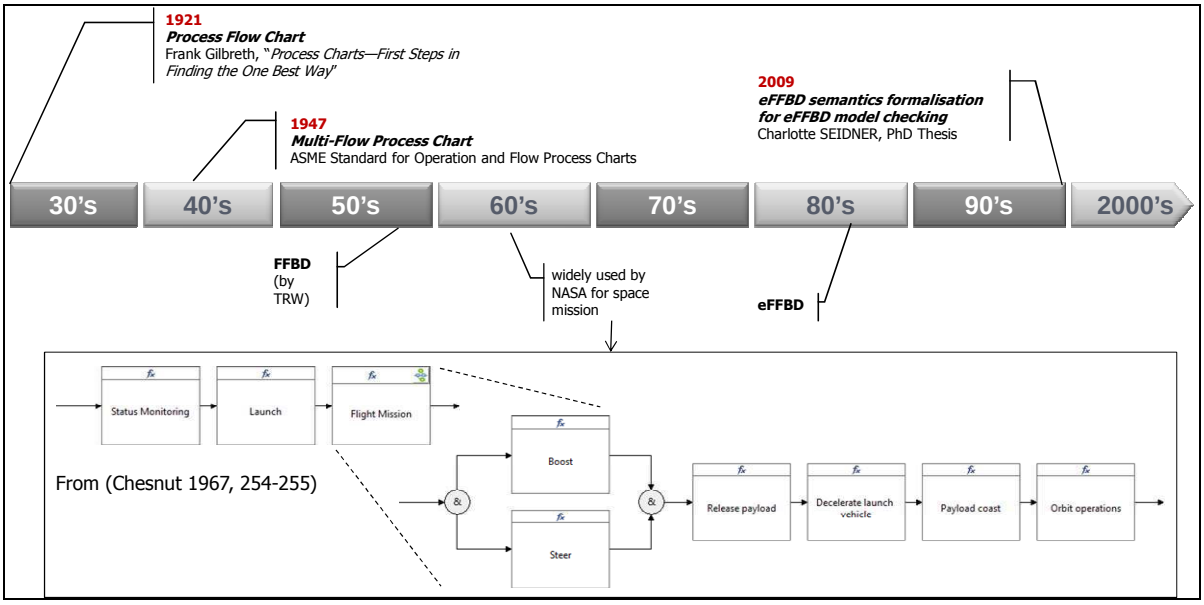


Figure 1: A short history of (e)FFBD

eFFBD as a readable, understandable, executable and non-ambiguous representation

As Albert Einstein once stated, "*Everything should be made as simple as possible, but not simpler*". We might add that the description of basic behaviors should also be as simple as possible but neither simpler, nor use "tricks". In this way, let us consider for instance the classical pattern of the access in mutual exclusion to a single resource, where two concurrent agents (of whatever nature they may be) are performing some activities or transformations and need at a certain time to use a single available resource before releasing it. If one agent wants to take the resource, while it is being used by the other agent, it has to wait and its related activity or function will be pending. This behavior is essential in concurrent processing modeling, from parallel computing systems to manufacturing or organizational systems. This behavior is rendered in eFFBD as illustrated in Figure 2.

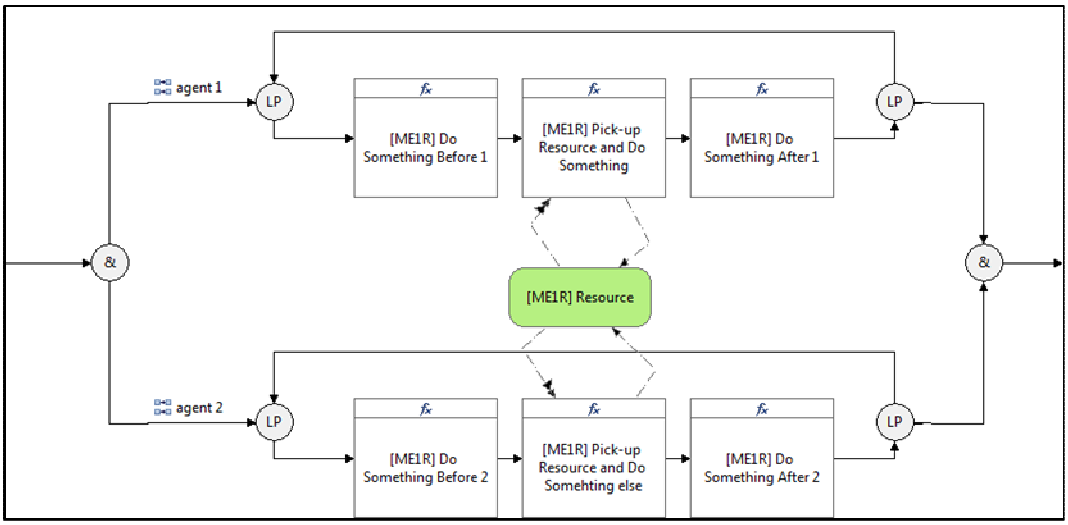


Figure 2: The mutual exclusion pattern in EFFBD

This model remains easily readable, understandable and non-ambiguous by a large variety of stakeholders. Such models are not restricted to a "pure" engineering community, but could be read, and thus validated, even by operational people. Moreover, a formal semantics

has been established in (Seidner *et al.* 2008). As a result, a simulation tool described in (Seidner 2009, Seidner *et al.* 2010) has been developed. So, eFFBD formalism is here considered expressive enough to permit a direct execution of the model *i.e.* it does not require any neither further model transformation nor additional information.

Such an execution is shown in Figure 3; one can then easily assess some performances such as a percentage of resource usage, the amount of time during which agents are waiting for the resource to be available and so on.

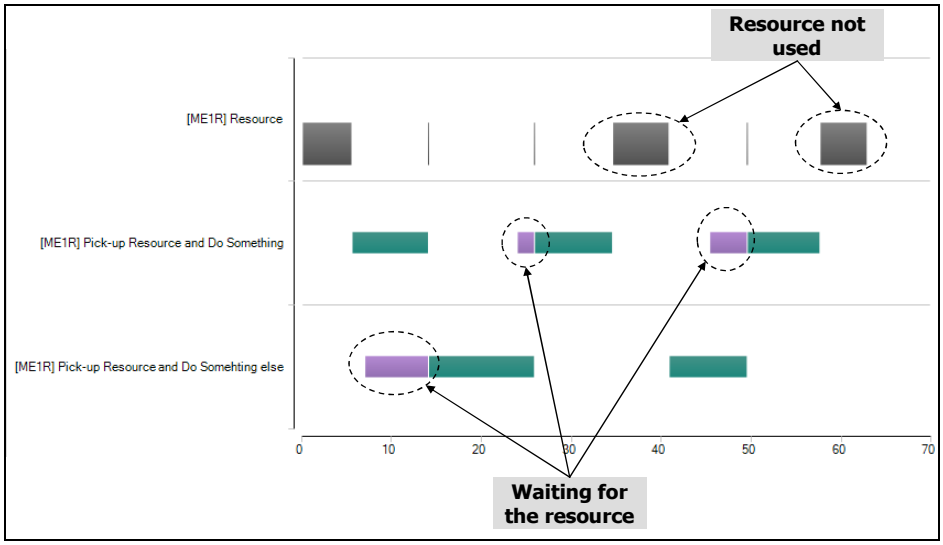


Figure 3: Execution trace of the mutual exclusion pattern

This basic functional (or process) model can also be described by more or less formal ones, such as Petri Nets and FCCS (Functional Charts for Control Systems), or by using activity diagrams from SysML, which are very close to Statecharts specifications.

Petri net (PN). This formalism and some extensions such as the Timed Petri Nets (TPN) or the Colored Petri Nets (CPN) have been developed precisely to describe and analyze such parallel, concurrent and communicating systems (Merlin 1974). Many theoretical results and tools can be used to assess such important properties as the presence of a deadlock or the liveness of a model. However, as developed in a following subsection, their abstract and mathematical nature often discourages their adoption by a large community. As an illustration, Figure 4 shows the PN (left) and CPN (right) models of mutual exclusion.

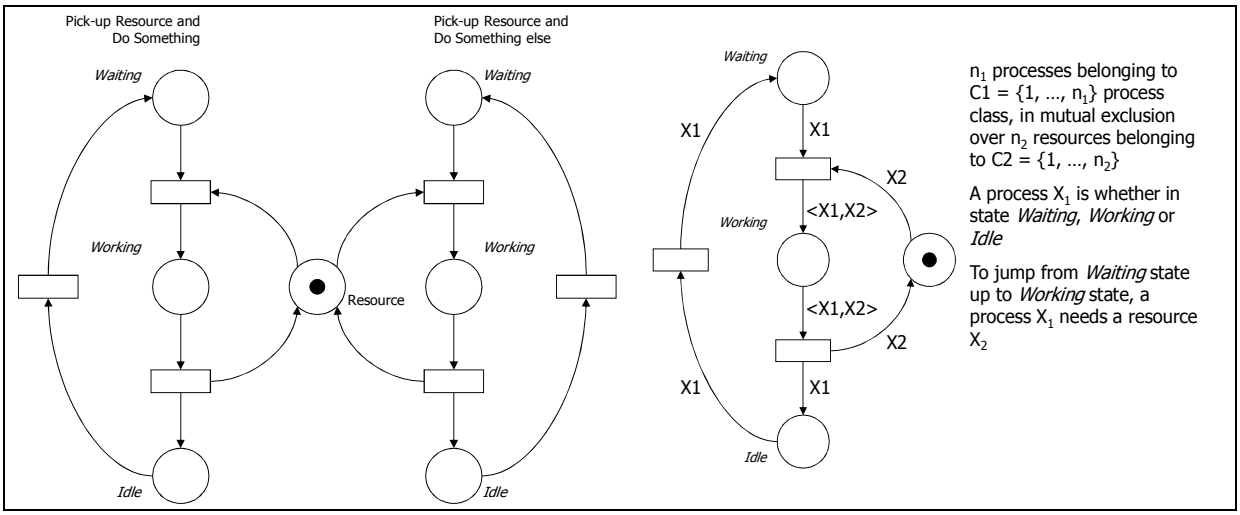


Figure 4: The mutual exclusion pattern in PN and CPN

SysML. It is possible to build the above-mentioned pattern using a SysML activity diagram (see Figure 5). However, we can note the following facts: activity diagrams (and their proposed artifacts) cannot capture all the desired behaviors. Informal extensions are needed to augment or enrich the knowledge described in the model. In this example, both a new stereotype, locally created, and informal textual notes are then required. As a consequence, no direct execution of the diagram is possible and extra work has to be performed (for instance adding non-standard pieces or artifacts to the model) before an execution can be carried out. Moreover, the large number of available modeling artifacts (about 30) increases the complexity of both writing and reading models, while not increasing the expressivity.

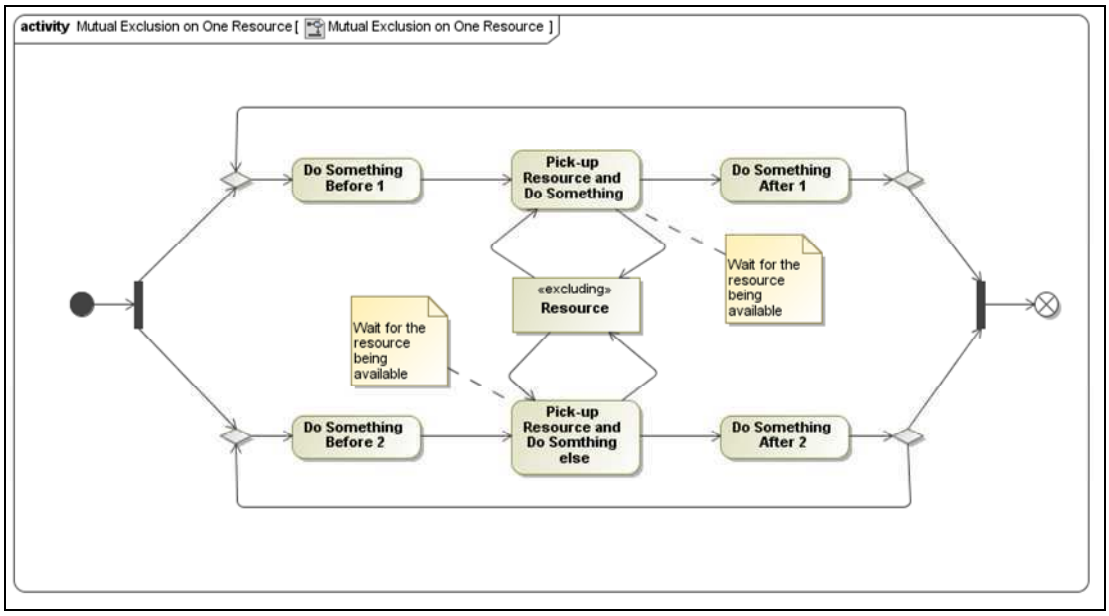


Figure 5: The mutual exclusion pattern in SysML

Finally, the fragmentation of modeling constructs into small pieces lead to potentially incorrect models, while eFFBD models are more easily qualified as "well-formed" as illustrated by Figure 6.

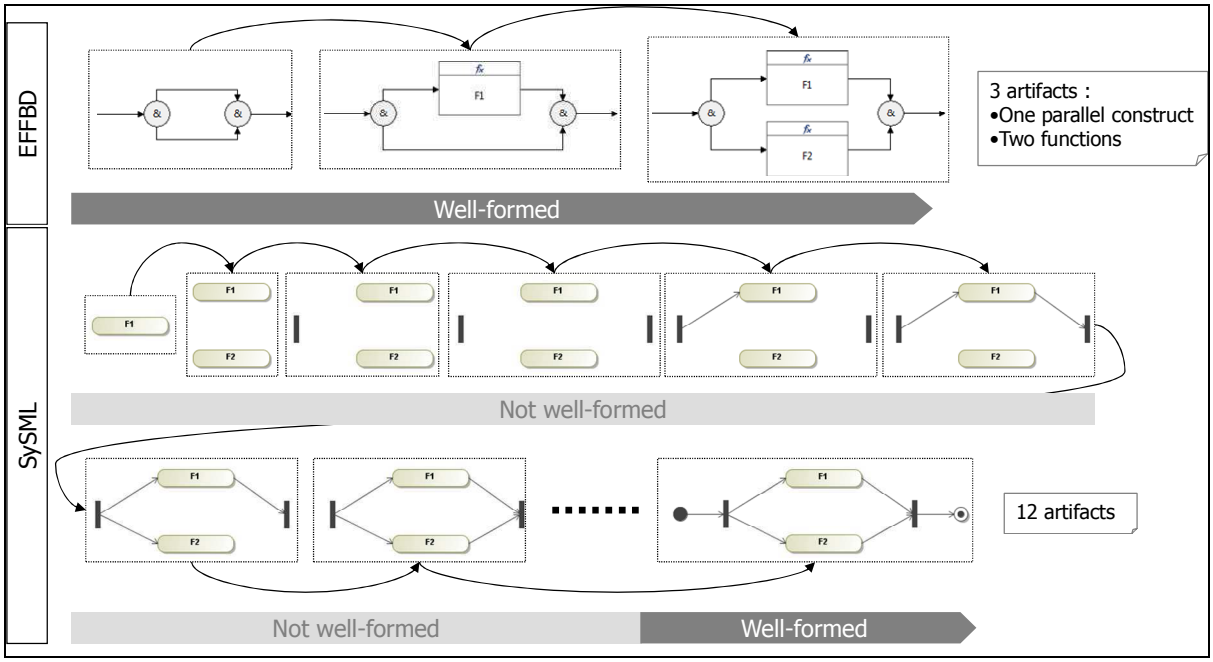


Figure 6: well-formed models comparison

At the opposite, SysML requires several steps to model complex behaviors: until the final step, the model cannot be considered as well-formed. Indeed SysML is more oriented towards building diagrams than helping to rapidly build models of complex behaviors easy to verify and express properties to be verified. As a result, the language is more suitable for capturing and sharing knowledge rather than allowing objective reasoning about the system being developed: the models are rather descriptive but not constructive enough.

We consider thus that eFFBD represents a relevant compromise for system designers in term of functional modeling. It is simple enough to be explained and shared by a wide variety of stakeholders. It remains sufficiently expressive to model process-oriented or transformation-oriented behaviors of the system. Lastly it is quite formal to allow assessment of some system properties leading to early verification and validation, objective comparison among alternatives, etc. Such a verification technique is described in the next subsection.

eFFBD models can be verified and simulated

As stated before, eFFBD models can be simulated, thus helping for example the designer to assess the system safety. Indeed, given today's trend to develop ever larger and more complex systems, verification actions have become a key element in the system design, all through its life-cycle, and performing either tests on the actual system or simulations on a behavioral model can help deciding whether the system behaves safely with regards to itself or its environment.

However, such an analysis cannot be exhaustive, even on "reasonably sized" systems, and carries the risk of missing potentially safety-critical situations. To overcome this limitation, the designer may use a formal method such as *model checking*, where the identified properties are first formally expressed, then confronted to a formal model of the system, using efficient algorithms and data structures. Previous works have shown that the inherent complexity of model checking can be overcome and efficiently used in a SE context (Seidner *et al.* 2010). Indeed, it is possible to:

- define a set of standard behavioral properties (such as "the execution always completes, and within a certain time bound" or "executing function *f* always triggers the execution of function *g* within a certain delay"), expressed in natural language;
- use the eFFBD formal semantics mentioned earlier to define the translation, without any information loss:
  - of any eFFBD model into an equivalent (timed) Petri Net;
  - of the above mentioned properties into equivalent logical formulas (written for instance using the Timed Computation Tree Logic or TCTL);
- use a model checker such as Romeo (Lime *et al.* 2009) to check the TCTL formula on the Petri net;
- translate the results back into high-level terms, that is in terms of functions and flows.

It should be noted that all these steps can be hidden from the user: the model checking technique can therefore contribute to the global dependability of the system as we have demonstrated that the eFFBD formalism can support the checking of complex safety and vivacity properties.

eFFBD limitations and required improvements

The current eFFBD modeling language has to overcome various challenges and is wanting a few improvements. These challenges arise essentially when performing the functional design of complex systems in a multidisciplinary environment:

- even if all engineering disciplines deal with functions, they represent the functional architecture with different tools, techniques and paradigms. Some are flow-oriented or

flow-driven (see for instance the use of block diagrams in signal processing or control theory) while other are event-oriented or state-oriented (such as sequential function charts or FCCS, Statecharts, Petri Nets, etc.);

- while the modeling of the system dynamics can, at a high level, be described efficiently by a discrete model, at one point a continuous modeling might be mandatory. Depending on the level of focus or interest about the system, it could be seen either continuous, discrete, or hybrid;
- the system itself could be recursively broken down into subsystems, thus leading to a multilevel functional architecture.

As stated before, eFFBD language can be used to capture various functional models into a single hierarchical model. The following figures shows the equivalent representation of a block diagram used in signal processing or control theory (Fig. 7) into an eFFBD (Fig. 8).

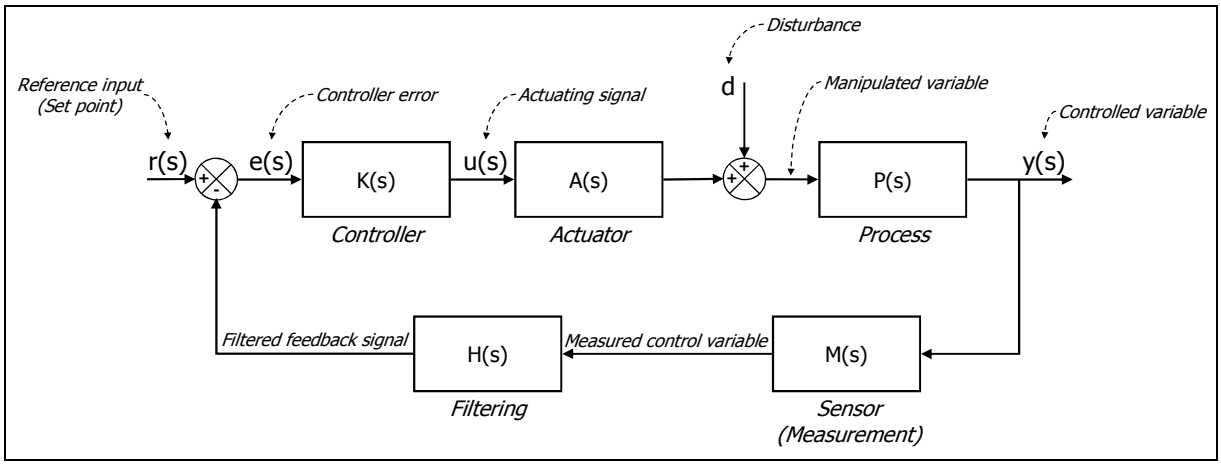


Figure 7: Classical control loop

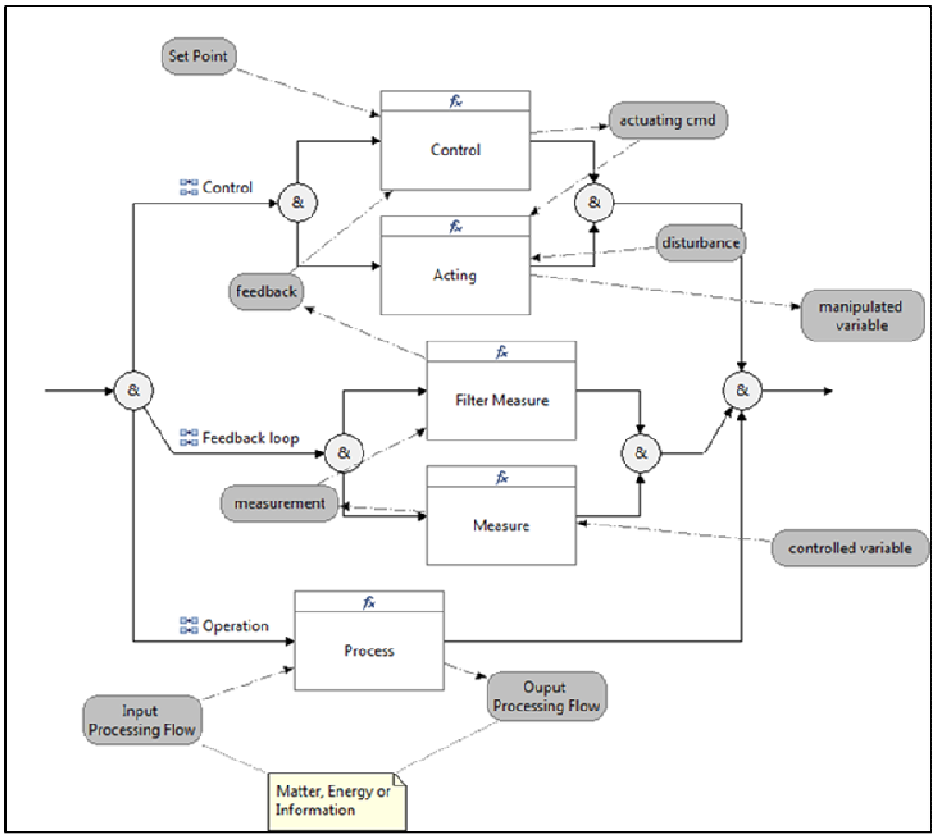


Figure 8: Translation of the classical control loop

It is indeed true that we lose the readability of the model here (the control loop itself does not appear anymore). However, the eFFBD can be simulated, and property checking techniques mentioned before can be applied.

Expectations, propositions and benefits for xFFBD

Considering the current capabilities of eFFBD shown in previous sections and interests for the context of a "functional paradigm driven system design process", the goal is now to overcome the limitations illustrated above and to propose and argue in what sense and how the eFFBD modeling language can evolve into xFFBD. From our point of view, such modeling language has to bring the following foreseen expectations to reach perceptible benefits.

Paradigm

An xFFBD model does not "throw away" other functional models; it just provides the "cement". Unlike the use of SysML, it should not require all engineers to perform deep changes in their functional modeling knowledge and know-how. The benefit is methodological considering the respect of established ways of thinking, various best practices (pattern driven approaches) and current level of experience / autonomy of systems designers and discipline oriented designers.

Towards "self contained" models

Even if the xFFBD modeling language misses some modeling aspects (considering for example the control theory domain, it is not possible to model the transfer functions described by using a Laplace or Z-transform notation), xFFBD has to constitute a suitable "container" for the various pieces required. In the same way, model transformation techniques are mature enough to envisage a quite complete and automatic transformation of these pieces into xFFBD or reverse from xFFBD to tierce modeling languages in order to be able to improve the checking, assessment and documentation of models. More generally, several "pieces" of functional models, obtained using various techniques, can thus be re-assembled into a whole containing model based, which is not possible directly on the present eFFBD language.

However, it should be kept in mind that, depending the envisaged usage scope of xFFBD (functional architecture, refined functional architecture ready for mapping to organic architecture, etc.), all the required model transformations (to or from xFFBD) must improve interoperability of models in an MBSE context. Indeed, it will be preferred to use specialized languages for assuming modeling activities of some points of view.

The benefits consist here to gain in model interoperability and then to facilitate the future developments and integration in existing systems engineering frameworks of modeling tools supporting xFFBD.

Semantic: formal and operational semantics

As illustrated by Figure 9, it is necessary to be capable to:

Build hybrid simulations of the system behavior hence not requiring model transformation or rewriting, inducing then a possible loss of semantic. In this sense and even if an eFFBD model allows describing the sequencing of functions or activities, there are some shortcomings for a full expressivity of eFFBD in terms of temporal sequencing model *i.e.* it is necessary to enrich its operational semantics.

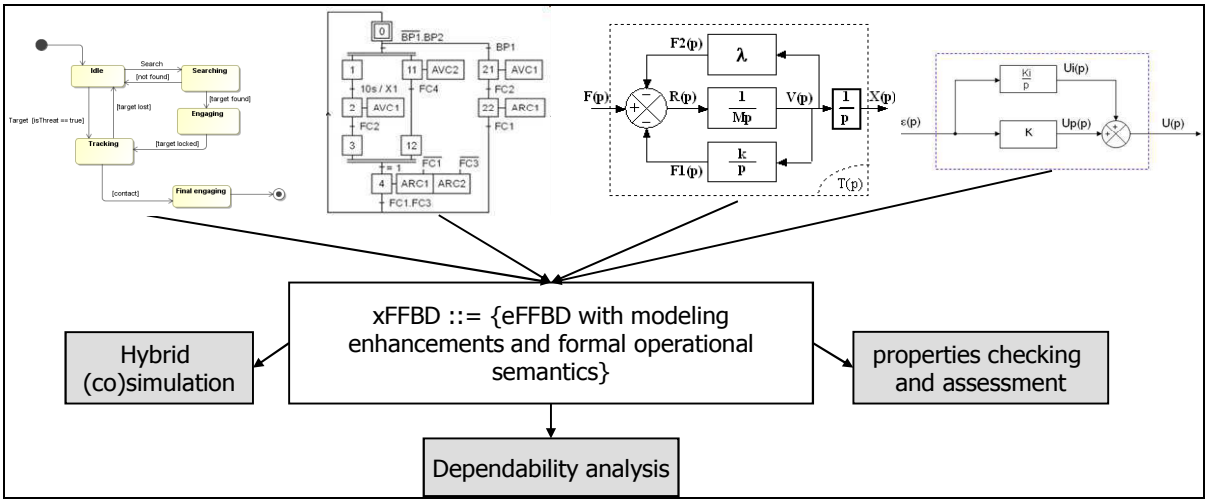


Figure 9: xFFBD modeling language expectations

Tasks are a good example of source for this enrichment. Task models are widely used by HMI community as a way to model the behavior of end-users through their interaction with the system. Tasks are activities that should be performed by the final user of the system to achieve a particular goal. GOMS (Goals, Operators, Methods, Selection Rules) (Card *et al.* 1986), UAN (User Action Notation) (Hamilton 1996), TKS (Task Knowledge Structures) (Hamilton 1996), or CTT (ConcurTaskTrees) (Mori *et al.* 2002) are best representatives of such task modeling languages. In a task model, various temporal relationships can be defined between tasks denoted T1 and T2 in the next through dedicated operators. A total of eight operators can be used (Paternò 1999):

- choice operator []: T1 [] T2 means that one of T1 and T2 will happen;
- order independence operator |=|: T1 |=| T2 means that T1 and T2 will happen in any order;
- concurrent operator |||: T1 ||| T2 means that T1 and T2 will happen concurrently (the operator |[]| is used to express information exchange between the tasks)
- disabling operator [>: T1 [> T2 means that T2 interrupts T1 (which will not be resumed);
- suspend/resume operator |>: T1 |> T2 means that T2 suspends T1, but T1 resumes once T2 has finished;
- enabling operator >>: T1 >> T2 means that T2 happens after T1 is finished ([]>> is used to express information exchange between the tasks);
- iterative operator *: T1* means task T1 happens repeatedly;
- optional operator [ ]: [T1] means task T1 might happen or not.

Although something similar does exist with the “kill branch” which permits to interrupt all other parallel functions when a function is finished, current eFFBD do not have the same expressiveness of interaction. Indeed, among these eight operators, two are missing in eFFBD: *Disabling* and *Suspend/Result*. These operators must be formally defined through an adequate formal semantic.

Extending the eFFBD language with the operators introduced above will thus allow eFFBD to fully encompass task model expressiveness, allow model transformation between task models and eFFBD models, and at least but not last provide a common unified framework for both system engineering and HMI communities.

In the same way, these extensions could also hold process business or enterprise process modeling. The envisaged extension to eFFBD could make it compatible with the PSL ontology (ISO 2004, Schelnoff *et al.* 1999).

Extend and complement eFFBD flow semantic. eFFBD have a unique and discrete way to model flows. According to the semantic definition, contents are all consumed when a function is activated and are then produced instantly when the execution of the function end. Implicitly, an eFFBD function behavior can be described by a simple Finite States Machine, as illustrated by Figure 10.

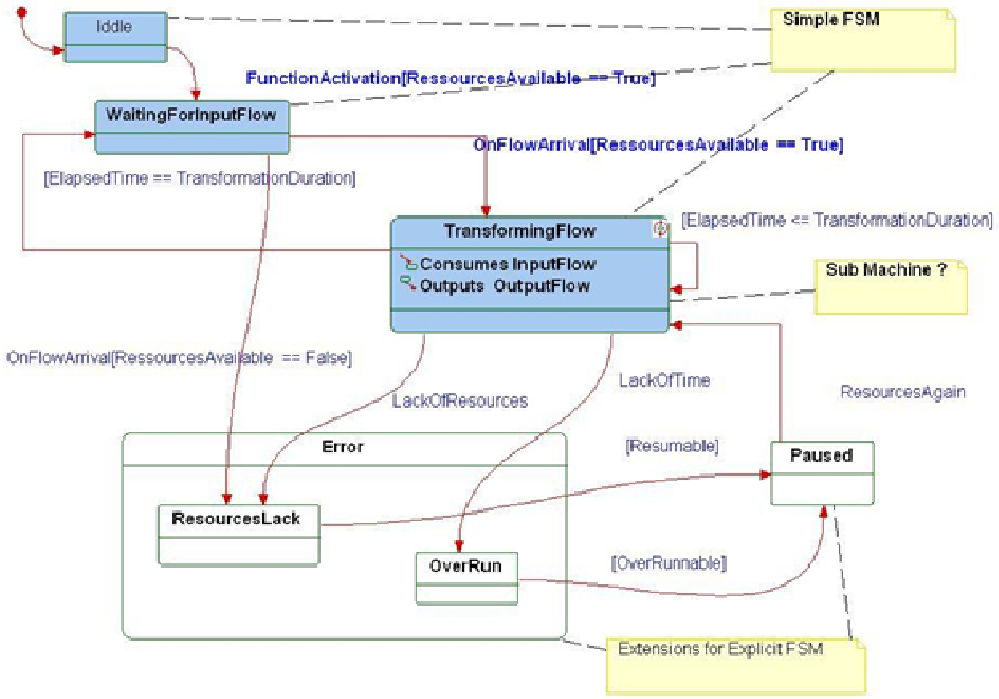


Figure 10: A model of a function and its flows

When modeling real cases, it is often needed to describe a continuous consummation and/or production of flows. Consequently, flows semantics must be clarified, classified and extended:

- there should be a classification between flows of discrete events, and streams (infinite, temporary etc.); moreover, it should be possible to distinguish between discrete or continuous variables representing the values of the flow over the time, itself discrete or continuous, leading to consider different types of systems behavior;
- it should be possible to describe different flow types: matter, energy, or information, and possibly extended to resource (something needed for the function but "not consumed", only "used"). Some constraint rules might be added in order to restrict the usage of such flows to "realistic" cases. E.g. an energy or a matter can be transformed, but once they are used, they are "lost" for further re-use, while information can be used several times without alteration (you can read an email, store it locally on your computer, and in the same time forward it to someone else);
- as a proposal, MOC (Models Of Computation) such as KPN (implementing classical FIFO behaviors) may be used, especially for streams (continuous consumption of input, delivery at output, time delay, sampling, jitter and coding/dynamic constraints, transformation, etc.). In addition, a stream meta model, capturing specifics their features, and standard streams transformation (coding, decoding, delaying, filtering, resampling etc.) may be added for flows characterization;
- basic algebra should be added, especially for information flows, in order, to add/group, subtract/split flows, to distinguish between content and container (e.g. a message can contain and convey information, a request holds parameters …).

Extend and complement the semantic of a function itself. In a same way, very few semantics is associated to function. In one way, functions are seen as black-box which consume and produce data and or event. On the other way, they are seen as white box, composed of sub functions modeled with eFFBD.

The formal description of the final flow transformation (as algebraic equations, transfer functions…) performed by leaves functions at the lower level of decomposition is out of the xFFBD envisaged semantic. However, xFFBD should provide some "extension points" allowing the addition of such semantic. We could quote some envisaged interface to specialized language with a huge benefit in terms of model analysis:

- notations defining the transformation itself such as Z-transform, Laplace transform…
- languages modeling the transformation function itself, and allowing execution of it such as MatLab Simulink, Modelica…
- languages modeling the dysfunctional aspect, allowing dependability analysis to take place (Altarica, for instance).

That means that xFFBD scope is not the whole modeling artifacts of Systems Engineering, firstly, that Systems Modeling may use with benefit multipoint of views models and hybrid modeling, supported by a tool chain. So, xFFBD needs interfacing capabilities with other languages. It is thus required, as stated before, to assume at least conceptual interoperability *i.e.* conceptual and formal compatibility of xFFBD with the other cited modeling languages such as Modelica or Altarica. Indeed, this allows disposing of existing and reliable proof mechanisms and tools.

Check model properties at the various levels. As stated in MIT system design lectures (MIT-ESD) "Architecting is the ***deliberate*** manipulation of structure to achieve desired system ***behavior*** and ***properties***".

Behavior verification relies on functional model simulation yet supported by eFFBD related tools (Seidner2009). Foreseen extensions are introduced in the *Build Hybrid Simulations* paragraph. To be fully compliant to MIT directives, it remains to provide to system designer the capability to define, to verify and to propagate properties.

By defining properties semantics, we can first consider the link to develop between xFFBD and formal properties modeling languages such as Temporal Logic (TCTL, CTL…), Property Specification Language (PSL) or Unified Properties Specification Language (UPSL) (Chapurlat *et al.* 2006, Aloui *et al.* 2008, Chapurlat *et al.* 2009). For instance, UPSL is based on the property concept named CREI used to formalize and check modeling requirements, the part of stakeholder requirements that can be formally checked in the models and various laws (physical, chemical, electrical…) which must always be applied but can be easily forgotten. An extension of UPSL called UPSL-SE is designed to complement the methodological and technical tool box supporting the verification process in SE context by considering formal checking techniques. However, the various definitions and semantics proposed for defining the concepts and relations used in the available SE languages do not allow or facilitate the use of such techniques.

The waited benefits for xFFBD are then to gain formal verification capabilities which requires that:

- the models can be translated without loss, ambiguities, and using automated methods xFFBD models into more formal ones on which formal proof techniques can be applied. In other words, xFFBD must be interoperable with formal checking tools, such as those presented in (Yahoda 2003);
- the requirements may be formally described (Micouin 2008, Chapurlat *et al.* 2012) and checked *i.e.* requirements modeling languages are themselves interoperable with xFFBD.

The next expected step would be the ability to propagate assessment of such properties or characteristics up or down into the levels of functional breakdown, in a way similar to the Critical Parameter Management of the DFFS (*Design for Six Sigma*) approach, implementing propagation algorithm that would take advantage of the functional to propagate properties.

Improve tools that will support xFFBD edition and simulation. Nowadays, there are a certain number of system engineering workbenches used by system designers in order to support their design activities and to manage in a consistent way engineering information produced by these activities. However such tools do not offer an efficient digital environment. Indeed, there is a need to offer a more agile environment that will allow a system designer team to better cooperate through system models sharing, and to improve capabilities to "play" on efficient views of the system being designed, that is to say to create and to bring more value-added. In this context, following ideas need to be explored:

- use of multi-touch surfaces to increase system engineers interaction;
- ontological (or semantic) oriented browsing in the system model through the exploitation of system engineering meta-model;
- visualization of system information or system model in alternative ways: while traditional views of the system information encompass the hierarchical aspects of things or their intrinsic properties, a new way to be explored is to focus on their relationships. Remember, *"A city is not a tree"*! (Alexander 1996);
- direct manipulation of system composite views to support system engineering activities such as allocation of functions to components, allocation of flows to physical links, …

Conclusion and perspectives

eFFBD represents a strong foundation for functional modeling in systems engineering. This position paper proposes and argues various improvements in order to make emerge or at least to act as a preliminary roadmap to the xFFBD modeling language. This language has to provide first a formal but understandable notation based on a set of required concepts and operators which remain absents for the moment from eFFBD. Second, xFFBD need the definition of a new and enriched formal operational semantics to become an expressive and executable functional modeling language. Third, it must be as interoperable as possible with other modeling languages handled by various tools so they can check properties, evaluate and compare various criteria (such as performance or safety) on functional models.

So, the goal here is not to contrast the functional paradigm for which xFFBD would be an appreciable contribution with the object paradigm as implemented for example by SysML. Through their own virtues, qualities and limitations, each of these paradigms is actively contributing to the development and use of systems engineering. They must co-exist and complement each other. As such, xFFBD is fully in the functional paradigm and this paper allows identifying roughly the required evolutions of eFFBD modeling language. At this moment, the work to be done first consists in formalizing the proposed formal extension to eFFBD, such a team is now ready to start!

References

Alexander, C., 1996, A city is not a tree, *Design* magazine, Council of Industrial Design.

Aloui S., Chapurlat V. 2008. "A System Modeling and Analysis Framework for Risk Analysis". In Socio-technical Systems, The best of France: Forum Académique et RobAFIS, INCOSE INSIGHT, 11(3)

Card, S.K., Moran, T.P., Newell, A. 1986. *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates.

Chapurlat V., Aloui S. 2006. “How to Detect Risks with a Formal Approach?” From Property Specification, Simulation Verification and Validation of Enterprise Information Systems (MSVVEIS'06), The 4th International Workshop on Modelling.

Chapurlat V., Roque M. 2009. “Interoperability Constraints and Requirements formal Modelling and Checking Framework”. In International Conference Advances in Production Management Systems (APMS'09).

Chapurlat V., Daclin N. 2012, System interoperability: definition and proposition of interface model in MBSE Context, to appear in INCOM 2012, IFAC's triennal symposium Information Control problems in Manufacturing, May 23-25, Bucharest, Romania

Chesnut, H. 1967. *System Engineering Methods*. Wiley & Sons.

FAA. 2006. NAS System Engineering Manual, version 3.1

Hamilton, F. 1996. “Predictive evaluation using task knowledge structures”, in Conference companion on Human factors in computing systems: common ground. ACM. 261– 262.

ISO. 2004. 18629:2004, Industrial automation systems and integration - PSL

Lime, D., Roux, O.H., Seidner, C., Traonouez, L.M. 2009. “Roméo: a Parametric Model-Checker for Petri Nets with Stopwatches”. In 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. LCNS (5505): 54–57.

Merlin, P.M. 1974. A Study of Recoverability of Computing Systems. Ph.D. thesis.

Micouin, P., 2008, Toward a Property Based Requirements Theory: System Requirements Structured as a Semilattice, in System Engineering, Volume 11, Issue 3: 235-245

Mori, G., Paternò, F., Santoro, C. 2002. “CTTE: Support for Developing and Analyzing Task Models for Interactive System Design”. IEEE Trans. on Software Engineering, 28(9).

NASA. 1967. NASA/SP-610S: *Systems Engineering Handbook.*

—. 2007. NASA/SP-2007-6105 Rev. 1: *Systems Engineering Handbook.*

Paternò, F. 1999. *Model-Based Design and Evaluation of Interactive Applications*, Springer-Verlag.

Schlenoff, C., Gruninger, M., Ciocoiu, M., Lee, J. 1999. “The Essence of the Process Specification Language”. Special Issue on Modeling and Simulation of Manufacturing Systems in the Transactions of the Society for Computer Simulation.

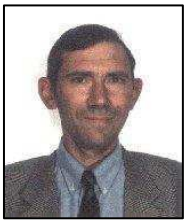
Seidner, C., Roux, O.H. 2008. “Formal methods for Systems Engineering behavior models.” IEEE Transactions on Industrial Informatics 4(4): 280–291.

Seidner, C. 2009. Vérification des eFFBD : Model-checking en Ingénierie Système. Ph.D. thesis, University of Nantes [in French].

Seidner, C., Lerat, J.P., Roux, O.H. 2010. “Simulation and Verification of [Dys]functional Behavior Models: Model Checking for SE”, 20th International Symposium of the INCOSE.

Yahoda. 2003. Formal verification tools overview web site (http://anna.fi.muni.cz/yahoda/)

Biographies



Bruno AIZIER received an engineer's degree in Electronics and Signal Processing from the ICPI-CPE (*Institut de Chimie et de Physique Industrielles*) engineering school of Lyon. As a research engineer at ENSTA Bretagne (Superior National School of Advanced Techniques of Brittany) since 2007, he is contributing to research works on Systems of systems modeling with NAF architecture framework (in partnership with the French Ministry of Defense), meta models engineering and hybrid modeling. His expertise on the engineering of software-intensive embedded systems is based on a course of about 25 years with major companies such as Alcatel, France Telecom R&D, Cap Gemini, DCN, Thales Underwater Systems, or Siemens VDO.

Vincent CHAPURLAT is a full professor at the École des Mines d'Alès, head of the ISOE research team at the LGI2P. After a PhD in Computer and Control Theory from the University of Montpellier II in 1994, he holds an HDR (accreditation to supervise PhD students). His research aims to develop and formalize concepts and tools for building and formally verifying complex systems models, applied in Enterprise Modeling and Systems Engineering domains. He is a member of the AFIS (French chapter of the INCOSE), of the 5.3 Technical Committee at the IFAC Board (on enterprise networking) and of the Enterprise interoperability WG at the IFIP Board.

Stéphanie LIZY-DESTREZ is an associate professor in System Engineering (SE) and head of Specialized Master in System Engineering at ISAE *(Institut Supérieur de l'Aéronautique et de l'Espace)* SUPAERO with more than 14 years of industrial experience in space systems engineering from design to operations. She was able to deploy inter-disciplinary approach for Earth Observation, Telecommunications or Human Spaceflight missions. In July 2009, she joined ISAE and the associated Department of Mathematics, Computer Science and Control Theory (DMIA). Her research aims at formalizing System Engineering processes through Human Space flights applications. She is a member of the AFIS (French chapter of the INCOSE).

Daniel PRUN is an associate professor in System Engineering (SE) at ENAC (French Civil Aviation University) with more than 15 years of industrial experience in the field of SE. In 1997 he obtained his PhD in Computer Science from the Pierre and Marie Curie University. He left academic research, became an SE consultant to several customers in various sectors (defense, air traffic control, aeronautic, railway, medical…) for their SE activities (specification, design, system integration, verification and validation). In 2009, he has joined ENAC school and the Interactive Computing Laboratory (LII) with the objective to develop SE teaching and research activities on interaction. He is a member of the INCOSE and the AFIS (French chapter of the INCOSE). He also participates at the BKCASE/GRCSE project.

Charlotte SEIDNER has obtained her master's degree in Engineering (2005) and in Control and Computer Theory (2006) from the École Centrale de Nantes. In 2009, she defended her PhD thesis, entitled *Verification of eFFBDs: model checking in system engineering* and carried out in collaboration with both the IRCCyN (the Research Institute on Communication and Cybernetics in Nantes) and Sodius, an SME in Nantes, highly involved in Systems Engineering. Since 2010 she has been an associate professor at the University of Nantes and carries out her research activities at the IRCCyN lab, on formal methods applied to high-level problems.

Jean-Luc WIPPLER obtained his master's degree at SUPELEC (École Supérieure d'Électricité) in signal processing. For the last 20 years he has worked mainly in the sectors of space, defense, and air-traffic control. As a system architect, he has participated in various projects in the field of Earth Observation or Satellite Navigation space based systems, and also contributed to systems engineering in the air-traffic control, medical and automobile fields. In addition to his job as a senior system architect, he devotes his time to teaching systems engineering at Master of Science level (ISAE-SUPAERO, ENAC) and continuing education.