System of Systems Architecting: A Behavioural and Properties Based Approach for SoS "–ilities" Modelling and Analysis

Vincent Chapurlat, Nicolas Daclin

System of Systems Architecting: A Behavioural and Properties based Approach for SoS "–ilities" Modelling and Analysis

Vincent Chapurlat, Nicolas Daclin

LGI2P Ecole des mines d'Alès, 69 rue G. Besse, 30035 Nîmes Cedex 1, France
{Vincent.Chapurlat, Nicolas.Daclin}@mines-ales.fr

Abstract. Architecting a System of Systems (SoS) is a complex task. Capabilities of heterogeneous and interactive sub-systems are composed to fulfil a mission, while preserving, as possible, the autonomy, independence, geographic distribution… of sub-systems and to face up efficiently while remaining as resilient as possible to disturbances and emergent phenomenon. The "-ilities" are relevant non-functional abilities (*e.g.* robustness, resilience, flexibility, adaptability, survivability, interoperability…) for guiding SoS architects and managers to choose and interface sub-systems. The goal is to become able to increase or decrease the value of these "–ilities" thanks to their interest for the SoS mission. The here presented work aims to support resilient SoS design and, in particular, their architecting by proposing a formalised model of property allowing to define and describe an "–ility" and a behavioural modelling approach to evaluate it.

Keywords: System of Systems, "–ilities", Non-functional Properties, Resilience, SoS behavioural modelling, Dependencies, Formalisation

1 Introduction

A System of Systems (SoS) is composed of (in most cases, existing) heterogeneous sub systems chosen for their capabilities, assembled and interfaced to interact during a time-frame and to provide capabilities to achieve a mission that each sub system cannot fulfil alone [1]. First, some characteristics of sub systems must be preserved: operational and managerial independence, evolutionary development, geographic distribution, and connectivity. Second, requested interactions induce emergent phenomenon (new properties and behaviours with more or less predictable and unwanted effects) at the SoS level that can favour or affect the achievement of its objectives and mission. Third, it is now recognized, for SoS, the relevance of specific properties called "-ilities". An "-ility" [1] is an "*ability to respond to changes, both*

[1] An "–ility" (plural "-ilities") [5] is "*a developmental, operational, and support requirements a program must address (e.g. availability, maintainability, vulnerability, reliability, supportability"* which are generally non-functional requirements.

foreseeable and unforeseeable" focusing on "*how the SoS should be and not what it should do*" [2]. For a SoS, "-ilities" differ from authors, but essentials ones remain: robustness, resilience, flexibility, adaptability, survivability, interoperability, sustainability, reliability, availability, maintainability and safety; each "-ility" can be strongly dependant or influenced by various causes considering SoS context, evolution period of its life cycle and emerging phenomenon. So, architecting a SoS implies to study these "-ilities" and how to increase or decrease their values thanks to their interest for the SoS purpose. Particularly, SoS has to face up, efficiently and accordingly to its mission, to various dynamic contexts in which disturbances can occur due to disruptive actions (from SoS environment or internal sub-systems failures [2]) or to new and enabled technological evolutions i.e., it has to maximise its resilience. The here presented work aims to help and support resilient SoS design, particularly architecting step of the design. Section 2 introduces two approaches for resilient SoS architecting that can be combined with a behavioural modelling approach named OMAG, briefly introduced. In section 3, retained SoS "–ilities" and their interdependence are discussed and formalised. Section 4 illustrates the proposed contributions before concluding.

2 Resilient SoS Architecting

Architecting a resilient SoS means to consider various interdependent "-ilities" and new design approaches. [4] insists on "*designing for resilience is about creating a system that can bounce back from something no one ever thought would happen*". Existing approaches are strongly based on system paradigm and follows Systems Engineering principles [24]. In the next we discuss about requested "–ilities" and two approaches.

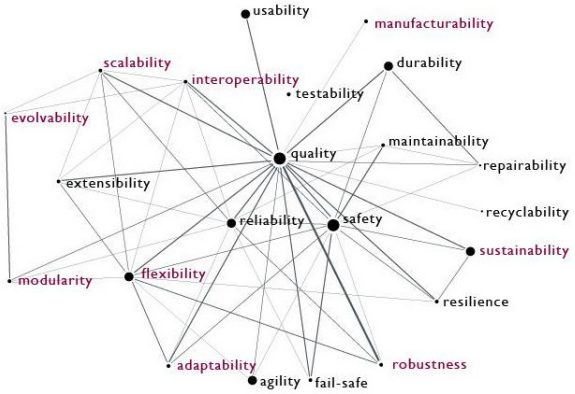

Fig. 1: Overview of "-ilities" dependencies [7]

Various "–ilities" definitions can be found [6] [7] [8]. The concept of "–ility" can be used to characterize both SoS and each of its sub-systems. As illustrated in **Fig. 1**, it is admitted that an "–ility" dynamically vary and is dependant or influenced all along SoS life cycle depending on 1) its characteristics (properties, behaviours and "–

ilities"), 2) the characteristics (properties, behaviours, and –ilities) of its sub-systems, and 3) emergent properties and behaviours from sub-systems' interactions.

For instance DoD [6] presents resilience as dependant from robustness, flexibility and protection "–ilities". Following the same dependence principles, robustness depends from reliability, availability, survivability and maintainability. In the same way, the DSTA framework [2] considers two levels of SoS "–ilities": Key SoS "–ilities" (robustness and evolvability) and Key Enabling "–ilities" (flexibility (operational and design) and interoperability) that have been enriched in **Fig. 2** by decomposing interoperability definition.

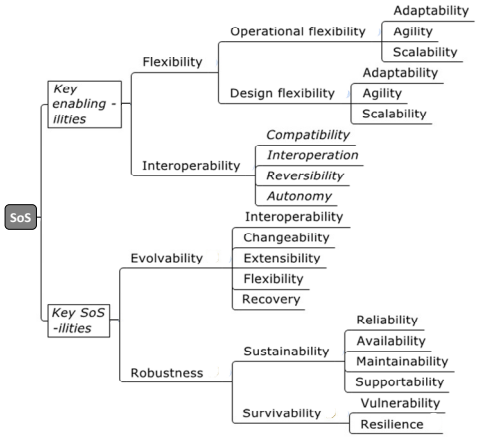

Fig. 2: Key SoS "–ilities" and Enabling "–ilities" (inspired from DSTA Framework [2])

In both cases, whatever the definition for "-ility" and its interdependence with other abilities of the SoS, an "–ility" remains difficult to conceptualize from a unified manner (that can be even not expected), to handle and to use in confidence in architecting process [10] [11] [12].

Concerning SoS architecting approaches, first SAI (SoS Architecting with Ilities) [8] focuses on value sustainment of both functional and non-functional requirements. Fig. **3** shows the essential steps of SAI, especially the steps 4 (Generate initial architecture alternatives) and 6 (Evaluate potential alternatives) in which a behavioural model of the SoS architecture is built and executed for evaluating the chosen "-ilities" defined in step 3. Second, DSTA framework [2] is a methodological framework for supporting SoS architecting and a reference framework for choosing the most relevant "–ilities" allowing practitioners and manager to drive and manage efficiently SoS architecting. In both approaches, the SoS architecting phase (i.e., defining strategy, requested operations and scenarios, and specifying architectural alternatives according to a more or less high level of abstraction) is apart from SoS designing phase (for instance, choice and interfacing of sub-systems, validating scenarios). In design phase, several solutions are proposed in [10] to improve various "–ilities" to gain resilience e.g. employing redundancy, reducing complexity or improving reparability. Alternatives solutions must be modelled and compared thanks to an expected value (quantitative or qualitative) level for each "-ility".

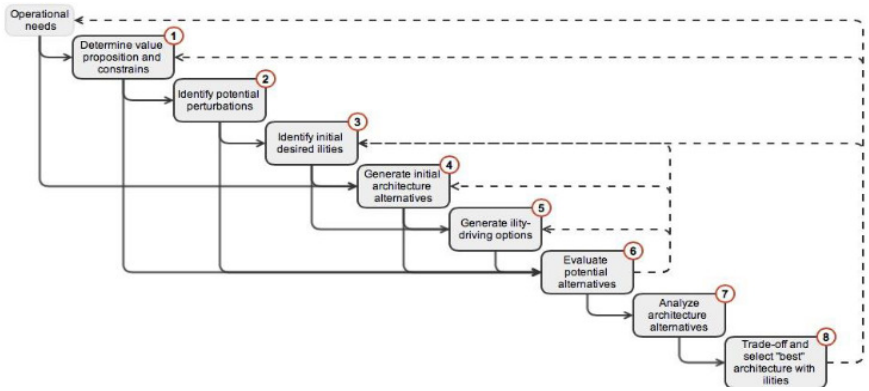

Fig. 3: SAI principles approach [8]

To this purpose, the behavioural modelling method OMAG can support both SAI steps and DSTA framework to describe simply and to link the behaviours of a SoS and of its sub-systems. This approach is enriched by the formalisation of a set of "–ilities" by properties allowing their checking or evaluation.

3 SoS Behavioural Modelling: OMAG Method

OMAG (Operating Modes Analysis Guide) [13] is a behavioural and functional modelling and analysis method allowing:

- System architect to select the **Operating Modes** [14] that characterize a system all along its life cycle.
- To determine gradually the expected **Properties** (functional and non-functional i.e. "–ilities") and **Parameters** of the system when evolving into a mode.
- To determine gradually and to model various **Operational Scenarios**. It is question of a functional model describing the dynamic of a system (what are the expected functions or activities and how they are chained and synchronised?) in a given operating mode. Various modelling languages can be used e.g. BPMN, eFFBD, or use case diagram. In OMAG approach, each operational scenario describes a part of the whole expected functional architecture of the system.

OMAG is based on a graphical grid shown in Appendix, detailed and illustrated in [13]. Briefly, the OMAG principles are summarized in **Fig. 4**. Considering a system, OMAG requires defining first systems' attributes and to gather them into a *ParametersAndPropertiesSet*. An attribute is modelled as a *Parameter*, a valued and typed data describing time (temporal aspect e.g. maximum delays for reaction), shape (structure e.g. geometric constraints) or space (situation e.g. non-functional expectation) characteristics of any element from the system or its environment.

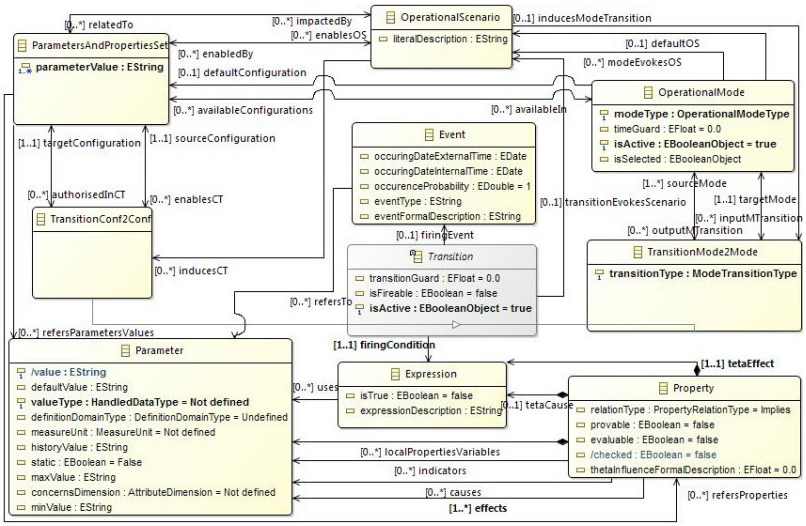

Fig. 4: OMAG main elements meta model (simplified view)

The *ParametersAndPropertiesSet* is set up and grows up gradually all along the architecting process. The set of generic Operating Modes and Transitions is then formalised by the grid and the architect must select those relevant for the system and to be studied. Each transition is characterized by a couple (condition/event) allowing system evolving from an operating to the next. For each mode and transition, architect has to model what are the expected operational scenarios allowed in the studied Mode or induced / expected to fire the transition. **Fig. 4** shows the main elements of a grid OMAG allowing to describe and link operating modes, operational scenarios associated to each mode, parameters and properties of a system. The result is a behavioural model of the studied system conforms to the underlying mathematical formalism (inspired from Finite State Machine model [15]). Each operating mode is modelled as a state and each mode transition as a state transition. An operational semantic of OMAG grid is given in [13] specifying formally with no ambiguity the interpretation and execution rules of an OMAG grid. This allows then defining and implementing OMAG grid simulation and proof mechanisms not detailed here. These mechanisms allow to model and verify properties (modelling properties as non-functional properties), to approximate "–ilities" values, or to highlight rapidly some disturbing or awkward situations the SoS has to face. This can be helpful for SoS architect who intends to test and compare various SoS architectural alternatives as requested, for instance, in SAI approach.

4 SoS –ilities and –ilities Dependence Modelling: Property Concept

A *'property'* is defined in [18] as "*an entity that can be predicated of a thing or, in*

other words, attributed to it (also called 'attribute', 'quality', 'feature', 'characteristic' or 'type')". [19] introduced a Property-Based Requirements (PBR) theory based on semi-lattices on which a property formalizes a portion of a well-formed requirement (denoted *wf-requirement*) and be owned by a system. In this case, a property is interpreted as a variable to be quantified or qualified to evaluate the relevance and adequacy of a system solution; such an evaluation is to be carried out by using a model of the system here, an OMAG grid. [20] postulates that a property is any descriptor of an artefact (i.e., a modelling artefact of the system). The property is represented as a mathematical function defined for this artefact, associating a (set of) value(s) allowing evaluating the solution described by this artefact. [21] and [22] define a property as "*the formal statement of an expectation by using a formal language*, *i.e., in the form of a logical formula to be proved later*".

These definitions are merged as follows and adopted in the *CREI* property modelling language (Cause Relation Effect Indicators) proposed in [16][17]:

> *[A property is] a provable or evaluable (i.e., quantifiable or qualifiable) characteristic of an artefact [that is 1) a system S, or 2) a model M of S built for achieving a design objective] that translates all or part of stakeholder expectations to be satisfied by this artefact.*

The goal is to help managers, architects and designers to formalise "what is" and "how" can evolve an "–ility" of a SoS i.e., how it can be influenced or dependent from 1) other characteristics of SoS, 2) conditions taking into account external and internal events, but also 3) characteristics related to each sub-systems of SoS or resulting from their interactions.

A *CREI* property is formalized as a composite entity made up of a group of causes (*C*) correlated with a group of effects (*E*) via a parameterized and constrained relation (*R*) between *C* and *E* describing the condition and the expected effects under which the property is satisfied. This relationship formally describes how the set of causes *C* induces a modification in the entire set of effects *E*. Moreover, a set *I* of indicators can be associated with *R* to make property assessable. These indicators are the *observation variable and Design variables*[2] [20]. For the formalization, we define the set Φ as the set of user-defined or predefined properties of the studied system. A *CREI Property CP* is defined as:

$$CP ::= < reference_{cp},\ C\ ,\ R\ ,\ E\ ,\ checkingValue_{cp},\ [I\ ,\ evaluationValue_{cp}] >$$

With:

- $reference_{cp} \in S$ is a handle (unique) for property proof traceability.
- $C = \{v_i / v_i \in ParametersSet \cup \Phi, i \in [0 ; card(ParametersSet \cup \Phi)]\}$, *i.e., C* can be empty ($C = \emptyset$): the property is then considered to be an *own property*[3], otherwise ($C <> \emptyset$) as a *composite property*[4].
- $E = \{ v_j / v_j \in F \cup \Phi, j \in]0 ; card(F \cup \Phi)]\}$ *i.e.,* $E <> \emptyset$.
- $R ::= <T_p, \theta_c , \theta_e, relationType, \theta_i >$, where:

---

[2] An *observation variable* allows modelling an expected performance level or an expected "*i-lity*" level. A *design variable* allows to handle and to set up values corresponding with potential design choices.

[3] An *own property* models expected values of a (set of) Parameter(s).

[4] A *composite property* is to be characterized by the causal relation.

– $T_p = C \cap E$ is the set of variables that may be simultaneously used for describing causes and effects.

– θ_c: $T_p^{\ k}$ x C^m x $\mathbb{R}^{+*n} \rightarrow$ { *True*, *False* } defines the Boolean function describing the condition under which the causes of *C* are interpreted. By default, the function θ_c returns *True (denoted* θ_c= True in the next):

$(t_1, \dots, t_k, c_1, \dots, c_m, r_1, \dots, r_n) \rightarrow \theta_c(t_1, \dots, t_k, c_1, \dots, c_m, r_1, \dots, r_n) \in$ { True, False }

– θ_e: $T_p^{\ o}$ x E^p x $\mathbb{R}^{+*q} \rightarrow$ { *True*, *False* } defines the Boolean function describing the condition under which the effects of *E* are interpreted. By default, the function θ_e returns *False (denoted* θ_e= False in the next):

$(t_1, \dots, t_o, e_1, \dots, e_p, r_1, \dots, r_q) \rightarrow \theta_e(t_1, \dots, t_o, e_1, \dots, e_p, r_1, \dots, r_q) \in$ { True, False }

– At this stage, *relationType* models the relation to be checked: *'C implies E'*, *'C is equivalent to E'*, or *'C influences E'* formalized as follows:

° '*C implies E*' is defined as the logical function: $\theta_c \Longrightarrow \theta_e$

° '*C is equivalent to E*' is defined as the logical function: $\theta_c \Longleftrightarrow \theta_e$

° *'C influences E'* is defined as the function: $\theta_c \dashrightarrow_{\theta i} \theta_e$. This relation is defined by [23] as "*in knowing with certainty C, we can then deduce E with certainty*", *i.e.,* knowing the values (and their variation) of the causes defined in *C* allows to deduce the possible values (and their potential variation) of effects defined in *E* by defining an *influence factor* $\theta_i \in$ [-1,1] allowing to interpret a beneficial *vs.* harmful influence as follows:

° $\theta_i \rightarrow 0$: the influence exists between the causes and effects remaining more or less neutral (by default, $\theta_i = 0$);

° $\theta_i \rightarrow 1$: each variation of the variables used in *C* induces a variation of the variables used in *E*, interpreted as beneficial for the system;

° $\theta_i \rightarrow -1$: each variation of the variables used in *C*, induces a variation of the variables used in *E*, interpreted as harmful for the system;

- *checkingValue*$_{cp}$ is set to True, else False (*i.e.,* if a checking technique can be applied on the model for proving the property *CP* and can conclude *CP* is satisfied, False otherwise, thus providing a counterexample).
- (*optional*) [*I*, *evaluationValue*$_{cp}$] $I \subseteq$ *ParametersSet* is a set of indicators that can be evaluated to characterize the truthfulness of the property *CP* (*e.g.* in case of simulation or for guiding the appraisal): $I = \{i_j / i_j \in ParametersSet, j \in [0, card(ParametersSet)]\}$, where i_j is a *Modeling Variable* extracted from the system model. In case of *I* is defined, then $CP.evaluationValue_{cp} = \mu(I)$, where μ is the *aggregation function* chosen for the evaluation, *e.g.* μ can be defined by $\mu ::= 1/card(I) * \sum_I (i.value)$. *CP* can be considered as satisfied as proposed in [19] *i.e.* $CP.checkingValue_{cp}$ = True if and only if $\forall i \in I$, i.*value* ∈ i.*objectif* ∧ i.*value* ∈ i.*valueset* otherwise not satisfied, *i.e.,* $CP.checkingValue_{cp}$ = False.

As an illustration, we focus on the next on the interoperability of each sub-system making up the SoS, considering the 'key enabling –ility' role of this characteristic [2] and its direct influence on the objectives of resilient SoS architecting project.

For this, we consider (see **Fig. 5**) there are some dependencies to respect (evaluated by using 5 levels of dependencies from preferred to forbidden) between the operating modes of the SoS and the operating modes of sub-system at each moment,

taking into account their dynamical evolution. For instance, when the SoS works and fulfil the mission (SoS is in Operating Mode O3), it seems important (++) that a maximum of sub-systems can be able to provide their own mission, and to operate efficiently (sub-systems have to be preferably in Operating Modes O1 to O5). Some of these sub-systems can also (+) be under deployment (operating Mode D1) for instance if these sub-systems have to replace existing sub-systems at a given moment, assuming then architectural evolution of the SoS. Conversely, it seems inacceptable (--) that a sub-system must be in dismantling operating mode (EL1 or EL2). These interdependencies are, of course, indicative: architect can modify them without any impact on the property formalisation and analysis that follows. Other tables can be built for maintainability, resilience or robustness as expected in [2].

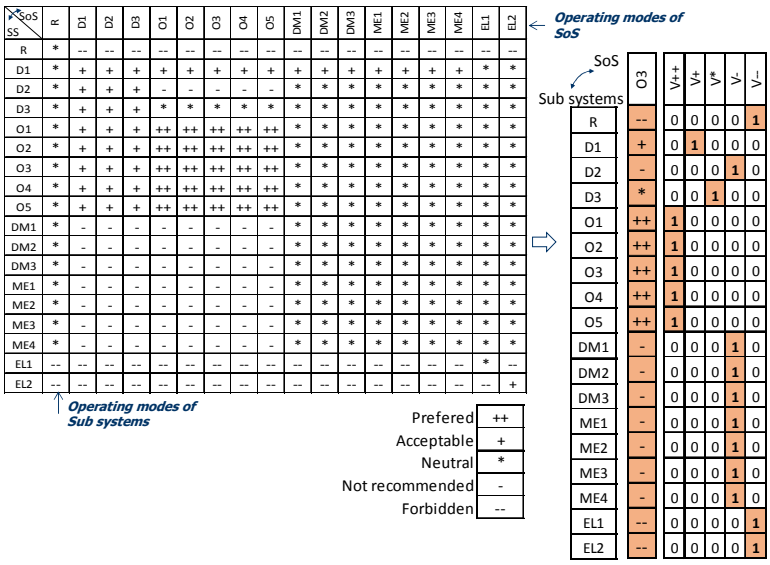

Fig. 5: operating modes dependencies considering requested interoperability

The interoperability characteristic of the SoS is formalized as the property *P* as follows:

1. **Cause** := $\forall OM_{SoS} \in$ OperatingModes(SoS) ; OMAG grid behaviour is translated into symbolic logico-temporal formulae. Each transition between Operating Modes selected by architect is modelled as an *Elementary Valid Formula* (*EVF*) [26] modified as follows[5]:

$$EVF ::= \quad (\mathbf{OM}_i \wedge \mathbf{event}_j \wedge \mathbf{condition}_k \supset \circ\mathbf{OM}_l \wedge \mathbf{scenario}_m)$$

With:

- $\mathbf{OM_i}$ and $\mathbf{OM_l}$ are propositional variables modelling the source and destination operating modes of the transition, and set to True if the studied system is in the corresponding operating mode, false otherwise.
- $\mathbf{event_j}$ and $\mathbf{condition_k}$ are propositional variables set to True if event and

[5] The formula o**A** means that the propositional variable **A** will be true at the next moment in a common logical and unified time scale.

condition associated to the transition are True, false otherwise.

The entire list of *EVFs* defines a symbolic and formal description of the behaviour of the OMAG grid. Similarly, a *Unified Valid Formula* (*UVF*) is computed by taking *EVFs* into consideration. Here considered, an *UVF* is the set of conditions i.e. θ_c which specifies how a given Operating Mode *OM* can be activated:

$$\theta_c := UVF(oOM_{SoS}) = \bigvee_{(p,q,r)} \text{OM}_{SoS} \wedge \text{event}_q \wedge \text{condition}_r$$

$$\text{OM}_{SoS} \wedge \text{event}_q \wedge \text{condition}_r \supset oOM_{SoS}$$

2. **Relation** := (influences) ; **Indicators** are user-defined and computed including parameters associated to sub-systems *e.g.* the latency time or interoperation time as proposed in [25].
3. **Effect** := $SS_i \in$ SubSystems(SoS) ; **Fig. 5** shows how dependency vectors (denoted V++, V+, V*, V- and V—) are computed regarding each state of the SoS for a top down analysis of the dependence relations. We focus on the preferred states of the sub-system i.e., those characterized by a dependence relation '++'. The state vector V of the SoS (resp. sub-system) is 1x18 vector defined as follows:

$$\exists! k \in [1,18], \left[(V(k) == 1) \underset{SoS \text{ is in } OM_i}{\Longrightarrow} \forall j \in [1,18], j \neq k, (V(j) == 0)\right]$$

(there are 18 possible operating modes in the current OMAG grid and the sub-system SS_i must be in one and only one operating mode OM_k)

So θ_e is defined as follows:

$$\forall i, \left[(V(OM(SS_i), t) * V_{++}^T == 1) \wedge \neg \prod_{T_k} [condition_i \wedge event_i)]\right] \vee [FVII(oOM(SS_i)]$$

Where:

- T_k is the set of output transitions from the preferred operating mode OM of the sub-system SS_i that is preferred when SoS is in operating mode OM_{SoS}.
- $(V(OM(SS_i), t) * V_{++}^T == 1) \wedge \neg \prod_{Ti}(condition_i \wedge event_i)]$==1 iff SS_i is in the state OM and will stay in this state at the next moment or if the state OM will be activated at the next moment.

The same computation process is used for determining θ_e in the case of dependence relation that indicates the operating mode of SS_i is acceptable, neutral, not recommended or forbidden. In the same way, the same computation approach is used in a bottom-up analysis regarding the dependence relations between each operating mode of each SS_i and SoS operating mode.

Simulation (following operational semantics given in [13]), evaluation of parameters and the generation of counter examples provided by checking technique proposed in [26] and developed in [27] are suitable for allowing architect to detect 1) modelling errors or mistakes, 2) unwanted or unexpected behaviour inducing non-functional properties variation. **Fig. 6** shows a big picture of the overall approach consisting to use OMAG and properties modelling, checking and evaluation in SAI approach.

6 Conclusion and Perspectives

A demonstrator of the OMAG grid and properties modelling tool is currently being tested. The automated properties building, taking into account various version of dependence tables, properties checking and OMAG grid simulation techniques are under development by using framework developed in [27]. The goal is now to test the overall approach on relevant case studies. The perspective is to enrich the two aforementioned analysis techniques when facing problematic of growing up models' size and complexity (*e.g.* due to number of OMAG to be considered).

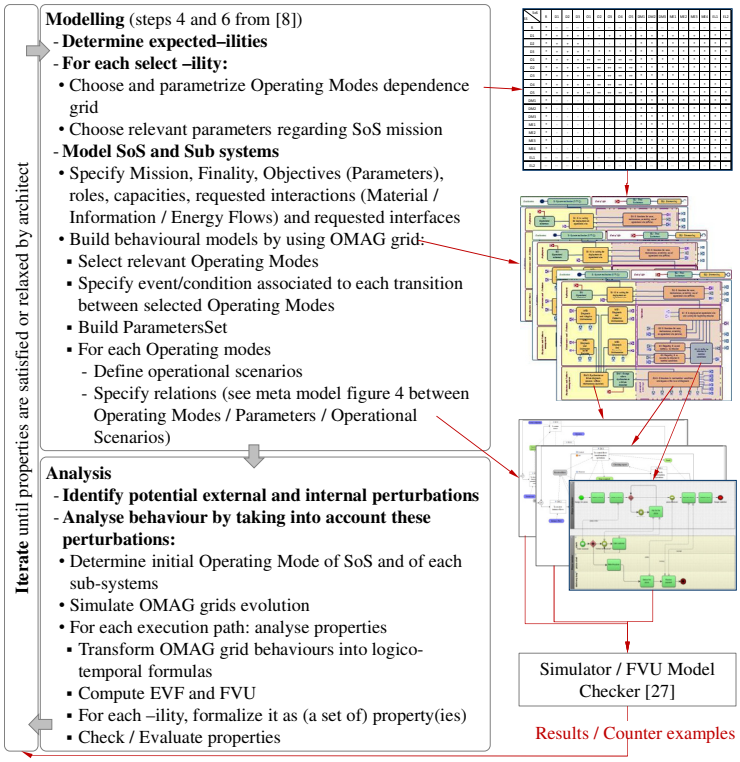

Fig. 6: Overview of the approach (big picture)

Acknowledgement. The authors thank A. Monfouga and R. Blainvillier for their involvement in the development of a demonstrator for modelling OMAG grid.

References

1. Maier, M.W.: Architecting principles for systems-of-systems. Syst. Eng. 1, 267–284, 1998
2. KANG Shian Chin, PEE Eng Yau, SIM Kok Wah, PANG Chung Khiang, Framework for managing System-of-systems ilities, DSTA HORIZONS, 2013/14

3. Timothy L.J. Ferris, It Depends: Systems of systems engineering requires new methods if you are talking about new kinds of systems of systems, INCOSE 2006, SoS panel (http://www.3milsys.com/sys_of_sys.asp [last visited 2011-07-12])
4. Weck O., Roos D. Magee C, Engineering Systems: Meeting Human Needs in a Complex Technological World, chapter 4, MIT Press, January 2012
5. INCOSE, System Engineering (SE) Handbook Working Group, System Engineering Handbook, A Guide For System Life Cycle Processes And Activities Version 3.2.1, INCOSE TP 2003 002 03.2., 2011
6. ESD Terms and Definitions (Version 12), ESD Symposium Committee, October 19, 2001, Massachusetts Institute of Technology Engineering Systems Division, Working Paper Series, SD-WP-2002-01
7. Adam M. Ross and Donna H. Rhodes, Towards a Prescriptive Semantic Basis for Change-type Ilities, 2015 Conference on Systems Engineering Research, Procedia Computer Science 44 (2015) 443 – 453
8. Nicola Ricci, Matthew E. Fitzgerald, Adam M. Ross, Donna H. Rhodes, Architecting Systems of Systems with Ilities: an Overview of the SAI Method, Systems Engineering Research (CSER 2014), March 21-22, 2014
9. Adam M. Ross, Adaptive and Resilient Space Systems Panel, AIAA Space 2011 Long Beach, CA 28 September, 2011
10. Warren K. Vaneman and Kostas Triantis, An Analytical Approach to Assessing Emergent Behavior is a System of System, SEDC 2014, Chantilly, VA, 3-5 April 2014
11. Hugh L. McManus, Matthew G. Richards, Adam M. Ross, and Daniel E. Hastings, A Framework for Incorporating “ilities” in Tradespace Studies, A Collection of Technical Papers - AIAA Space 2007 Conference, Vol. 1, pp. 941-954
12. Ke Dou, Xi Wang, Chong Tang, Adam Ross, Kevin Sullivan, An Evolutionary Theory-Systems Approach to a Science of the Ilities, Systems Engineering Research (CSER 2015)
13. Chapurlat V., Daclin N., Proposition of a guide for investigating, modeling and analyzing system operating modes: OMAG, Complex Systems Design and Management CSDM, 2013
14. Charles S.Wasson, System Analysis, Design and Development: concepts, principles and practices, Wiley Intersciences Eds., 2014
15. K.T.Cheng, A.S.Krishnakumar ‘Automatic functional test generation using the Extended Finite State Machine Model’, 30th ACM/IEEE Design Automation Conference, USA, 1993
16. Chapurlat V., UPSL-SE: A Model Verification Framework for Systems Engineering, Computers in Industry, Computers in Industry 64 (2013), pp. 581–597
17. Chapurlat V., Property concept and modelling language for Model-Based Systems Engineering (MBSE) context, Internal Research Report (access on demand),
18. Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/properties/ [last visited 2012-08-31]
19. Micouin, P., 2008, Toward a Property Based Requirements Theory: System Requirements Structured as a Semilattice, System Engineering, Vol. 11, Issue 3: 235-245, INCOSE/Wiley
20. A.Qamar, C.J.J.Paredis, Dependency modeling and model management in mechatronics, proceedings of ASME 2012, International Design Engineering Technical Conference and Computers and Information Engineering Conference, IDET/CIE 2012, August 2012, USA
21. Pallab Dasgupta, A roadmap for formal property verification, Springer, 2010, ISBN: 978-90-481-7185-9 (Bérard et al. 2001)
22. Bérard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen Ph. McKenzie P. Systems and Software verification: model checking techniques and tools, Springer, 2001
23. J. Pearl, "Reasoning with cause and effect" , UCLA Cognitive Systems Laboratory, Technical Report (R-265), July 1999. In AI Magazine, Vol. 23(1), 95-111, Spring 2002.

24. BKCASE Editorial Board. 2015. The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 1.3.2 R.D. Adcock (EIC). Hoboken, NJ: The Trustees of the Stevens Institute of Technology [last visited 2015-04-23].
25. Billaud S., Daclin N., Chapurlat V., Interoperability as a key concept for the control and evolution of the System of Systems (SoS) , 6th International Workshop on Enterprise Interoperability, to appear in IWEI Proc. LNCS Springer Verlag Series, May 2015, France
26. Chapurlat V., Larnac M., Dray G., Analysis and formal verification of Grafcet (FCCS) using Interpreted Sequential Machine, IEEE CESA'96, July 1996, Lille, France
27. Nastov B., Chapurlat V., Dony C., Pfister F., A verification approach from MDE applied to Model Based Systems Engineering: xeFFBD dynamic semantics, Complex Systems Design and Management CSDM 2014, December 2014, Paris, France

Appendix: OMAG grid